Identity Metasystem Interoperability Version 1.0

Committee Draft 02

19 February 2009

Specification URIs:

This Version:

http://docs.oasis-open.org/imi/identity/v1.0/cd/identity-1.0-spec-cd-02.html
http://docs.oasis-open.org/imi/identity/v1.0/cd/identity-1.0-spec-cd-02.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/cd/identity-1.0-spec-cd-02.pdf

Previous Version:

http://docs.oasis-open.org/imi/identity/v1.0/cd/identity-1.0-spec-cd-01.html
http://docs.oasis-open.org/imi/identity/v1.0/cd/identity-1.0-spec-cd-01.doc (Authoritative)
http://docs.oasis-open.org/imi/identity/v1.0/cd/identity-1.0-spec-cd-01.pdf

Latest Version:

http://docs.oasis-open.org/imi/identity/v1.0/identity.html
http://docs.oasis-open.org/imi/identity/v1.0/identity.doc
http://docs.oasis-open.org/imi/identity/v1.0/identity.pdf

Technical Committee:

OASIS Identity Metasystem Interoperability (IMI) TC

Chair(s):

Marc Goodner
Anthony Nadalin

Editor(s):

Michael B. Jones
Michael McIntosh

Related work:

This specification replaces or supersedes:

- None

This specification is related to:

- WS-Trust
- WS-SecurityPolicy
- WS-Addressing

Declared XML Namespace(s):

http://docs.oasis-open.org/imi/ns/identity-200810
http://schemas.xmlsoap.org/ws/2005/05/identity
http://schemas.xmlsoap.org/ws/2006/02/addressingidentity
http://schemas.xmlsoap.org/ws/2007/01/identity

Abstract:

This document is intended for developers and architects who wish to design identity systems and applications that interoperate using the Identity Metasystem Interoperability specification.

An Identity Selector and the associated identity system components allow users to manage their Digital Identities from different Identity Providers, and employ them in various contexts to access online services. In this specification, identities are represented to users as “Information Cards”. Information Cards can be used both at applications hosted on Web sites accessed through Web browsers and rich client applications directly employing Web services.

This specification also provides a related mechanism to describe security-verifiable identity for endpoints by leveraging extensibility of the WS-Addressing specification. This is achieved via XML [XML 1.0] elements for identity provided as part of WS-Addressing Endpoint References. This mechanism enables messaging systems to support multiple trust models across networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

Status:

This document was last revised or approved by the Identity Metasystem Interoperability TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/imi/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/imi/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/imi/.

Notices

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

1 Introduction

The Identity Metasystem Interoperability specification prescribes a subset of the mechanisms defined in [WS-Trust 1.2], [WS-Trust 1.3], [WS-SecurityPolicy 1.1], [WS-SecurityPolicy 1.2], and [WS-MetadataExchange] to facilitate the integration of Digital Identity into an interoperable token issuance and consumption framework using the Information Card Model. It documents the Web interfaces utilized by browsers and Web applications that utilize the Information Card Model. Finally, it extends WS-Addressing's endpoint reference by providing identity information about the endpoint that can be verified through a variety of security means, such as https or the wealth of WS-Security specifications.

This profile constrains the schema elements/extensions used by the Information Card Model, and behaviors for conforming Relying Parties, Identity Providers, and Identity Selectors.

1.1 Notational Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC 2119].

This specification uses the following syntax to define outlines for assertions:

- The syntax appears as an XML instance, but values in italics indicate data types instead of literal values.
- Characters are appended to elements and attributes to indicate cardinality:
  - "?" (0 or 1)
  - "*" (0 or more)
  - "+" (1 or more)
- The character "|" is used to indicate a choice between alternatives.
- The characters "(" and ")" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.
- The characters "[" and "]" are used to call out references and property names.
- Ellipses (i.e., "...") indicate points of extensibility. Additional children and/or attributes MAY be added at the indicated extension points but MUST NOT contradict the semantics of the parent and/or owner, respectively. By default, if a receiver does not recognize an extension, the receiver SHOULD ignore the extension; exceptions to this processing rule, if any, are clearly indicated below.
- XML namespace prefixes (see Table 2) are used to indicate the namespace of the element being defined.

