Reference Communication Model

Includes Messaging Security Guide
and Application Messaging Guide
(IATA SCR Volume 8)

Version 1.0

Date: January 2016

This document is maintained by the Communications Standards Coordination Committee (CSCC) of PADIS. Workgroup members or interested parties should send their comments and requests to CSCC Vice-Chair Mansour Rezaei-Mazinani (Mansour.Rezaei-Mazinani@sita.aero) with a copy to Marie Zitkova (zitkovam@iata.org) at IATA PADIS Secretariat.

Revision History

Date	Version	Description	Author
September 2015	1.0	New document incorporating individual document developed by the CSCC over the last two years	Secretary

Table of Contents

Preface

This is the 1st Edition of the Reference Communication Model including Messaging Security Guide and Application Messaging Guide. It will be reissued periodically to incorporate changes as business requirements and actual usage of communications protocols evolve.

Any comments, questions, or suggestions concerning this Manual should be addressed by e-mail to padis.secretariat@iata.org.

Purpose

The IATA Reference Communications Model is issued by IATA on the authority of the IATA Passenger and Airports data Interchange Standards (PADIS) Board and it is maintained by the Communications Standards Coordination Committee (CSCC) which reports to the PADIS Board.

The CSCC is tasked to promote and oversee interoperability on the level of communications protocol and general implementation of data exchanges. The IATA Reference Communications Model and associated guide provides, in a single manual, comprehensive documentation of common components of communications standards (including data security) that can be used by industry applications.

The purpose of this Manual is to promote a much broader understanding, usage and acceptance of the usage of various communications protocols in the context of the airline industry. Readers of this document are highly encouraged to refer to this document when developing business requirements for new messaging standards and designing pilots of messaging standards.

The International Air Transport Association

The International Air Transport Association (IATA) is the world organization of the scheduled airlines. Its Members carry the bulk of the world's scheduled international and domestic air traffic, under the flags of over 125 nations and membership presently totals 240 carriers. The aims of IATA are to:

- promote safe, regular, and economical air transport for the benefit of passengers, to foster air travel and to study related problems;
- provide means for collaboration among the air transport enterprises engaged directly or indirectly in air transport services and with enterprises associated in the travel industry;
- co-operate with the International Civil Aviation Organization (ICAO), the Airlines for America (A4A), and other international organizations.

IATA functions as the international air transport link with governments and the public. For the airlines and other travel related organizations, IATA provides machinery for finding joint solutions to problems beyond the resources of any single company. It has developed procedures and practices which produce a worldwide public service system, despite the differences in languages, currencies and laws.

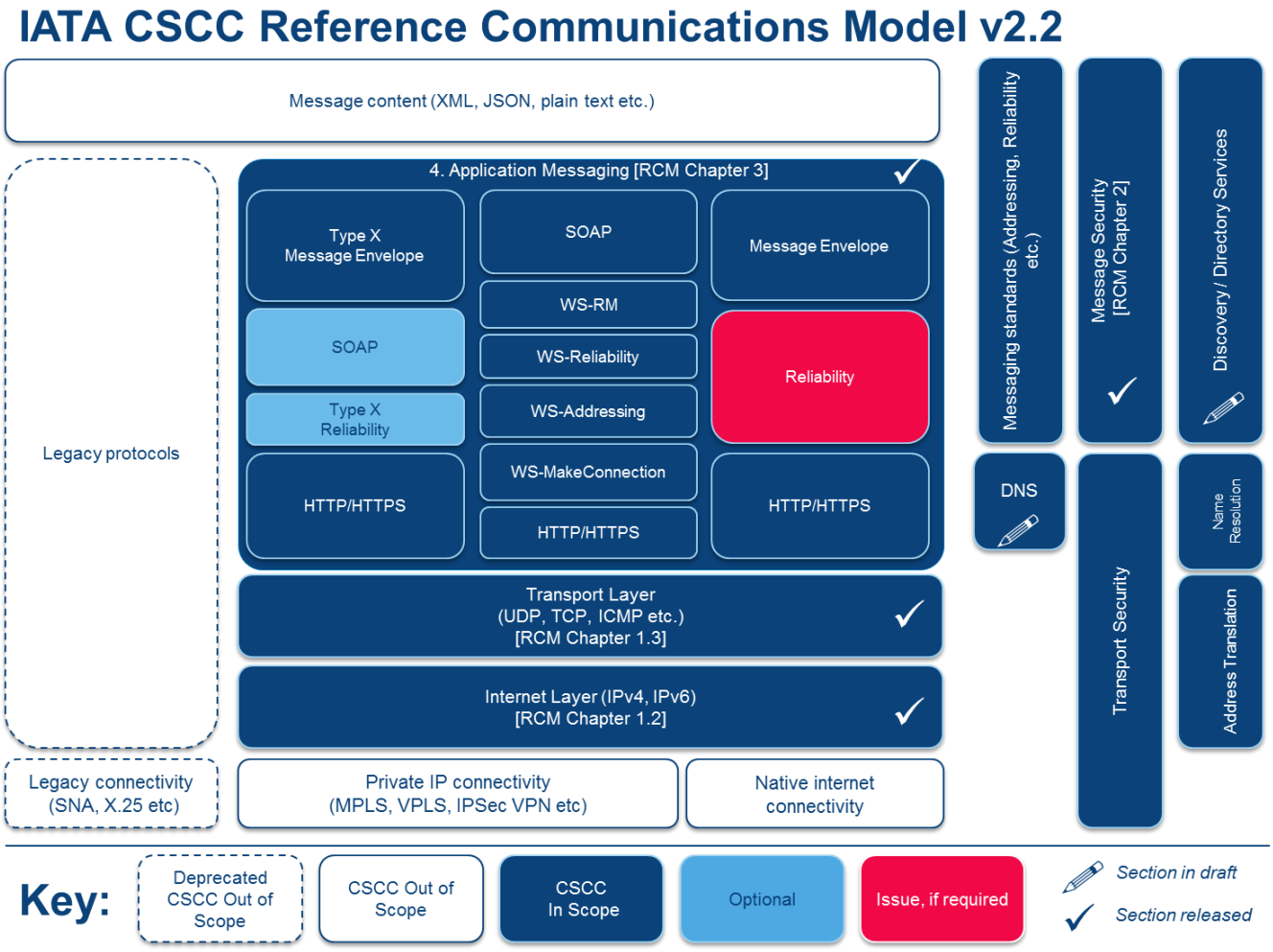
For the public, IATA ensures high standards of efficient operation and proper business practices by airlines and other travel related organizations and their agents, simplification of government regulations, and fares and rates consistent with sound economy. Airline cooperation through IATA enables a passenger to undertake journeys involving many countries, and the systems of several scheduled airlines, by making one telephone call, and one payment in a single currency.

IATA is subdivided into Committees and Conferences, each being responsible for particular aspects of air transportation, e.g. Tariffs, Passenger Agency, Passenger Services, etc. The handling of data exchange standards for passenger services including associated coordination for the deployment of communications protocols is the responsibility of the Passenger Services Conference and its respective Committees.

Section 1: Reference Communication Model

1.1 Reference Communication Model Overview

The Communications Standards Coordination Committee (CSCC) developed a reference model, it was agreed that this document, renamed to Messaging Guide, will serve as the master repository of text describing the Communications Reference Model. Initially, the text describing the Model will be developed in the form of Annexes. Once the initial content of the Model has been developed, this document will be restructured accordingly and the content will be incorporated here.



1.2 Internet Layer (IPv4, IPv6)

1.2.1 Background

The TCP/IP protocol suite is the basis for almost all modern data communications, and forms the underlying protocol transport for the majority of messaging protocols.

Proper design of the IP communications infrastructure and addressing scheme avoids many problems which subsequently need to be solved at higher layers.

The Internet Protocol (IP) provides an abstraction from the underlying physical network and enables an end-to-end network addressing scheme. It forms the basis for all communications across the Internet, and is also the prevailing standard used in private communications networks.

IP Version 4 (IPv4) is the most widely deployed version of the protocol. This provides an address space with 2^32 addresses. When the protocol was originally designed, it was envisaged that this address space would be sufficient for the requirements of the global internet.

The subsequent exponential growth in the number of nodes with internet connectivity, driven largely by the consumerization of the internet, has led to rapid exhaustion of this address space, and hence a new version of IP, known as IP Version 6 (IPv6) has been developed. This provides an address space with 2^128 addresses.

At the time of writing IPv6 has not been widely adopted, although there are significant indications that it will gain traction during the next 2-3 years, with some major players now actively deploying the protocol.

1.2.2 IPv4 Public vs. Private Addressing

Because of the scarcity of IPv4 addressing, IETF Request for Comments (RFC) 1918 allocated certain defined ranges of IP addressing for allocation within private networks. These ranges are: 192.168.0.0/16, 172.16.0.0/12, and 10.0.0.0/8. The intention was for organizations to use these addresses in their internal networks and to conserve globally unique, public IP addresses for communication across organizational boundaries. Organizations can either: assign public addresses to hosts which require external connectivity; translate private addresses into public addresses at the network boundary [n.b. this can be a static 1:1 mapping, or a mapping of multiple private addresses to a single public address using multiple UDP/TCP port numbers on the public side - this is known as Port Address Translation]; deploy proxy or reverse proxy servers to handle the translation; use publically-addressed Virtual IP interfaces on load-balancers to front connectivity to pools of privately addressed resources. Section C.1 of the Reference Communications Model discusses address translation in more detail.

Because multiple organizations can use the private IP address ranges in their own networks, there is a high probability of overlapping addressing. This generally renders it highly undesirable to use private IP addressing between organizations; even if two organizations are able to agree bilaterally to use a non-conflicting private IP address range, it is highly likely that a future user of the same services would not be able to use the same address range for communications.

1.2.3 IPv6 Addressing

Because the IPv6 address space is so large, in most cases there is no requirement to use private IPv6 address space.

1.2.4 Performance

As a logical abstraction layer, IP networks have to be able to deal with a wide range of different physical media with differing performance characteristics.

Key performance metrics for IP networks are bandwidth, latency, packet loss and jitter. Most higher layer messaging protocols will use TCP as the underlying transport, so packet loss is addressed at the transport protocol layer. More relevant considerations are bandwidth - how much data can be sent in a given timeframe; latency - what is the delay to transmit the information across a network; jitter - what is the variation in delay which may be experienced.

Connections within a Local Area Network will have the lowest latency. Typically, within a datacentre, sub-millisecond latency is the norm, with some connections able to achieve microsecond latencies.

Connections over a Wide Area Network will typically have much higher latencies. Factors affecting this are whether the connection is terrestrial or satellite. In both cases, the speed at which the signal can travel is constrained by the speed of light; in practice this means that a satellite connection will typically have a minimum latency of 300-500 milliseconds. Terrestrial connections may achieve latency in the tens or hundreds of milliseconds.

The choice of transport layer protocol has implications on end-to-end performance (see section 3.1).

On private network connections, it is possible to specify Service Level Agreements with service providers that set expectations as to these metrics. It is also possible to specify multiple Quality of Service (QoS) classes to treat different types of traffic preferentially, or to reserve a percentage of link bandwidth for a particular traffic type.

With internet connections, it is generally only possible to specify the performance characteristics of the last-mile circuit that connects a network to the internet. Notwithstanding, because internet bandwidth is often cheaper than private WAN connectivity of the same bandwidth, in many cases in practice the performance of an internet connection may exceed the performance of a private connection, albeit without guarantees that this will be consistently maintained.

When deploying messaging applications it is important to bear in mind that throughputs achievable in a lab environment may be significantly better than those achievable in real-world deployments where slower, and possibly congested WAN topologies are in place.

1.2.5 Security

Typically firewalls are used to provide a secure boundary between an organization's network and other organizations, or an organization's network and the internet. The firewall implements rules that determine what resources can talk to each other, usually based on IP addresses, and transport protocol/logical port number.

1.2.6 Implications for messaging applications

Messaging applications within organizations may use RFC1918 private IP addressing between messaging endpoints. Messaging applications between organizations should always use publicly registered globally unique IP addresses, even if the underlying transport network is a private network, not the internet. No organization should mandate the use of private addressing between organizations as part of a messaging solution. Organizations who bilaterally agree to use private addressing over private network transports between themselves may do so; however they must be prepared to present the same services using publicly valid IP addresses if another party requires them to do so.

Messaging applications should not use IP addresses directly as endpoints. Doing so makes it very difficult to subsequently re-architect an organization's network, or migrate applications between different server platforms. IP-based applications should always use fully-qualified DNS (Domain Name System) names to refer to endpoints, either directly or implicitly as part of a URI.

When defining firewall rules, there is a trade-off between having very fine-grained rulesets, which may have to be updated very frequently, or very coarse-grained, which may need to be updated less frequently but arguably are less secure. We note that many cloud-based application service providers are providing coarse-grained address ranges for the services they provide rather than individual service IP addresses; this allows greater flexibility for applications to move between server platforms and to scale to introduce additional nodes.

1.3 Transport Layer

1.3.1 Background

The transport layer within the TCP/IP protocol suite builds on the Internet layer to provide communication between network services on either the same or different hosts. The most common transport layer protocols, User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) both allow for service to be provided on logical ports on the host. Addressing network traffic to these logical ports allows network traffic to be passed to the appropriate application on the host. The combination of source and destination IP addresses, protocol, and port numbers is known as a socket, and can be thought of as a logical channel between the two hosts.

1.3.2 User Datagram Protocol

UDP is a connectionless protocol that allows datagrams to be delivered across a network with low protocol overhead. It provides no guarantees to an application that datagrams will be delivered in sequence, no handling of dropped packets, no acknowledgements that a datagram has been received, and (from the perspective of the receiving host) only weak assurance that a packet has not been altered in flight, and no mechanism for detecting and discarding duplicate packets. Because it is a "fire-and-forget" protocol, there is no session establishment process. Typically UDP is used for very simple request-response protocols like Domain Name System (DNS) resolution, and for real-time streaming applications where it is undesirable that the server keep track of connection state, and on-time delivery of packets is more critical than absolute reliability.

1.3.3 Transmission Control Protocol

TCP is a connection-oriented protocol that provides added assurance to an application that packets will be delivered reliably and in sequence. Duplicate packets are discarded. TCP sessions are established using a 3-way handshake that negotiates parameters for the connection including datagram size, sequence numbers and initial transmission control windows.

The majority of higher-level protocols such as FTP, SMTP, HTTP, SSL, and so on, are built on top of TCP as it provides a standard set of services that can be reused.

1.3.4 Other protocols

The Internet Control Message Protocol (ICMP) is not generally used to route application data between hosts, but provides for a range of diagnostic capabilities, including the standard echo-request and echo-reply mechanism commonly known as 'ping'.

A newer protocol, Stream Transmission Control Protocol (SCTP) provides improvements in performance and security over TCP, however at the time of writing, its adoption is not widespread. Obstacles to deployment include lack of native support in Microsoft operating systems, the need to re-code applications to use it, and the lack of support for SCTP in many firewall, loadbalancer and network address translation devices. If SCTP becomes more widely adopted in future it is likely to have significant advantages to messaging applications.

1.3.5 Performance

In general, messaging applications will be deployed using TCP as the transport layer protocol. Although there is additional overhead in session establishment, the use of TCP avoids the requirement for bespoke development to handle connection-oriented traffic. However, in order to maximise throughput, certain factors need to be taken into consideration.

TCP/IP has a 'slow start' mechanism that progressively increases the window size of the connection (the number of bytes of traffic that can be sent before having to wait for an acknowledgement). When traffic is delivered with no packet loss, the window progressively increases, if packets are lost it will be decreased. This mechanism is designed to cause traffic rates to be backed off under network congestion conditions. Window size settings need to be considered to enable this mechanism to work optimally. Additionally, high latency links can affect the ability for a TCP connection to achieve maximum link throughput. There are a number of optimization techniques which can be applied within hosts or using external devices to

help improve TCP performance over a Wide Area Network.

1.3.6 Security

Many organizations do not allow, or seriously restrict inbound UDP traffic from entering their networks through firewalls.

Also, many organizations restrict ICMP traffic across firewall boundaries. It is strongly recommended that ICMP echo-request and echo-reply messages be allowed to transit firewalls between messaging hosts to allow for a basic level of diagnostics.

1.3.7 Implications for messaging applications

In contrast with legacy messaging protocols, TCP/IP provides a generic set of reliability mechanisms. In designing messaging applications and protocols, the need for the mechanisms that existed in legacy protocols to be re-implemented over and above the capabilities that already exist in IP should be challenged, particularly in applications deployed within a controlled and managed infrastructure where basic levels of reliability and robustness can be assumed.

Applications written to use TCP should be extremely careful about the potential for a negative interaction between application and protocol-level acknowledgement mechanisms. The TCP windowing mechanism allows for multiple packets to be sent across a network before an acknowledgement is required. This improves performance across high-latency links. Messaging applications which are too aggressive in their implementation of application-level acknowledgements may prevent the TCP slow-start mechanism from ever reaching maximum throughput. It is highly desirable to maximize the length of IP flows and to avoid the set up of many short-lived TCP sessions as far as possible.

1.4 Domain Name System (DNS)

1.4.1 Overview

When you type a web address or send an email, DNS is the system that tells the machine where to go. It is a hierarchical distributed naming system for computers, services, or any resource connected to the Internet or a private network. It associates information with domain names assigned to each of the participating entities (such as airlines). Most frequent use is to translate domain names to IP addresses needed for the purpose of locating computer services and devices worldwide.

Different types of information are stored in different resource records such as an address record (A for IPv4 address or AAAA for IPv6 address), mail exchange record used for mail servers associated with the domain (MX) or a certificate record (CERT) intended for storing of a certificate in DNS.

4.7.1. Security / DNSSEC

DNS Security Extensions (DNSSEC) adds security to the Domain Name System by providing origin authentication of DNS data, data integrity, and authenticated denial of existence of DNS data. It was designed to protect the Internet from certain attacks, such as DNS cache poisoning.

The Internet root has been digitally signed already and many registries allow users to sign their record. The Root Trust Anchor can be found at the IANA DNSSEC website. See www.dnssec.net for more information.

4.7.2. ENUM, Internet of Things

Number of applications rely on domain name system for mapping between identifiers and corresponding internet addresses. Examples of such applications are ENUM (mapping of phone numbers to internet services used in VoIP applications or Internet of Thing which uses DNS to map object identifiers (EPC codes) to internet addresses. In the airline world, applications of this concept could include identification of devices at an airport, identification of RFID tagged aircraft parts or addressing of aircraft in future networks. Research projects for all these applications exist.

4.7.3. Internationalized Domain Names

Domain names use a limited set of ASCII characters preventing the representation of names in many languages and scripts. RFC 3492 [2] specifies Punycode to allow user applications such web browsers map Unicode strings into the valid DNS character set. Many registries allow registration of internationalized domain names.

4.7.4. Implications on messaging applications

DNS is used by Messaging applications between organizations should consider using DNSSEC to authenticate the DNS data used.

Section 2: Messaging Security Guide

2.1 Introduction

2.1.1 Purpose

The *Message Level Security* chapter discusses encryption and digital signing principles using W3C and OASIS standards and describes its use with IATA Type X Messaging specification.

A tutorial section is first provided, before discussing W3C and OASIS specifications related to message level security.

The *TypeX Security* chapter presents a new TypeX Security binding which enables digital signing and encryption of TypeX envelopes in an independent and interoperable fashion.

2.2 Transport Layer Security - TLS

This section presents an overview of TLS (Transport Layer Security), the successor of SSL, whose usage is so prevalent on the internet today providing a level point to point security at transport level. It is useful to understand how this protocol functions in order to better appreciate risks involved when carrying out secure transactions on the internet.

It is interesting to note that TLS makes use of the usual mechanisms such as public key encryption, symmetric encryption and cryptographic hash function.

TLS has only few differences with respect to SSL, notably the use of more secure hashing.

The following diagram presents the basic exchange between a client and a secure server. In this example, the client is not required be authenticated by the server; this is typical of a web transaction. Concretely this means that the client is not in possession of keys; only the server has public and private keys.

