Data Transport Standard Specification

Version 2.0

May 21, 2007

Prepared by

PESC DTS Technical Workgroup

Executive Summary

Business Problem

Over time, the Higher Education community has developed multiple methods to exchange data and information with multiple trading partners and operating systems. Each partner has developed their own protocols, standards or utilized commercial products independently. The lack of a common standard to transport electronic data has caused each player in the Higher Education community to develop, maintain and support a wide variety of transport solutions and hindered its ability to use a common protocol across the Higher Education community. Differing transport methods make it difficult to support each trading partners proprietary needs. As technologies advance more organizations are looking for a real-time transport solution. Due to the lack of a standard transport, entities are forced to develop proprietary processes or purchase commercial products with licensing and royalties issues.

The primary means of transporting information today is the use of FTP and/or an electronic mail processes. Each of these has limitations and is not suitable for the exchange of information for real time processing or acknowledgement. Differing transport definitions and standards make it difficult to build interfaces that ensure simplicity and reliability as it relates to electronic exchange of data.

Solution

Since the Higher Education community share common points of interaction, the Data Transport workgroup was formed to develop a common solution for data transport with representation from each of the major players in the community.

The Data Transport Standard (DTS) is a specification for a web service architecture that enables entities to send and respond to any type of request (transaction, inquiry, report) utilizing standard web service protocols. DTS is a culmination of different specifications and defines how to use them together. DTS is a result of a PESC initiative to create a standard method to exchange data within the Higher Education community, regardless of the business process. It is a recommended replacement for POP3/SMTP (e-mail) and an industry wide solution for real-time or immediate requests. DTS offers a solution for transport and may coexist or replace FTP.

Web services are based on request-response patterns. These request-response patterns occur in a synchronous (uninterrupted) mode. DTS defines the request-response pattern to be used in a SOAP format. SOAP (Simple Object Access Protocol) is widely accepted and an open industry standard for exchanging messages. Some of the benefits of the SOAP standard are: speed, extensibility, interoperability and tool integration.

Below is a list of some of the advantages DTS offers to the Higher Education community as a transport standard.

- Guaranteed delivery – real time communication that confirms delivery and receipt, regardless if payload is processed real-time or not.

- Supports immediate and deferred processing requests.
- Single transport method for all business applications (payload insensitive – XML, flat file, binary file, etc).
- No need to analyze payload to determine type and destination.
- Can accommodate a variation of technical platforms (among schools, servicers, lenders, FAMS, guarantors and FSA).
- Highly secure – transparent encryption via HTTPS and authentication via X.509 digital certificates.
- Supports larger payloads.
- No distribution royalties.
- Uses open standards.
- Can be used both for internal and external communications.

The current landscape of the industry offers a unique opportunity for the Higher Education community to promote a common data transport standard. Many in the higher education community are either looking to develop a transport process or streamline and consolidate transports by the adoption or creation of a new transport process. The industry can be proactive and develop a standard that can meet the needs of multiple business sectors within our industry. PESC does not want to be reactive, as each sector develops its own solutions for data transport.

Table of Contents

Tables and Figures

Tables

1 Introduction

1.1 Overview

This document defines the Data Transport Standard (DTS), utilizing a set of non-proprietary Web Services specifications to transport data, along with clarifications and amendments to those specifications that provide interoperability. The contents were derived by creating reference implementations and from many Information Technology industry resources including, but not limited to:

- Simple Object Access Protocol (SOAP) 1.1
- Web Services Description Language (WSDL) 1.1
- Extensible Markup Language (XML) 1.0 (Second Edition)
- RFC2616: Hypertext Transfer Protocol -- HTTP/1.1
- Web Services Interoperability Basic Profile 1.0
- RFC1950: Zlib Protocol
- WS-Security Specification v1.1

1.2 Purpose

The purpose of this specification is to describe the Data Transport Standard (DTS) web service through the use of Web Service Definition Language (WSDL). The SOAP elements defined are the minimum required to transport data in a manner that meets the business problem. This document provides the exact minimum structure for the SOAP that is transmitted between a web service and its client. If two entities created a web service client and service that transmits the SOAP described here, the applications should communicate correctly without further work.

It is intended to be a guide for PESC members and the education community in general, to use when implementing the Data Transport Standard throughout the community.

1.3 Scope

This specification provides a technical guideline for a web service to electronically exchange data between organizations. It focuses on defining the transport layer, the DTS WSDL, and DTS SOAP.

While this document identifies the necessary SOAP elements for data exchange using web services, the actual values contained in the SOAP elements and those industry segments will specify usage definitions for DTS in different industry segments. Those industry segments will define these values.

The following topics are **not** within the scope of this specification:

- The business reasons for adopting DTS as a data transport
- The integration of DTS with the file processing backend system
- How immediate or deferred (batch) data processing is performed

1.4 Intended Audience

This document is intended for experienced developers. It targets readers that have an intermediate to expert level of knowledge on Web Services, XML and cryptography. It is not intended to be a comprehensive tutorial on these topics. Only those topics that must be clarified for the purposes of standardization are covered.

1.5 Organization of the Document

The Guidelines for the DTS Specification consists of the following sections:

- **Section 1 – Introduction:**
  - provides a high level overview, scope, and assumptions of this document
- **Section 2 – Overview and Description of DTS Specification:**
  - provides an introduction to DTS and highlights topics that readers should be familiar with before reading this document
- **Section 3 - DTS WSDL (Web Services Description Language)**
  - provides a high level overview of the purpose of the DTS WSDL and outlines the requirements for constructing the request and response SOAP
- **Section 4 - DTS SOAP Descriptions:**
  - describes how a DTS Client and Service produces the SOAP requests and responses that meet the requirements of the WSDL
- **Section 5 - Communication or Framework Errors Generated Outside DTS:**
  - describes a list of possible external errors that can be encountered when creating a web service
- **Section 6 - DTS Error Handling Requirements:**
  - provides a list of all error conditions defined within the DTS framework and how they are reported
- **Section 7 – Recommendations:**
  - provides recommendations for topics of consideration that are outside the scope of the specification but may impact an organizations implementation of DTS
- **Section 8 – Glossary:**
  - provides a list of terms and definitions
- **Section 9 – Acknowledgements**

2 OVERVIEW AND DESCRIPTION OF DTS SPECIFICATION

DTS is designed to be a standard for creating web services that provide real-time transfer of data. It is built on the standard request/response scenario of web services. It is designed so any type of business could use this as a transport mechanism. DTS is based on computer industry standards in the Web Services arena:

- **WSDL –** Describes what functionality a Web Service offers, how it communicates and where you can find it.
- **SOAP** - Specialized XML used as the basis for transporting data for Web Services
- **WS-Security -** Used for authentication and protection against "man in the middle" attacks.
- **HTTPS -** Used for encryption and communication layer

DTS is designed to provide functionality in the following areas:

- **Core Services –** Build SOAP based on the specification
- **Security** (Encryption/Authentication) - Ensure that the information is encrypted and authenticated
- **Assured Delivery –** Assures that the request gets to the designated endpoint
- **Application Integration –** Provides business application integration to gain access to the low level SOAP elements for processing
- **Error Handling –** Provides error handling for each area in the specification

3 SECURITY (Encryption/Authentication)

- **SSL (HTTPS)**

Organizations implementing a DTS Service, must obtain an X.509 certificate suitable for the site's web server thus enabling secure communications. The certificate must be obtained from a recognized certificate authority such as VeriSign® or similar organization. Each organization should consult its web server's documentation (or Technical Support staff) for details on obtaining and installing an X.509 certificate.

NOTE

Test certificates or self-signed certificates must not be used in production environments.

Communication using SSL (https) works as follows:

- The (DTS) client connects to the web server and as part of the normal connection process ("*handshake*") asks for the certificate.
- The web server, prior to the DTS Service, returns the certificate to the client,
- The client then determines which certificate authority authorized and signed the certificate and verifies that the signature is valid by using the public key for that authority.
- If the certificate is found to be valid, the two computers begin communication with encryption.
- If the certificate, domain name, or keys do not match up correctly, the connection is aborted.

While this is an over simplification of SSL encryption, there are two key items to point out. First, all of these steps are part of the https protocol and lie completely outside of a DTS implementation. Second, the public key exchange occurs during the "handshake" of client and server, so there is no need for manual exchange of the public key contained in the X.509 certificate. Commonly available Web Server applications and Web Browser applications have security via SSL built in.

- **WS-Security Specification**
  - This version of the DTS specification requires the inclusion of parts of the WS-Security Specification v1.1 to provide authentication.
    - Timestamp
      The <Timestamp> block is required. It provides a mechanism to ensure timely delivery of a message and protect against "man in the middle" attacks.

```
<wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        <wsu:Created>2006-10-24T15:58:33.984Z</wsu:Created>
        <wsu:Expires>2006-10-24T15:59:03.984Z</wsu:Expires>
</wsu:Timestamp>
```

    - Signature
      The <Signature> block is required. It provides a mechanism to ensure that a digital signature is included with the transmission. It is the primary way to authenticate the originator of the transmission.