Elements and Attributes defined by this specification are referred to in the text of this document using XPath 1.0 expressions. Extensibility points are referred to using an extended version of this syntax:

- An element extensibility point is referred to using {any} in place of the element name. This indicates that any element name can be used, from any namespace other than the namespace of this specification.
- An attribute extensibility point is referred to using @{any} in place of the attribute name. This indicates that any attribute name can be used, from any namespace other than the namespace of this specification.

Extensibility points in the exemplar may not be described in the corresponding text.

1.2 Namespaces

Table 1 lists the XML namespaces that are used in this document.

Prefix	XML Namespace	Specification(s)
ds	http://www.w3.org/2000/09/xmldsig#	XML Digital Signatures
ic	http://schemas.xmlsoap.org/ws/2005/05/identity	This document
ic07	http://schemas.xmlsoap.org/ws/2007/01/identity	Namespace for additional elements also defined by this document
ic08	http://docs.oasis-open.org/imi/ns/identity-200810	Namespace for new elements defined by this document
S	*May refer to either* http://schemas.xmlsoap.org/soap/envelope *or* http://www.w3.org/2003/05/soap-envelope *since both may be used*	SOAP
S11	http://schemas.xmlsoap.org/soap/envelope	SOAP 1.1 [SOAP 1.1]
S12	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2 [SOAP 1.2]
saml	urn:oasis:names:tc:SAML:1.0:assertion	SAML 1.0
sp	*May refer to either* http://schemas.xmlsoap.org/ws/2005/07/securitypolicy *or* http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 *since both may be used*	WS-SecurityPolicy
sp11	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy	WS-SecurityPolicy 1.1 [WS-SecurityPolicy 1.1]
sp12	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702	WS-SecurityPolicy 1.2 [WS-SecurityPolicy 1.2]
wsa	http://www.w3.org/2005/08/addressing	WS-Addressing [WS-Addressing]
wsdl	*May refer to either* http://schemas.xmlsoap.org/wsdl/ *or* http://www.w3.org/TR/wsdl20 *since both may be used*	Web Services Description Language
wsdl11	http://schemas.xmlsoap.org/wsdl/	Web Services Description Language [WSDL 1.1]
wsdl20	http://www.w3.org/TR/wsdl20	Web Services Description Language [WSDL 2.0]
wsid	http://schemas.xmlsoap.org/ws/2006/02/addressingidentity	Identity Extension for Web Services Addressing also defined by this document
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	WS-Policy [WS-Policy]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	WS-Security Extensions [WS-Security]
wst	*May refer to either* http://schemas.xmlsoap.org/ws/2005/02/trust	WS-Trust

	or http://docs.oasis-open.org/ws-sx/ws-trust/200512 *since both may be used*	
wst12	http://schemas.xmlsoap.org/ws/2005/02/trust	WS-Trust 1.2 [WS-Trust 1.2]
wst13	http://docs.oasis-open.org/ws-sx/ws-trust/200512	WS-Trust 1.3 [WS-Trust 1.3]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	WS-SecurityUtility
wsx	http://schemas.xmlsoap.org/ws/2004/09/mex	WS-MetadataExchange [WS-MetadataExchange]
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]

It should be noted that the versions identified in the above table supersede versions identified in referenced specifications.

1.3 Schema

A copy of the XML Schemas for this document can be found at:

```
http://docs.oasis-open.org/imi/identity/200810/identity.xsd
http://docs.oasis-open.org/imi/identity/200810/addr-identity.xsd
http://docs.oasis-open.org/imi/identity/200810/claims.xsd
```

1.4 Terminology

The following definitions establish the terminology and usage in this document.

Information Card Model – The "*Information Card Model*" refers to the use of Information Cards containing metadata for obtaining Digital Identity claims from Identity Providers and then conveying them to Relying Parties under user control.

Information Card – An *Information Card* provides a visual representation of a Digital Identity for the end user. Information Cards contain a reference to an IP/STS that issues Security Tokens containing the Claims for that Digital Identity.

Digital Identity – A "*Digital Identity*" is a set of Claims made by one party about another party.

Claim – A "*Claim*" is a piece of information about a Subject that an Identity Provider asserts about that Subject.

Subject – A "*Subject*" is an individual or entity about whom claims are made by an Identity Provider.

Service Requester – The term "*Service Requester*" means software acting on behalf of a party who wants to obtain a service through a digital network.