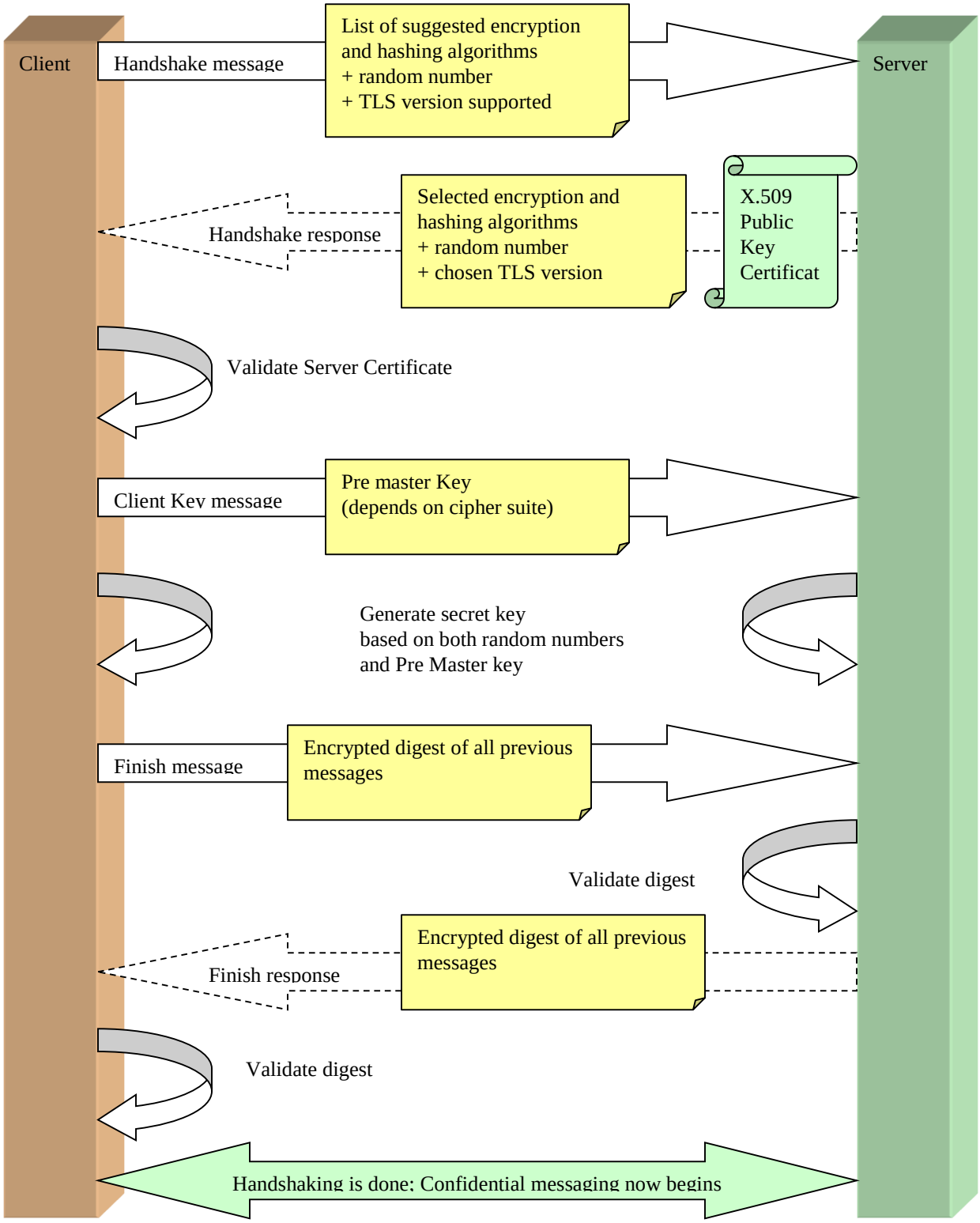


Figure 1 TLS exchange (client not authenticated)

2.2.1 Considerations

- Complexity of X.509
- Tools (including browsers) have loose implementations of X.509
- SSL has a number of problems (for example see www.blackhat.com)
- TLS protocol of choice for Internet transactions, but still suffers of problems related to PKI and those seen with the use of SSL.

2.3 Message Level Security

This chapter discusses message level security. Various fundamental concepts are first defined in order to present the possible solutions to enable message level security.

2.3.1 Definition

Message level security is essentially concerned with the protection of the message end to end, regardless of the underlying transport or environment which may or may not be secured.

For example, a message could be encrypted and sent over the Internet. The Internet is not secure in that anyone can intercept the message; however the message is undecipherable except by the intended recipient.

2.4 W3C Security

2.4.1 What is W3C Security?

The W3C organization defines two specifications of interest, XMLSIG [XSIG] and XMLENC [XENC] concerning digital signature and encryption, respectively.

2.4.2 XML Signature

The XML Signature specification defines XML syntax and processing rules for creating and representing digital signatures. XML Signatures can be applied to any digital content (data object), including XML. The syntax includes necessary elements such the signing algorithm, digest method and the signature.

An XML Signature may be applied to the content of one or more resources. Enveloped or enveloping signatures are over data within the same XML document as the signature; detached signatures are over data external to the signature element. TypeX attachments for example can be external resources, and thus could be signed and sent in TypeX message without having to transport the signed resource.

This specification also includes useful types that identify methods for referencing collections of resources, algorithms, and keying and management information. The XML Signature is a method of associating a key with referenced data (octets) and it itself is *not sufficient to address all application security/trust concerns*.

The basic structure of the XML Signature is presented in the following diagram:

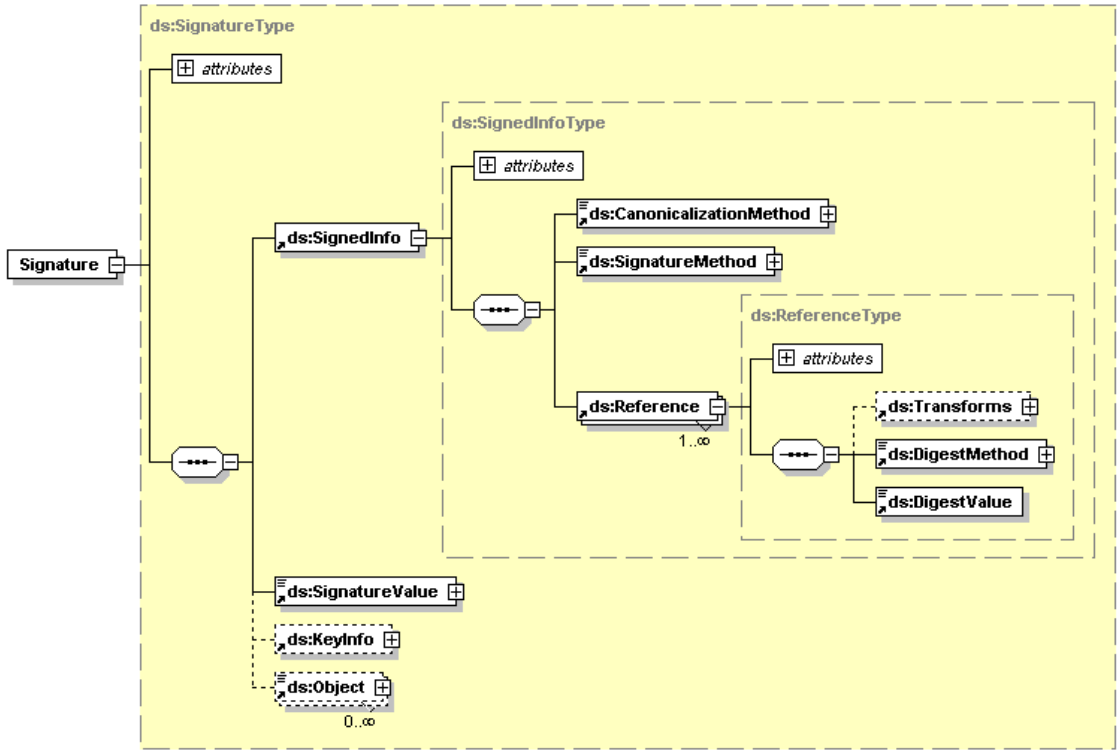


Figure 2 W3C XML Signature Schema

As shown in the diagram the XML Signature consists of these components:

- SignedInfo: The structure contains the references to all of the documents or portions thereof that are signed. A common way (and recommended) to specify the references is to use an XPATH transform, which is one of a number of transforms defined by the XML Signature specification. For each signed reference, there is an associated digest (unencrypted). There are two other elements:
  - Canonicalization Method : method to standardize the *SignedInfo* element (i.e. its parent)
  - Signature Method: method to used to generate the signature value
- SignatureValue: This is the signature generated by the hashing of the *ds:SignedInfo* element and its subsequent encryption.
- KeyInfo: Optional element that enables a recipient to obtain information concerning the key used for the encryption of the signature; if this element is not provided, the recipient is expected to obtain the key information in some other manner.
- Object: This is an element that may contain any data and can be used in a number of ways. This element can be viewed as an extension mechanism to enrich the XML Signature by introducing new elements, or it can contain the object being signed for enveloping signatures where the document being signed is to be included in the Signature element.

A sample XML signature follows. This example uses the RSA algorithm with SHA-1 hashing.

```
[s01] <Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
[s02]  <SignedInfo>
[s03]  <CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
[s04]  <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
[s05]  <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
[s06]   <Transforms>
[s07]    <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
[s08]   </Transforms>
[s09]   <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
[s10]   <DigestValue>dGhpcyBpcyBub3QgYSBzaWduYXR1cmUK.../DigestValue>
[s11]  </Reference>
[s12] </SignedInfo>
[s13]  <SignatureValue>...</SignatureValue>
[s17] </Signature>
```

As can be seen in the XML sample, there are two elements (s03 and s07) indicating the canonicalization method with value …c14n11. These elements define the method used to transform the XML resource being signed into a standard form, termed canonical, before transforming it into an octet stream suitable for digesting; the resulting octet stream of the transform is hashed.

This is necessary for XML since it is processed using standard XML parsing and processing techniques which frequently changed the message syntactically. Common examples are:

- Positioning and inclusion of namespaces
- Elimination of white spaces (e.g. <MyTag > is equivalent to <MyTag>)
- Transformation of end of line characters
- Attribute ordering

Thus the signing process differs slightly than previously presented the Methods section, in that the message is canonicalized before applying the hashing function to generate the digest. It should be noted that canonicalization is only one of a number of transforms that can be applied to a message.

→ *It is important to note that it is the transformed message that is signed and not the original message.*

2.4.2.1 Processing

This section presents a summary of the signing procedure. The W3C procedure differs somewhat than the generic procedure presented in the *Fundamentals* section as follows. It is useful to refer to the W3C signature schema diagram above.

The signing procedure begins by populating a *ds:SignedInfo* element:

1. Select a portion of the message (termed a *ds:Reference*) to sign (for example using the XPATH transform)
2. Standardize (i.e. canonicalize) the selection using a W3C canonicalization transform
3. Hash the output of the canonicalization (step 2.)
4. Place the hash of step 3 in the W3C *ds:SignedInfo/ds:Reference/ds:DigestValue* element
5. Specify the algorithm (*ds:DigestMethod*) and transforms (*ds:Transforms*).
6. Repeat steps 1 to 5, for each portion of the message to sign

The final steps will now create the signature proper:

7. Specify the signature algorithm (*ds:SignatureMethod*) and canonicalization method (*ds:CanonicalizationMethod*) in the *ds:SignedInfo* element.
8. Canonicalize the *ds:SignedInfo* element.
9. Apply the signature algorithm to the output of the canonicalization method (step 8).
10. Place the result of the signature algorithm (step9) in the W3C *ds:SignatureValue* element (the signature of the message).

The validation process is the reverse of the signing procedure.

2.4.2.2 Considerations

In general, it must be appreciated that XML Signature specification is still in its infancy; the W3C group will release a new version of XMLSIG (version 1.1) in the near future, and is considering more radical changes in future versions to address flaws in the design of the specification as well as current security issues [XSIG, XSBP, XSTS, XSUC]. The core difficulties can be directly attributed to the nature and processing of XML; this is in contrast to signing binary or non-XML data.

A few important issues are:

- Canonicalization performance: very expensive; judicious to use hardware appliance where performance is required.
- Interoperability is an issue due to the problems with canonicalization of the xml as well as the complexity and flexibility of the specification; for example, even with a given *CanonicalizationMethod*, the resulting XML may yield a different digest when depending on the processing.
- Attacks The current specification is open to a number of attacks such as denial of service attacks when using XSLT or XPATH transforms. The proposed TypeX Security Extension in this document addresses this point by restricting the transforms to XPATH Filter 2.
- Complexity The specification is very (read *too*) flexible with numerous features; the result is that it can be difficult for an application to know what is signed. The proposed TypeX Security Extension in this document addresses this point by constraining the available features and imposing the XPATH Filter 2 transform.
- Limitations This will be alleviated in upcoming releases, to allow a richer set of encryption and hashing algorithms.

2.4.2.3 Signature Extension: XAdES

W3C has published an extension for XML Signature called XAdES [XADES] which stands for Xml Advanced Electronic Signatures. It is a W3C note that extends XMLSIG in the domain of long term non-repudiation in order to be compliant with the European Directive on electronic signatures. The reader is referred to European Telecommunications Standards Institute (ETSI) web site for the latest specification.

The XAdES extension is achieved using the *dsig:object* component of *ds:Signature*, as shown in the schema above. XAdES extension includes signature properties for time stamping, since standardized time stamping is *not* provided in the W3C XML Signature specification.

Time-stamping is achieved using a trusted authority which basically countersigns the signature of the document with a timestamp and its private key.

In order to provide more robust and longer protection of a signature, the XAsES specification provides for including additional data, so called archive validation data. The motivation is that over time, the cryptographic algorithms and hash functions used to create the signature are no longer secure. These data include:

- the references to the CA certificates used to validate the signature, which consist of the digest of each certificate, the issuer and the serial number identifier.
- the values of certificates used to validate the signature, capturing all of the certificates from the certification path from the signer to the trusted root.
- the references to the full set of revocation information used for the verification of the electronic signature, defining the type of revocation information (e.g. CRL), the issuer, etc…
- the full set of revocation information used for the verification of the electronic signature containing the all the revocation information.

These validation data and other attributes are time-stamped along with the document signature.

Another consideration, although of somewhat lesser importance, is the possibility that over time the hash function and encryption method of the Time-Stamping trusted authority may be deemed not secure; it assumed that the trusted authorities will use very strong algorithms, lessening the probability of a breach. To protect against this, XAdES defines a mechanism for nesting timestamps using a different algorithm; thus signature properties data may contain multiple embedded timestamps.

In addition, new validation data may also be added to the signature if some certificates have expired, needing to be replaced by new ones.

The XAdES specification defines different forms, where a form is a set of signature properties. Each form can be viewed as corresponding to a level of non-repudiation. For example, a signature with a simple time-stamp is defined by the form XAdES-T; the form XAdES-X-L called *Extended long electronic signatures with time*, includes all the archive validation data providing the strongest non-repudiation over time.

There are commercial and open source tools for XAdES; there is currently no standard Java XAdES package.

A sample W3C signature with a XAdES extension is presented below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ds:Signature Id="Signature-ID1" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
                <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
                <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                   <ds:Reference Id="SignedData" URI="
    http://www.sitatest.org/mydoc.pdf">...</ds:Reference>                    <ds:Reference
    Id="XAdESProps"                          Type="http://uri.etsi.org/01903#SignedPropertie
    URI="#SignedProperties"> …</ds:Reference>