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#id-15101782">
      <ds:Transforms>
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      </ds:Transforms>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>d4xBa1dR/Lp0ExBkaNQxT7oYrVw=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
<ds:SignatureValue>DoRhJCLVe4H3xZD71jEXizvOi03WD+psQdZj1pS4r8W33awYE01SX4r1xJBkB/MMO9u1fUoGJ
dgfus2ZCQh2zyC67bAErQIZdj9FgAZ9z1xXRmPwO4SYyxTqkix4YkaI1DIf7Fsk8WMm/zK+cu1i/w5jWmcNOe3uaI
IOrwM26b8=</ds:SignatureValue>
    <ds:KeyInfo Id="KeyId-20201168">
    <wsse:SecurityTokenReference wsu:Id="STRId-12413535" xmlns:wsu="http://docs.oasis-
     open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <wsse:Reference URI="#CertId-1563400" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
       wss-x509-token-profile-1.0#X509v3"/>
       </wsse:SecurityTokenReference>
    </ds:KeyInfo>
  </ds:Signature>
```

- Binary Security Token
  The <BinarySecurityToken> is required. The value contained in this element must be the X.509 certificate that is used to verify the digital signature in the <Signature> block. This will allow the receiver of the transmission to verify the signature without having to exchange the X.509 certificate out of band prior to the transmission.

<wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="CertId-1563400" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">

MIICXjCCAccCCQCMGanSPbP8XzANBgkqhkiG9w0BAQQFADCBhjELMAkGA1UEBhMCVVMxCzAJBgNVB
AgTAlBBMQ4wDAYDVQQHEwVFbm9sYTEPMA0GA1UEChMGUklTQ2lUMQwwCgYDVQQLEwNEVFMxF
zAVBgNVBAMTDk1hcmsgTWFsaW5vc2tpMSIwIAYJKoZIhvcNAQkBFhNtbWFsaW5vc0ByaXNjaXQuY29tMB
4XDTA2MDgyMzIzMjM1NVoXDTA5MDgyMjIzMjM1NVowYDELMAkGA1UEBhMCVVMxCzAJBgNVBAgT
Ak1BMQ8wDQYDVQQHEwZCb3N0b24xDzANBgNVBAoTBkNsaWVudDEMMAoGA1UECxMDRFRTMRQ
wEgYDVQQDEwtDbGllbnQgQ2VydDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA4Kv0JJj3R38AY2
N8NU8P9LQVgn//Wt7dFXzWIJrP7bvp434pSszPf3DIH183TlgMjrW39zlHxjvMDBa5bEqS+Rdgi/FbJFxyaa3Oa
uAuc75uvWYoPW/dLyP4bvygtewEy+aFV3zOuJKc76H5OLxsOHaOwQ3rl9UmGdC/x9nvF+cCAwEAATAN
BgkqhkiG9w0BAQQFAAOBgQCOUvZ7tP6beZEcly7JOkg+ctuMMRhAlw4R51SyJnKpcPPS1k39uquSF9zVCJY
Ru1CXLFj07yCGOfRr/+Eavfhnp0NJnF0q3YboP5b4CjnYrK34PkGDlAtJZ2znZ3LcyJmuHKrSrIPjAj46Qibk2MU
uoza4oCT8DSWs4VkPiI71Ng==

</wsse:BinarySecurityToken>

Certificate Validation

The certificate will also need to be validated to ensure its authenticity. The sections below outline the three areas of X.509 certificate validation:

1. Validating the Trust Chain
   This is a required operation of both a DTS service and client. The trust chain must be checked to ensure that a valid certificate from the PESC designated certificate authority(s) is in the certificate chain.

2. Expiration
   This is a required operation of both a DTS service and client. This check must be done to ensure the certificate is still within its current validity dates.

3. Revoked
   This is a required operation of both a DTS service and client. The certificate must be checked against the PESC designated certificate authority(s) revocation list to ensure that the X.509 certificate has not been compromised.

See the Section 7 for DTS error requirements for X.509 validation.

4 DTS WSDL (Web Service Definition Language)

The Web Services Definition Language (WSDL) is an XML-based language used to describe the services a business offers and to provide a way for individuals and other businesses to access those services electronically. It also provides a roadmap to how the SOAP will look in the Web Service transaction. The definition of the SOAP provides the foundation for this entire specification. Any organization implementing the specification must ensure that the SOAP produced from their applications conforms to the DTS WSDL.

There are five high level elements contained within a WSDL document. The table below summarizes each section.

Table 4.1 – WSDL Elements

Element Name	Description
Types	A container for abstract type definitions defined using XML Schema
message	A definition of an abstract message that may consist of multiple parts, each part may be of a different type
portType	An abstract set of operations supported by one or more endpoints (commonly known as an interface); operations are defined by an exchange of messages
binding	A concrete protocol and data format specification for a particular portType
Service	A collection of related endpoints, where an endpoint is defined as a combination of a binding and an address (URI)

4.1 Types

The WSDL **types** element is a container for XML Schema type definitions. The type definitions placed here are referenced from higher-level message definitions in order to define the structural details of the message. The **types** element contains zero or more schema elements from the http://www.w3.org/2001/XMLSchema namespace.

Below is an example of part of the **types** section defined in the DTS WSDL:

```
<xsd:element name="DTSRequestHeader">
        <xsd:complexType>
                <xsd:all>
                <xsd:element maxOccurs="1" minOccurs="1" name="DTSRequestRouting" type="tns:DTSRouting"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="DTSRequestServiceExpectation" type="xsd:string"/>
                 <xsd:element maxOccurs="1" minOccurs="1" name="DTSRequestPayloadType" type="xsd:string"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="DTSRequestSignature" type="xsd:string"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="DTSRequestPayloadBytes" type="xsd:string"/>
                </xsd:all>
        </xsd:complexType>
</xsd:element>
```

4.2 Messages

The WSDL **message** element defines an abstract message that can serve as the input or output of an operation. Messages consist of one or more part elements where each part is associated with an element. The messages and their parts must be named making it possible to refer to them from elsewhere in the WSDL definition.

Below is an example of part of the message section defined in the DTS WSDL:

```
<wsdl:message name="DTSRequest">
  <wsdl:part element="tns:DTSRequest" name="DTSRequest"/>
</wsdl:message>
<wsdl:message name="DTSResponse">
  <wsdl:part element="tns:DTSResponse" name="DTSResponse"/>
</wsdl:message>
```

4.3 Interfaces (portTypes)

The WSDL **portType** element defines a group of operations also known as an interface in most environments. A **portType** element contains zero or more operation elements. Each portType must be given a unique name making it possible to refer to it from elsewhere in the WSDL definition. Each operation element contains a combination of **input** and **output** elements, and when you have an **output** element you can also have a **fault** (exception) element.

Below is an example from the portTypes section defined in the DTS WSDL:

```
<wsdl:portType name="submitDTS">
  <wsdl:operation name="submitDTS">
    <wsdl:input message="tns:DTSRequest"/>
    <wsdl:output message="tns:DTSResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

4.4 Bindings

The WSDL **binding** element describes the concrete details of using a particular portType with a given protocol. The **binding** element contains several extensibility elements as well as a WSDL **operation** element for each operation in the portType which it describes. A binding must be given a unique name so you can refer to it from elsewhere in the WSDL definition. The binding must also specify which portType it describes through the type attribute.

Below is an example from the **bindings** section defined in the DTS WSDL:

```
<wsdl:binding name="submitDTS" type="tns:submitDTS">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="submitDTS">
    <wsdlsoap:operation soapAction="http://www.datatransportstandard.com/submitDTS"/>
    <wsdl:input>
        <wsdlsoap:body use="literal"/>
        <wsdlsoap:header message="tns:DTSRequestHeader" part="DTSRequestHeader" use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <wsdlsoap:body use="literal"/>
        <wsdlsoap:header message="tns:DTSResponseHeader" part="DTSResponseHeader" use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

4.5 Services

The WSDL **service** element defines a collection of ports, or endpoints, that expose a particular binding. You must give each port a name and assign it a binding. Then, within the port element, you use an **extensibility** element to define the address details specific to the binding.

Below is an example from the services section defined in the DTS WSDL:

```
<wsdl:service name="submitDTS">
  <wsdl:port binding="tns:submitDTS" name="submitDTS">
    <wsdlsoap:address location="http://localhost:8080/DTS2.0/services/ReferenceImplementation"/>
  </wsdl:port>
</wsdl:service>
```

Note: The Supplemental Documentation section of this document includes the full DTS WSDL.

5 DTS SOAP Descriptions

A DTS Client and Service must produce SOAP requests and responses that meet the requirements of the WSDL located in the supplemental documentation section of this document.

5.1 DTS SOAP Header Elements

The specification defines several header elements designed to allow any line of business the capability of transferring data without interrogating the request and/or response payload. Most of the header elements within the specification do not have any pre-determined values associated with them. This will allow each line of business to determine what values are necessary to perform their business functions. See the annotations (Specification defined vs. Business defined) beside each header element to determine if the value is necessary within the specification or to be defined by a line of business.

The following sections describe each of the DTS request and response SOAP packets.

5.2 DTS Request SOAP (See the supplemental section of this document for example SOAP)

Header Elements

- **<DTSRequestHeader> Element (Complex Type)**
  This element groups all of the header information for the DTS request.

  <DTSRequestRouting> Element (Complex Type - DTSRouting)

  <DTSRequestServiceExpectation> Element (Simple Type – xsd:string)
  (Business Defined)
  This element is to be used to identify how the transaction should be processed.

  <DTSRequestPayloadBytes> Element (Simple Type – xsd:string)
  (Specification Defined)
  This element holds the decompressed byte count of the request payload. Every DTS Service will have system resource constraints (memory, disk space, etc.) that should be taken into consideration when attempting to decompress the request payload. The byte count is required to provide a reliable method to assist the DTS Service in determining the best method to decompress the request payload.

<DTSRequestPayloadType> Element (Simple Type – xsd:string)
(Business Defined)
This element identifies the type of payload within the request.

The follow are examples of the <DTSRequestHeader>:

Java Example DTSRequestHeader SOAP Element

```
<dts:DTSRequestHeader soapenv:actor="urn:org:pesc:datatransport/dts"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" soapenv:mustUnderstand="1" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:dts="urn:org:pesc:datatransport">
 <dts:DTSRequestRouting>
    <dts:DTSUUID>936977f0-afd1-11db-b2d3-eaeb1a53a454</dts:DTSUUID>
    <dts:DTSSourceID>Mark</dts:DTSSourceID>
    <dts:DTSSourceIDSubCode>Malinoski</dts:DTSSourceIDSubCode>
    <dts:DTSRecipientID>Nathan</dts:DTSRecipientID>
    <dts:DTSRecipientIDSubCode>Chitty</dts:DTSRecipientIDSubCode>
 </dts:DTSRequestRouting>
 <dts:DTSRequestServiceExpectation>Immediate</dts:DTSRequestServiceExpectation>
 <dts:DTSRequestPayloadType>CRC:APPSEND</dts:DTSRequestPayloadType>
 <dts:DTSRequestPayloadBytes>54</dts:DTSRequestPayloadBytes>
</dts:DTSRequestHeader>
```

.Net Example DTSRequestHeader SOAP Element

```
<DTSRequestHeader xmlns:dts="urn:org:pesc:datatransport" soap:mustUnderstand="1"
soap:actor="urn:org:pesc:datatransport/dts" xmlns="urn:org:pesc:datatransport">
    <dts:DTSRequestRouting>
    <dts:DTSUUID>99309714-176a-42fa-80fe-29736753f1e0</dts:DTSUUID>
    <dts:DTSSourceID>Nathan</dts:DTSSourceID>
    <dts:DTSSourceIDSubCode>Chitty</dts:DTSSourceIDSubCode>
    <dts:DTSRecipientID>Mark</dts:DTSRecipientID>
    <dts:DTSRecipientIDSubCode>Malinoski</dts:DTSRecipientIDSubCode>
    </dts:DTSRequestRouting>
    <dts:DTSRequestPayloadType>CRC:APPSEND</dts:DTSRequestPayloadType>
    <dts:DTSRequestPayloadBytes>53</dts:DTSRequestPayloadBytes>
    <dts:DTSRequestServiceExpectation>Immediate</dts:DTSRequestServiceExpectation>
</DTSRequestHeader>
```

Body

- **<DTSRequest> (Type xsd:string)**
  (Business Defined)
  This is the payload of the transaction. The data in this element must be zlib compressed and base64 encoded.

Java Example submitDTSRequest SOAP Element

```
<soapenv:Body>
    <DTSRequest xmlns="urn:org:pesc:datatransport">
        eJwLSk1LLUrNS05V8MwtyEnNTc1LLMnMz1MISKzMyU9MUQQAyZIMDQ==
    </DTSRequest>
</soapenv:Body>
```

.Net Example submitDTSRequest SOAP Element