Relying Party – The term "*Relying Party*" (RP) means a network entity providing the desired service, and relying upon Digital Identity.

Identity Provider – The term "*Identity Provider*" (IP) means a network entity providing the Digital Identity claims used by a Relying Party.

Security Token Service – The term "*Security Token Service*" (STS) refers to a WS-Trust endpoint.

Identity Provider Security Token Service – The term "Identity Provider Security Token Service" (IP/STS) refers to the Security Token Service run by an Identity Provider to issue tokens.

Relying Party Security Token Service – The term "Relying Party Security Token Service" (RP/STS) refers to a Security Token Service run by a Relying Party to accept and issue tokens.

Identity Selector – The term "*Identity Selector*" (IS) refers to a software component available to the Service Requester through which the user controls and dispatches her Digital Identities.

Trust Identity – A *trust identity* is a verifiable claim about a principal (e.g. name, identity, key, group, privilege, capability, etc).

Security Token – A *security token* represents a collection of claims.

Signed Security Token – A *signed security token* is a security token that is asserted and cryptographically endorsed by a specific authority (e.g. an X.509 certificate, a Kerberos ticket, or a self-issued Information Card).

Unsigned Security Token – An un*signed security token* is a security token that is not cryptographically endorsed by a specific authority (e.g. a security token backed by shared secrets such as usernames and passwords).

Proof-of-Possession – The *proof-of-possession* information is data that is used in a proof process to demonstrate the sender's knowledge of information that SHOULD only be known to the claiming sender of a security token.

Integrity – *Integrity* is the process by which it is guaranteed that information is not modified in transit.

Confidentiality – *Confidentiality* is the process by which data is protected such that only authorized actors or security token owners can view the data

Digest – A *digest* is a cryptographic checksum of an octet stream.

Signature - A *signature* is a cryptographic binding of a proof-of-possession and a digest. This covers both symmetric key-based and public key-based signatures. Consequently, non-repudiation is not always achieved.

1.5 Normative References

[DOM]

"Document Object Model (DOM)", November 2000. http://www.w3.org/DOM/

[EV Cert]

CA / Browser Forum, "Guidelines for the Issuance and Management of Extended Validation Certificates, Version 1.1", April 2008. http://cabforum.org/EV_Certificate_Guidelines_V11.pdf

[HTTP]

R. Fielding et al., "IETF RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", June 1999. http://www.ietf.org/rfc/rfc2616.txt

[HTTPS]

E. Rescorla, "RFC 2818: HTTP over TLS", May 2000. http://www.ietf.org/rfc/rfc2818.txt

[RFC 1274]

P. Barker and S. Kille, "RFC 1274: The COSINE and Internet X.500 Schema", November 1991. http://www.ietf.org/rfc/rfc1274.txt

[RFC 2119]

S. Bradner, "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", March 1997. http://www.ietf.org/rfc/rfc2119.txt

[RFC 2256]

M. Wahl, "RFC 2256: A Summary of the X.500(96) User Schema for use with LDAPv3", December 1997. http://www.ietf.org/rfc/rfc2256.txt

[RFC 2459]

R. Housley, W. Ford, W. Polk, and D. Solo, “RFC 2459: Internet X.509 Public Key Infrastructure - Certificate and CRL Profile”, January 1999. http://www.ietf.org/rfc/rfc2459.txt

[RFC 2898]

B. Kaliski, “PKCS #5: Password-Based Cryptography Specification, Version 2.0”, September 2000. http://www.ietf.org/rfc/rfc2898.txt

[RFC 3066]

H. Alvestrand, “Tags for the Identification of Languages”, January 2001. http://www.faqs.org/rfcs/rfc3066.html

[SOAP 1.1]

W3C Note, "SOAP: Simple Object Access Protocol 1.1," 08 May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SOAP 1.2]

M. Gudgin, et al., “SOAP Version 1.2 Part 1: Messaging Framework”, June 2003. http://www.w3.org/TR/soap12-part1/

[URI]

T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998. http://www.ietf.org/rfc/rfc2396.txt

[WS-Addressing]

W3C Recommendation, “Web Service Addressing (WS-Addressing)”, 9 May 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WS-MetadataExchange]

“Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1”, August 2006. http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

[WSDL 1.1]

W3C Note, "Web Services Description Language (WSDL 1.1)," 15 March 2001. http://www.w3.org/TR/wsdl