        </ds:SignedInfo>
        <ds:SignatureValue>cxlmyba...</ds:SignatureValue>
        <ds:Object xmlns:xa="http://uri.etsi.org/01903/v1.3.2#">
          <xa:QualifyingProperties Target="#Signature-ID1">
                <xa:SignedProperties Id="SignedProperties">
                  <xa:SignedSignatureProperties>
                        <SigningTime>2009-10-18T11:20:00Z</SigningTime>
                        <xa:SigningCertificate>
                          <xa:Cert>
                            <xa:CertDigest>
                                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                <ds:DigestValue>iKad...</ds:DigestValue>
                            </xa:CertDigest>
                            <xa:IssuerSerial>
                                 <ds:X509IssuerName>OU=SITA-TYPEX, O=SITA-TYPEX-ROOTC
SIG, C=FR</ds:X509IssuerName>
                                <ds:X509SerialNumber>1234876599</ds:X509SerialNumber>
                            </xa:IssuerSerial>
                          </xa:Cert>
                        </xa:SigningCertificate>
                  </xa:SignedSignatureProperties>
                </xa:SignedProperties>
          </xa:QualifyingProperties>
        </ds:Object>
</ds:Signature>
```

Reference to data to be signed

Reference to XAdES signature properties.

Timestamp of the signature

Digest of the timestamping authority's X-509

Issuer and serial number of the timestamping authority's certficate

2.4.2.4 How to implement?

The implementation of a signing tool could be achieved using Java security and encryption packages, as well as a PKI tool (this can also be achieved with the *java.security.cert* package) to validate the certificate and to provide time stamping. Signatures can be implemented using the *java.xml.crypto* package.

The implementation can be made simpler by imposing constraints, which is the motivation for the TypeX Security Extension described later in this document. Some of the constraints include:

- What can be signed: for example enforcing that the entire document be signed always, rather than individual portions.
- Allowed transforms: for example disallow the use of XSLT.
- Complexity of the signing: for example limit the depth of counter-signatures.

The implementation can be made more complex by removing the above constraints and adopting an extension scheme such as XAdES.

There are also a number of commercial and open source tools, as well as signing services. Of possible interest is the OASIS Digital Signature Service (DSS) core specification [DSSC]; DSS defines

request/response protocols for the processing of digital signatures for Web services and other applications.

2.4.3 XML Encryption

The W3C XML Encryption specification specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. This specification makes use of some of the XML Signature definitions, specifically *ds:KeyInfo* and *ds:Transform*.

The result of encrypting data is an *EncryptedData* element which contains (via one of its children's content) or identifies (via a URI reference) the cipher data.

When encrypting an XML element or element content the *EncryptedData* element replaces the element or content (respectively) in the encrypted version of the XML document.

Thus the specification provides fine encryption granularity, enabling to selectively encrypt sensitive elements. This capacity is also useful when certain portion of a document must be manipulated by intermediary nodes, and thus cannot be encrypted.

When encrypting arbitrary data (including entire XML documents), the EncryptedData element may become the root of a new XML document or become a child element in an application-chosen XML document.

A sample XML message with encrypted data is shown below. This example demonstrates the granularity of encryption that is possible with this specification. It also demonstrates data hiding: the malicious eavesdropper does not know how payment was made, since the *<Credit Card>* tag is hidden; a less secure alternative would be to only encrypt the content of the *<Number>* tag.

Before	**Encrypted**
<?xml version='1.0'?> <PaymentInfo xmlns='http://example.org/paymentv2'> <Name>John Smith</Name> <CreditCard Limit='5,000' Currency='USD'> <Number>4019 2445 0277 5567</Number> <Issuer>Example Bank</Issuer> <Expiration>04/02</Expiration> </CreditCard> </PaymentInfo>	<?xml version='1.0'?> <PaymentInfo xmlns='http://example.org/paymentv2'> <Name>John Smith</Name> <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element' xmlns='http://www.w3.org/2001/04/xmlenc#'> <CipherData> <CipherValue> A23B45C56… </CipherValue> </CipherData> </EncryptedData> </PaymentInfo>

Similarly to the XML Signature specification, the XML Encryption provides syntax to specify encryption metadata such as the algorithm, key information, etc…

2.4.3.1 Considerations

- Encrypted data may contain dangerous data such as virus, malicious executable code, etc…; therefore the receiving application needs to anticipate and be ready to deal with such potential hazards.

- Simultaneous use of encryption and signing may cause problems; this is discussed below.

The application of both encryption and digital signatures over portions of an XML document can make subsequent decryption and signature verification difficult. In particular, when verifying a signature one must know whether the signature was computed over the encrypted or unencrypted form of elements.

The recommendation is to include the data and the signature in the encryption, to avoid potential attacks and issues in the decryption process. In other words, it is recommended to sign first, followed by the encryption. This approach ensures confidentiality as well as integrity and authentication.

2.4.3.2 How to implement?

The solutions are the same as for the digital signature:

- Java security and encryption packages (*java.security* and *javax.crypto*);
- If public key encryption is used, then a PKI tool is required for obtaining validating the certificate; note that the Java package *java.security.cert* may also be used to achieve this.

2.4.4 W3C Best Practices

The following list summarizes the W3C best practices:

- Ensure that the application can easily discover what is signed
- Avoid the use of XSLT and XPATH in transforms to minimize attacks
- If a document needs to be signed and encrypted, then sign the document before encrypting it, including the signature.
- Sign as much of the document as possible to avoid substitution of data
- Use the transform XPATH Filter 2.0 to select portions of a document to sign, so that application can easily discover what is signed, and to avoid denial of service attacks.
- Timestamp the signature to avoid replay
- Use namespaces and avoid default schema value in order to minimize issues when validating a signature

More details may be found in the references.

2.5 WS-Security

The WS-Security suite of specifications is aimed at secure XML messaging over SOAP.

2.5.1 What is WS-Security?

WS-Security is an OASIS standard specification that describes a protocol [WSS] for securing web service message exchanges over SOAP. The protocol specifies how to enforce and declare security token transport, message integrity and message confidentiality in SOAP messages. In addition to the protocol, WS-Security specifies an XML schema that defines how to communicate metadata such as binary tokens, signatures and encryption algorithms. The WS-Security specification allows for a variety of security tokens: username/password, X.509 certificate, Kerberos and SAML.

WS-Security is a core specification for service security, relying on two other specifications: XML Encryption [XENC] and XML Digital Signatures [XSIG], which are described in the previous section.

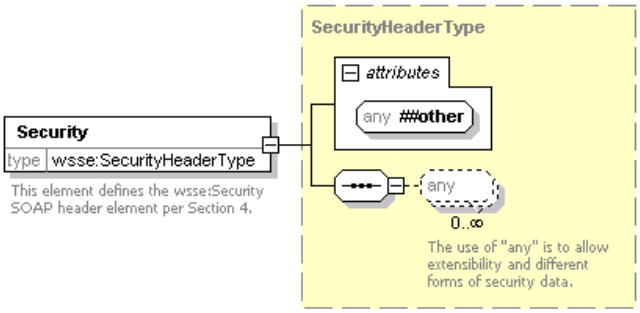
The WS-Security specification is broader in scope than these two W3C specifications and provides the basis for a secure communication framework. For example, WS-Security can be used to exchange security tokens such as a Kerberos ticket.

WS-Security addresses end to end security, where end to end security is defined as the preservation of message integrity and confidentiality between the originator of the message and the intended recipient of that message, whether there are intermediate nodes or not. In contrast, a protocol such TLS addresses point to point security.

It introduces several notions, in particular:

- **Claim**: a declaration made by an entity (e.g. name, key, group, privilege)
- **Security Token**: a collection (one or more) of claims
- **Security Token Service (STS)**: a system authority that issues renews releases and validates security tokens.

WS-Security also introduces a new header entry with the `wsse:Security` element.



Schema 1 – WS-Security header

There are a number of extensions to the WS-Security specifications to provide a complete secure communication framework which is presented in the following sub-sections. In case these extensions add new header elements these are always added as child nodes of `wsse:Security`.

2.5.1.1 WS-Trust

This specification [WST] builds on the extensibility of WS-Security. Its purpose is to define a mechanism to disseminate credentials across different trust domains using a secure token service. In addition, the specification defines means to negotiate, create and renew security tokens among partners within different trust domains. A secure token service (STS) is a web service that issues security tokens by mapping tokens.

For each token exchange, the STS may validate the incoming token. For example, if the STS is required to validate a X-509 certificate, it will contact the appropriate certificate authority (PKI) to validate the incoming certificate.

An STS is typically extended to play the role of identity provider (IP) which authenticates token requestors.

The following diagram demonstrates a contrived use of the WS-Trust specification. The WS-Trust messaging uses the request/response pattern with the messages RequestSecurityToken (RST) and the RequestSecurityTokenResponse (RSTR).

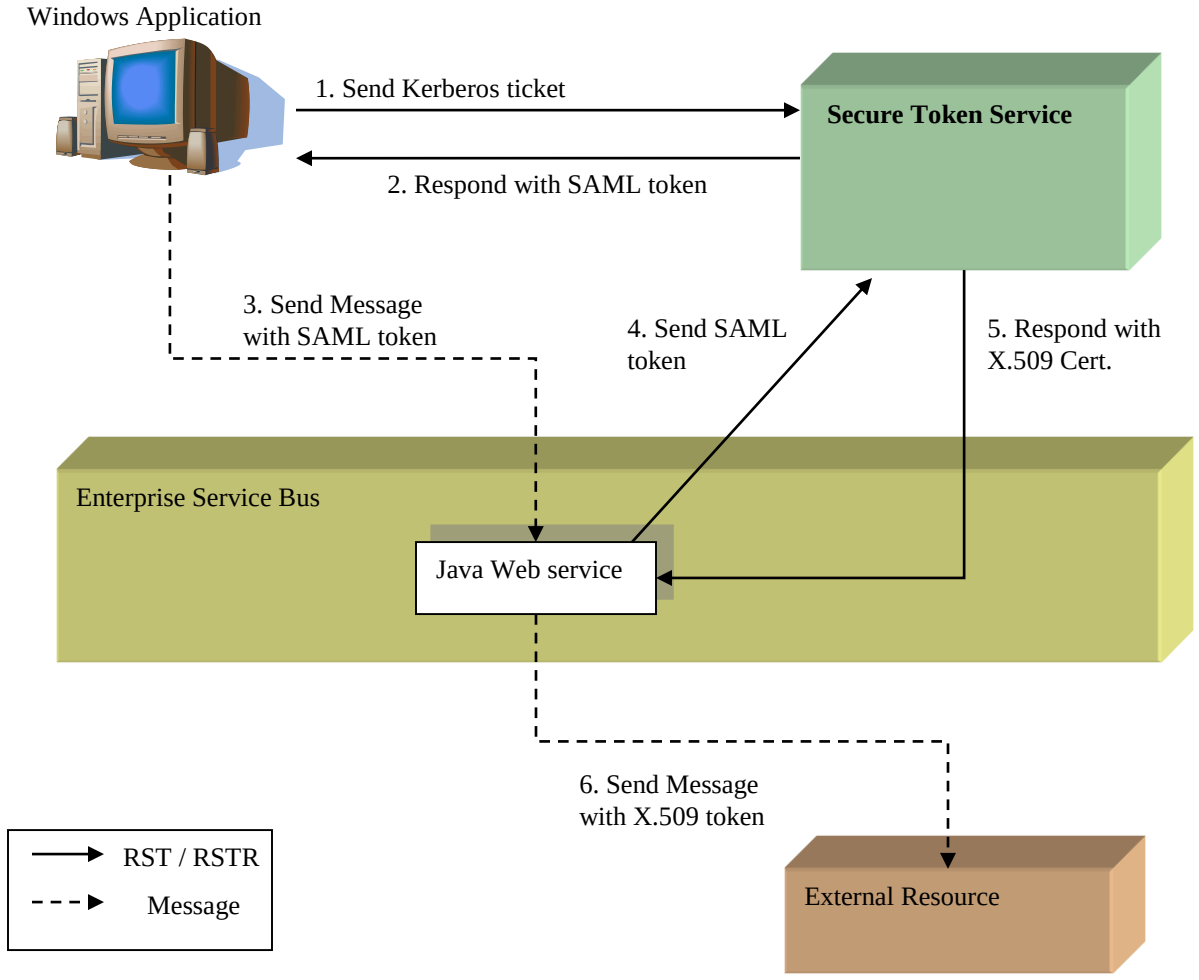


Figure 5-1 WS-Trust example

The sequence is the previous figure is as follows:

1. The Windows application requests a SAML token based on its Kerberos ticket (RST message).
2. The STS returns a SAML token in a WS-Security header (RSTR response).
3. The Window application posts a message to the Java Web Service with the SAML toke in a WS-Security header.
4. The Java Web Service requests a certificate based on the SAML token in order to access the external service (RST message).
5. The STS returns a certificate (RSTR response).
6. The Web service posts a message to the external resource with the certificate in a WS-Security header.

The specification WS-Federation [WSF] builds on WS-Trust to provide brokerage of trust claims. Concretely, it aims to facilitate the management of identities and authentication across security domains in such way as to eliminate centralization of user identities and authorizations, enabling requestors to access resources outside their security domains.

A simple example, taken for the specification [WSF] is shown in the following diagram (note IP stands for Identity Provider). In this instance there are two security domains. The requestor first obtains the necessary token from its STS (1). The requestor then sends its request to the resource, including its credentials (2). The resource will validate the credentials sent by the requestor with its own STS (3) before returning a response. The STS exchange Federation metadata that describe any information deemed necessary for the federated relationship.

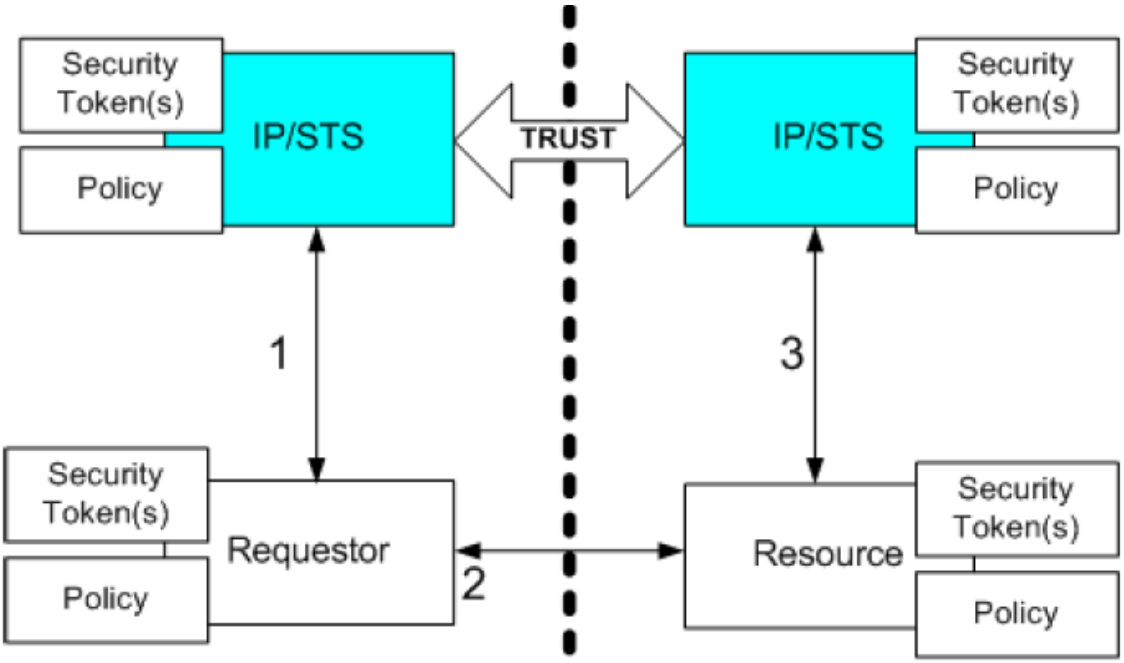


Figure 5-2 WS-Federation example

As another example, consider an enterprise that has many partners; this enterprise offers services with different policies which require different credentials. Each partner has its own trust domain with a STS. In order to avoid centralization of user accounts of each partner, the enterprise uses the WS-Federation protocol to obtain dynamically credentials from each partner's trust domain whenever one of its services receives a request. The only requirements are that the partners be known to the enterprise, and that the type of claims and other metadata must be agreed upon between all partners (for example there could be a Purchasing Claim requiring special authorization)

2.5.1.2 WS-SecureConversation

This specification [WSC] defines extensions to WS-Security and builds on WS-Trust to establish a security context that is valid for the life of an exchange session; this is useful for multiple message exchanges, essentially to increase the overall performance. For permanent TypeX connections, where many messages are exchanged, this protocol is recommended.

The specification basically defines a protocol to establish and maintain a security context token that is shared by the participating parties for the lifetime of a session. Concretely this means that a shared secret key is used by the exchanging participants; this key will often be dynamically altered during a session in order to minimize the possibility of compromising the secret key. This specification also leverages, optionally, WS-Addressing [WSA] to provide references of the issuer of the security token.

The TLS protocol utilizes this approach by first establishing a secure context with X.509 certificates in order to distribute a shared secret key.

2.5.1.3 WS-SecurePolicy

The WS-Policy framework was specified to enable a web service to express as well as communicate constraints, requirements and properties. The constraints and requirements are expressed as policy assertions.

Concretely, these assertions are XML sentences. In a security context, the WS-SecurityPolicy [WSP] standard defines a set of security policy assertions compatible with the WS-Policy framework. These security policy assertions define how a message is to be secured.

A related specification is WS-PolicyAttachment [WSPA]. This specification defines how policies may be referenced in WSDL and UDDI.

2.5.1.4 Security tokens

The standard defines three types of security tokens:

- **Simple token**: based on username password (e.g. X.509 certificate token)
- **Binary token**: based on certificates keys or tickets (e.g. Kerberos token)
- **XML token**: based on XML-formatted claims (e.g SAML token)

Hereafter is a sample of a Request security token/Request security token response (RST/RSTR) exchange for SAML token issuance:

```
(01)      <S:Envelope>
(02)       <S:Header>
(03)        <wsse:Security S:mustUnderstand="1">
(04)         <wsse:BinarySecurityToken wsu:Id="SecurityToken-id"
   EncodingType="http://.../oasis-200401-wss-soap-message-security-
   1.0#Base64Binary" ValueType="http://.../oasis-200401-wss-x509-token-
   profile-1.0#X509v3">
(05)         client X509 Certificate
(06)         </wsse:BinarySecurityToken>
(07)         <ds:Signature>
(08)          <ds:SignedInfo>
(09)          ...
(10)          </ds:SignedInfo>
(11)          <ds:SignatureValue>
(12)          ...
(13)          </ds:SignatureValue>
(14)          <ds:KeyInfo>
(15)           <wsse:SecurityTokenReference>
(16)             <wsse:Reference URI="#SecurityToken-id"
   ValueType="http://.../oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
(17)           </wsse:SecurityTokenReference>
(18)          </ds:KeyInfo>
(19)         </ds:Signature>
(20)         ...
(21)        </wsse:Security>
(22)       </S:Header>
(23)       <S:Body>
(24)        <wst:RequestSecurityToken>
(25)         <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-
   trust/200512/Issue</wst:RequestType>
(26)         <wsp:AppliesTo>
(27)          <wsa:EndpointReference>
(28)
   <wsa:Address>https://test.webservices.amadeus.com/1ASIWPOC1A</wsa:Addre
   ss>
(29)          </wsa:EndpointReference>
(30)         </wsp:AppliesTo>
(31)         <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-
   token-profile-1.1#SAMLV2.0</wst:TokenType>
(32)        </wst:RequestSecurityToken>
(33)       </S:Body>
(34)      </S:Envelope>
```

Listing 1 – Request Security Token

Lines (07) to (19) are used to establish the identity of the client through an XML signature couple with a certificate X.509 key (Lines (04) to (06)).

Line (25) indicates the request for a token issuance for the endpoint defined at line (28) and which is expected to be of type SAML version 2.0 as stated at line (31).

```
(01)     <S:Envelope>
(02)      <S:Header>
(03)       <wsse:Security S:mustUnderstand="1">
(04)        <wsu:Timestamp>
(05)         <wsu:Created>2009-11-08T11:03:20Z</wsu:Created>
(06)         <wsu:Expires>2009-11-08T11:08:20Z</wsu:Expires>
(07)        </wsu:Timestamp>
(08)       </wsse:Security>
(09)      </S:Header>
(10)      <S:Body>
(11)       <wst:RequestSecurityTokenResponseCollection>
(12)        <wst:RequestSecurityTokenResponse>
(13)         <wst:TokenType>http://docs.oasis-open.org/wss/oasis-
   wss-saml-token-profile-1.1#SAMLV2.0</wst:TokenType>
(14)         <wst:RequestedSecurityToken>
(15)          <saml2:Assertion ID="uuid-76a197ba-c2e8-4704-97a2-
   147db7619f2c" IssueInstant="2009-11-08T11:03:20Z" Version="2.0">
(16)           ...
(17)           <saml2:Subject>
(18)            <saml2:NameID
   NameQualifier="STS">jsmith</saml2:NameID>
(19)            <saml2:SubjectConfirmation
   Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches"/>
(20)           </saml:Subject>
(21)           ...
(22)          </saml:Assertion>
(23)         </wst:RequestedSecurityToken>
(24)         <wst:RequestedAttachedReference>
(25)          <wsse:SecurityTokenReference>
(26)           <wsse:KeyIdentifier ValueType="http://.../wss/oasis-
   wss-saml-token-profile-1.1#SAMLID">uuid-76a197ba-c2e8-4704-97a2-
   147db7619f2c</wsse:KeyIdentifier>
(27)          </wsse:SecurityTokenReference>
(28)         </wst:RequestedAttachedReference>
(29)         ...
(30)         <wsp:AppliesTo>
(31)          <wsa:EndpointReference>
(32)           <wsa:Address>
(33)           https://test.webservices.amadeus.com/1ASIWPOC1A
(34)           </wsa:Address>
(35)          </wsa:EndpointReference>
(36)         </wsp:AppliesTo>
(37)         <wst:Lifetime>
(38)          <wsu:Created>2009-11-08T11:03:20.344Z</wsu:Created>
(39)          <wsu:Expires>2009-11-08T11:09:20.344Z</wsu:Expires>
(40)         </wst:Lifetime>
(41)        </wst:RequestSecurityTokenResponse>
(42)       </wst:RequestSecurityTokenResponseCollection>
(43)      </S:Body>
(44)     </S:Envelope>
```

Listing 2 – Request Security Token Response

Lines (15) to (22) contain the returned SAML assertion. This assertion will be then included in the Security header of subsequent requests to the Web Service.

Note

In order to access the STS the requestor has to give a proof of his identity. It can be done with any security token type, for example by providing a username token or an X.509 certificate token previously delivered by a STS (that can be the same).

2.5.1.5 UsernameToken Profile

Trust specifications introduce dedicated operations for the security token management. It is described in a WSDL. The STS therefore implements the WSDL as a server and the Customer application as a client.

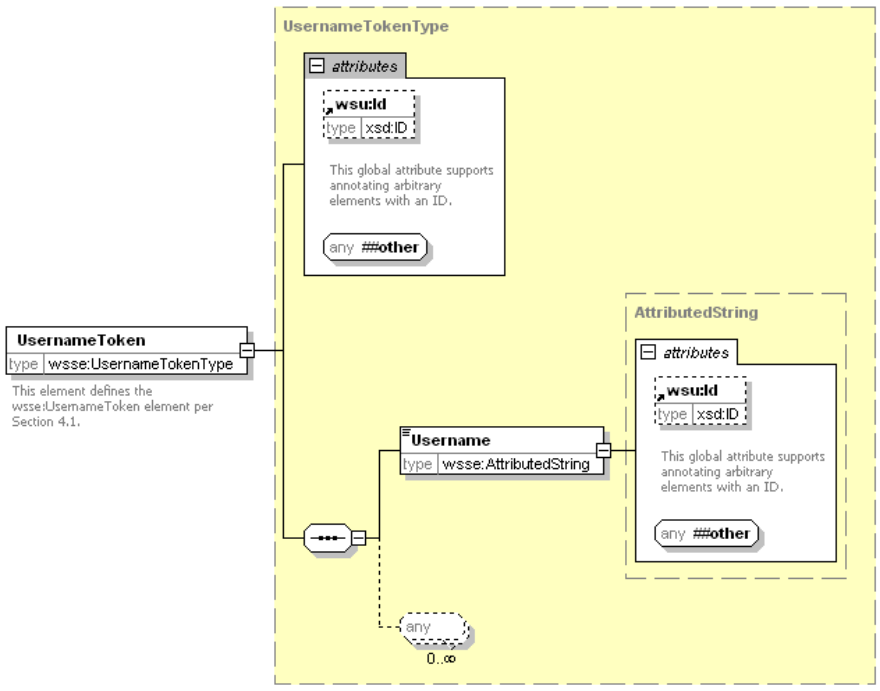
Besides Trust specifications offer the ability to specify some policies for the STS WSDL. In particular some security policies apply to the STS services, as the usage of a security token. This can be a security token issued by another STS or a simple token.

In order to keep the process as simple as possible and to minimize the number of actors, the STS will expect a Username token to be used.

The Username Token Profile 1.0 specifications [UTPS] describe use of elements defined by the WS-Security in order to allow a web service consumer to supply a UsernameToken as a means of identifying the requestor by “username”, and optionally using a password.

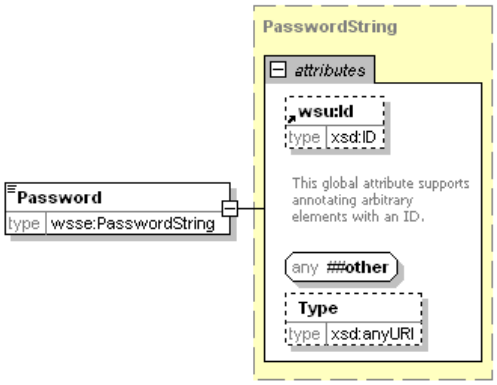
. The used elements are:

- The wsse:UsernameToken including by default a wsse:Username element and allowing extensions



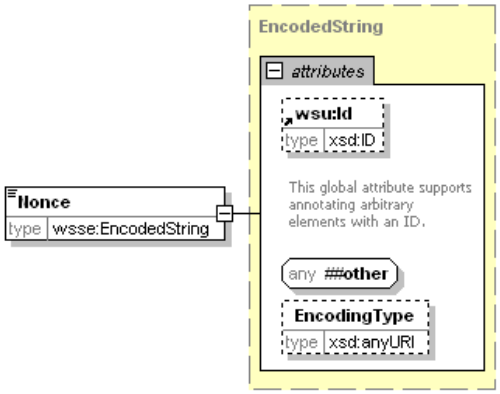
WS-Security: UsernameToken

- A wsse:Password element as optional extension. When a password is used for authentication, the password needs to be properly protected using a digest algorithm for example.



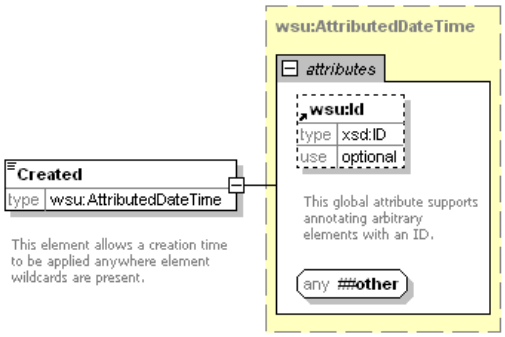
WS-Security: Password

- A wsse:Nonce element as optional extension. It contains the base64 encoded value of a nonce. A nonce is a random value generated by the sender and for which the server maintains a cache of used nonces. This element is used to avoid replay attacks.



WS-Security: Nonce

- A wsu:Created element as optional extension. It is a timestamp that enables to limit the cache to freshness time period.



WS-Security: Created

2.5.2 Soap example

This section proposes an implementation of the WS-Security in a SOAP context.

- User ID: used for the identification of the user
- Password: to ensure the security

- Nonce: a random number used to encode the password and avoid replicate attacks
- Timestamp: the date and time of the message

2.5.2.1 Example

Here is an example of implementation for a SOAP request:

XPath	Element / Attribute	Properties	Content
wsse:Security/wsse:UsernameToken/wsse:Username	<wsse:Username>		UserID used
wsse:Security/wsse:UsernameToken/wsse:Password	<wsse:Password>	Encrypted	Hashed Password
wsse:Security/wsse:UsernameToken/wsse:Password#Type	@Type	Fixed value	"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest"
wsse:Security/wsse:UsernameToken/wsse:Nonce	<wsse:Nonce>	Unique (a user in an Organization cannot re-use the same Nonce during 30minutes)	Filled according to the standard (random value)
wsse:Security/wsse:UsernameToken/wsse:Nonce#EncodingType	@EncodingType	If provided, only one encoding type is accepted	"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
wsse:Security/wsse:UsernameToken/wsse:Created	<wsu:Created>	UTC Format: yyyy-mm-ddTHH:MM:SSZ or yyyy-mm-ddTHH:MM:SS.sssZ GMT time Allowed offset with server time is +/- 30 min	The value is set according to the W3C XML schema dateTime type definition It is set to the creation time of the message.

The following namespaces must be used:

- wsu: WS-Security utility version 1.0: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
- wsse: WS-Security version 1.0: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

This leads to the following example:

```
<SOAP:Envelope
   xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
   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"
   <!-- Omitted Namespaces --> >
   <SOAP:Header>
      <wsse:Security>
         <wsse:UsernameToken>
            <wsse:Username>User</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
username-token-profile-1.0#PasswordDigest">KNEdk3v33PQjby6BUB3ehf3nPXg=</wsse:Password>
            <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary">VZzFq9RQL69dDrf0bcoNeg==</wsse:Nonce>
            <wsu:Created>2014-01-23T12:27:39.173Z</wsu:Created>
         </wsse:UsernameToken>
      </wsse:Security>
      <!-- Omitted "WS-Addressing" & "Security" headers  -->
   </SOAP:Header>
   <SOAP:Body>
      <!-- Omitted Request -->
   </SOAP:Body>
</SOAP:Envelope>
```

2.6 Considerations

WS-Security is aimed at SOAP based web service security

- complexity considering the interdependence of specs
- tools immaturity mileage may vary with different implementations;
- exploitable or buggy tool may jeopardize security
- must choose security model carefully so that it is compatible with the quality of service requirements
- considerations described in the W3C Security section are applicable
- performance

2.6.1 How to implement?

Both open source (e.g. Apache's Rampart and WSS4J) and commercial implementations (e.g. Bloombase, Oracle) are available.

In addition, there is a demonstration of WS-Security use with Type X over SOAP, using its Type X Java Framework in IATA Type X Implementation Guide..

2.7 Type X Security

This chapter presents guidelines on how to secure Type X messaging at message level by proposing a new Type X message level security binding. The binding addresses encryption and digital signing of a *TXM_Envelope*.

The goal of this binding is provide an interoperable mechanism to enable secure TypeX message exchanges regardless of the underlying transport protocol.

2.7.1 Guiding Principles

The current binding proposition describes the binding in terms of application processing rules; this is in contrast to W3C schema extensions.

The following principles are observed in this binding proposition:

- Adherence to existing standards: concretely this imposes the use of the W3C recommendations for signature and encryption.
- Simplicity: only allow what is necessary in the simplest manner possible; discard the rest.
- Independence: TypeX Security binding must maintain the standalone aspect of TypeX
- Evolutivity: Impose very loose coupling in order to evolve simply and quickly.

> **Important**: The TypeX Security binding proposed in this document may evolve after discussions with interested parties, and after it is implemented and tested against a number of typical use cases.

As with the W3C XML Signature and Encryption specifications, this TypeX Security binding is not sufficient to address all application security/trust concerns such as authentication and authorization. It is expected that the exchanging participants address these concerns by using PKI and/or a suitable protocol.

2.7.2 TypeX Security Extension

This section proposes a new TypeX extension to enable digital signature and encryption. The extension is termed "TXM_Security" for TypeX Message Security.

2.7.2.1 Namespaces

The following new namespaces are defined for signature and encryption, respectively:

- txmsig : http://www.iata.org/txm/sig
- txmenc : http://www.iata.org/txm/enc

2.7.2.2 Data Structure

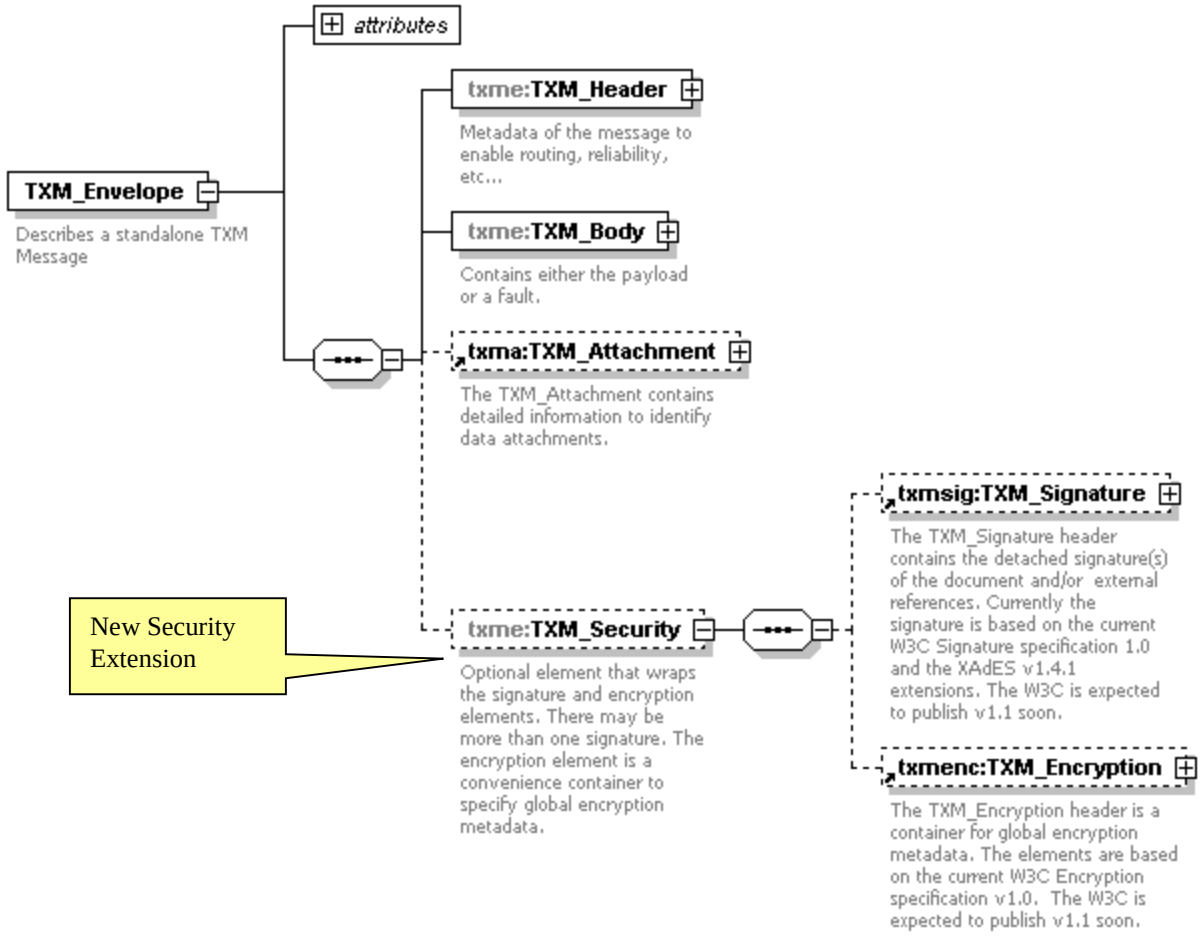


Figure 6-2 TypeX Security Extension

2.7.3 Encryption

This section presents the encryption portion of the TypeX Security binding, termed *TXM_Encryption*.

The proposed *TXM_ Encryption* serves as a container for global encryption metadata, and encrypted keys for multiple recipients. Thus the encryption processing does not differ than that specified in the W3C XML Encryption specification:

- Composes with the W3C XML Encryption standard
- Obeys W3C best practices
- Achieves simplicity by imposing restrictions
- Achieves usability by imposing restrictions
- Anticipates proposed future W3C XML Encryption syntax and processing
- Facilitates interoperability and adoption

2.7.3.1 TypeX Constraints

A TypeX message contains data that need to be updated in order for the message to be routed to the intended recipients. Thus a secured TypeX message must allow legitimate routing intermediaries and gateways to update the message.

The elements of a TypeX envelope that **may** be encrypted are listed here; all other elements **must** never be encrypted.

1. */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader*
2. */txme:TXM_Envelope/txme:TXM_Body/txme:TXM_Payload*
3. */txme:TXM_Envelope/txme:TXM_Body/txmr:TXM_Report*
4. */txme:TXM_Envelope/txme:TXM_Body/txmf:TXM_Fault*
5. */txme:TXM_Envelope/txma:TXM_Attachment*
6. */txme:TXM_Envelope/txme:TXM_Security/txmsig:TXM_signature*

The elements from the *//txmm:TXM_MessageHeader* that **must** be excluded from encryption are:

- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Destination*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Originator*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/PossibleDuplicateMessage*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/RepeatedMessageId*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/MessageId*

In some contexts, it may be judicious to sign the TypeX envelope, before encryption to ensure message integrity, concerning notably the list of recipients (*//Destination*) and the originator (*//Originator*). A possible alternative to signing could be to use an encryption algorithm that uses the message digest as an explicit parameter.

2.7.3.2 Multiple Recipients

Multiple recipients present another challenge. In this instance, the encrypted message must be decrypted by more than one recipient. Two alternatives are addressed.

The first is concerned with recipients having the capacity to decrypt the data using their own private keys. The general solution is to encrypt the data using a random symmetric key and chosen algorithm (e.g. AES). This random symmetric key is then encrypted using each user's public key; thus N recipients will result in N encrypted keys. It should be noted that each encrypted key is distinguished with a unique identifier for each recipient; for example using the W3C XML Encryption, this can be achieved using the *ds:EncryptedKey/@Recipient* or the *ds:KeyInfo/ds:KeyName* elements.

The process of encrypting a key is termed key encapsulation, in contrast to encrypting data which is termed data encapsulation.

The generic encryption process is as follows:

1. Generate a random symmetric key, ***K***, which will be used for encrypting the data
2. Encrypt the data using the key ***K*** and the chosen algorithm
3. For each recipient: encrypt the key ***K*** using the recipient's public key

The generic decryption process is as follows for each recipient:

1. Read the corresponding encrypted key
2. Decrypt that encrypted key using the recipient's secret private key, obtaining the original key ***K***.
3. Decrypt the data using the key ***K*** and the indicated algorithm.

→ A sample encrypted TypeX envelope using this approach is provided at the end of this chapter.

In the second alternative, the targeted recipients and the sender share a long term secret symmetric key, which we will call ***S***. In this instance, the encryption process is as follows:

1. Generate a random symmetric key ***K*** which will be used for encrypting the data
2. Encrypt the data using the key ***K*** and a chosen algorithm
3. Wrap* (encrypt) the key ***K*** using the shared secret key ***S***.

The decryption for *all* recipients is simply the reverse process:

1. Decrypt that encrypted key using the shared key ***S***, obtaining the original key ***K***.
2. Decrypt the data using the key ***K*** and the indicated algorithm.

* The term “wrap” is used to refer to key wrapping (a form of key encapsulation) in order to distinguish it from the previous key encapsulation mechanism presented in the first scenario using asymmetric key encryption.

These key encapsulation approaches are **recommended** for all encryption of a TypeX envelope, even for a single recipient. As well as enabling encryption of multiple recipients, the use of symmetric encryption greatly improves performance in terms of size and speed, since the message is *encrypted only once* using the random symmetric key.

Upcoming updates to the W3C specification (e.g. version 1.1 and beyond) may enable more possibilities.

The most recent document available is the W3C Working Draft 30 July 2009 of version 1.1; there is also an extension document for hybrid ciphers (i.e. the use asymmetric and symmetric encryption as described in the first scenario above).

2.7.3.3 Data Structure

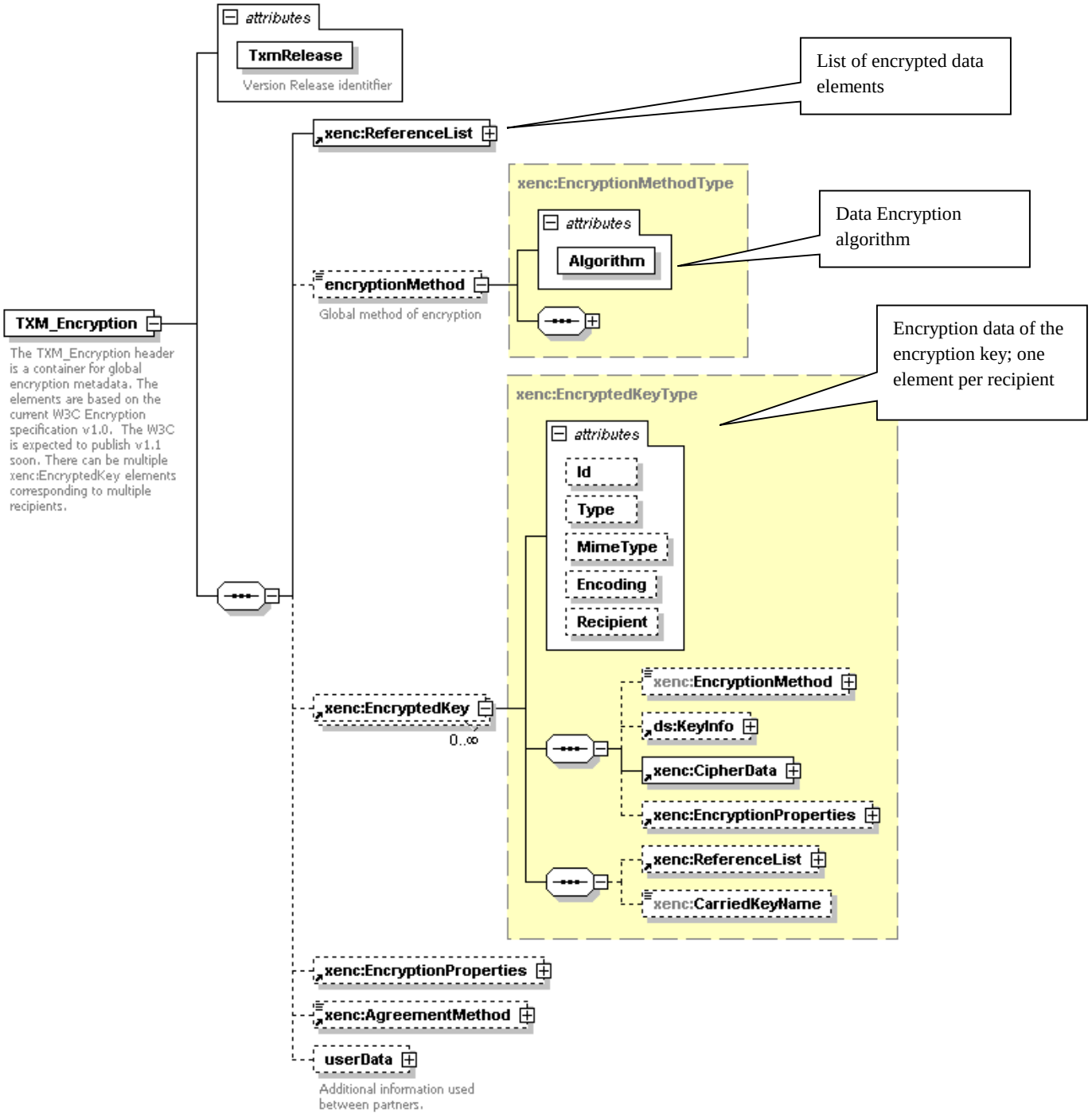


Figure 6-3 TypeX Encryption Extension

The *txme:TXM_Envelope/txme:TXM_ Security/txmsig:TXM_Encryption* extension serves as a container for global encryption metadata as well as multiple encrypted keys, each corresponding to a recipient. It also specifies the list of encrypted elements in *xenc:ReferenceList*.

2.7.3.4 Invariants

- All global (i.e. shared) encryption metadata **should** be contained in the element *txme:TXM_Envelope/txme:TXM_ Security/txmsig:TXM_Encryption*, in order to simplify the decryption process by centralizing the data.
- The *xenc:ReferenceList/xenc:DataReference* elements of the *txmenc:TXM_Encryption* element **must** be used to specify the encrypted elements. The *xenc:DataReference/@URI* **must** refer to the *xenc:EncryptedData/@Id* of the encrypted elements. For example, the URI="#Elem3" refers to the Id="Elem3". The essential purpose is to signal a receiving application that the envelope is encrypted.
- If the document is signed, then the signature **must** be encrypted, in order to avoid attacks.
- The *ds:RetrievalMethod* element of **must** be used with care and only if necessary; it is recommended not to use this element, in order to avoid attacks.
- All encrypted keys **must** be contained in *txmenc:TXM_Security*, in order to simplify the decryption process.
- All *xenc:cipherdata* elements **must** always contain the *xenc:cipherValue*; the element *xenc:cipherReferences* is not allowed; this is to eliminate potential dangerous references and transforms.
- The *xenc:ReferenceList* elements of the *xenc:EncryptedKey* elements **must** not be used for the same reasons.
- The *xenc:EncryptedData/@Type* attribute **must** be "element"; this means that the *child* elements (and their content) of the elements listed in the *TypeX Constraints section* (numbered 1 to 6) are encrypted. This approach has the advantage of hiding the data.
- The only exception to the previous rule: If a signature is present, then it **must** be encrypted as "content" to enable optional signing after encryption.
- Default namespaces **must** not be used in the *TXM_Envelope*, in order to avoid canonicalization issues; in other words, namespaces should be explicitly declared.
- If *xenc:EncryptedKey/@Recipient* attribute **should** contain the TypeX Address in string format (i.e. the fields of a TypeX Address separated by underscores) *if* the encrypted key is specific to a single recipient (see section 3.3.2) to facilitate processing. For recipients that are not assigned TypeX addresses (i.e. the recipient has a TypeX sub-address), then the *ds:keyInfo* element **must** to be provided to distinguish multiple recipients through a single gateway; in this instance the *Recipient* **must** bethe gateway.

2.7.3.5 Processing Rules

2.7.3.5.1 Encryption

*→ Encryption processing **must** be performed after all XML processing that may affect the content of the TXM_Envelope to be sent. In the current TypeX specification, the only processing that may affect content is the inclusion (inlining) of external attachments in the TypeX envelope.*

The following sequence **must** be respected in order to encrypt a TXM_Envelope:

1. Select the encryption algorithm
2. Obtain or create an encrypting key.
3. Encrypt the TypeX envelope elements
4. Process the TypeX envelope by inserting the encrypted data
5. If multiple recipients are targeted, create the encrypted keys for each recipient

2.7.3.5.2 Decryption

The following sequence **must** be respected in order to encrypt a TXM_Envelope:

1. Determine the encryption algorithm
2. Determine the encryption key.
3. Decrypt the encryption key.
4. Decrypt the TypeX envelope elements
5. Process the TypeX envelope by inserting the decrypted data

2.7.3.5.3 Intermediary Nodes

Intermediary TypeX Nodes receiving need to be careful in validating encrypted TypeX envelopes. Since the elements are not encrypted, but only their content, an intermediary node should only validate structure for the encrypted portions of the TypeX envelope and not content. For the unencrypted elements, full validation is possible and recommended.

2.7.4 Signature

This section presents the signature portion of the TypeX Security binding, termed *TXM_Signature*.

The proposed *TXM_Signature* binding has the following essential characteristics:

- Composes with the W3C XML Signature standard
- Employs a minimal set of recommended transforms: XAPTH Filter 2.0 and Canonicalization
- Selection of nodes to be signed is achieved with XPATH Filter 2.0 so that it is simple for an application to determine what is signed.
- Obeys W3C best practices
- Achieves simplicity by imposing restrictions as described in the remaining sections
- Achieves usability by imposing restrictions as described in the remaining sections
- Anticipates proposed future W3C XML Signature syntax and processing
- Facilitates interoperability and adoption

2.7.4.1 TypeX Constraints

A TypeX message contains data that need to be updated in order for the message to be routed to the intended recipients, as discussed in the Encryption section. In addition, in order to minimize certain attacks, all of the relevant portions of a TXM_Envelope must be signed.

It should be noted that the concern is the integrity of the TypeX envelope as a whole; thus a payload may be already signed or not before being included in a TypeX envelope.

The applicable constraints for signing a TypeX Envelope differ slightly from those for TypeX encryption; the included and excluded elements are presented here.

The elements of TypeX envelope that **must** be signed are listed here:

- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader*
- */txme:TXM_Envelope/txma:TXM_Body/txmr:TXM_Report*
- */txme:TXM_Envelope/txma:TXM_Body/txmf:TXM_Fault*
- */txme:TXM_Envelope/txme:TXM_Body/txme:TXM_Payload*
- */txme:TXM_Envelope/txma:TXM_Attachment*

All other elements **must** never be signed. Thus, there is always at least two internally signed references (i.e. the *txme:TXM_Body* elements and the *txmm:TXM_MessageHeader*) ; if the optional *txma:TXM_Attachment* is present, then it must also be signed.
Similarly, the list of elements from the *//txmm:TXM_MessageHeader* that **must** be excluded from the signature is:

- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Destination/RecipientInformation/ResponsibilityFlag*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Destination/NodeTrace*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/PossibleDuplicateMessage*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/RepeatedMessageId*
- */txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/MessageId*

All other elements of *//txmm:TXM_MessageHeader* **must** be signed.

2.7.4.2 Data Structure

The extension serves as a container for all signatures of a *TXM_Envelope*.

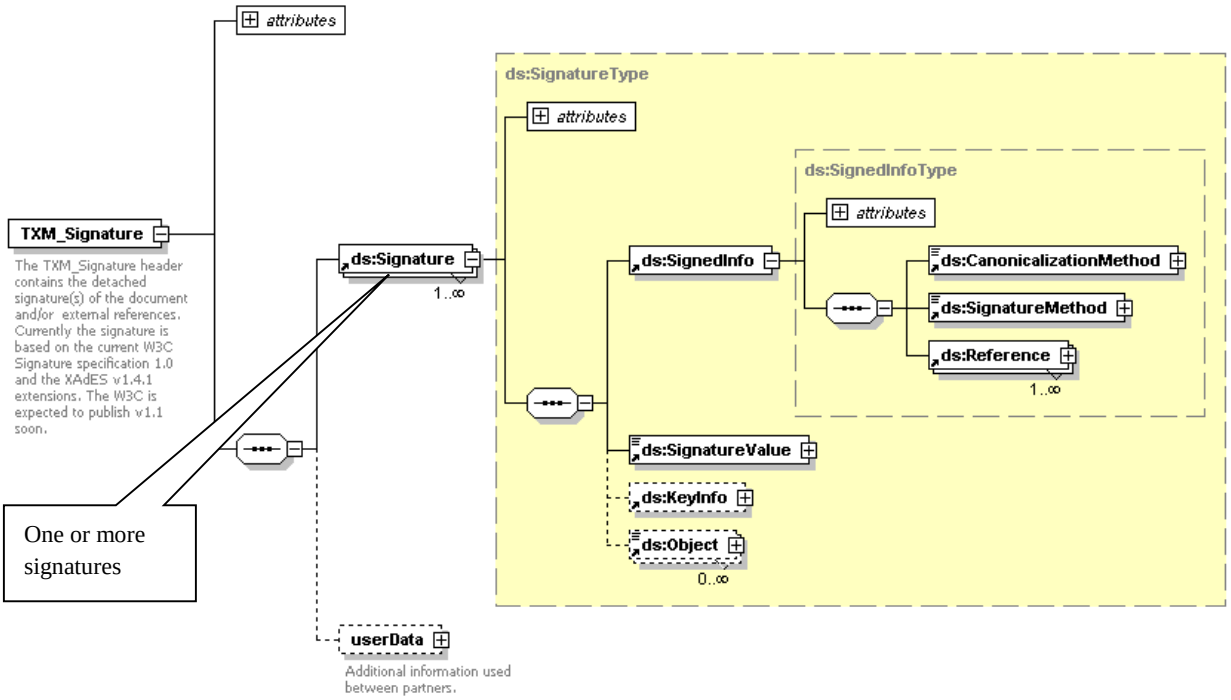


Figure 6-4 TypeX Signature Extension

In order to better understand the content of this section, the W3C XML Signature schema for a data object to be signed, i.e. a dsig:SignedInfo/*dsig:Reference*, is shown in the following diagram.

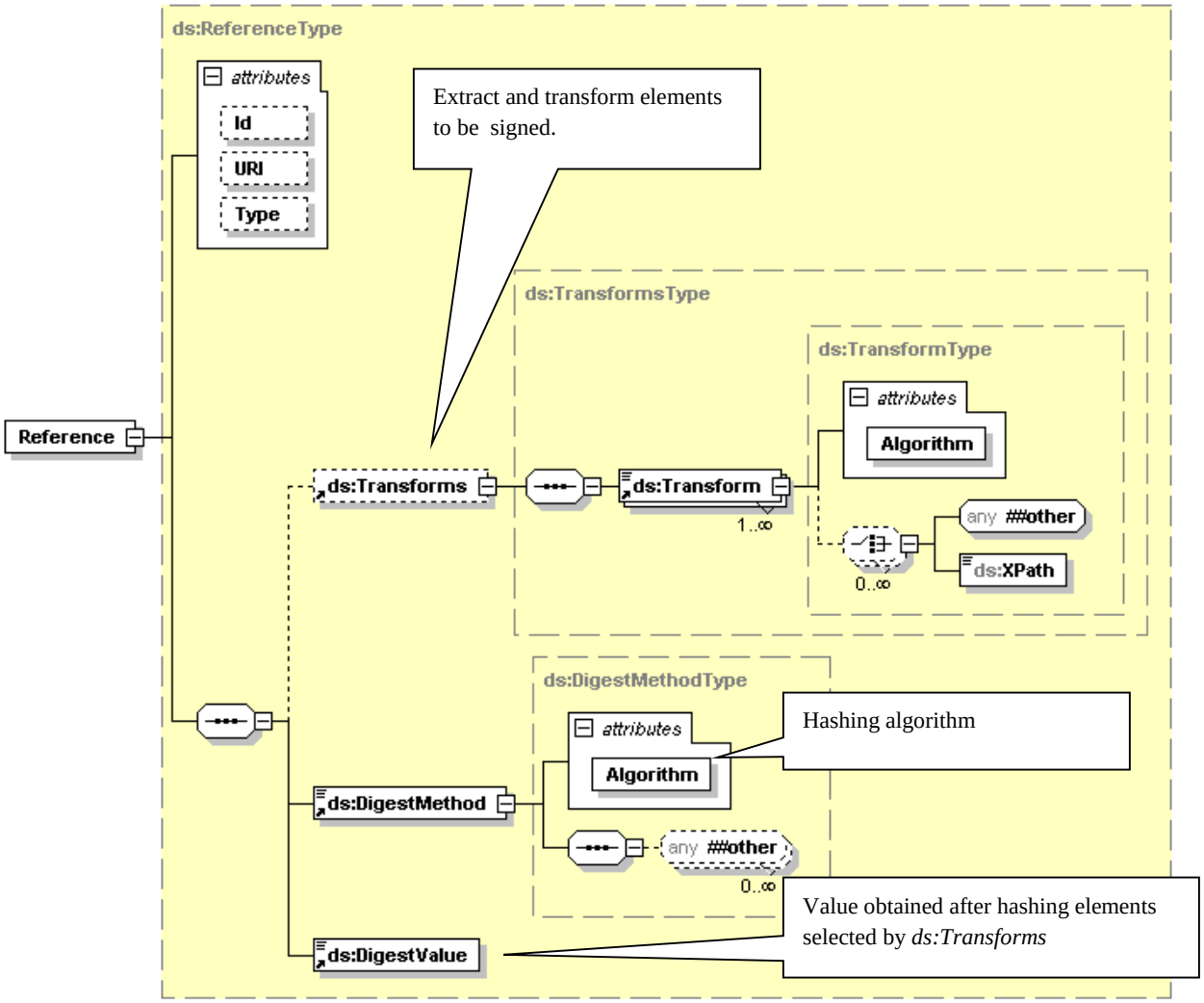


Figure 6-5 XML Signature *ds:Reference*

2.7.4.3 Invariants

This section presents all of the rules associated to signing a TypeX envelope. Any deviation from these rules should result in the rejection of the signed document by the verifier.

The following presents a graphical overview of the TypeX Signature rules.

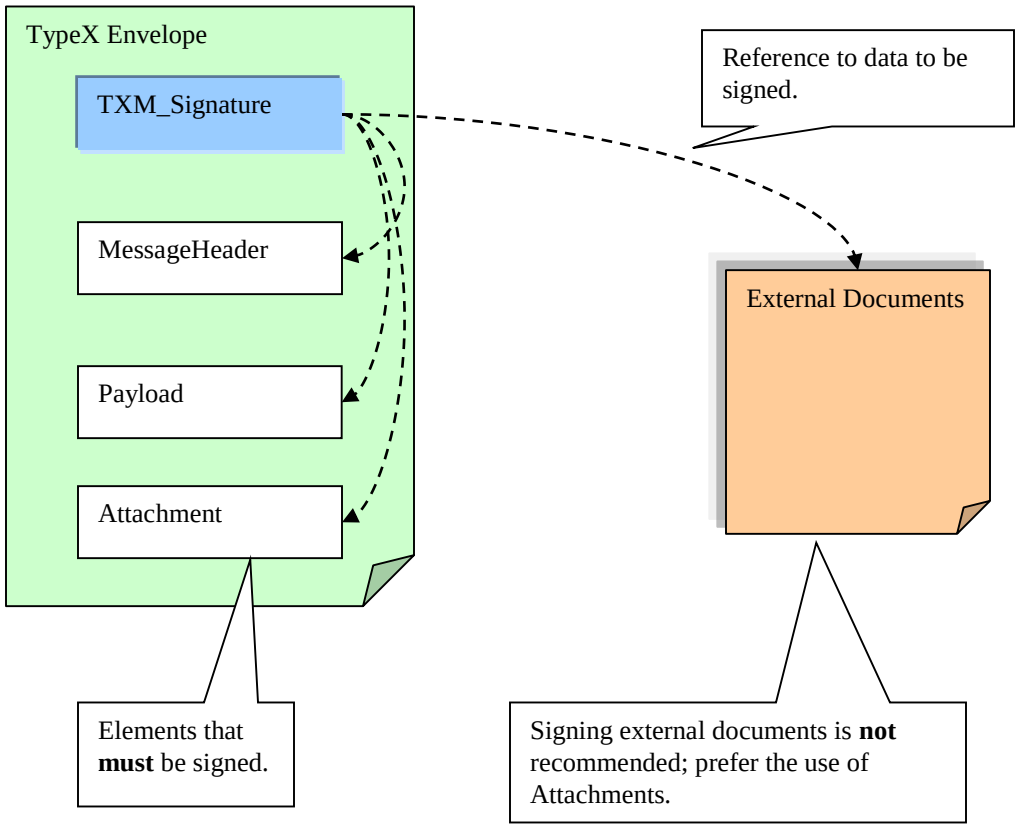


Figure 6-6 TypeX Signature Rules

The TypeX Signature rules are all contained in the following list:

- All Signatures **must** be contained in the elements *txme:TXM_Envelope/txme:TXM/txmsig:TXM_Signature*; this form of signature is termed a *detached* signature.
- The *txme:TXM_Payload* **must** be signed.
- The *txmm:TXM_MessageHeader* **must** be signed.
- The *txma:TXM_Attachment* **must** be signed if present. These last three rules are equivalent to saying that the entire TypeX envelope must be signed, while respecting the TypeX constraints presented above.
- The transform algorithm *XPATH Filter2.0* **must** be used for selecting nodes to be signed, in order to eliminate potential attacks and to ensure that the receiving application may clearly discover what is signed.
- The transform algorithm *XPATH Filter2.0* **must** be the only transform used for selecting nodes,

for the same reasons just mentioned.

- For internal signatures, i.e. signatures referring to content in the *TXM_Envelope*, the URI of the reference **must** be empty (null); the only exception is if the URI refers to a manifest of references, in which case the URI **must** refer to the *Id* of the *dsig:Object/dsig:Manifest* which is simply a list of references. Again, the rationale is to eliminate potential attacks; since the URI is empty, the XPATH transforms define the XML node set (portion of the TypeX envelope) to sign.
- For internal signatures, the syntax **must** conform to that specified in the next section (see XML snippets); to summarize, one mandatory *intersect* filter followed by zero or more *subtract* filters. This rule simply enforces XAPTH Filter 2.0 syntax.
- For multiple signatures, the signed reference elements **must** be placed in a *dsig:Object/dsig:Manifest*, to maintain simplicity and performance.
- External signatures, being the signature of documents external to the TypeX Envelope are **not recommended**; again the rationale is to eliminate potential attacks (see also section 3.4.5).
- If an external document is required to be signed, then the URI of the reference **must** either be known (at least its root) by the verifier or **must** conform to the URI standard (for example XPOINTER is not allowed); the preferred approach is to copy the external document into *txma:TXM_Attachment*, again to avoid attacks (see also section 3.4.5).
- All external documents **must** be assumed to be binary, requiring the base64 transform as the only transform; the idea is to ensure that the document is signed verbatim, regardless of its format.
- A single canonicalization transform **must** the last transform to be executed after the reference transforms and before hashing. Since the document being signed is a TXM_Envelope, the Canonical XML 1.1 transform is recommended. This rule simply constrains possible transforms.
- Each reference **must** be canonicalized unless it is a binary external document, to ensure interoperability and that the signature remains valid.
- The *dsig:RetrievalMethod* element **must** not be used; this means that each signature has its own *dsig:KeyInfo*, if required. Again, the rationale is to minimize attacks, such as denial of service or the use of harmful transforms.
- It is strongly recommended to timestamp signatures, such as the proposed in the XAdES extension, in order to provide for stronger non-repudiation.
- Default namespaces **must** not be used in the *TXM_Envelope*; this may cause issues during the signature validation since processing of the received envelope may inject the empty namespace *xmlns=""* to get rid of ancestry namespaces.
- Default element values **should** not be used. Again, this may cause issues during the signature validation since processing of the received envelope may inject default value elements, which will cause the validation of the signature to fail.

The current version 1.0 of XML Signature restricts the choice of digest and signature algorithms. For example the only digest algorithm that is specified is SHA-1, which is now considered vulnerable (see for example here). The upcoming version 1.1, currently a working draft, and extensions will enable the use of more robust algorithms (see also the section 2.2.2.4 Hash Function).

2.7.4.4 Internal Signature

This section presents in detail the syntax for internal signatures, where internal refers to the TypeX envelope itself, including attachments that may be themselves a reference to an external document. The assumption is the document to be signed is a single TypeX envelope.

External signatures which are signatures of documents residing outside the TypeX envelope are treated in the next section.

2.7.4.4.1 Signing txme:TXM_Payload

For the signing the *txme:TXM_Payload*, the following base *dsig:reference* structure is provided. It is recommended to use this structure as presented.