```
<soap:Body>
    <DTSRequest xmlns="urn:org:pesc:datatransport">eJwLzswtyEIVCEktLgEAGNYEKw==</DTSRequest>
</soap:Body>
```

5.3 DTS Response SOAP (See the supplemental section of this document for example SOAP)

Header Elements

- **<DTSResponseHeader> Element (Complex Type)**
  This element identifies all of the header information for the response.

  <DTSResponseRouting> Element (Complex Type - DTSRouting)

  <DTSResponsePayloadType> Element (Simple Type – xsd:string)
  (Business Defined)
  This element is to be used to identify the type of payload within the response.

  <DTSResponseAcknowledge> Element (Simple Type – xsd:string)
  (Business Defined)
  This element is used to identify how the Service handled or will handle the request payload.

  <DTSResponsePayloadBytes> Element (Simple Type – xsd:string)
  (Specification Defined)
  This element holds the decompressed byte count of the response payload. Every DTS Client will have system resource constraints (memory, disk space, etc.) that should be taken into consideration when attempting to decompress the response payload. The byte count is required to provide a reliable method to assist the DTS Client in determining the best method to decompress the response payload.

  The follow are examples of the <DTSResponseHeader>:

  Java Example DTSResponseHeader SOAP Element

```
<dts:DTSResponseHeader soapenv:actor="urn:org:pesc:datatransport/dts"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" soapenv:mustUnderstand="1" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:dts="urn:org:pesc:datatransport">
    <dts:DTSResponseAcknowledge>Deferred</dts:DTSResponseAcknowledge>
    <dts:DTSResponsePayloadBytes>62</dts:DTSResponsePayloadBytes>
    <dts:DTSResponsePayloadType>CRC:RESPONSE</dts:DTSResponsePayloadType>
    <dts:DTSResponseRouting>
        <dts:DTSRecipientID>Nathan</dts:DTSRecipientID>
        <dts:DTSRecipientIDSubCode>Chitty</dts:DTSRecipientIDSubCode>
        <dts:DTSSourceID>Mark</dts:DTSSourceID>
        <dts:DTSSourceIDSubCode>Malinoski</dts:DTSSourceIDSubCode>
        <dts:DTSUUID>99309714-176a-42fa-80fe-29736753f1e0</dts:DTSUUID>
    </dts:DTSResponseRouting>
</dts:DTSResponseHeader>
```

  .Net Example DTSResponseHeader SOAP Element

```
<DTSResponseHeader xmlns:dts="urn:org:pesc:datatransport" xmlns="urn:org:pesc:datatransport">
   <dts:DTSResponseRouting>
            <dts:DTSUUID>ed5a7410-afd3-11db-8ded-f1b1cb25f748</dts:DTSUUID>
            <dts:DTSSourceID>Nathan</dts:DTSSourceID>
            <dts:DTSSourceIDSubCode>Chitty</dts:DTSSourceIDSubCode>
            <dts:DTSRecipientID>Mark</dts:DTSRecipientID>
            <dts:DTSRecipientIDSubCode>Malinoski</dts:DTSRecipientIDSubCode>
   </dts:DTSResponseRouting>
```

```
<dts:DTSResponsePayloadType>CRC:RESPONSE</dts:DTSResponsePayloadType>
<dts:DTSResponsePayloadBytes>54</dts:DTSResponsePayloadBytes>
<dts:DTSResponseAcknowledge>Deferred</dts:DTSResponseAcknowledge>
</DTSResponseHeader>
```

Body

- **<DTSResponse> (Type xsd:string)**
  (Business Defined)
  This is the response of the transaction. The data in this element must be zlib compressed and base64 encoded.

Java Example DTSResponse SOAP Element

```
<soapenv:Body>
    <DTSResponse xmlns="urn:org:pesc:datatransport">
        eJxzCQlWCEpNSy1KzUtOVfDMLchJzU3NK0ksyczPA0oUF+TnFacCAP0rDfA=
    </DTSResponse>
</soapenv:Body>
```

.Net Example DTSResponse SOAP Element

```
<soap:Body>
    <DTSResponse xmlns="urn:org:pesc:datatransport">
        eJwLTi0qy0xOdc7PLShKLS5OTdExstAJhgiG5iEJGxrChMNSizLTKn2DdYxNjAHxPBa4
    </DTSResponse>
</soap:Body>
```

5.4 DTS Common Header Element (See the supplemental section of this document for example SOAP)

<DTSRouting> Element (Complex Type)
Common header element used by both the <DTSRequestHeader> and <DTSResponseHeader>.

<DTSSourceID> (Simple Type – xsd:string) (Business Defined)
Identifies the source of the request/response.

<DTSSourceSubCode> (Simple Type – xsd:string) (Business Defined)
Identifies a secondary code that helps identify the source of the request/response.

<DTSRecipientID> (Simple Type – xsd:string) (Business Defined)
Identifies the recipient of the request/response.

<DTSRecipientSubCode> (Simple Type – xsd:string) (Business Defined)
Identifies a secondary code that helps identify the recipient of the request/response.

<DTSUUID> (Simple Type – xsd:string) (Business Defined)
Is a unique identifier for the request/response. See the recommendations section of this document or further clarification of UUID.

6 COMMUNICATION OR FRAMEWORK ERRORS GENERATED OUTSIDE DTS

The following is a non-inclusive list of possible external errors that should be given consideration in conjunction with a DTS Implementation:

TCP/IP Specification Errors

HTTP/S Specification Errors
e.g. 400/500/505

Web Service errors generated natively by programming toolkits (SDK)
e.g. De-serialzation Errors for child elements of a SOAP header element missing.

7 DTS ERROR HANDLING REQUIREMENTS:

Each component of a DTS enabled application must provide error handling and report exceptions to the appropriate component.

7.1 DTS Service Error Handling Table

The table below defines the three categories that can produce an error.

Table 7.1 – DTS Fault Summary Table

Element or Condition	Element is Missing	Element Exists but Length is Zero	Processing Failure
WS-Security Aggregate	DTS.SECURITY.001	N/A	See the WS-Security Error Handling section below.
DTSRequestHeader Aggregate	DTS.HEADER.001		
DTSRouting Aggregate			
DTSSourceID		DTS.HEADER.002	
DTSRecipientID		DTS.HEADER.002	
DTSUUID		DTS.HEADER.002	
DTSRequestPayloadType		DTS.HEADER.003	
DTSRequestServiceExpectation		DTS.HEADER.004	
DTSRequestPayloadBytes		DTS.HEADER.005	
Compressed, encoded Payload		DTS.PAYLOAD.001	
Payload cannot be decoded			DTS.PAYLOAD.002
Payload cannot be decompressed			DTS.PAYLOAD.003
Payload too large to decompress			DTS.PAYLOAD.004
Application unable to process transaction			DTS.APPLICATION

7.2 WS-Security Error Handling Requirements (Processing Errors)

Error handling for WS-Security is described in Section 12 of the WS-Security Specification v1.1 . The WS-Security specification explicitly outlines the fault requirements if errors occur while processing the WS-Security header block. Any organization implementing this version of DTS should follow the WS-Security specification for reporting errors while processing the WS-Security header block.

If the WS-Security element is not present in the SOAP packet see the DTS WS-Security Error Handling section below.

7.3 DTS Faults Summary

Detailed fault descriptions are provided below:

Detail Error Conditions Reported to DTS Client via SOAP Fault

- **Error handling for DTSRequestHeader Element**

  Condition 1 - DTSRequestHeader Element is null (not found in SOAP)
  Fault Code: DTS.HEADER.001
  Fault String: DTS SOAP Header Error with DTS Request Header Information.
  Fault Detail: DTS Request Header Information is not set in the SOAP Packet

- **Error handling for DTSRouting Element**

  The DTSRouting Element fault conditions should be combined if more than 1 of the error conditions exists.

  Condition 1 - Length of DTSSourceID = 0
  Fault Code: DTS.HEADER.002
  Fault String: DTS SOAP Header Error with DTS Routing Information.
  Fault Detail: DTS Source Id value is missing.

  Condition 2 - Length of DTSRecipientID = 0
  Fault Code: DTS.HEADER.002
  Fault String: DTS SOAP Header Error with DTS Routing Information.
  Fault Detail: DTS Recipient Id value is missing.

  Condition 3 - Length of DTSUUID = 0
  Fault Code: DTS.HEADER.002
  Fault String: DTS SOAP Header Error with DTS Routing Information.
  Fault Detail: DTS UUID value is missing.

- **Error handling for DTSRequestPayloadType Element**

  Condition 1 - Length of DTSRequestPayloadType = 0
  Fault Code: DTS.HEADER.003
  Fault String: DTS SOAP Header Error with DTS Payload Type Information
  Fault Detail: DTS Request Payload Type value is missing.

- **Error handling for DTSRequestServiceExpectation Element**

  Condition 1 - Length of DTSRequestServiceExpectation = 0
  Fault Code: DTS.HEADER.004
  Fault String: DTS SOAP Header Error with DTS Request Service Expectation Information
  Fault Detail: DTS Request Service Expectation value is missing

- **Error handling for DTSRequestPayloadBytes Element**

  Condition 1 - Length of DTSRequestPayloadBytes = 0
  Fault Code: DTS.HEADER.005
  Fault String: DTS SOAP Header Error with DTS Request Payload Bytes Information
  Fault Detail: DTS Request Payload Bytes value is missing

- **Error handling for DTSRequest**

  Condition 1 - Length of Compressed and Encoded Payload = 0
  Fault Code: DTS.PAYLOAD.001
  Fault String: DTS Payload Error
  Fault Detail: DTS Payload is missing.

Note: This is a valid check since compressing and encoding a zero byte string will always result in a non-zero byte length string.

Condition 2 - Payload cannot be decoded
Fault Code: DTS.PAYLOAD.002
Fault String: DTS Payload Error
Fault Detail: DTS Payload decoding error.

Condition 3 - Payload cannot be decompressed
Fault Code: DTS.PAYLOAD.003
Fault String: DTS Payload Error
Fault Detail: DTS Payload decompress error.

Condition 4 - Payload too large
Fault Code: DTS.PAYLOAD.004
Fault String: DTS Payload Error
Fault Detail: DTS Payload could not be decompressed because of size limitations.

- **Error handling for DTS WS-Security**

Condition 1 –Security Header Element not found.

Fault Code: DTS.SECURITY.001
Fault String: DTS Security Error
Fault Detail: Could not find WS-Security Element.

Service Backend Processing Error Handling

If the backend processing of a DTS Service cannot be invoked or encounters errors during processing the error conditions must be reported to the DTS Client via the following SOAP fault:

Any Condition

Fault Code: DTS.APPLICATION
Fault String: Backend Processing Error
Fault Detail: Provided by the DTS Service Application.

Example DTSRequestRouting Fault SOAP Element