[WSDL 2.0]

“Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language”, June 2007. http://www.w3.org/TR/wsdl20

[WS-Policy]

“Web Services Policy Framework (WS-Policy), Version 1.2”, March 2006. http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf

[WS-Security]

A. Nadalin et al., “Web Services Security: SOAP Message Security 1.0”, May 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[WS-SecurityPolicy 1.1]

“Web Services Security Policy Language (WS-SecurityPolicy), Version 1.1”, July 2005. http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf

[WS-SecurityPolicy 1.2]

OASIS, “WS-SecurityPolicy 1.2”, July 2007. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

[WS-Trust 1.2]

“Web Services Trust Language (WS-Trust)”, February 2005. http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

[WS-Trust 1.3]

OASIS, “WS-Trust 1.3”, March 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf

[XML 1.0]

W3C Recommendation, “Extensible Markup Language (XML) 1.0 (Fourth Edition)”, September 2006. http://www.w3.org/TR/xml/

[XMLDSIG]

Eastlake III, D., Reagle, J., and Solo, D., “XML-Signature Syntax and Processing”, March 2002. http://www.ietf.org/rfc/rfc3275.txt

[XMLENC]

Imamura, T., Dillaway, B., and Simon, E., “XML Encryption Syntax and Processing”, August 2002. http://www.w3.org/TR/xmlenc-core/

[XML Schema, Part 1]

H. Thompson et al., “XML Schema Part 1: Structures”, May 2001. http://www.w3.org/TR/xmlschema-1/

[XML Schema, Part 2]

P. Biron et al., “XML Schema Part 2: Datatypes”, May 2001. http://www.w3.org/TR/xmlschema-2/

Non-Normative References

[Addressing-Ext]

J. Alexander et al., “Application Note: Web Services Addressing Endpoint References and Identity”, July 2008. http://schemas.xmlsoap.org/ws/2006/02/addressingidentity

[ISIP]

A. Nanda and M. Jones, “Identity Selector Interoperability Profile V1.5”, July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Guide]

Microsoft Corporation and Ping Identity Corporation, “An Implementer’s Guide to the Identity Selector Interoperability Profile V1.5”, July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

[ISIP Web Guide]

M. Jones, “A Guide to Using the Identity Selector Interoperability Profile V1.5 within Web Applications and Browsers”, July 2008. http://www.microsoft.com/downloads/details.aspx?FamilyID=b94817fc-3991-4dd0-8e85-b73e626f6764&DisplayLang=en

2 Relying Party Interactions

This section defines the constructs used by a Relying Party Web service for specifying and conveying its Security Token requirements to the Service Requester.

2.1 Expressing Token Requirements of Relying Party

A Relying Party specifies its Security Token requirements as part of its Security Policy using the primitives and assertions defined in WS-SecurityPolicy. The primary construct in the Security Policy of the Relying Party used to specify its requirement for a Security Token from an Identity Provider is the `sp:IssuedToken` policy assertion. The basic form of the issued token policy assertion as defined in WS-SecurityPolicy is as follows.

```
<sp:IssuedToken sp:Usage="xs:anyURI" sp:IncludeToken="xs:anyURI" ...>
  <sp:Issuer>
    wsa:EndpointReference | xs:any
  </sp:Issuer>
  <sp:RequestSecurityTokenTemplate>
    ...
  </sp:RequestSecurityTokenTemplate>
  <wsp:Policy>
    ...
  </wsp:Policy>
  ...
</sp:IssuedToken>
```

The attributes and elements listed in the schema fragment above are described in WS-SecurityPolicy.

The ensuing subsections describe special parameters added by this profile as extensions to the `sp:IssuedToken` policy assertion that convey additional instructions to the Identity Selector available to the Service Requester.

2.1.1 Issuer of Tokens

The `sp:IssuedToken/sp:Issuer` element in an issued token policy specifies the issuer for the required token. More specifically, it should contain the endpoint reference of an Identity Provider STS that can issue the required token.

A Relying Party MUST specify the issuer for a required token in one of the following ways:

- Indicate a *specific* issuer by specifying the issuer’s endpoint as the value of the `sp:Issuer/wsa:Address` element.
- Indicate that the issuer is *unspecified* by omitting the `sp:Issuer` element, which means that the Service Requester should determine the appropriate issuer for the required token with help from the user if necessary.