```
<dsig:Reference URI="">
   <dsig:Transforms>
    <dsig:Transform  Algorithm=”http://www.w3.org/2002/06/xmldsig-filter2”
          xmlns:txme=”http://www.iata.org/txm/envelope”>
      <dsig-xpath:XPath Filter="intersect">
          /txme:TXM_Envelope/txme:TXM_Body/txme:TXM_Payload
      </dsig-xpath:XPath>
     </dsig:Transform>
     < dsig:Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </dsig:Transforms>
       ...
</dsig:Reference>
```

2.7.4.4.2 Signing txma:TXM_Attachment

For the signing the *txme:TXM_Attachment*, the following base *dsig:reference* structure is provided. It is recommended to use this structure as presented. It should be noted that if the attachment is a global reference (i.e. an external document), then only the reference to the document is signed. If the external document needs to be signed, then an external signature is required (see the external signature section).

```
<dsig:Reference URI="">
   <dsig:Transforms>
    <dsig:Transform  Algorithm=”http://www.w3.org/2002/06/xmldsig-filter2”
          xmlns:txme=”http://www.iata.org/txm/envelope”
          xmlns:txma=”http://www.iata.org/txm/attachment”>
      <dsig-xpath:XPath Filter="intersect">
          /txme:TXM_Envelope/txma:TXM_Attachment
      </dsig-xpath:XPath>
     </dsig:Transform>
     < dsig:Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </dsig:Transforms>
       ...
</dsig:Reference>
```

2.7.4.4.3 Signing txmr:TXM_Report

For the signing the *txmr:TXM_Report*, the following base *dsig:reference* structure is provided. It is recommended to

use this structure as presented.

```
<dsig:Reference URI="">
   <dsig:Transforms>
    <dsig:Transform  Algorithm=”http://www.w3.org/2002/06/xmldsig-filter2”
          xmlns:txme=”http://www.iata.org/txm/envelope”
          xmlns:txmr=”http://www.iata.org/txm/report”>
      <dsig-xpath:XPath Filter="intersect">
          /txme:TXM_Envelope/txme:TXM_Body/txmr:TXM_Report
      </dsig-xpath:XPath>
     </dsig:Transform>
     < dsig:Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </dsig:Transforms>
       ...
</dsig:Reference>
```

2.7.4.4.4 Signing txmf:TXM_Fault

For the signing the *txmf:TXM_Fault*, the following base *dsig:reference* structure is provided. It is recommended to use this structure as presented.

```
<dsig:Reference URI="">
   <dsig:Transforms>
    <dsig:Transform  Algorithm=http://www.w3.org/2002/06/xmldsig-filter2
          xmlns:txme=http://www.iata.org/txm/envelope
          xmlns:txmf=http://www.iata.org/txm/fault>
      <dsig-xpath:XPath Filter="intersect">
          /txme:TXM_Envelope/txme:TXM_Body/txmf:TXM_Fault
      </dsig-xpath:XPath>
     </dsig:Transform>
     <dsig:Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
    </dsig:Transforms>
       ...
</dsig:Reference>
```

2.7.4.4.5 Signing txmm:TXM_MessageHeader

For the signing the *txmm:TXM_MessageHeader*, the following base *dsig:reference* structure is provided. It is recommended to use this structure as presented.

```
<dsig:Reference URI="">
```

```
<dsig:Transforms>
 <dsig:Transform  Algorithm=”http://www.w3.org/2002/06/xmldsig-filter2”
       xmlns:txme="http://www.iata.org/txm/envelope"
       xmlns:txmm="http://www.iata.org/txm/msgheader">
  <dsig-xpath:XPath Filter="intersect">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader
  </dsig-xpath:XPath>
  <dsig-xpath:XPath Filter="subtract">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Destination/RecipientInformation/ResponsibilityFlag
  </dsig-xpath:XPath>
  <dsig-xpath:XPath Filter="subtract">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Destination/NodeTrace
  </dsig-xpath:XPath>
  <dsig-xpath:XPath Filter="subtract">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/PossibleDuplicateMessage
  </dsig-xpath:XPath>
  <dsig-xpath:XPath Filter="subtract">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Information/MessageId
  </dsig-xpath:XPath>
 </dsig:Transform>
 <dsig:Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
</dsig:Transforms>
    ...
</dsig:Reference>
```

2.7.4.5 External Signature

This section present in detail the syntax for external signatures, where external refers to a document external to the TypeX envelope that is to be signed.
We recall that the *ds:SignedInfo* element contains *ds:Reference* elements, each identifying a document or XML node set to sign. For external signatures, the *ds:Reference/@URI* attribute refers to some document external to the TypeX Envelope carrying the signature. The danger is that the URI may be constructed as to cause problems.
Thus the strongly **recommended** approach is not to allow external references, but rather include the external document as an attachment in */TXM_Envelope/txma:TXM_Attachment*.
If the attachment approach is too restrictive, then the following recommendations for external signatures should be heeded. Again, if unknown URIs are accepted, then the verifier must be aware that for some URIs, there may be

negative side-effects.
All external documents **must** be assumed binary, so that only the base64 transform needs to be used.
The only acceptable syntax of URI **must** conform to the URI standard [URI].
An example would be to sign a PDF document referenced in the payload, where the *dsig:Reference/@URI* ="`http://www.boguscorp.example.com/mydoc.pdf`"
According to the rules previously presented in the *Invariants* section, the verifier should at least know or trust the root of the URI, which in this case: "`http://www.boguscorp.example.com/`".
The following example presents a reference to the external document *mydoc.pdf*.

```
<dsig:Reference URI="http://www.boguscorp.example.com/mydoc.pdf">
    <dsig:Transforms>
        <dsig:Transform  Algorithm=http://www.w3.org/2000/09/xmldsig#base64 />
    </dsig:Transforms>
     <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <ds:DigestValue>UjBsR09EbGhGUX... </ds:DigestValue>
</dsig:Reference>
```

2.7.4.6 Multiple Signatures

If multiple signatures are required, e.g. there are co-signers, then all *ds:Reference*s **must** be placed in the *dsig:Object* /*dsig:Manifest* element which is simply a container of *ds:Reference* elements. Thus, each signature is calculated on the same manifest. The W3C XML Signature specification provides the optional *dsig:Manifest* element as a means to meet requirements not addressed by the mandatory elements. The *ds:Reference/URI* of the manifest **must** obey the syntax as presented, which is the string "#<manisfest Id>".
The idea here is to centralize all document references in a single location, since for each signature there is a *ds:SignedInfo* element containing the references to the documents being signed.
If these references are not centralized, the encryption and decryption processes will be required to process the same references as many times as there are signatures. Thus with the manifest (list) of references, the references will be digested only once, and each signature being calculated on the manifest of references. During validation, the manifest of reference needs to validated only once for all signatures, since all signatures are calculated on the manifest.
The following snippet presents an example:

```
<dsig:Reference URI="#MySampleManifest">
        Type="http://www.w3.org/2000/09/xmldsig#Manifest">
      <Transforms>
          <Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
      </Transforms>
       ...
</dsig:Reference>
…
<dsig:Object>
  <dsig:Manifest Id="MySampleManifest">
        <dsig:Reference URI=""> …</dsig:Reference>
```

```
        <dsig:Reference URI=""> …</dsig:Reference>
        <dsig:Reference URI="http://www.my-external-ref.com/doc.pdf"> …</dsig:Reference>
    </ dsig:Manifest>
        ...
</dsig:Object>
```

2.7.4.7 Processing Rules

2.7.4.7.1 Signature Generation

→ The signature generation processing **must** be performed after all XML processing that may affect the content of the TXM_Envelope to be sent; for example, external attachments must be included before signing. If the TXM_Envelope is to be wrapped within another protocol (e.g. SOAP envelope), then the TXM_Envelope **must** be signed before it is wrapped.

The following sequence **must** be respected in order to generate a signed TXM_Envelope:

1. Create the *dsig:SignedInfo* element containing all referenced node sets (i.e. portions of the document), calculating the digest for each reference.
2. Calculate the *dsig:SignatureValue* by :
   2.1. apply the canonicalization transform to *dsig:SignedInfo*
   2.2. calculate the digest of the result of 2.1
   2.3. encrypt the result of 2.2
3. Construct the *dsig:Signature* element

2.7.4.7.2 Validation

→ The signature validation processing **must** performed before XML validation of the document, or any other form of XML processing that may affect the content of the received message.

The following sequence **must** be respected in order to validate a signed TXM_Envelope:

1. Validate the key of the signer, i.e. authenticate and establish trust.
2. For each *dsig:SignedInfo*
   - Validate the *dsig:*
3. Validate the *dsig:SignatureValue*

2.7.4.8 Sample TypeX Envelopes

This section presents two TypeX envelopes; the first is signed and the second is encrypted. Both examples use the same original TypeX envelope. The portion of XML pertaining to signing and encryption are highlighted in bold.

2.7.4.8.1 Signed TypeX Envelope

This section presents a signed TypeX Envelope. The MessageHeader and the Payload are both signed.