```
<soapenv:Fault>
   <faultcode>soapenv:DTS.HEADER.001</faultcode>
   <faultstring>DTS SOAP Header Error with DTS Routing Information</faultstring>
   <detail>
      <string> DTS Source Id value is missing.</string>
   </detail>
   <detail>
      <string>DTS UUID value is missing.</string>
   </detail>
</soapenv:Fault>
```

7.4 DTS Client Error Handling

The DTS Client must process any SOAP faults received by the called service in accordance to the implementation specific requirements for error handling.

In addition, the DTS response SOAP must meet the following requirements:

- The WS-Security block must be present in the SOAP packet and error handling should be done in accordance to Section 12 of the WS-Security Specification v1.1.

- DTSResponseHeader Element is not null (must be present in SOAP)

- Length of DTSSourceID must be > 0
- Length of DTSRecipientID must be > 0
- Length of DTSUUID must be > 0

- Length of DTSResponsePayloadType must be > 0
- Length of DTSResponseAcknowledge must be > 0
- Length of DTSResponsePayloadBytes must be > 0

If the DTS response SOAP violates any of the above requirements, action taken by the DTS client will be implementation specific for each receiving entity.

7.5 SUPPLEMENTAL DOCUMENTATION

This section contains additional documentation to assist entities to understand the specification. It includes the DTS Web Service Definition Language (WSDL), which describes the web service that is created using the specification and example SOAP Request and Response packets for both .Net and JAVA.

DTS 2.0 WSDL

```
<!-- ********************************************************************************************************************************************************* -->
<!--                                                  Data Transport Standard                                                                                  -->
<!--                                                  Web Service Definition Language (WSDL)                                                                   -->
<!--                                                  Version 2.0.0                                                                                            -->
<!--                                                  01/29/2007                                                                                               -->
<!-- ********************************************************************************************************************************************************* -->
<wsdl:definitions targetNamespace="urn:org:pesc:datatransport" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:org:pesc:datatransport" xmlns:intf="urn:org:pesc:datatransport" xmlns:tns="urn:org:pesc:datatransport"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
  <xsd:schema elementFormDefault="qualified" targetNamespace="urn:org:pesc:datatransport"
  xmlns:tns="urn:org:pesc:datatransport" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:import namespace="ttp://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
 schemaLocation="http://docs.oasis-open.org/wss/v1.1/oasis-200401-wss-wssecurity-secext-1.0.xsd"/>
<xsd:element name="DTSRequestHeader">
    <xsd:complexType>
     <xsd:all>
                <xsd:element maxOccurs="1" minOccurs="1" name="DTSRequestRouting" type="tns:DTSRouting"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="DTSRequestServiceExpectation" type="xsd:string"/>
                <xsd:element maxOccurs="1" minOccurs="1" name="DTSRequestPayloadType" type="xsd:string"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="DTSRequestPayloadBytes" type="xsd:string"/>
    </xsd:all>
    </xsd:complexType>
</xsd:element>
<xsd:element name="DTSResponseHeader">
    <xsd:complexType>
     <xsd:all>
                <xsd:element name="DTSResponseRouting" type="tns:DTSRouting"/>
                <xsd:element name="DTSResponseAcknowledgement" type="xsd:string"/>
                <xsd:element name="DTSResponsePayloadType" type="xsd:string"/>
                <xsd:element maxOccurs="1" minOccurs="0" name="DTSResponsePayloadBytes" type="xsd:string"/>
     </xsd:all>
     </xsd:complexType>
</xsd:element>
<xsd:complexType name="DTSRouting">
    <xsd:all>
               <xsd:element maxOccurs="1" minOccurs="1" name="DTSUUID" type="xsd:string"/>
               <xsd:element maxOccurs="1" minOccurs="1" name="DTSSourceID" type="xsd:string"/>
               <xsd:element maxOccurs="1" minOccurs="0" name="DTSSourceSubCode" type="xsd:string"/>
               <xsd:element maxOccurs="1" minOccurs="0" name="DTSRecipientID" type="xsd:string"/>
               <xsd:element maxOccurs="1" minOccurs="0" name="DTSRecipientSubCode" type="xsd:string"/>
    </xsd:all>
</xsd:complexType>
   <xsd:element name="DTSRequest" type="xsd:string"/>
   <xsd:element name="DTSResponse" type="xsd:string"/>
  </xsd:schema>
 </wsdl:types>
   <wsdl:message name="DTSRequest">
      <wsdl:part element="tns:DTSRequest" name="DTSRequest"/>
   </wsdl:message>
   <wsdl:message name="DTSResponse">
      <wsdl:part element="tns:DTSResponse" name="DTSResponse"/>
   </wsdl:message>
```

```
  <wsdl:portType name="submitDTS">
    <wsdl:operation name="submitDTS">
      <wsdl:input message="tns:DTSRequest"/>
      <wsdl:output message="tns:DTSResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="submitDTS" type="tns:submitDTS">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="submitDTS">
      <wsdlsoap:operation soapAction="http://www.datatransportstandard.com/submitDTS"/>
      <wsdl:input>
          <wsdlsoap:body use="literal"/>
          <wsdlsoap:header message="tns:DTSRequestHeader" part="DTSRequestHeader" use="literal"/>
      </wsdl:input>
      <wsdl:output>
          <wsdlsoap:body use="literal"/>
          <wsdlsoap:header message="tns:DTSResponseHeader" part="DTSResponseHeader" use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="submitDTS">
    <wsdl:port binding="tns:submitDTS" name="submitDTS">
      <wsdlsoap:address location="http://localhost:8080/DTS2.0/services/ReferenceImplementation"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Entire Java DTS SOAP Request:


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
      <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
        <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
wsu:Id="CertId-1563400" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">MIICXjCCAccCCQCMGanSPbP8XzANBgkqhkiG9w0BAQQFADCBhjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAlBBMQ
4wDAYDVQQHEwVFbm9sYTEPMA0GA1UEChMGUklTQ2lUMQwwCgYDVQQLEwNEVFMxFzAVBgNVBAMTDk1hcmsgTWFsaW
5vc2tpMSIwIAYJKoZIhvcNAQkBFhNtbWFsaW5vc0ByaXNjaXQuY29tMB4XDTA2MDgyMzIzMjM1NVoXDTA5MDgyMjIzMjM1NVow
YDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk1BMQ8wDQYDVQQHEwZCb3N0b24xDzANBgNVBAoTBkNsaWVudDEMMAoGA
1UECxMDRFRTMRQwEgYDVQQDEwtDbGllbnQgQ2VydDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA4Kv0JJj3R38AY2N
8NU8P9LQVgn//Wt7dFXzWIJrP7bvp434pSszPf3DIH183TlgMjrW39zIHxjvMDBa5bEqS+Rdgi/FbJFxyaa3OauAuc75uvWYoPW/dLyP
4bvygtewEy+aFV3zOuJKc76H5OLxsOHaOwQ3rl9UmGdC/x9nvF+cCAwEAATANBgkqhkiG9w0BAQQFAAOBgQCOUvZ7tP6beZEc
ly7JOkg+ctuMMRhAlw4R51SyJnKpcPPS1k39uquSF9zVCJYRu1CXLFj07yCGOfRr/+Eavfhnp0NJnF0q3YboP5b4CjnYrK34PkGDlAt
JZ2znZ3LcyJmuHKrSrIPjAj46Qibk2MUuoza4oCT8DSWs4VkPiI71Ng==</wsse:BinarySecurityToken>
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <ds:Reference URI="#id-1670071">
              <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              </ds:Transforms>
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>n9tgS63MW6NC4aLSadxxkji904o=</ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
<ds:SignatureValue>1rxlMRHDMZ3fORxQ+9Rvqm+wdNCz2ro61Oa/zc37hc0hGGWCEeIgvhS0uKNJUzANpBZknifT/dHiEA+wKFTa
L1CRvv4M77exDi9rcWBBqNu8wZmaRHsxhHsnAIwUFE5zsFfi/k7LES8tcmqLRqHvPyiVjXxGsFr51ZT9NoA1ITQ=</ds:SignatureVal
ue>
          <ds:KeyInfo Id="KeyId-12413535">
            <wsse:SecurityTokenReference wsu:Id="STRId-9936523" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd">
              <wsse:Reference URI="#CertId-1563400" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"/>
            </wsse:SecurityTokenReference>
          </ds:KeyInfo>
        </ds:Signature>
        <wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
          <wsu:Created>2007-01-29T20:01:40.796Z</wsu:Created>
          <wsu:Expires>2007-01-29T20:02:10.796Z</wsu:Expires>
        </wsu:Timestamp>
      </wsse:Security>
      <dts:DTSRequestHeader soapenv:actor="urn:org:pesc:datatransport/dts"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" soapenv:mustUnderstand="1" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:dts="urn:org:pesc:datatransport">
        <dts:DTSRequestRouting>
          <dts:DTSUUID>889ccfa0-afd3-11db-b8aa-c72490605f03</dts:DTSUUID>
          <dts:DTSSourceID>Mark</dts:DTSSourceID>
          <dts:DTSSourceIDSubCode>Malinoski</dts:DTSSourceIDSubCode>
          <dts:DTSRecipientID>Nathan</dts:DTSRecipientID>
          <dts:DTSRecipientIDSubCode>Chitty</dts:DTSRecipientIDSubCode>
        </dts:DTSRequestRouting>
        <dts:DTSRequestServiceExpectation>Immediate</dts:DTSRequestServiceExpectation>
        <dts:DTSRequestPayloadType>CRC:APPSEND</dts:DTSRequestPayloadType>
        <dts:DTSRequestPayloadBytes>54</dts:DTSRequestPayloadBytes>
      </dts:DTSRequestHeader>
    </soapenv:Header>
    <soapenv:Body wsu:Id="id-1670071" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
      <ns1:DTSRequest
xmlns:ns1="urn:org:pesc:datatransport">eJwLTsxTcMlMTc9XyCxWSFRIL0pNLFEoyElMTlUoyVcoyyzOLFFIzEsBCVUqlGeWZCi4h
ATrAQAAQxKQ</ns1:DTSRequest>
    </soapenv:Body>
  </soapenv:Envelope>
```

Entire Java DTS SOAP Response:


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
wsu:Id="CertId-1563400" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">MIICXjCCAccCCQCMGanSPbP8XzANBgkqhkiG9w0BAQQFADCBhjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAlBBMQ
4wDAYDVQQHEwVFbm9sYTEPMA0GA1UEChMGUklTQ2lUMQwwCgYDVQQLEwNEVFMxFzAVBgNVBAMTDk1hcmsgTWFsaW
5vc2tpMSIwIAYJKoZIhvcNAQkBFhNtbWFsaW5vc0ByaXNjaXQuY29tMB4XDTA2MDgyMzIzMjM1NVoXDTA5MDgyMjIzMjM1NVow
YDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk1BMQ8wDQYDVQQHEwZCb3N0b24xDzANBgNVBAoTBkNsaWVudDEMMAoGA
1UECxMDRFRTMRQwEgYDVQQDEwtDbGllbnQgQ2VydDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA4Kv0JJj3R38AY2N
8NU8P9LQVgn//Wt7dFXzWIJrP7bvp434pSszPf3DIH183TlgMjrW39zIHxjvMDBa5bEqS+Rdgi/FbJFxyaa3OauAuc75uvWYoPW/dLyP
4bvygtewEy+aFV3zOuJKc76H5OLxsOHaOwQ3rl9UmGdC/x9nvF+cCAwEAATANBgkqhkiG9w0BAQQFAAOBgQCOUvZ7tP6beZEc
ly7JOkg+ctuMMRhAlw4R51SyJnKpcPPS1k39uquSF9zVCJYRu1CXLFj07yCGOfRr/+Eavfhnp0NJnF0q3YboP5b4CjnYrK34PkGDlAt
JZ2znZ3LcyJmuHKrSrIPjAj46Qibk2MUuoza4oCT8DSWs4VkPiI71Ng==</wsse:BinarySecurityToken>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#id-10893564">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>SuXOn6+mD8eowNt3VtajUVEsrvw=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>

<ds:SignatureValue>sIFP5FPgcnaQ4lIUFK8WsHn5tdXVW64OH0J5N/w+NEQunXa2a4WdsEQHcx3AttASw/F+b6/kGNqpFoowUnB
+z4rE4TzBRU7l7xie8xBkpihdqeef/CqbEe4eRPYgcPKazUexNme66n6SU/wsc2VBmnPn1cAw/8/GiGvu3A9lDuE=</ds:SignatureValu
e>
      <ds:KeyInfo Id="KeyId-5484244">
        <wsse:SecurityTokenReference wsu:Id="STRId-33223668" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd">
          <wsse:Reference URI="#CertId-1563400" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"/>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </ds:Signature>
    <wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <wsu:Created>2007-01-29T20:01:41.265Z</wsu:Created>
      <wsu:Expires>2007-01-29T20:02:11.265Z</wsu:Expires>
    </wsu:Timestamp>
  </wsse:Security>
  <dts:DTSResponseHeader soapenv:actor="urn:org:pesc:datatransport/dts"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" soapenv:mustUnderstand="1" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:dts="urn:org:pesc:datatransport">
    <dts:DTSResponseAcknowledge>Deferred</dts:DTSResponseAcknowledge>
    <dts:DTSResponsePayloadBytes>62</dts:DTSResponsePayloadBytes>
    <dts:DTSResponsePayloadType>CRC:RESPONSE</dts:DTSResponsePayloadType>
    <dts:DTSResponseRouting>
      <dts:DTSRecipientID>Mark</dts:DTSRecipientID>
      <dts:DTSRecipientIDSubCode>Malinoski</dts:DTSRecipientIDSubCode>
      <dts:DTSSourceID>Nathan</dts:DTSSourceID>
      <dts:DTSSourceIDSubCode>Chitty</dts:DTSSourceIDSubCode>
      <dts:DTSUUID>889ccfa0-afd3-11db-b8aa-c72490605f03</dts:DTSUUID>
    </dts:DTSResponseRouting>
  </dts:DTSResponseHeader>
 </soapenv:Header>
 <soapenv:Body wsu:Id="id-10893564" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
  <DTSResponse
xmlns="urn:org:pesc:datatransport">eJwLTi0qy0xOdc7PLShKLS5OTdExM9AJhgiG5iEJm5rAhMNSizLTKn2DdQx0ANpeFn0=</DT
SResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Entire .NET DTS SOAP Request:


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <soap:Header>
    <DTSRequestHeader xmlns:dts="urn:org:pesc:datatransport" soap:mustUnderstand="1" soap:actor="urn:org:pesc:datatransport/dts" xmlns="urn:org:pesc:datatransport">
      <dts:DTSRequestRouting>
        <dts:DTSUUID>99309714-176a-42fa-80fe-29736753f1e0</dts:DTSUUID>
        <dts:DTSSourceID>Nathan</dts:DTSSourceID>
        <dts:DTSSourceIDSubCode>Chitty</dts:DTSSourceIDSubCode>
        <dts:DTSRecipientID>Mark</dts:DTSRecipientID>
        <dts:DTSRecipientIDSubCode>Malinoski</dts:DTSRecipientIDSubCode>
      </dts:DTSRequestRouting>
      <dts:DTSRequestPayloadType>CRC:APPSEND</dts:DTSRequestPayloadType>
      <dts:DTSRequestPayloadBytes>53</dts:DTSRequestPayloadBytes>
      <dts:DTSRequestServiceExpectation>Immediate</dts:DTSRequestServiceExpectation>
    </DTSRequestHeader>
    <wsse:Security soap:mustUnderstand="1">
      <wsu:Timestamp wsu:Id="Timestamp-ba2a92ea-e368-4049-a796-a2fd65581db9">
        <wsu:Created>2007-01-29T19:48:59Z</wsu:Created>
        <wsu:Expires>2007-01-29T19:53:59Z</wsu:Expires>
      </wsu:Timestamp>
      <wsse:BinarySecurityToken ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="SecurityToken-122cb9a7-82de-40a5-91e5-4a4a5cf00b90">MIICcjCCAdsCCQDMhQGu43qMhzANBgkqhkiG9w0BAQQFADCBhjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAlBBMQ4wDAYDVQQHEwVFbm9sYTEPMA0GA1UEChMGUklTQ2IUMQwwCgYDVQQLEwNEVFMxFzAVBgNVBAMTDk1hcmsgTWFsaW5vc2tpMSIwIAYJKoZIhvcNAQkBFhNtbWFsaW5vc0ByaXNjaXQuY29tMB4XDTA2MDgyMTE1NDI1NloXDTA5MDgyMDE1NDI1NlowdDEUMBIGA1UEAxMLamF4LTMyMDUzMGwxEjAQBgNVBAsTCU5hdGhhbkRUUzESMBAGA1UEChMJTmF0aGFuRFRTMRUwEwYDVQQHEwxKYWNrc29udmlsbGUxEDAOBgNVBAgTB0Zsb3JpZGExCzAJBgNVBAYTAlVTMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC0nwGCXlgD9MmxQtDu/HyFicJG64p3fRGHFgEuucOWrIJaWQBxl+9M5Lyq9eVaCNv3odINkSKMI5z/oAlshfew5uPK3rT+h/iSAiUB4AkxQ+p8FT0YpVFqVJRz31FffopcvIskGBfx9p0oH2b5GBhjz/wtAnIYroOUJlgpWAmQrwIDAQABMA0GCSqGSIb3DQEBBAUAA4GBAFkh3kiwegZlW8MzQEjtUPGPxRm0JutI05wGJYB7Ivmq96S5sm5oLV4KjBXQd30sKd2p2d+wDlvwHBN9YhlK4w5OvDEKE11eA/ryFdNukwmftbQo/DyiqI2ZDLEC+0ADdUsSpGPCjQlydCh6gK4L42nFtyM9NV7MsDEkrpbxiz1B</wsse:BinarySecurityToken>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <Reference URI="#Id-2fc19e22-84e5-4b4a-9379-e19ec31db9e7">
            <Transforms>
              <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>z1vyRGTtJDgW28yPPisPlxa5oio=</DigestValue>
          </Reference>
        </SignedInfo>

<SignatureValue>QzYgLTcY4bimSe2AXAARO/tcTHsMZUGVZlSxPbAosH2LV7J1+0SttyGVO44SlxbD+l2aM2qH0QBTXSEncDO/utFl0TTJS9JfZvc5q8RSbJ21Y/1wced1rsVFK5gZKWe09oXAp0/9/c/FR8huoR+9P/zh7mxhEPy61JfwTvCdq+s=</SignatureValue>
        <KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#SecurityToken-122cb9a7-82de-40a5-91e5-4a4a5cf00b90" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
          </wsse:SecurityTokenReference>
        </KeyInfo>
      </Signature>
    </wsse:Security>
  </soap:Header>
  <soap:Body wsu:Id="Id-2fc19e22-84e5-4b4a-9379-e19ec31db9e7">
    <DTSRequest xmlns="urn:org:pesc:datatransport">eJwLyUhVyEmsqlRlyU9XyCrNLUhNUcgvSy1SKAFKJOeXFyskJSZn6+npGRgaGZuYmplbWBooAgDxtBCz</DTSRequest>
  </soap:Body>
</soap:Envelope>
```

Entire .NET DTS SOAP Response:


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <soap:Header>
      <DTSResponseHeader xmlns:dts="urn:org:pesc:datatransport" xmlns="urn:org:pesc:datatransport">
        <dts:DTSResponseRouting>
          <dts:DTSUUID>ed5a7410-afd3-11db-8ded-f1b1cb25f748</dts:DTSUUID>
          <dts:DTSSourceID>Nathan</dts:DTSSourceID>
          <dts:DTSSourceIDSubCode>Chitty</dts:DTSSourceIDSubCode>
          <dts:DTSRecipientID>Mark</dts:DTSRecipientID>
          <dts:DTSRecipientIDSubCode>Malinoski</dts:DTSRecipientIDSubCode>
        </dts:DTSResponseRouting>
        <dts:DTSResponsePayloadType>CRC:RESPONSE</dts:DTSResponsePayloadType>
        <dts:DTSResponsePayloadBytes>54</dts:DTSResponsePayloadBytes>
        <dts:DTSResponseAcknowledge>Deferred</dts:DTSResponseAcknowledge>
      </DTSResponseHeader>
      <wsse:Security soap:mustUnderstand="1">
        <wsu:Timestamp wsu:Id="Timestamp-5de36a39-5165-4c24-87ab-8975ff8847c5">
          <wsu:Created>2007-01-29T20:04:23Z</wsu:Created>
          <wsu:Expires>2007-01-29T20:09:23Z</wsu:Expires>
        </wsu:Timestamp>
        <wsse:BinarySecurityToken ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="SecurityToken-e3cd258e-b9f1-4a7a-8287-
8752ef077feb">MIICcjCCAdsCCQDMhQGu43qMhzANBgkqhkiG9w0BAQQFADCBhjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAl
BBMQ4wDAYDVQQHEwVFbm9sYTEPMA0GA1UEChMGUklTQ2lUMQwwCgYDVQQLEwNEVFMxFzAVBgNVBAMTDk1hcmsgTW
FsaW5vc2tpMSIwIAYJKoZIhvcNAQkBFhNtbWFsaW5vc0ByaXNjaXQuY29tMB4XDTA2MDgyMTE1NDI1NloXDTA5MDgyMDE1NDI
1NlowdDEUMBIGA1UEAxMLamF4LTMyMDUzMGwxEjAQBgNVBAsTCU5hdGhhbkRUUzESMBAGA1UEChMJTmF0aGFuRFRTM
RUwEwYDVQQHEwxKYWNrc29udmlsbGUxEDAOBgNVBAgTB0Zsb3JpZGExCzAJBgNVBAYTAlVTMIGfMA0GCSqGSIb3DQEBA
QUAA4GNADCBiQKBgQC0nwGCXlgD9MmxQtDu/HyFicJG64p3fRGHFgEuucOWrlJaWQBxl+9M5Lyq9eVaCNv3odINkSKMI5z/oaI
shfew5uPK3rT+h/iSAiUB4AkxQ+p8FT0YpVFqVJRz31FffopcvlskGBfx9p0oH2b5GBhjz/wtAnIYroOUJlgpWAmQrwIDAQABMA0GCS
qGSIb3DQEBBAUAA4GBAFkh3kiwegZlW8MzQEjtUPGPxRm0Jutl05wGJYB7Ivmq96S5sm5oLV4KjBXQd30sKd2p2d+wDlvwHBN9
YhlK4w5OvDEKE11eA/ryFdNukwmftbQo/DyiqI2ZDLEC+0ADdUsSpGPCjQlydCh6gK4L42nFtyM9NV7MsDEkrpbxiz1B</wsse:Binar
ySecurityToken>
        <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
          <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
            <Reference URI="#Id-e5b57f73-0afa-4a99-b7b1-5cf7a88af5e9">
              <Transforms>
                <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
              </Transforms>
              <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
              <DigestValue>RbkUhmToq17rYW8dBQis8RiwXx8=</DigestValue>
            </Reference>
          </SignedInfo>