When requiring a specific issuer, a Relying Party MAY specify that it will accept self-issued Security Tokens by using the special URI below as the value of the `wsa:Address` element within the endpoint reference for the issuer.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

Following is an example of using this URI within an issued token policy.

Example:

```
<sp:IssuedToken ...>
  <sp:Issuer>
    <wsa:Address>
      http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
    </wsa:Address>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

A Relying Party MAY specify the value of the `sp:Issuer/wsa:Address` element in policy as a “logical name” of the token issuer instead of an actual network address where the token is issued. An Identity Selector SHOULD resolve the logical name to an appropriate endpoint for the token issuer by matching the issuer name in Information Cards available to it.

If a Relying Party specifies the token issuer as a network endpoint in policy, then it MUST also specify the location of issuer metadata from where the issuer’s policy metadata can be obtained. This is done using the mechanism defined in [WS-Addressing] for embedding metadata within an endpoint reference. The following example shows a token policy where the issuer endpoint and its corresponding metadata location are specified.

Example:

```
<sp:IssuedToken ...>
  <sp:Issuer>
    <wsa:Address>http://contoso.com/sts</wsa:Address>
    <wsa:Metadata>
      <wsx:Metadata>
        <wsx:MetadataSection
            Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
          <wsx:MetadataReference>
            <wsa:Address>https://contoso.com/sts/mex</wsa:Address>
          </wsx:MetadataReference>
        </wsx:MetadataSection>
      </wsx:Metadata>
    </wsa:Metadata>
  </sp:Issuer>
  ...
</sp:IssuedToken>
```

2.1.2 Type of Proof Key in Issued Tokens

An Identity Selector SHOULD request an asymmetric key token from the Identity Provider to maximize user privacy and security if no explicit key type is specified by the Relying Party.

A Relying Party MAY explicitly request the use of an *asymmetric* or *symmetric* key in the required token by using the `wst:KeyType` element within its issued token policy assertion. The key type URIs are defined in [WS-Trust]. The following example illustrates the use of this element in the Relying Party’s Security Policy to request a symmetric key in the issued token.

Example:

```
<sp:IssuedToken>
  <sp:RequestSecurityTokenTemplate>
    <wst:KeyType>
      http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
    </wst:KeyType>
  </sp:RequestSecurityTokenTemplate>
</sp:IssuedToken>
```

2.1.3 Claims in Issued Tokens

The claims requirement of a Relying Party can be expressed in its token policy by using the optional `wst:Claims` parameter defined in [WS-Trust 1.2] and [WS-Trust 1.3]. However, the `wst:Claims` parameter has an open content model. This profile defines the `ic:ClaimType` element for use as a child of the `wst:Claims` element. A Relying Party MAY use this element to specify an individual claim type required. Further, each required claim MAY be specified as being *mandatory* or *optional*. Multiple `ic:ClaimType` elements can be included to specify multiple claim types required.

The outline for the `ic:ClaimType` element is as follows:

Syntax:

```
<ic:ClaimType Uri="xs:anyURI" Optional="xs:boolean"? /> *
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:ClaimType

Indicates the required claim type.

/ic:ClaimType/@Uri

The unique identifier of the required claim type.

/ic:ClaimType/@Optional

Indicates if the claim can be absent in the Security Token. By default, any required claim type is a mandatory claim and must be present in the issued Security Token.

Two `<ic:ClaimType>` elements refer to the same claim type if and only if the values of their XML attribute named `Uri` are equal in a case-sensitive string comparison.

When the `ic:ClaimType` element is used within the `wst:Claims` parameter in a token policy to specify claims requirement, the `wst:Dialect` attribute on the `wst:Claims` element MUST be qualified with the URI value below.