```
<?xml version="1.0" encoding="UTF-8"?>
<txme:TXM_Envelope TxmRelease="TXM2009A0"
xmlns:txme="http://www.iata.org/txm/envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <txme:TXM_Header>
              <txme:TXM_BodyHeader>
                     <txmm:TXM_MessageHeader TxmRelease="TXM2009A0"
                           xmlns:txmm="http://www.iata.org/txm/msgheader">
                            <Information>
```

```
                              <MessageId>
                                    <TypeXAddress>
                                          <TYPEX_Address>
                                                <Airline>TST</Airline>
                                                <City>CLT</City>
                                                <Department>TXM</Department>
                                          </TYPEX_Address>
                                    </TypeXAddress>
                                    <OriginDate>2008-07-15+02:00</OriginDate>
                                    <OriginTime>23:58:09.609+02:00</OriginTime>
                                    <Number>1</Number>
                              </MessageId>
                              <MessageReference>test it</MessageReference>

     <DeliveryNotificationRequest>No</DeliveryNotificationRequest>
     <PossibleDuplicateMessage>No</PossibleDuplicateMessage>
        <Priority>3</Priority>
        <SpecialAttentionNotification>0</SpecialAttentionNotification>
      <Subject>sub it</Subject>
     </Information>
                        <Originator>
                              <OriginatorAddress>
                                    <TYPEX_Address>
                                          <Airline>TST</Airline>
                                          <City>CLT</City>
                                          <Department>TXM</Department>
                                    </TYPEX_Address>
                              </OriginatorAddress>
                        </Originator>
                        <Destination>
                              <RecipientInformation>
                                    <ResponsibilityFlag>Yes</ResponsibilityFlag>
                                    <ActionType>TO</ActionType>
                                    <DestinationAddress>
                                          <TYPEX_Address>
                                                <Airline>1G</Airline>
                                                <City>NYC</City>
                                                <Department>AA</Department>
                                          </TYPEX_Address>
                                    </DestinationAddress>
                              </RecipientInformation>
                              <NodeTrace/>
                        </Destination>
                  </txmm:TXM_MessageHeader>
            </txme:TXM_BodyHeader>
     </txme:TXM_Header>
     <txme:TXM_Body>
            <txme:TXM_Payload>
                  …some XML payload …
            </txme:TXM_Payload>
</txme:TXM_Body>
<txme:TXM_Security>
  <txmsig:TXM_Signature TxmRelease="TXM2009A0"
      xmlns:txmsig="http://www.iata.org/txm/sig">
```

```
       <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-
c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
         <ds:Reference URI="">
          <ds:Transforms xmlns:ds-xpath="http://www.w3.org/2002/06/xmlds-filter2">
     <ds:Transform Algorithm="http://www.w3.org/2002/06/xmlds-filter2">
      <ds-xpath:XPath Filter="intersect">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader
         </ds-xpath:XPath>
       <ds-xpath:XPath Filter="subtract">
/txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader/Destin
ation/RecipientInformation/ResponsibilityFlag
        </ds-xpath:XPath>
       <ds-xpath:XPath Filter="subtract">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader
/Destination/NodeTrace
        </ds-xpath:XPath>
       <ds-xpath:XPath Filter="subtract">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader
/Information/PossibleDuplicateMessage
        </ds-xpath:XPath>
       <ds-xpath:XPath Filter="subtract">
       /txme:TXM_Envelope/txme:TXM_Header/txme:TXM_BodyHeader/txmm:TXM_MessageHeader
/Information/MessageId
          </ds-xpath:XPath>
       </ds:Transform>
       <ds:Transform Algorithm="http://www.w3.org/2002/06/xmlds-filter2">
          <ds-xpath:XPath Filter="intersect">
/txme:TXM_Envelope/txme:TXM_Body/txme:TXM_Payload</ds-xpath:XPath>
          </ds:Transform>
          <ds:Transform Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>UjBsR09EbGhGUX... </ds:DigestValue>
        </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>UjBsR09EbG ...</ds:SignatureValue>
       </ds:Signature>
     </txmsig:TXM_Signature>
   </txme:TXM_Security>
</txme:TXM_Envelope>
```

2.7.4.8.2 Encrypted TypeX Envelope

This section presents an encrypted TypeX Envelope. The MessageHeader and the Payload are both encrypted using a symmetric random key (***K***) and the AES algorithm. For each targeted recipient, the key ***K*** is then encrypted using the recipient's public key and the RSA algorithm, and stored in the *txmenc:TXM_Encryption/xenc:EncryptedKeys* element.

```
<?xml version="1.0" encoding="UTF-8"?>
<txme:TXM_Envelope TxmRelease="TXM2009A0"
```

```
 xmlns:txme="http://www.iata.org/txm/envelope"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<txme:TXM_Header>
           <txme:TXM_BodyHeader>
                 <txmm:TXM_MessageHeader TxmRelease="TXM2009A0"
                       xmlns:txmm="http://www.iata.org/txm/msgheader">
                       <Information>
                             <MessageId>
                                   <TypeXAddress>
                                         <TYPEX_Address>
                                               <Airline>TST</Airline>
                                               <City>CLT</City>
                                               <Department>TXM</Department>
                                         </TYPEX_Address>
                                   </TypeXAddress>
                                   <OriginDate>2008-07-15+02:00</OriginDate>
                                   <OriginTime>23:58:09.609+02:00</OriginTime>
                                   <Number>1</Number>
                             </MessageId>
<xenc:EncryptedData Id="#MH1" Type='http://www.w3.org/2001/04/xmlenc#Element">
     <xenc:CipherData>
       <xenc:CipherValue>ydUNqHkMrD...</xenc:CipherValue>
     </xenc:CipherData>
</xenc:EncryptedData>
     </Information>
                       <Originator>
                             <OriginatorAddress>
                                   <TYPEX_Address>
                                         <Airline>TST</Airline>
                                         <City>CLT</City>
                                         <Department>TXM</Department>
                                   </TYPEX_Address>
                             </OriginatorAddress>
                       </Originator>
                       <Destination>
                             <RecipientInformation>
                                   <ResponsibilityFlag>Yes</ResponsibilityFlag>
                                   <ActionType>TO</ActionType>
                                   <DestinationAddress>
                                         <TYPEX_Address>
                                               <Airline>1G</Airline>
                                               <City>NYC</City>
                                               <Department>AA</Department>
                                         </TYPEX_Address>
                                   </DestinationAddress>
                             </RecipientInformation>
                             <NodeTrace/>
                       </Destination>
                 </txmm:TXM_MessageHeader>
           </txme:TXM_BodyHeader>
     </txme:TXM_Header>
     <txme:TXM_Body>
           <txme:TXM_Payload>
<xenc:EncryptedData Id="#PAYLOAD" Type='http://www.w3.org/2001/04/xmlenc#Element">
```