<SignatureValue>oy34LhSYxeWyx/rsoNzOCqthyy4FX4xEIzFYIzt6UHxIQgq8sxcDVtcCfuL+5709++B2BjsK0+4HBtm8gQ7wStZnJTa
/KDDrL95XUIYUrj4qjZbORGY0xgMfGjUWwCZRUEH5jn8ZuszVbEyfFCII2PdtmLCf3Ibv2IMqgCQMZYc=</SignatureValue>
          <KeyInfo>
            <wsse:SecurityTokenReference>
              <wsse:Reference URI="#SecurityToken-e3cd258e-b9f1-4a7a-8287-8752ef077feb" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
            </wsse:SecurityTokenReference>
          </KeyInfo>
        </Signature>
      </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="Id-e5b57f73-0afa-4a99-b7b1-5cf7a88af5e9">
      <DTSResponse
xmlns="urn:org:pesc:datatransport">eJwLTsxTcMlMTc9XyCxWSFRIL0pNLFEoyElMTlUoyVcoyyzOLFFIzEsBCVUqlGeWZCi4hATr
AQAAQxKQ</DTSResponse>
    </soap:Body>
  </soap:Envelope>
```

8 Recommendations

It is the recommendation of the PESC DTS Technical Committee that the <DTSUUID> sub-element of the <DTSRouting> header element for both a request and a response be formatted according to the IETF Draft Uuid Specification. Supporting documentation can be found at the following link: http://en.wikipedia.org/wiki/UUID.

It is the recommendation of the PESC DTS Technical Committee that the uncompressed source data not exceed 50 megabytes. Technology will allow for data sizes greater than 50 megabytes, but this requires modifications to the implementation of the specification and web server configuration parameters.

The error handling section of the specification does not require values being present in the <DTSRouting> sub-elements <DTSSourceIDSubCode> and <DTSRecipientIDSubCode> within the <DTSRequestHeader> and <DTSResponseHeader> blocks. It is the recommendation of the PESC DTS Technical Committee that the IDSubCode sub elements be populated with the assigning organization of the SourceId and RecipientId codes respectively.

For example: SourceId = 12345678, SourceIDSubCode = ABCDE

It is recognized entities may have internal uses of the specification. When used internally entities may chose to not implement the security portions of this specification.

9 Appendix

Below is the PKI WSS Proposal submitted to the PESC Steering Committee in October 2006. It provides technical information supporting the move to DTS v2.0.

Problem Statement:

DTS Version 1.0 provides the mechanism to authenticate and validate the request/response through asymmetric PKI using X.509 certificates. Currently, the only way to exchange public keys is to physically send the X.509 Certificates to the trading partners you want to do DTS transactions with.

The DTS technical workgroup proposed three technical solutions for key exchange, referenced in both the specification and reference implementation guide documents. However, the industry landscape and technical choices available at the time eliminated adoption of any method that would reduce public key management complexities.

New Technical solution for key exchange:

With the advance in technology and standards, the WS-S (Web Service Security) specification has provided a solution for key exchange, simple management, and eliminates the need for any of the original three technical solutions proposed. The DTS technical workgroup has been working on proving out this new solution and would like to present it for approval to PESC before making the appropriate changes to the DTS specification.

Actual Solution:

The WS-S specification allows for the X.509 certificate to be sent within the security header of the SOAP message. Extending the DTS specification to use WS-S would allow the recipient of the SOAP transaction to extract the public key of the sender's certificate and for the recipient to verify the digital signature without any separate exchange of public keys.

Issues/Benefits:

- Trusting that the X.509 certificate is valid.
    - A root Certificate Authority (CA) should sign the certificate.
    - For added trust, a second Certificate Authority could sign the certificate.
    - By "trusting" that the Certificate Authority executed due diligence to ensure that the certificate provided to them to sign was indeed the agency represented in the certificate, the certificate of the Certificate Authority is all each trading partner needs in the beginning
    - Root certificates from major Certificate Authorities come already installed on most computers.
- Checking revocation lists.

- Version 3 of X.509 certificates allows for Certificate Revocation List (CRL) Distribution Endpoints, which provides the URL to retrieve information concerning the status of certificates that is maintained by the CA.
- Utilizing this list allows for an agency to communicate a problem with their key to one institution (the CA) and everyone they trade with (who checks that status with the CA) will immediately be aware of the potential compromise.
- If the CA becomes aware of a potential problem the certificate can be immediately revoked thus allowing all others to know.

- Key Expiration
  - CA's typically only issue certificates that are valid for one year.
  - Periodically changing keys becomes a necessity at a higher level than just one institution's corporate policy.

Recommendations:

1. PESC becomes a Certificate Authority allowing their certificate to be part of the trust chain. Action items:
   - Research the feasibility of PESC becoming a CA.
   - Research outsourcing to Verisign or some other third party CA acting on behalf of PESC.
   - Ensure Verisign or other third party provides the appropriate safeguards and procedures to perform CA responsibilities correctly.

2. Have the DTS technical workgroup create a Version 2.0 of the DTS specification to include this new solution for key exchange using WS-S. This would include restructuring the current DTS header elements to align more closely to WS-* specifications.

Supplemental Information

The following sections and diagrams explain the technology behind, viability of, validity of, and justification for the proposed solution.