```
      <xenc:CipherData>
        <xenc:CipherValue>ksdfJHsfL...</xenc:CipherValue>
      </xenc:CipherData>
  </xenc:EncryptedData>
            </txme:TXM_Payload>
 </txme:TXM_Body>
 <txme:TXM_Security>
    <txmenc:TXM_Encryption TxmRelease="TXM2009A0"
           xmlns:txmenc="http://www.iata.org/txm/enc">
      <xenc:ReferenceList>
         <xenc:DataReference URI="#MH1"/>
         <xenc:DataReference URI="#PAYLOAD"/>
       </xenc:ReferenceList>
       <txmenc:encryptionMethod
           Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes256"/>
      <xenc:EncryptedKey Recipient="1G_NYC_AA">
       <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
       <xenc:CipherData>
            <xenc:CipherValue>UjBsR0...</xenc:CipherValue>
       </xenc:CipherData>
      </xenc:EncryptedKey>
    </txmenc:TXM_Encryption>
  </txme:TXM_Security>
</txme:TXM_Envelope>
```

2.8 Conclusions

General Security Aspects:

- Even though there are sufficiently good cryptographic algorithms available, message level security is still "bleeding edge" technology
- PKI use is important and should be further defined in terms policy and appropriate certificate profile.
- To achieve better interoperability encryption and signature profiles should be formalized.
- Need to carefully evaluate tools: your mileage will vary greatly with respect to compliance, interoperability and performance.
- Exploitable or buggy tools will jeopardize security
- A complete security analysis is required in order to implement an appropriate security solution compatible with the requirements and constraints of a given context. This analysis should include all security aspects, not just message level security.

TXM_Security:

- In order to address current shortcomings of the W3C specification, the TypeX Security extension simplifies greatly digital signing of TypeX envelopes by constraining how signing is done, facilitating adoption and interoperability.
- The TypeX Security Encryption extension enables and facilitates encryption and signing for multiple recipients.
- Composes with W3C XML Signature and Encryption standards

2.9 References

NB: These references were current at the time of writing, but may have been superseded by more recent versions.

[DSSC] *Digital Signature Service Core protocols,* version 1.0, April 2007, http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf
[TYPEXSPEC] *TypeX Specification*, version 1.0, IATA SCR volume 7. Access via http://www.iata.org/whatwedo/workgroups/Pages/padis.aspx
[URI] *Uniform Resource Identifiers (URI): Generic Syntax,* January 2005, http://www.ietf.org/rfc/rfc3986.txt
[WSA] WS-Addressing 1.0, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509
[WSC] WS-SecureConversation 1.4, http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.html
[WSF] WS-Federation 1.2, http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html
[WSP] WS-SecurePolicy 1.2, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html
[WSPA] WS-PolicyAttachment 1.5, http://www.w3.org/TR/2007/CR-ws-policy-attach-20070228/
[WSS] WS-Security 1.1, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-176 spec-os-SOAPMessageSecurity.pdf
[WST] WS-Trust 1.3, http://docs.oasis-open.org/ws-sx/ws-trust/200512
[UTPS] Username Token Profile 1.0 specifications http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
[XADES] XML Advanced Electronic Signature version 1.4.1, 2009-06, http://webapp.etsi.org/XAdES
[XENC] XML Encryption Syntax and Processing, version 1.0, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/
[XSBP] XML Signature Best Practices, working draft, http://www.w3.org/TR/2009/WD-xmldsig-bestpractices-20090730/
[XSIG] XML Signature Syntax and Processing, version 1.0, http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/
[XSTS] XML Signature Transform Simplification: Requirements and Design, working draft, http://www.w3.org/TR/2009/WD-xmldsig-simplify-20090730/
[XSUC] XML Security Use Cases and Requirements, working draft, http://www.w3.org/TR/2009/WD-xmlsec-reqs-20090226/

2.10 Appendix

2.10.1 IATA Security-Extended Schemas

The IATA TypeX schemas that included the new Security Binding.
Schemas are available together with IATA Type X standard schemas.
More details may be found in the references.

Section 3: Application Messaging Guide

3.1 Purpose

The purpose of the IATA XML reliable messaging guide is to provide a standalone document that clearly describes the features and identifies the benefits and impacts of implementing XML reliable messaging based on the IATA reliable messaging requirements. The intent of this document is to help potential IATA standard users and implementer to make a technically informed decision on the choice of appropriate XML reliable messaging standard and related technology. Standards and technologies described in the current version of this document include IATA Type X Standard SCR Volume 7, JMS - Message Queuing and Web Service Reliable Messaging.

3.2 IATA Reliable Messaging Requirements

This section presents the reliable messaging requirements for air transport business data exchanges as they are identified and defined as a part of Type X Work Group. Some requirements may vary to a degree from one sector to another. These requirements are based on the premise that all IATA message exchanges are carried out over SOAP, HTTP or JMS. This suggests that reliable messaging using SOAP or JMS or over HTTP are possible ways of meeting this requirement.

3.2.1 Messaging Context

The following diagram presents the IATA messaging context that will be treated in this document.

IATA messages are exchanged with the messaging stack over a transport appropriate for each application (shown as dashed lines). The receiving point may in turn send the message to an appropriate end point or application.

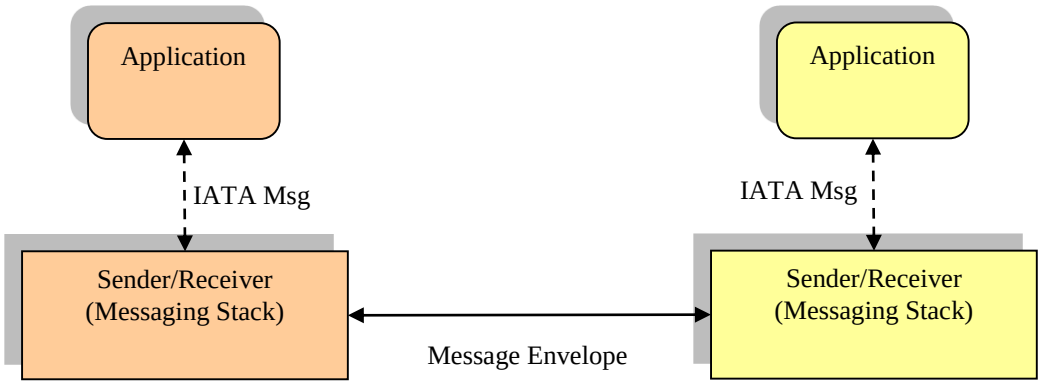


Figure 7 Messaging context

3.2.2 Reliable Messaging: Definition

For the purpose of this document, reliability is defined as the ability to guarantee the delivery of a message between two business applications, otherwise termed as an end-to-end exchange.

Reliable messaging is thus defined as the exchange of messages with reliability between two business applications, regardless of the message exchange pattern.

3.2.3 Reliable Messaging Requirements

The essential principle stipulated as prime requirement and driving factor is to have a transport and platform independent messaging specification for secure and reliable XML messaging compatible with air transport business practices.

The messaging requirements are as follows:

1. Specification must support for wider range of transport protocols including SOAP, HTTP, MQ and TCP

2. Specification must support Message Exchange Patterns consisting of:

   2.1. One-way messaging with no expectation of a response, there may be a response but not explicitly correlated to the message sent

   2.2. Asynchronous request/response exchange, the exchange consists of one or more request/response pairs

   2.3. Synchronous request/response exchange over a single two-way connection.

3. Specification shall be open and freely available
4. Specification shall support assured and secured delivery
5. Specification shall support message delivery to more than one recipient or end point

6. Specification shall contain elements supporting standard IATA addressing based IATA coding principles

7. Specification may support exchanges with X.400, URL, SMTP, Fax, Telex

8. Specification shall provide capability to detect duplicated messages

9. The sending application shall be informed of any failure of the message to arrive at its destination

10. Specification shall provide capability for sequence control

11. Specification shall have the ability of a sender to deliver a message once and only once to its intended receiver(s)

12. The sender shall be notified of message delivery failure

13. The sender shall be able to define a maximum life time for the message

14. Messages confirmed to be received by any given node will never be lost

15. Messages shall contain a priority indicator element

16. Error Messages are used to report permanent or transient problems or errors in a message

17. Support for global interoperability including a level of backward interoperability with legacy to manage transition
18. A Message can optionally include a Digital Signature so that:
    18.1. the identity of the party sending the message can be authenticated

    18.2. any change to the message can be detected.

19. Ability to authenticate a partner connection

20. Capability for message encryption

21. Capability for flow control and recovery

3.2.4 Glossary

Back Channel: response channel in a two-way transport protocol

Duplex exchange: bidirectional exchange over one or two connections, equivalent to two(2) One-Way exchanges

Endpoint: Node responsible for sending and receiving a message; the endpoint may or may not be the initiator of the message or the ultimate receiver of a message

End-to-End: Defined to be application to application, where the application is the entity that generates a message (initial sender) or is targeted by a message (ultimate receiver)

JMS : Java Messaging Service is a standard Java queue-based MOM API specification for loosely coupled, distributed , asynchronous message exchange between two or more endpoints

MEP: Message Exchange Pattern

MOM: Message Oriented Middleware (e.g. a JMS implementation such as Sun's Open MQ or IBM MQ)

Non-addressable endpoint: an endpoint that does not permit incoming connections

One-Way exchange: request-only (event) exchange over a single connection, with no response (i.e. payload) expected on the back channel

Session: A logical connection between two applications. The concept of session is decoupled from the underlying message transport and the exchange pattern

TXM: Type X Messaging

TXM Node: Endpoint that understands the Type X schema and implements the Type X protocol

Two-Way exchange: request/response exchange over a single connection

Web Service: [???]

3.3 WS-RX Overview

3.3.1 Introduction

This chapter presents an overview of the OASIS suite of specifications related to reliable messaging over SOAP, termed WS-RX. The WS-RX suite, as presented in the diagram below, includes:

- WS-RM: a reliable messaging protocol for SOAP
- WS-MakeConnection: a messaging protocol using the transport back-channel (e.g. HTTP reply)
- WS-RM Policy: basic assertions specific to WS-RM
- WS-Addressing: is used by WS-RX to identify SOAP endpoints (not applications) and specify exchange patterns, which are essential to both the WS-RM and WS-MakeConnection protocols.

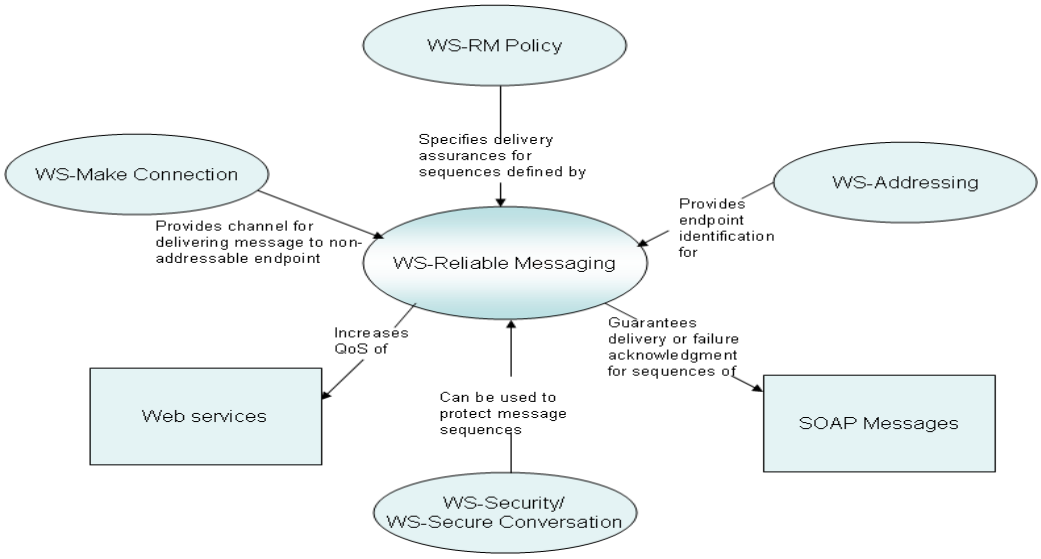


Figure 2 WS-RX components

3.3.2 WS-RX Messaging Context

The following diagram presents the WS-RX messaging context.

3.3.3 WS-RM

3.3.3.1 Description

The main goal of WS-RM is to provide a mechanism to assure a sending SOAP endpoint that the message arrives at the receiving SOAP endpoint. WS-RM specifies a SOAP based protocol for sending a message and getting an acknowledgment when that message is received. WS-RM also specifies the resending of messages that have not been acknowledged.

Exchange patterns are not known by WS-RM; for example in the case of a request/response, both the request and the response are separate reliable one-way messages with the correlation being achieved by the optional WS-Addressing *RelatesTo* element.

Messages are identified with a number that is unique within the context of a WS-RM sequence, each message contiguously numbered. A sequence itself is also uniquely identified. A sequence can be terminated when no more messages are to be sent or in case of some error.

WS-RM is essentially aimed at asynchronous exchanges, since it potentially makes all exchanges asynchronous, relying on WS-MakeConnection to obtain responses.

3.3.3.2 Interoperability

WS specifications have very specific focus to avoid overlap with any other WS specification. Enabling a complete functional area like reliable messaging requires composition with other WS specifications such as WS-Addressing, etc. which through time and versioning progress may create an interoperability issue in a community wide and global context. There are also interoperability issues between SOAP versions. WS-I organization spends significant effort in tightening the specifications by providing documents that essentially detail real world scenarios and how each participant should behave.

3.3.4 WS-RM Messaging Model

The WS-RM messaging model is presented in the following diagram.

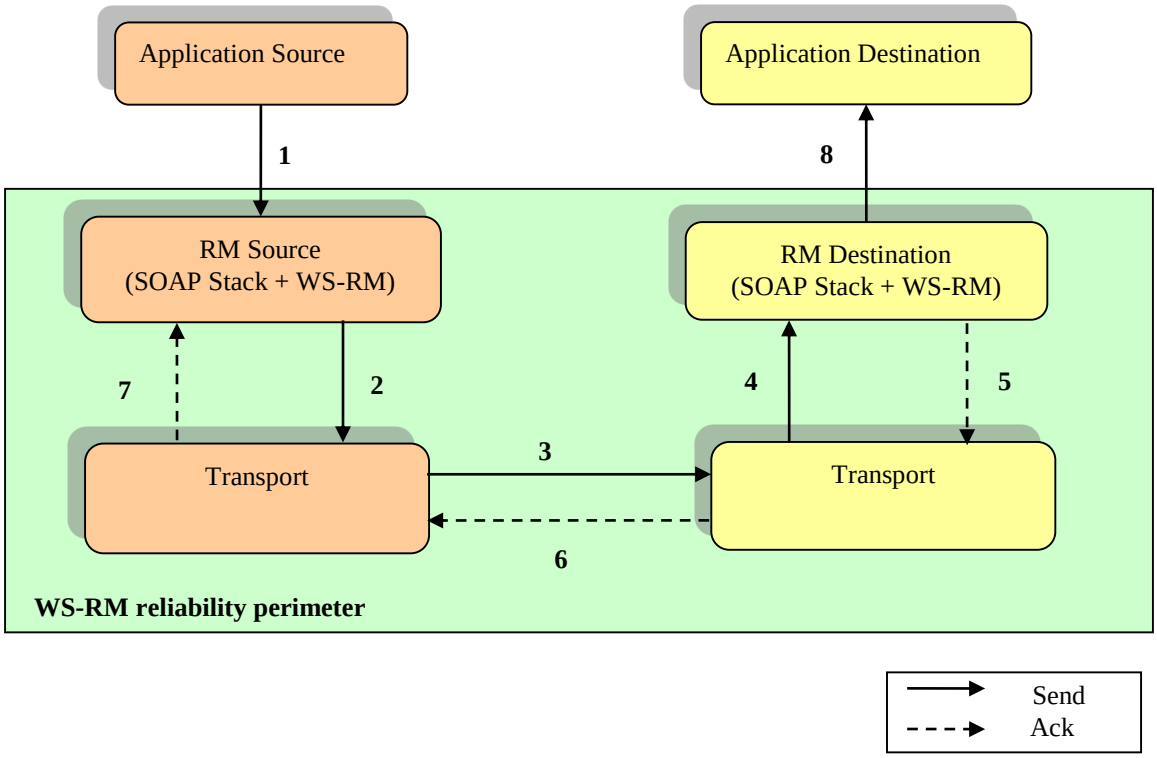


Figure 3 WS-RM messaging model

The reliability context of WS-RM is between SOAP endpoints. The SOAP stack component is either the receiving and sending component depending on the exchange pattern which can be one of one-way, two-way or duplex exchanges.

3.3.4.1 Sessions

There is no specific session concept in WS-RM and the relationship between applications and sequences or about the visibility of these sequences.

3.3.4.2 Essential Properties

The important aspects of the WS-RM protocol are;

- Allows two WS-RM enabled SOAP endpoints or systems to send messages to each other reliably.
- WS-RM remains a wire protocol with no API of its own (unlike JMS) instead it composes with existing SOAP based systems.
- Provides various delivery assurances (once only, at most once, etc) that applies to the SOAP endpoints (assumes that the link between the SOAP endpoint and the application is always reliable).

The reader is referred to the OASIS specification WS-RM for details and schemas [WS-RM].

3.3.5 WS-MakeConnection

WS-MakeConnection is a one way synchronous exchange, with the response provided in the back-

channel, meaning the response flow in HTTP. This specification is targeted at non-addressable endpoints which must explicitly obtain responses from the server.

A WS-MakeConnection message is not required to be reliable, but we only address its use in a reliable context.

The sequence of steps is:

1. The client sends a WS-MakeConnection message with the Sequence identifier used for the request(s). Note that the message does not contain the original request.

2. The server responds with a response if available in the back channel; if more messages are waiting, a *MessagePending* flag is set to true.

3. Steps 1 and 2 are repeated until no more messages are pending.

It should be noted that a WS-MakeConnection message can be sent in the context of live WS-RM Sequence or over a separate connection.

3.3.5.1 Essential Properties

The WS-MakeConnection enables a non-addressable endpoint to obtain a message from the server in a standard manner. This avoids the necessity for the client to resend (replay) the message. Correlation between request and response at the application level is not addressed by this specification.

The reader is referred to the OASIS specification WS-MakeConnection for details and schemas [WS-MakeConnection].

3.4 IATA Messaging Requirements Crosscheck

This section provides a crosscheck of WS-RX against IATA reliable messaging requirements.It should be noted that certain characteristics that are not part of WS-RM, but other composable WS specification explained above.

Requirements	WS-RM functionality	Comment
Specification shall support a range of transport protocols; HTTP, MQ, SOAP	✘	WS-RM makes use of SOAP only
Specification shall support one-way messaging, push or pull pattern	✓	
Specification shall support asynchronous request/response	✓	
Specification shall support synchronous request response	✘	
Specification shall be open and freely available	✓	
Specification shall support assured and secured delivery	✓	
Specification shall support message delivery to more than one recipient	✘	
Specification shall contain support for IATA addressing based on IATA coding principles	✘	Addressing is based on WS-Addressing using a single recipient URI
Specification may support exchanges with SMTP, X.400, URL, Fax	✘	
Specification shall provide capability to detect and remove duplicated messages	✓	
Specification shall provide capability for sequence control	✓	
Sending application shall be informed of any failure of the message to arrive at its destination	✓	.
Sender application shall be notified of message delivery or non delivery	✘	This is an explicit end to end delivery or non-delivery notification that the sender can request from the receiving application on a per message basis
Sender shall be able to define the life time for the message	✘	This is also defined as message durability
Message confirmed to be received by any node will never be lost	✓	This is not part of the WS-RM specification.
Messages shall contain a priority indicator	✘	
Specification shall support a level of backward compatibility with legacy to allow business continuity during transition	✘	

Requirements	WS-RM functionality	Comment
Specification shall provide support for message level digital signature	✓	
Specification shall provide support for message level encryption	✓	
Ability to authenticate a partner connection	✗	
Ability to authenticate the sender	✓ ✗	
Capability for session management flow control	✗	

3.5 IATA Type X Messaging Standard – SCR Volume 7

3.5.1 Introduction

This section is a guide for IATA Type X messaging standard [IATATYPEXSPEC] to enable reliable and secure exchange of business specific messages defined in various industry groups with business partners. It describes the use of IATA Type X messaging standard to provide reliable messaging for use where a reliable message exchange practice is required for appropriate business messages. Some background is provided on Type X, as well as examples demonstrating IATA Type X messaging over different message exchange patterns.

Type X is a public IATA standard for platform and transport independent XML messaging that incorporates application-to-application reliability, session management and reporting, as well as compatibility with the air transport industry addressing and business practices. The specification is aimed at main implementation types (e.g. SOA based, web services, legacy architectures) over mostly used transports or protocols (e.g. SOAP, JMS, HTTP).

3.5.1.1 Scope

The document describes the use of Type X over SOAP/HTTP and JMS. Type X can be also used over HTTP and RESTfull framework or directly over TCP. Message Level security by composing with W3C XML encryption and digital signature is also described.

3.5.1.2 Out of Scope

The document does not deal with the following aspects of reliable messaging:

- Performance benchmarks
- Transport Level security (use of SSL/TLS)
- Conversations: duplex asynchronous multiple request/response (this is equivalent to IATA Host-to-host which is also included in IATA Type X specification)

This document provides a summary description of Type X standard various sections. All details can be found in the core standard [IATATYPEXSPEC] and the Implementation guide [IATATYPEXIMPGUIDE]

The Appendix contains XML message samples and document references.

3.5.2 IATA Type X Messaging Overview

The purpose of IATA Type X Messaging is to provide an efficient protocol for reliable and secure messaging to deliver an XML payload to one or multiple applications or recipients compatible with air transport business practices and standards. The Type X protocol is decoupled from underlying transport and may be used over HTTP, JMS or binded to SOAP used as a Web service. Message exchange patterns supported by Type X are also described in this chapter.

Type X core components are:

- Type X envelope: XML container that carries the payload, the list of recipients, as well as metadata which include:
    - Payload attributes
    - Reliability level for the payload
    - Exchange pattern used (e.g. whether a response is expected)
    - Delivery report request

- Type X reliability protocol termed XATAP: optional reliability protocol that guarantees end-to-end delivery, to be used if similar capability is not provided by the underlying transport

- Type X session management protocol termed XSM : optional session management protocol to manage logical connections and traffic flow between endpoints, to be used if similar capability is not provided by the underlying transport

3.5.2.1 Interoperability

Type X specification strongly facilitates interoperability since the specification covers in detail the behavior of the Type X protocols and the participating nodes during message exchanges. In addition, Type X is not composed with other specifications eliminating interoperability issues incurred by evolution of individual composed external specifications, providing a complete and standalone set of protocols.

3.5.2.2 IATA Type X Messaging Context

The following diagram presents the Type X messaging context.

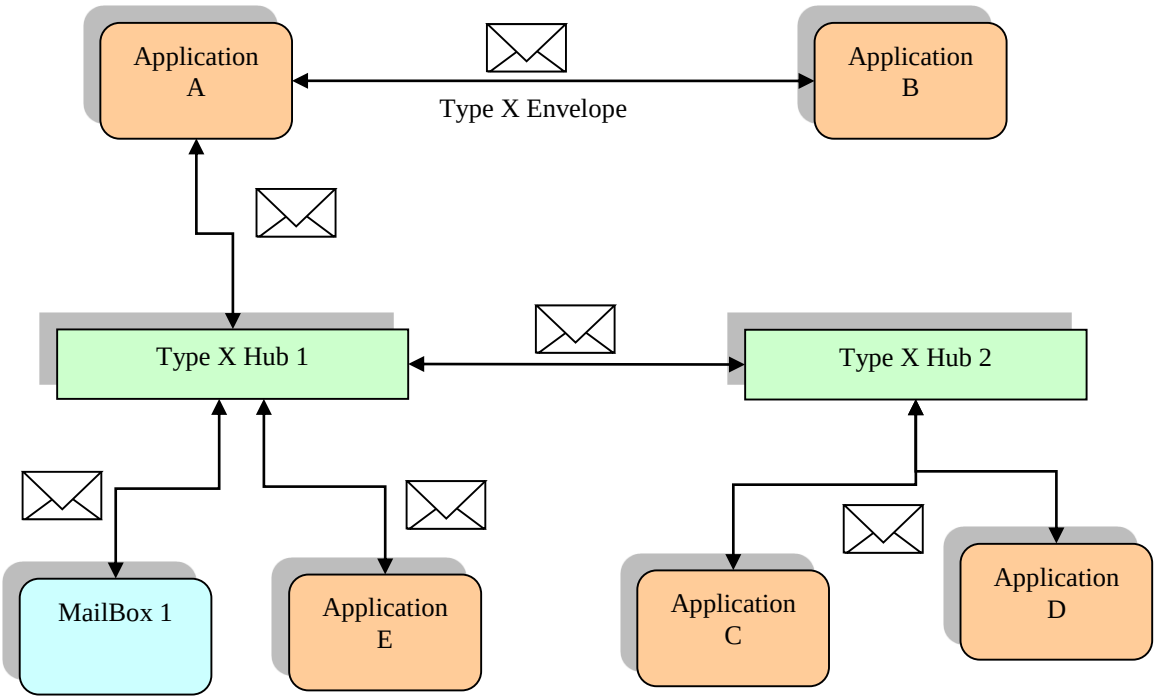


Figure 4 IATA Type X messaging context

Important Points:

- There can be one or multiple receiving applications (i.e. multicasting is supported), as shown in the diagram

- It can be used bilaterally or multilaterally, Type X standard specifies global message routing and looping detection

- Independent of transport

- Open architecture, so that Type X may be deployed anywhere from a service oriented implementation to a web service

- Bindings specified for the following transports: SOAP, HTTP, JMS and TCP.

3.5.2.3 Essential Properties

IATA Type X specification is designed as reliable and secure messaging protocol to meet the defined requirements from the outset and thus provides all core functionalities necessary for reliable and secure messaging needs; this is in contrast to the SOAP protocol.

Only essential properties of the Type X protocol that are pertinent to the IATA requirements are presented in this section. The reader is referred to the IATA Type X specification which enumerates and discusses the complete list of features [IATATYPEXSPEC].

- Public IATA standard: open availability enables broad adoption in the industry.
- Transport independence: all features in Type X are supported for all transports; currently there are specified Type X bindings for SOAP/HTTP, JMS and TCP.
- End-To-End reliability: guaranteed application to application delivery provided by the Type X extension protocol and delivery reporting.
- Session management: management of a logical connection between two applications in order to achieve flow and connection control.
- Multiple Recipients: a message can sent to one or more applications or recipients.
- Web Service use: a Type X envelope is simply carried as the payload in the SOAP envelope.
- Compatibility IATA, ATA and ICAO message communications: compatible with other related IATA and ICAO standards to facilitate interoperability during the transition.
- Support for all standard message exchange patterns
- Permits detection of duplicate messages
- Permits messaging ordering
- Grouping of message with message ordering
- End-to-End reporting (ultimate receiver or recipient to originator)
- Message Priority: high priority messages are processed first.
- Strong message level security by composing with W3C and WS-Security standards

3.5.2.4 Type X Envelope

This section presents the essential aspects of the Type X envelope. The Type X envelope consists of a number of essential components, which are specified by the source application:

- Payload: the business message to send, possibly with attachments
- Recipients: the list of targeted application destinations
- MEP: the end-to-end exchange pattern is specified by the application

- Reliability Level: this parameter indicates the criticality of the message; the default value (2) requires that the message be acknowledged and safe stored. For informational messages, the value of zero (0) requires no acknowledgement and no safe store. The intermediate value of one (1) requires an acknowledgement but no safe store. The receiving node needs to meet this policy parameter.

- Expiration Time: a parameter that defines the date and time after the message is no longer valid, and thus delivery should no longer be attempted. This parameter is important in meeting business rules.

- Priority: a parameter that defines the urgency of the message; messages with the highest priorities are processed first. This parameter is important in meeting business rules.

The Type X agent, responsible for the message exchange, optionally utilizes the following protocols:

- XATAP: the application-to-application reliability protocol.

- XSM: establishes a session for the duration of the exchange which may encompass many exchanges.

Appendix A includes an example of XML schema defining a Type X envelope carrying an IATA business data as payload targeting a single recipient with no response expected from the recipient.

3.5.2.5 Transports

Type X messaging may be implemented over several transports. A number of Type X bindings are specified for the transports SOAP/HTTP, JMS and TCP [IATATYPEXIMPGUIDE].

Type X is naturally transported over SOAP as a payload, and is compatible with any WS-* specifications.

3.5.2.6 Message Exchange Patterns

Type X supports all possible end-to-end message exchange patterns (MEP).

The Type X exchange patterns are as follows:

- **One-way messaging** with no expectation of a response.

- **Asynchronous request/response exchange**; the exchange consists of one or more request/response pairs. The correlation between requests and responses is achieved with the field *InReplyTo* and the identifiers set by the sender and the responder, each participant choosing a suitable form for its identifier.

- **Synchronous request/response exchange**. The requesting application may or may not block until a response is received. In non-blocking mode the requesting application is invoked via a callback when the response is received. The request and response may be correlated via the field *InReplyTo*. If multiple application instances are multiplexed over one or more connections, then the *SessionId* (see XSM) is also used to uniquely target a response.

3.5.2.7 Non-Addressable Sender

Type X supports the ability for a non-addressable sender to explicitly obtain messages from another TXM node. This is achieved using the TXM_AuthHeader as the sole element in a TXM message.

Non-addressable sender can be used for exchanges that are of “pull” type, whereby the sender only provides the TXM_AuthHeader to an application that will return messages specific to that sender. A typical example is invoking a “Get” service that returns a message that is targeted to the address of the sender; in the event that the sender is not authorized, the service would return a fault to the sender. The UserData field can be to refine the selection of returned message (see [IATATXPEXSPEC] for more

details). A response with an empty payload indicates that no further messages are available for that sender.

3.5.2.8 Reliability

There are three aspects of reliability addressed in Type X. This capacity makes Type X not only robust, but also flexible in the selection of the reliability level required by an application depending on the message criticality.

The three aspects addressed are:

- Point to point: addresses reliability between two adjacent Type X endpoints. The Type X extension XATAP provides this capacity. If JMS and underlying MOM is used for underlying transport with persistence and notification, then the use of XATAP is not required (JMS/MQ with persistent queues provide similar level of reliability).
- End to End: addresses reliability between two end applications. In Type X this is provided by positive and negative end to end delivery reports. Additionally use of point to point reliability between each pair of intermediate points enforces the reliability in data exchange.
- Durability: this refers to the ability to safe store messages in order to be robust in case of severe failures such as server crash. In Type X durability and reliability are prescribed by the field *ReliabilityLevel*, which should be viewed a reliability policy parameter.

Each of these aspects are discussed in more detail in the following sections.

3.5.2.9 Type X Reliability Protocol

The Type X specification defines a lightweight reliability protocol termed XATAP, type X Application-To-Application Protocol. The purpose of XATAP is to respond to the airline reliable messaging requirements that stipulate the need for application to application reliability.

The use of XATAP is optional. When Type X is used over JMS and the underlying MOM provides adequate reliability such as persistence and notification use of XATAP is not required.

This protocol defines an approach for messaging point-to-point reliability at the application level, where reliability is defined as:

1. Guaranteed delivery to the application: It is the receiving application or agent that acknowledges the reception of a message, not the messaging transport stack.
2. Identification of potential duplicate messages: The protocol notifies a node that a message maybe a duplicate. The deletion of a duplicate is left to either hub nodes or recipients, based on the *MessageId*.

The reader is referred to the IATA Type X specification for Type X reliability protocol details and schemas [IATATYPEXSPEC].

3.5.2.10 Type X Reliability Model

The XATAP reliability model is presented in the following diagram.

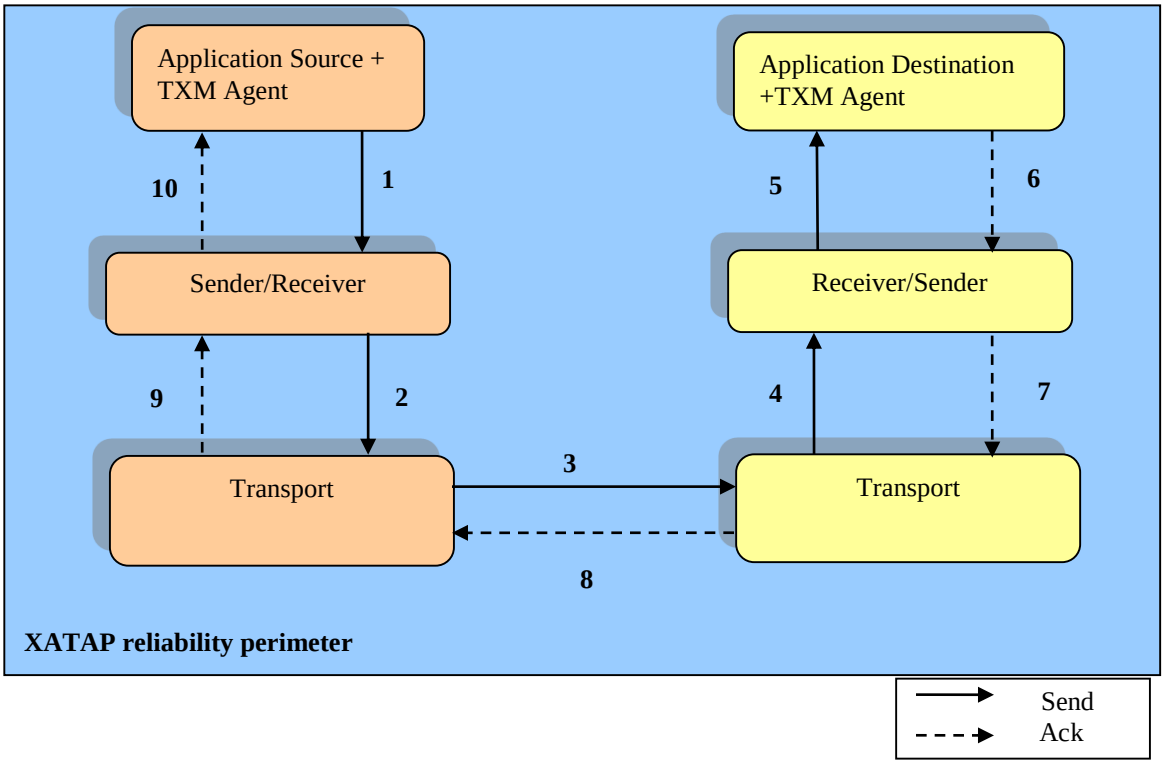


Figure 5 XATAP reliability model

As is shown, the XATAP reliability perimeter extends to the applications (strong reliability).

The Sender/Receiver component is either the receiving or sending component depending on the exchange pattern which can be one of one-way or two-way. The sequence of events in a Type X reliable exchange is as follows:

A. The application (or agent) source sends a message (1).

B. The sender (e.g. SOAP stack) sends the message (2,3).

C. Once received (4), the receiver forwards the message to the application (5).

D. The receiving application (or agent) acknowledges the message (6).

E. The acknowledgement is sent back to application (7, 8, 9 and 10).

The application may be reliably decoupled from the XATAP component (e.g. a Type X agent on a JMS bus). In this manner, legacy applications can easily integrate TXM.

The delivery reliability is achieved by the replay model. A message that is not acknowledged is simply resent with same *MessageId*, but a different XATAP identifier (*SerialNumber*).

XATAP has the advantage of being simple and robust to failures.

3.5.2.11 Delivery Reporting

The Type X specification provides for an explicit form of end-to-end reliability in the form of a delivery report from the targeted endpoint application or agent. When the optional Boolean field *DeliveryNotificationRequest* is set to *true* in the outgoing message, the endpoint receiving application or

agent must send back a delivery report. The delivery report is treated as a normal reliable message.

It should be noted that a Non-Delivery report is systematically sent to the sender if the message could not be delivered to one or more targeted recipients. This is useful when there are intermediate nodes between the participant applications.

3.5.2.12 Durability

The Type X specification defines a policy parameter termed *ReliabilityLevel* that enables to specify the persistence or durability of a message. This parameter is dynamic, being specific to each message. By default, the parameter requires the receiving node to safe store the message. The duration of storage is specified in the message information metadata (*LifeTimeDays*).

3.5.2.13 Session Management

This section presents a simple session mechanism connecting two applications or agents to authenticate and ensure that the remote application or agent is available to receive traffic reliably. The session management component is termed XSM for type X Session Management.

The use of XSM is optional. When Type X is used over JMS and the underlying MOM provides adequate connection management, the use of XSM is not required.

Depending on the communication context between two TXM Nodes, a TXM session handling mechanism may be required to ensure better traffic flow management as well as to uniquely identify a traffic flow. The explicit opening and closing of a session is also an advantage of using session management, preventing dangling connections.

The prime importance of XSM is related to permanent or long life sessions. A permanent session provides better performances and better operator visibility to control and manage the traffic flow between two TXM Nodes.

Three service messages (or commands) are used to manage the TXM Sessions:

- Open: Request to establish a new session.
- OpenConfirm: Acknowledge the request for the new session.
- Close: Terminate a session

Two additional service messages are used to maintain TXM Sessions:

- StatusQuery: this is a heartbeat message during no traffic periods to determine if the receiving application is alive.
- StatusResponse: Acknowledgement of the StatusQuery.

For one-way exchanges, only the Client initiates the XSM session and sends XSM messages. The Server Node only replies with an acknowledgement to the XSM message received from the Client.

If the flow is bidirectional, either participant may initiate the opening of XSM session. Once established, each participant may send messages.

3.5.2.14 XSM Benefits

XSM benefits include:

- Determine if the application is available: This is essential to avoid sending and resending messages in a futile fashion, with the associated resource consumption if the application or agent is not available.

- Connection management: Better visibility and management in operations; also used to control user access and message flow.

- Uniquely identify channels: Exchanges can be uniquely identified by XSM *SessionId* in addition to the pair identifiers (logical addresses) for the sender and receiver, and the response *InReplyTo* field. This also permits a given sender to have several, albeit distinct, sessions with a given receiver.

- Authenticate- avoid resource attack: The establishment of a secured XSM session, using the *TXM_AuthHeader* (see examples in the appendix), enables a receiving application to validate and authorize a sender. This also reduces the possibility of denial of service attacks.

3.6 IATA Messaging Requirements Crosscheck

This section provides a crosscheck of Type X specification against IATA reliable messaging requirements.

Requirements	Type X	Comment
Specification shall support a range of transport protocols; HTTP, MQ, SOAP	✓	Type X standard in decoupled from the underlying transport
Specification shall support one-way messaging, push or pull pattern	✓	
Specification shall support asynchronous request/response	✓	
Specification shall support synchronous request response	✓	
Specification shall be open and freely available	✓	
Specification shall support assured and secured delivery	✓	
Specification shall support message delivery to more than one recipient	✓	
Specification shall contain support for IATA addressing based on IATA coding principles	✓	Type X standard support IATA and ICAO coding principles indifferently
Specification may support exchanges with SMTP, X.400, URL, Fax	✓	
Specification shall provide capability to detect and remove duplicated messages	✓	Type X marks messages are potential duplicates; the receiving application is responsible to eliminate duplicates
Specification shall provide capability for sequence control	✓	
Sending application shall be informed of any failure of the message to arrive at its destination	✓	.
Sender application shall be notified of message delivery or non delivery	✓	This is an explicit end to end delivery or non-delivery notification that the sender can request from the receiving application on a per message basis
Sender shall be able to define the life time for the message	✓	This is also defined as message durability
Message confirmed to be received by any node will never be lost	✓	This is not part of the WS-RM specification.
Messages shall contain a priority indicator	✓	
Specification shall support a level of backward compatibility with legacy to allow business continuity during transition	✓	

Mozilla Firefox.lnk

Requirements	Type X	Comment
Specification shall provide support for message level digital signature	✓	
Specification shall provide support for message level encryption	✓	
Ability to authenticate a partner connection	✓	
Ability to authenticate the sender	✓	
Capability for session management flow control	✓	

3.7 Appendix

3.7.1 Sample Messages with WS-RM enabled

The following XML samples present IATA messages using WS-RM.

→ It is important to note if Type X is used to transport the IATA message, then the only change in the following examples would be to simply replace the IATA message by the Type X envelope carrying that same IATA message.

3.7.1.1 Synchronous Request/Response

Request:

An RM Sequence element is added in the sample SOAP message as shown in the figure below. The Sequence element represents the location of the current message in relation to the overall sequence of messages within which it is being delivered. The Identifier element contains an ID value associated with the sequence itself, while the *MessageNumber* element contains a number indicating the position of the message within the overall of sequence of messages sent.

Also note that the WS-Addressing *ReplyTo* element in the SOAP header can have an anonymous URL for WS-MakeConnection. This is added by the RM Source. If the RM source can detect that the expected sequence acknowledgement or response from the service provider is missing, it will send a MakeConnection to establish a back channel for the service provider to send a message back.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
   <wsa:ReplyTo>
    <wsa:Address>http://docs.oasis-open.org/ws- rx/wsmc/200702/anonymous?id=http://Business123.com
      /guid/6733e337c0a901036f206f2089a1870b
    </wsa:Address>
  </wsa:ReplyTo>
   <wsa:MessageID>
      http://Business456.com/guid/71e0654e-5ce8-477b-bb9d-34f05cfcbc9e
   </wsa:MessageID>
   <wsa:To>http://example.com/X_Service/123</wsa:To>
   <wsa:From>
     <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>
   </wsa:From>
   <wsa:Action>http://example.com/X_Service/123/request</wsa:Action>
   <wsrm:Sequence>
     <wsrm:Identifier>http://Business123.com/RM/ABC</wsrm:Identifier>
     <wsrm:MessageNumber>1</wsrm:MessageNumber>
   </wsrm:Sequence>
 </S:Header>
```

```
  <S:Body>
    <!--  IATA Message Payload  -->
  </S:Body>
</S:Envelope>
```