While the "DTS Version 1 Specification" requires the use of digital signatures, the means by which to do so is not defined through a proven and standardized method. However, the "DTS Version 1 Reference Implementation" does state "Signing and verifying require the use of X.509 certificates."

Version 2 of DTS would require the use of X.509 Certificates and that they must be issued by a Certificate Authority. The inclusion of proven, interoperable, portions of the Web Services – Security specification would also be explicitly required; thus also stating the means by which to implement the Digital Signature.

Other changes to the DTS specification are limited to regrouping the SOAP Headers defined in Version 1. In doing so, the header elements required by DTS will be brought into a format that follows all other Web Services specifications. The reorganization doesn't eliminate any of the data elements being provided in Version 1. Rather, they are either replaced by elements included in the WS-S specification or simply located elsewhere in the message structure. This means that any Version 1 interface already implemented will not have to change.

The following information discusses: what are X.509 certificates, what do they contain, and how will/are they used by DTS and WS-S.

Digital Signature Summary

A Digital Signature is used to ensure that 1) the contents of the message have not been changed and 2) to authenticate the sender of the message. In order to validate a digital signature, the receiver must have the sender's public key. A public key is not a secret and may be freely distributed. A trusted "certificate authority" validates the owner of the public key.

What is a Digital Signature?

Technically, a digital signature is an encrypted hash of the signed message presented in a standard format. A hash is an algorithm (mathematical function) which produces a short string (or number) which is very likely to change if the content of the message is changed. The output of the hash algorithm is called a digest.

A digital signature does two things. First, the receiver is assured that the message received is the message sent and, the content of the message has not been tampered with. The hash algorithm provides this guarantee.

Second, the digital signature provides authentication of the signer. Digital signatures use a public-private key pair. The public key is made available to the receiver of the message in an X.509 certificate. The certificate includes the name or organization which owns the certificate and information about the certificate authority which vouches for the certificate's authenticity.

What is an X.509 Certificate?

An X.509 Certificate is a container for a public key. In addition, the certificate is digitally signed by the issuer of the certificate, a trusted Certificate Authority.

If an X.509 certificate is to be trusted, then the user must believe that the Certificate Authority has adequately validated the identity of the certificate's owner. In addition, the Certificate Authority must maintain the ability to revoke the certificate if the owner reports that the matching private key has been compromised. The certificate contains the information necessary for programmatically contacting the revocation list. Often there is a chain of trusted certificate authorities. For example, an entity may obtain a certificate from a well known and trusted commercial certificate authority. Then the entity may act as a Certificate Authority itself and issue certificates. Users who choose to trust the entity can verify the identity of the entity via the commercial Certificate Authority.

How is a Digital Signature created?

The first step in creating a digital signature is to hash the contents of the message. Standard hash algorithms are used to "compress" the contents of the message into a relatively short number.

The next step is to encrypt the hash number. The signer uses its private key to perform the encryption. Only the signer's public key will decrypt the hash.

The final step is to create the digital signature itself which includes some information in addition to the encrypted digest, such as the algorithm used to create the digest.

The digital signature is then sent with the message.

The receiver must also have access to an X.509 Digital Certificate which contains the signer's public key. Often the Digital Certificate is included with the signed message.

How is a Digital Signature validated?

The first step in validating a digital signature is to obtain and validate the public key of the signer. Usually, the signer's public key is delivered in an X.509 Certificate. To validate the certificate, and thus validate that the public key belongs to the signer, the Certificate Authority's X.509 Certificate must be in the user's collection of trusted Certificate Authorities.

The next step is to ensure that the Certificate Authority has not revoked the certificate. All Certificate Authorities provide a mechanism(s) for determining if a certificate has been revoked.

Knowing that the certificate is issued by a trusted authority and has not been revoked, the hash digest in the digital signature is decrypted using the public key in the X.509 Certificate.

Finally, the user computes the hash digest of the received message. The signature is valid if the computed digest matches the decrypted digest in the digital signature.

How are public-private key pairs obtained and distributed?

A user creates a public-private key pair on their own computer using commonly available software. Microsoft operating systems provide this ability. UNIX based systems may use open source software to create the key pair.

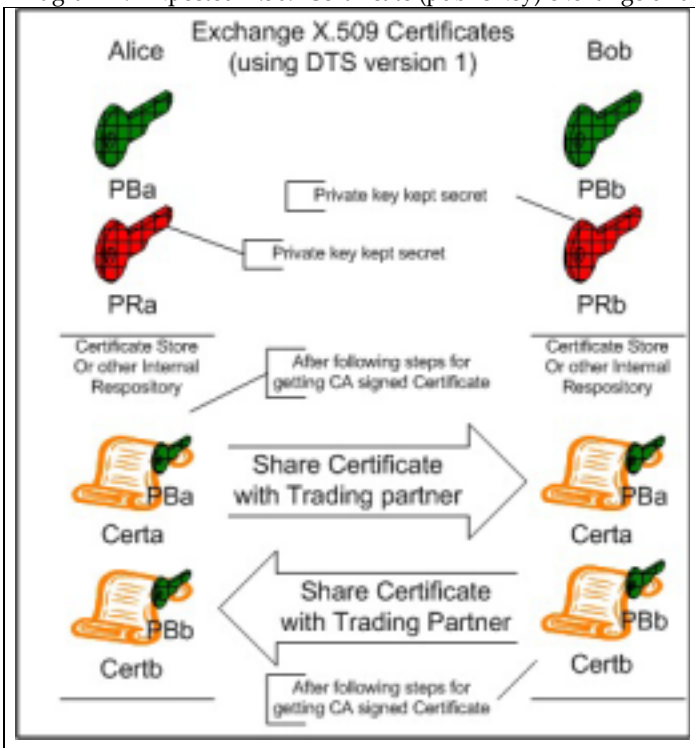
Once the key pair is created, the user is responsible for guarding the secrecy of the private key. The public key, however, is not a secret and may be safely disclosed to anyone.

Users want assurance of the public key owner's identity. This is accomplished by obtaining an X.509 Certificate from a trusted Certificate Authority. The public key is sent to the Certificate Authority and the authority returns a signed X.509 Certificate.