Response:
Just as in the request, the RM Sequence element is added to the SOAP header.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <S:Header>
    <wsa:RelatesTo>
          http://Business123.com/guid/71e0654e-5ce8-477b-bb9d-34f05cfcbc9e
    </wsa:RelatesTo>
    <wsa:MessageID>
       http://Business123.com/guid/71e0654e-5ce8-477b-bb9d-45o05cfkji8p
    </wsa:MessageID>
    <wsa:To> http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous </wsa:To>
    <wsa:From>
      <wsa:Address>http://example.com/X_Service/123</wsa:Address>
    </wsa:From>
```

```
    <wsa:Action>http:// air.com/X_Service/456/response</wsa:Action>
    <wsrm:Sequence>
      <wsrm:Identifier>http://Business123.com/RM/XYZ</wsrm:Identifier>
      <wsrm:MessageNumber>1</wsrm:MessageNumber>
    </wsrm:Sequence>
    <wsrm:SequenceAcknowledgement>
      <wsrm:Identifier>http://Business456.com/RM/ABC</wsrm:Identifier>
      <wsrm:AcknowledgementRange Upper="1" Lower="1"/>
    </wsrm:SequenceAcknowledgement>
  </S:Header>
  <S:Body>
    <!--  IATA Message Payload  -->
  </S:Body>
</S:Envelope>
```

3.7.1.2 Asynchronous Request/Response

Request:

```

```

Response:

```

```

3.7.2 Sample Messages with Type X enabled

The following XML samples define Type X envelopes targeted to a single recipient using different message exchange patterns.

The *ReliabilityLevel* is the default value (2) which implies that the message must be acknowledged and safe stored.

The agent responsible for sending the message will add the following components in the *TXM_Header*:

- a *TXM_Header/TXM_AuthHeader* which never changes for a given sender,
- a *TXM_BodyHeader/TXM_XATAPHeader* **if** required by the *ReliabilityLevel.*

In a SOAP context, The Type X envelope is sent in the SOAP envelope's body. Bindings are also defined when HTTP or MQ is used as the underlying transport. The defined decoupling enables use of Type X as a web service (SendMessage, GetMessage) with REST framework or use of SOAP, or with JMS/MQ.

3.7.2.1 Fire and Forget Exchange

In this section a simple one-way Type X envelope is presented; no response is expected.

The MEP is implicitly Fire-and-Forget; the tag <FireAndForget> could explicitly be included.

```
<?xml version="1.0" encoding="UTF-8"?>
<txme:TXM_Envelope xmlns:txme="http://www.iata.org/txm/envelope"
TxmRelease="TXM2009A0">
        <txme:TXM_Header>
                <txme:TXM_BodyHeader>
            <txmm:TXM_MessageHeader xmlns:txmm="http://www.iata.org/txm/msgheader"
            TxmRelease="TXM2009A0">
                                <Information>
                                        <MessageId>
                                                <TypeXAddress>
                                                        <TYPEX_Address>
                                                                <Airline>TST</Airline>
                                                                <City>CLT</City>

        <Department>TXM</Department>
                                                        </TYPEX_Address>
                                                </TypeXAddress>
                                                <OriginDate>2008-07-24+02:00</OriginDate>

        <OriginTime>20:40:02.890+02:00</OriginTime>
                                                <Number>1</Number>
                                        </MessageId>
                    <MessageReference>My ref</MessageReference>
                                </Information>
                                <Originator>
                                        <OriginatorAddress>
                                                <TYPEX_Address>
                                                        <Airline>TST</Airline>
                                                        <City>CLT</City>
                                                        <Department>TXM</Department>
                                                </TYPEX_Address>
                                        </OriginatorAddress>
                                </Originator>
                                <Destination>
                                        <RecipientInformation>

        <ResponsibilityFlag>Yes</ResponsibilityFlag>
                                                <ActionType>TO</ActionType>
                                                <DestinationAddress>
                                                        <TYPEX_Address>
                                                                <Airline>1G</Airline>
                                                                <City>NYC</City>
                                                                <Department>AA</Department>
                                                        </TYPEX_Address>
                                                </DestinationAddress>
                                        </RecipientInformation>
                                        <NodeTrace/>
                                </Destination>
                        </txmm:TXM_MessageHeader>
                </txme:TXM_BodyHeader>
        </txme:TXM_Header>
        <txme:TXM_Body>
                <txme:TXM_Payload>
```

Message Id

Sender

Recipient

Payload

```
                    <!-- IATA Message  -->
                </txme:TXM_Payload>
        </txme:TXM_Body>
</txme:TXM_Envelope>
```

3.7.2.2 Synchronous Exchange

In this section a synchronous request/response Type X envelope is presented. The request and response envelopes are presented separately.

Request :

```
<?xml version="1.0" encoding="UTF-8"?>
<txme:TXM_Envelope xmlns:txme="http://www.iata.org/txm/envelope"
TxmRelease="TXM2009A0">
        <txme:TXM_Header>
                <txme:TXM_BodyHeader>
            <txmm:TXM_MessageHeader xmlns:txmm="htp://www.iata.org/txm/msgheader"
            TxmRelease="TXM2009A0">
                            <Information>
                                    <MessageId>
                                            <TypeXAddress>
                                                    <TYPEX_Address>
                                                            <Airline>TST</Airline>
                                                            <City>CLT</City>

        <Department>TXM</Department>
                                                    </TYPEX_Address>
                                            </TypeXAddress>
                                            <OriginDate>2008-07-24+02:00</OriginDate>

        <OriginTime>20:40:02.890+02:00</OriginTime>
                                            <Number>1</Number>
                                    </MessageId>
                    <MessageReference>My ref</MessageReference>
                            </Information>
                            <Originator>
                                    <OriginatorAddress>
                                            <TYPEX_Address>
                                                    <Airline>TST</Airline>
                                                    <City>CLT</City>
                                                    <Department>TXM</Department>
                                            </TYPEX_Address>
                                    </OriginatorAddress>
                            </Originator>
                            <Destination>
                                    <RecipientInformation>

        <ResponsibilityFlag>Yes</ResponsibilityFlag>
                                            <ActionType>TO</ActionType>
                                            <DestinationAddress>
                                                    <TYPEX_Address>
                                                            <Airline>1G</Airline>
                                                            <City>NYC</City>
```

Message Id

Sender

Recipient

```
                                                    <Department>AA</Department>
                                                </TYPEX_Address>
                                            </DestinationAddress>
                                        </RecipientInformation>
                                        <NodeTrace/>
                                    </Destination>
                                    <ExchangePattern>
                        <SendReceive>0</SendReceive>
                                    </ExchangePattern>
                                </txmm:TXM_MessageHeader>
                        </txme:TXM_BodyHeader>
                </txme:TXM_Header>
                <txme:TXM_Body>
                        <txme:TXM Payload>
                                <!-- IATA Message -->
                        </txme:TXM_Payload>
                </txme:TXM_Body>
</txme:TXM_Envelope>
```

MEP

Payload

Response :

```
<?xml version="1.0" encoding="UTF-8"?>
<txme:TXM_Envelope xmlns:txme="http://www.iata.org/txm/envelope"
TxmRelease="TXM2009A0">
        <txme:TXM_Header>
                <txme:TXM_BodyHeader>
            <txmm:TXM MessageHeader xmlns:txmm="http://www.iata.org/txm/msgheader"
            TxmRelease="TXM2009A0">
                                <Information>
                                        <MessageId>
                                                <TypeXAddress>
                                                        <TYPEX_Address>
                                                                <Airline>1G</Airline>
                                                                <City>NYC</City>
                                                                <Department>AA</Departme
                                                        </TYPEX_Address>
                                                </TypeXAddress>
                                                <OriginDate>2008-07-24+02:00</OriginDate>

        <OriginTime>20:40:49.890+02:00</OriginTime>
                                                <Number>1</Number>
                                        </MessageId>
                                        <InReplyTo>My ref<InReplyTo>
                                </Information>
                                <Originator>
                                        <OriginatorAddress>
                                                <TYPEX_Address>
                                                                <Airline>1G</Airline>
                                                                <City>NYC</City>
                                                                <Department>AA</Department>
                                                </TYPEX Address>
```

Message Id

Reply Ref

Sender

```
                                        </OriginatorAddress>
                                </Originator>
                                <Destination>
                                        <RecipientInformation>

        <ResponsibilityFlag>Yes</ResponsibilityFlag>
                                                <ActionType>TO</ActionType>
                                                <DestinationAddress>
                                                        <TYPEX_Address>
                                                          <Airline>TST</Airline>
                                                          <City>CLT</City>
                                                          <Department>TXM</Department>
                                                        </TYPEX_Address>
                                                </DestinationAddress>
                                        </RecipientInformation>
                                        <NodeTrace/>
                                </Destination>
                        </txmm:TXM_MessageHeader>
                </txme:TXM_BodyHeader>
        </txme:TXM_Header>
        <txme:TXM_Body>
                <txme:TXM_Payload>
                        <!-- IATA Message -->
                </txme:TXM_Payload>
        </txme:TXM_Body>
</txme:TXM_Envelope>
```

Recipient

Payload

3.7.2.3 Asynchronous Exchange

Request :

```
<?xml version="1.0" encoding="UTF-8"?>
<txme:TXM_Envelope xmlns:txme="http://www.iata.org/txm/envelope"
TxmRelease="TXM2009A0">
        <txme:TXM_Header>
                <txme:TXM_BodyHeader>
            <txmm:TXM_MessageHeader xmlns:txmm="htp://www.iata.org/txm/msgheader"
            TxmRelease="TXM2009A0">
                                <Information>
                                        <MessageId>
                                                <TypeXAddress>
                                                        <TYPEX_Address>
                                                                <Airline>TST</Airline>
                                                                <City>CLT</City>

        <Department>TXM</Department>
                                                        </TYPEX_Address>
                                                </TypeXAddress>
                                                <OriginDate>2008-07-24+02:00</OriginDate>

        <OriginTime>20:40:02.890+02:00</OriginTime>
                                                <Number>1</Number>
```

Message Id

```
                                        </MessageId>
                        <MessageReference>My ref</MessageReference>
                                </Information>
                                <Originator>
                                        <OriginatorAddress>
                                                <TYPEX_Address>
                                                        <Airline>TST</Airline>
                                                        <City>CLT</City>
                                                        <Department>TXM</Department>
                                                </TYPEX_Address>
                                        </OriginatorAddress>
                                </Originator>
                                <Destination>
                                        <RecipientInformation>

        <ResponsibilityFlag>Yes</ResponsibilityFlag>
                                                <ActionType>TO</ActionType>
                                                <DestinationAddress>
                                                        <TYPEX_Address>
                                                                <Airline>1G</Airline>
                                                                <City>NYC</City>
                                                                <Department>AA</Department>
                                                        </TYPEX_Address>
                                                </DestinationAddress>
                                        </RecipientInformation>
                                        <NodeTrace/>
                                </Destination>
                                <ExchangePattern>
                            <SendCallback>
                                                <ConversationId>
                                                        <InitiatorId>123</InitiatorId>
                                                        <RecipientId>TXM
_VOID</RecipientId>
                                                </ConversationId>
                                                <Phase>continue</Phase>
                                        </SendCallback>
                                </ExchangePattern>
                            </txmm:TXM_MessageHeader>
                    </txme:TXM_BodyHeader>
        </txme:TXM_Header>
        <txme:TXM_Body>
                <txme:TXM_Payload>
                        <!-- IATA Message -->
                </txme:TXM_Payload>
        </txme:TXM_Body>
</txme:TXM_Envelope>
```

Sender

Recipient

MEP

Payload

Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<txme:TXM Envelope xmlns:txme="http://www.iata.org/txm/envelope"
TxmRelease="TXM2009A0">
        <txme:TXM Header>
                <txme:TXM BodyHeader>
            <txmm:TXM MessageHeader xmlns:txmm="htp://www.iata.org/txm/msgheader"
            TxmRelease="TXM2009A0">
                                <Information>
                                        <MessageId>
                                                <TypeXAddress>
                                                        <TYPEX Address>
                                                                <Airline>1G</Airline>
                                                                <City>NYC</City>
                                                                <Department>AA</Departme
                                                        </TYPEX Address>
                                                </TypeXAddress>
                                                <OriginDate>2008-07-24+02:00</OriginDate>

        <OriginTime>20:40:49.890+02:00</OriginTime>
                                                <Number>1</Number>
                                        </MessageId>
                                        <InReplyTo>My ref<InReplyTo>
                                </Information>
                                <Originator>
                                        <OriginatorAddress>
                                                <TYPEX Address>
                                                                <Airline>1G</Airline>
                                                                <City>NYC</City>
                                                                <Department>AA</Department>
                                                </TYPEX_Address>
                                        </OriginatorAddress>
                                </Originator>
                                <Destination>
                                        <RecipientInformation>

        <ResponsibilityFlag>Yes</ResponsibilityFlag>
                                                <ActionType>TO</ActionType>
                                                <DestinationAddress>
                                                        <TYPEX_Address>
                                                          <Airline>TST</Airline>
                                                          <City>CLT</City>
                                                          <Department>TXM</Department>
                                                        </TYPEX_Address>
                                                </DestinationAddress>
                                        </RecipientInformation>
                                        <NodeTrace/>
                                </Destination>
                                <ExchangePattern>
                                        <ReceiveCallback>
                                                <ConversationId>
                                                        <InitiatorId>myrowid</InitiatorId>
```

Message Id

Reply Ref

Sender

Recipient

MEP

```
                                                    <RecipientId>123</RecipientId>
                                                </ConversationId>
                                                <Phase>continue</Phase>
                                            </ReceiveCallback>
                    </ExchangePattern>
                            </txmm:TXM_MessageHeader>
                    </txme:TXM_BodyHeader>
            </txme:TXM_Header>
            <txme:TXM_Body>
                    <txme:TXM_Payload>
                            <!--  IATA Message   -->
                    </txme:TXM_Payload>
            </txme:TXM_Body>
</txme:TXM Envelope>
```

Payload

3.7.3 References

NB: These references were current at the time of writing, but may have been superseded by more recent versions.

[TYPEXIMPGUIDE] *TypeX Implementation Guide*, final draft Dec 31 2008, IATA document. http://www.iata.org/padis
[TYPEXSPEC] *TypeX Specification*, version 1.0, expected august 2009, IATA SCR volume 7. http://www.iata.org/padis
[WSMC] *Web Service Make Connection*, version 1.1, 2 Feb 2009, http://docs.oasis-open.org/ws-rx/wsmc/200702/wsmc-1.1-spec-os.pdf
[WSRM] *Web Service Reliable Messaging*, version 1.2, 2 Feb 2009, http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.pdf
[WSRMP] *Web Service Reliable Messaging Policy Assertion*, version 1.2, 29 Nov 2008, http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.2-spec-cs-02.pdf
[JMS] *Java Message Service*, version 1.1, http://java.sun.com/products/jms/docs.html

3.7.4 Definitions

A **Web service** is a method of communication between two electronic devices over the Web (Internet). (source Wikipedia)

"Web service" as "a software system designed to support interoperable machine-to-machine interaction over a network" (source W3C)

"We can identify two major classes of Web services, REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and arbitrary Web services, in which the service may expose an arbitrary set of operations." (source W3C)

Web API is a development in Web services (in a movement called Web 2.0) where emphasis has been moving away from SOAP based services towards representational state transfer (REST) based communications.[3] REST services do not require XML, SOAP, or WSDL service-API definitions. (source W3C)