The X.509 Certificate may be distributed to anyone. Since the certificate contains only public information, it does not need to be encrypted before transmission. Because the X.509 Certificate is signed by a trusted Certificate Authority, the authenticity of the certificate may be independently validated.

The simplest distribution method is to include the X.509 Certificate with the message rather than using a separate "out-of-band" mechanism. This is safe because 1) none of the information in the certificate is secret and 2) the ownership and contents can be verified by the issuing Certificate Authority.

Diagram 1: Expected X.509 Certificate (public key) exchange under Version 1



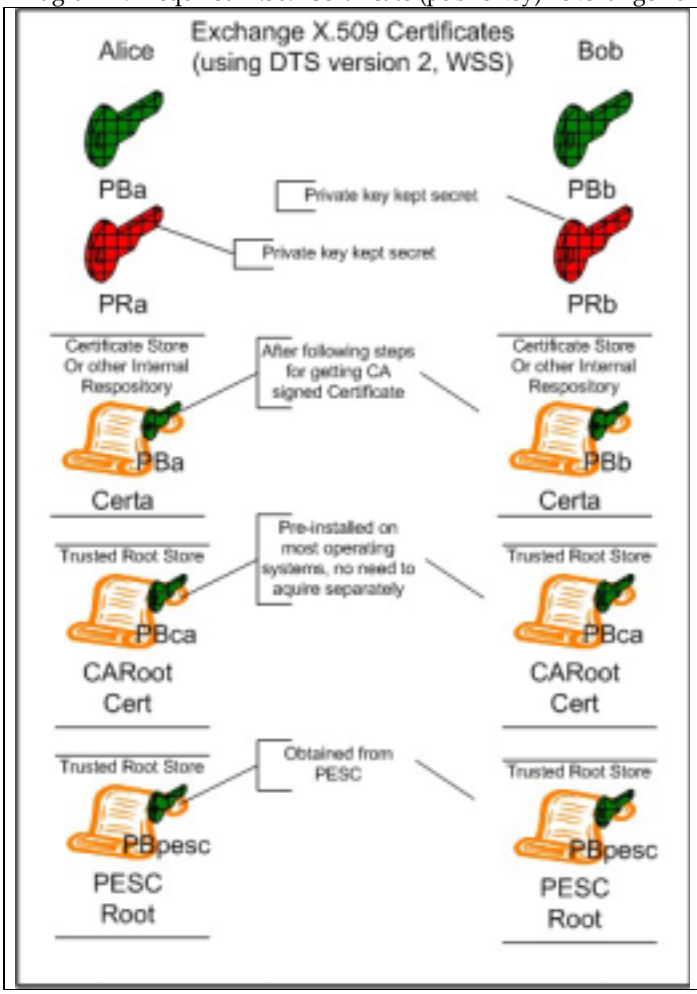
Benefits:

- Decreased cost (though minimal) if not requiring CA to issue the certificate.
- By using X.509 as the container for the public key, interoperability between platforms is guaranteed.

Shortcomings:

- New certificates and updates must be exchanged with all trading partners
- The certificates are exchanged, stored, and validated outside of the specification.
- Validation outside of specification could allow an entity to be compromised, **if the entity does not create due diligent programmatic processes that are not explicitly stated in the specification**

Diagram 2: Required X.509 Certificate (public key) "exchange" under Version 2



Benefits:

- **No** manual exchange of certificates.
- Interoperability is guaranteed by using X.509 as the container for the public key.
- DTS version 2 Specification would include WS-S and thus the key exchange and validation are within the specification (see Diagram 6).
- Minimal key management.
- Single point of revocation.
- Single point of key update.
- Only trusted certificate(s) is required to be obtained before transport via DTS

Shortcomings:

- By requiring CA to issue the certificate there is an increased cost (TBD based on result of action items in recommendation 1).

Diagram 3: Obtaining X.509 Certificate from Certificate Authority

The following diagram depicts the steps an organization would follow for obtaining a certificate from a certificate authority.

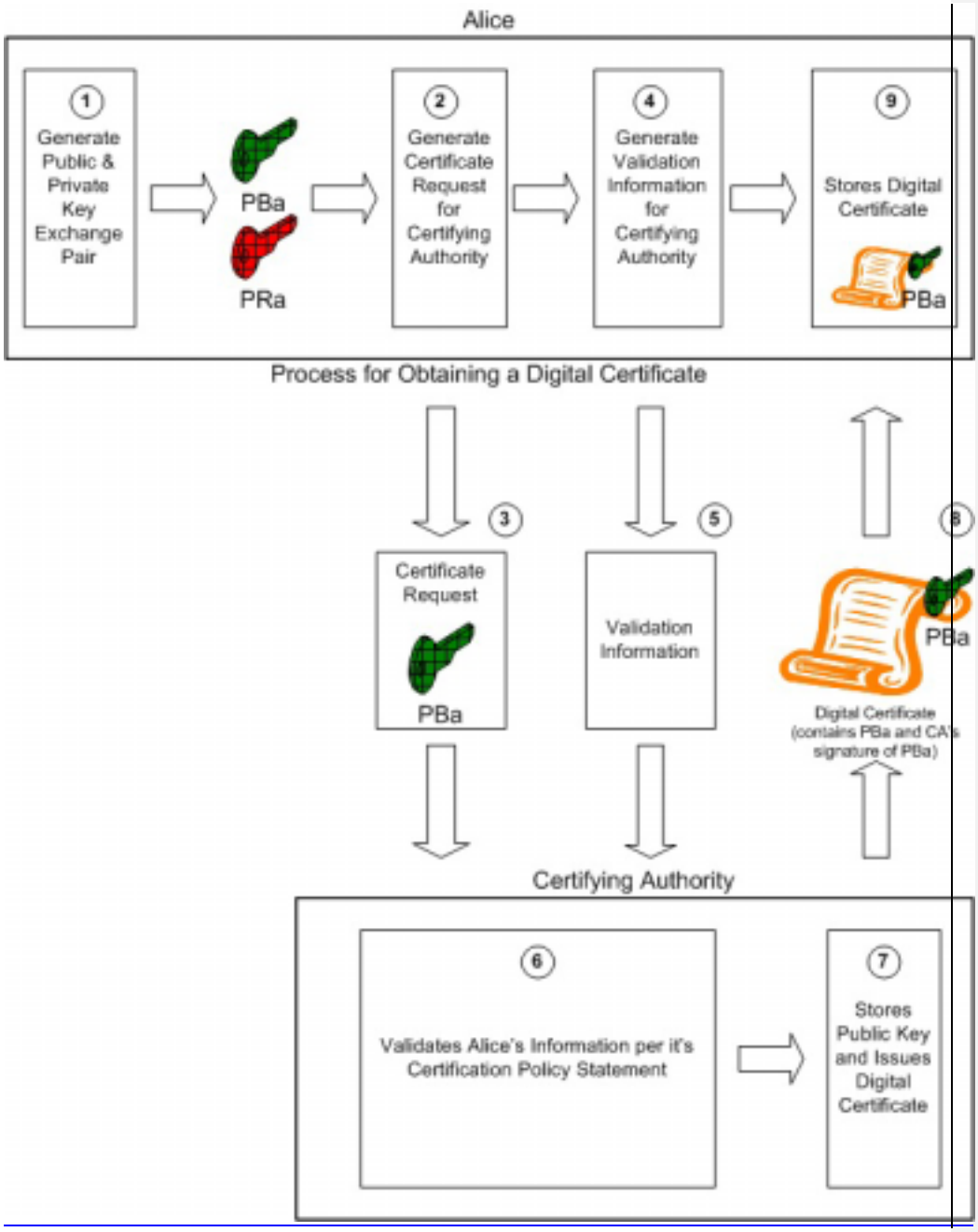
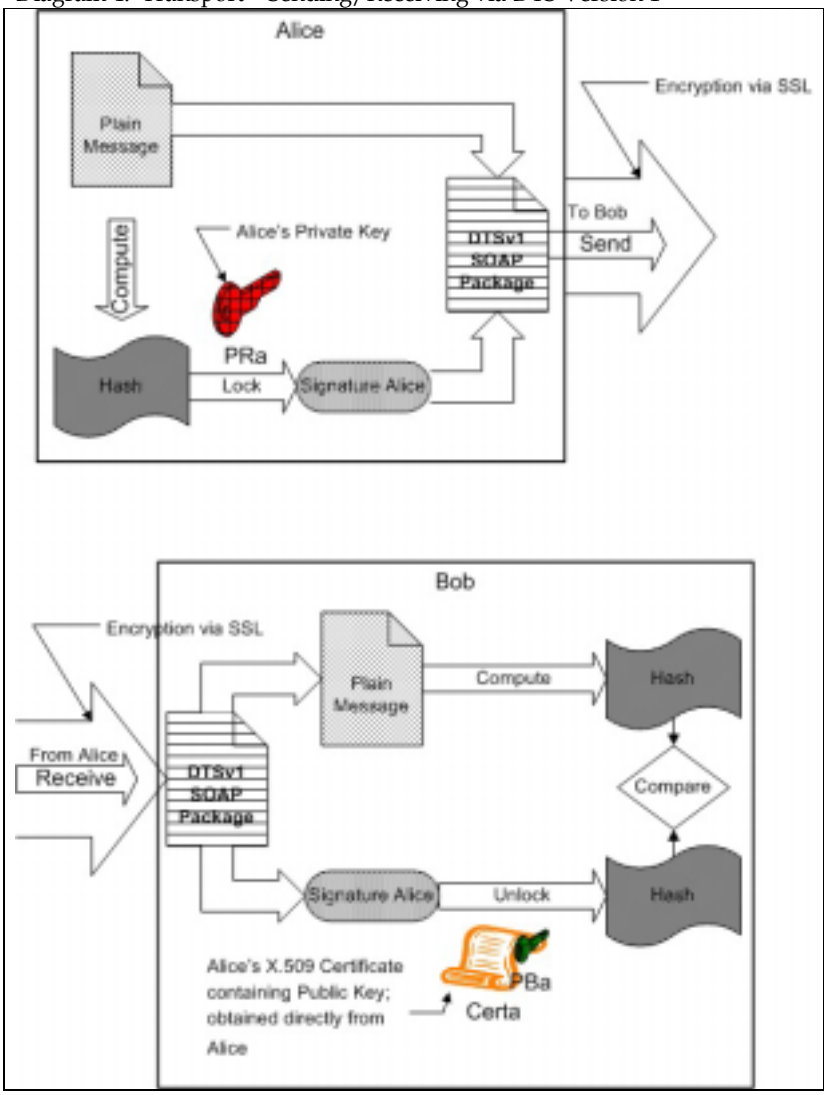
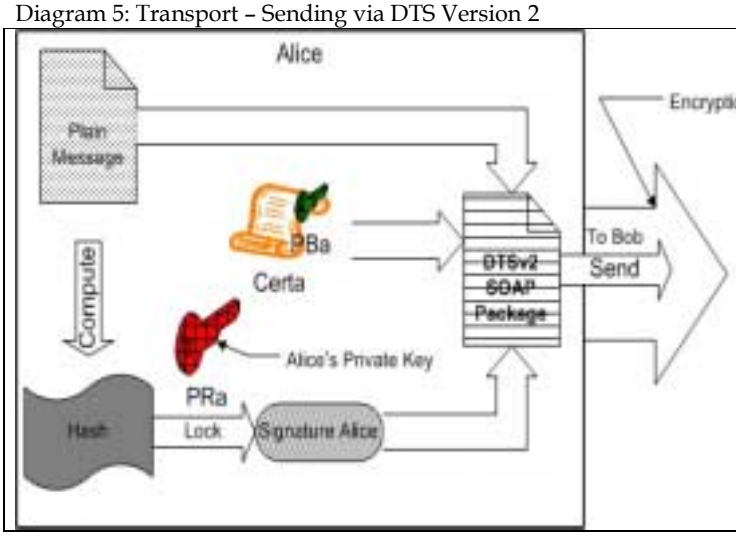

Alice
1
Generate Public & Private Key Exchange Pair
PBa
PRa
2
Generate Certificate Request for Certifying Authority
4
Generate Validation Information for Certifying Authority
9
Stores Digital Certificate
PBa
Process for Obtaining a Digital Certificate
3
Certificate Request
PBa
5
Validation Information
8
PBa
Digital Certificate (contains PBa and CA's signature of PBa)
Certifying Authority
6
Validates Alice's Information per it's Certification Policy Statement
7
Stores Public Key and Issues Digital Certificate

Diagram 4: Transport – Sending/Receiving via DTS Version 1



Notes:
Alice's certificate is obtained directly from Alice via a previous out-of-band, non-DTS communication

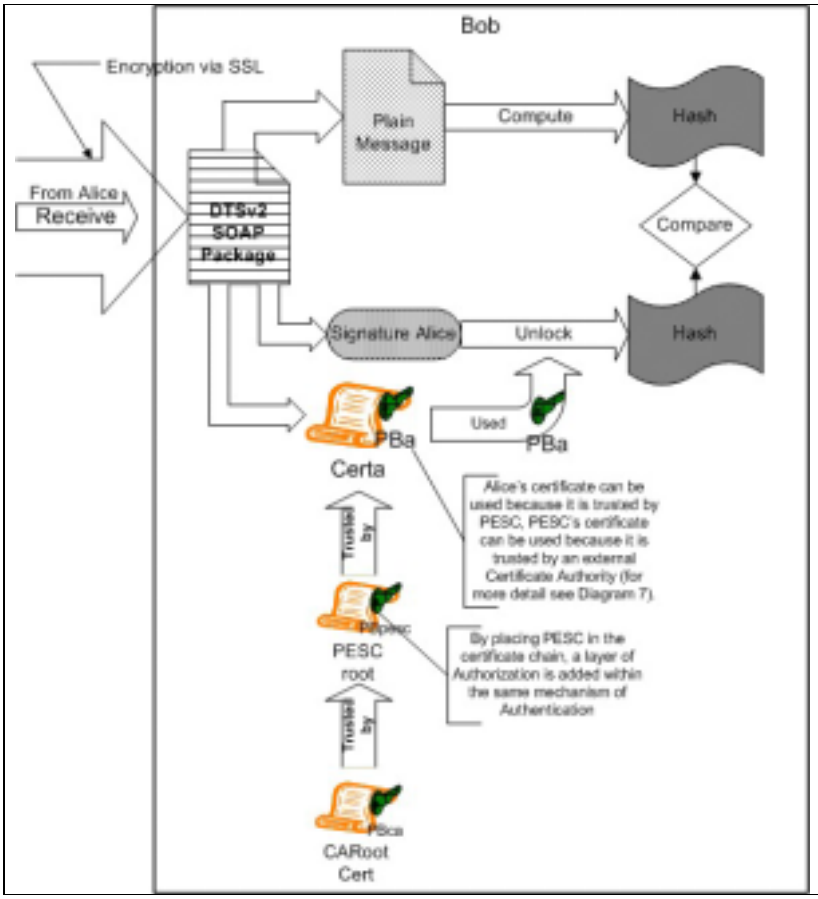
Diagram 5: Transport – Sending via DTS Version 2



Notes:
The certificate is added to the SOAP package under the WS-S specification

The certificate contains Alice's public key, the Certificate Authority's signature of the public key, and the entire certificate trust chain

Diagram 6: Transport – Receiving/Verifying via DTS version 2

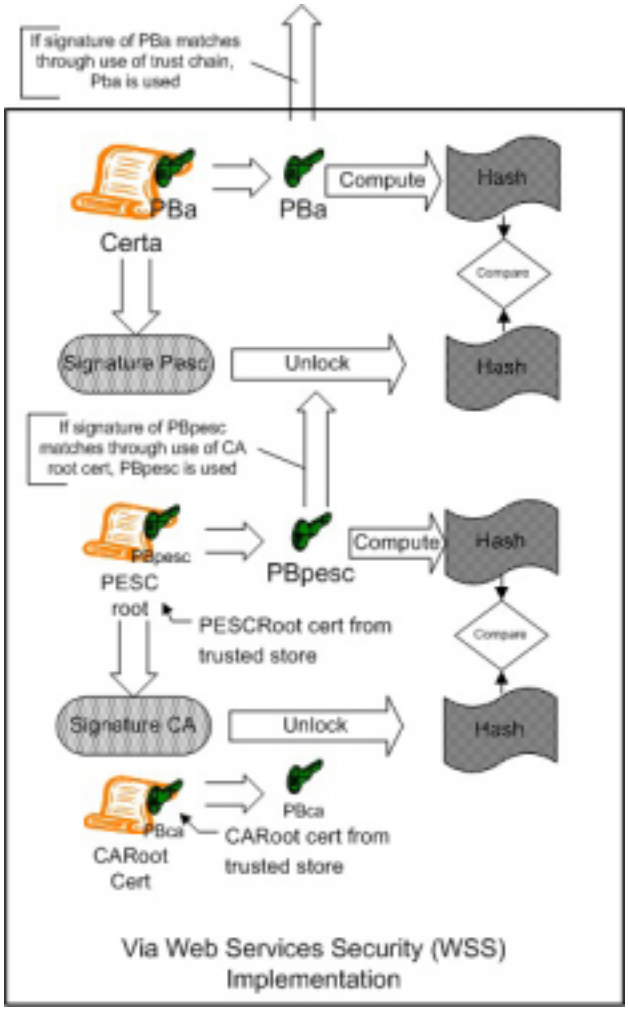


Notes:
Alice's certificate is received from Alice during the same DTS communication.

Formatted: Bullets and Numbering

Diagram 7: Detailed Certificate Chain Validation

The following diagram depicts the X.509 trust chain verification at a detailed level. Alice's certificate is transmitted with the DTS Version 2 SOAP package and is extracted. Its' authenticity is verified by use of the PESC certificate already installed on the computer. The certificate from PESC is authenticated by use of the root certificate of a Certificate Authority, preinstalled on most computers.


If signature of PBa matches through use of trust chain, Pba is used
PBa
Certa
PBa
Compute
Hash
Compare
Signature Pesc
Unlock
Hash
If signature of PBpesc matches through use of CA root cert, PBpesc is used
PBpesc
PESC root
PBpesc
Compute
Hash
PESCRoot cert from trusted store
Compare
Signature CA
Unlock
Hash
PBca
PBca
CARoot Cert
CARoot cert from trusted store
Via Web Services Security (WSS) Implementation

Glossary

This section should list a glossary of terms and their definitions helpful in deciphering this document. – Will be provided at a later date.

Acknowledgements

PESC DTS Workgroup would like to recognize the following individuals and their respective organizations for their time and effort with the development of the DTS Specification

Mark Malinoski – AES
Nathan Chitty – Nelnet
Laura Damkoehler – ELM
John Gill – TG
Barry Needalman – ASA
Gary Sandler – ELM
Kim Shiflette – USA Funds
Richard Henninger – Datatel
Carl Romanik – NextStudent
Greg Zink – NY HESC
Larry Casey – accessGroup
Chet Sharrar – accessGroup

Revision History

DATE	SECTION/ PAGE	DESCRIPTION	REQUESTED BY	MADE BY
5/24/05	Whole Document	Initial Version		Mark Malinoski Nathan Chitty
8/2/05	Whole Document	Format, grammar, and style review and changes.	Gary Sandler	Laura Damkoehler
05/19/06	Whole Document	Header and footer date changes	Kim Shiflette	Kim Shiflette
10/23/06	Whole Document	Typographical changes and example consistency corrections	Mark Malinoski	Nathan Chitty
10/24/06	Security Section	Added security section to describe WS-S (San Diego)	DTS Tech Group	Mark Malinoski
2/1/07	Whole Document	Modified Document to include all of the WS-S requirements and the reformatting of the DTS headers (Las Vegas and Santa Fe)	DTS Tech Group	Mark Malinoski
2/6/07	Whole Document	Modified Document based on comments made during webanar review session	DTS Tech Group	Mark Malinoski
2/26/07	Whole Document	Minor formatting changes.		Mark Malinoski

DATE	SECTION/ PAGE	DESCRIPTION	REQUESTED BY	MADE BY