Dialect URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity
```

The above dialect URI value indicates that the specified claim elements are to be processed according to this profile.

Following is an example of using this assertion within an issued token policy to require two claim types where one claim type is optional.

Example:

```
<sp:IssuedToken ...>
  ...
  <sp:RequestSecurityTokenTemplate>
    ...
    <wst:Claims
        Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/givenname"/>
      <ic:ClaimType
          Uri="http://.../ws/2005/05/identity/claims/surname"
          Optional="true" />
    </wst:Claims>
  </sp:RequestSecurityTokenTemplate>
  ...
</sp:IssuedToken>
```

This profile also defines a standard set of claim types for common personal information about users that MAY be requested by Relying Party Web services in Security Tokens and supported by any Identity Provider. These standard claim types are defined in Section 7.4.

2.2 Expressing Privacy Policy of Relying Party

A Relying Party Web service SHOULD publish its “Privacy Policy”. Users may decide to release tokens and interact further with that service based on its Privacy Policy. No assumptions are made regarding the format and content of the Privacy Policy and an Identity Selector is not required to parse, interpret or act on the Privacy Policy programmatically.

To express the location of its privacy statement, a Web service MUST use the optional policy assertion `ic:PrivacyNotice` defined below:

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt"?> xs:anyURI </ic:PrivacyNotice>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:PrivacyNotice

This element is used to express the location of the privacy statement of a Web service.

/ic:PrivacyNotice/@Version

This optional attribute provides a version number for the privacy statement allowing changes in its content to be reflected as a change in the version number. If present, it MUST have a minimum value of 1.

Following is an example of using this policy element to express the location of the privacy statement of a Web service.

Example:

```
<wsp:Policy>
  ...
  <ic:PrivacyNotice Version="1">
    http://www.contoso.com/privacy
  </ic:PrivacyNotice>
  ...
</wsp:Policy>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

Because the Privacy Policy assertion points to a “privacy statement” that applies to a service endpoint, the assertion MUST apply to [Endpoint Policy Subject]. In other words, a policy expression containing the Privacy Policy assertion MUST be attached to a `wsdl:binding` in the metadata for the service.

Further, when an Identity Selector can only render the privacy statement document in a limited number of document formats (media types), it MAY use the HTTP request-header field “Accept” in its HTTP GET request to specify the media-types it can accept. For example, the following request-header specifies that the client will accept the Privacy Policy only as a plain text or a HTML document.

```
Accept: text/plain, text/html
```

Similarly, if an Identity Selector wants to obtain the privacy statement in a specific language, it MAY use the HTTP request-header field “Accept-Language” in its HTTP GET request to specify the languages it is willing to accept. For example, the following request-header specifies that the client will accept the Privacy Policy only in Danish.

```
Accept-Language: da
```

A Web service, however, is not required to be able to fulfill the document format and language requests of an Identity Selector. It may publish its privacy statement in a fixed set of document formats and languages.

2.3 Employing Relying Party STSs

The Security Policy of a Relying Party MAY require that an issued token be obtained from a Relying Party STS. This can create a chain of STSs. The Identity Selector MUST follow the RP/STS chain, contacting each referenced STS, resolving its Policy statements and continuing to the STS it refers to.

When following a chain of STSs, when an STS with an `ic:RequireFederatedIdentityProvisioning` declaration is encountered as per Section 3.2.1, this informs the Identity Selector that the STS is an IP/STS, rather than a member of the RP/STS chain. Furthermore, if an RP or RP/STS provides an incomplete Security Policy, such as no issuer or no required claims, the Identity Selector MUST be invoked so a card and requested claims can be selected by the user, enabling a Request for Security Token (RST) to be constructed and sent to the selected IP/STS.

The RP/STS's Policy is used for card matching. If the RP/STS requests a PPID, the RP/STS's certificate is used for calculating the PPID – not the certificate of the Relying Party. This enables a single RP/STS to service multiple Relying Parties while always receiving the same PPID for a given user from the Identity Selector.

Identity Selectors MUST enable users to make Relying Party trust decisions based on the identity of the Relying Party, possibly including displaying attributes from its certificate. By trusting the RP, the user is implicitly trusting the chain of RP/STSs that the RP employs.

Each RP/STS endpoint MUST provide a certificate. This certificate MAY be communicated either via Transport (such as HTTPS) or Message (such as WS-Security) Security. If Message Security is employed, transports not providing security (such as HTTP) may be used.

Like IP/STSs, RP/STSs publish endpoint metadata. This metadata MAY be retrieved via either WS-MetadataExchange or HTTPS GET in the same manner that IP/STS metadata can be, as described in Section 3.1.1.2.

Like IP/STSs, no changes to the syntax used to specify metadata locations occurs when RP/STS metadata is published by the Relying Party STS as a page retrievable using HTTPS GET. Relying Parties and Identity Providers MAY consequently support either or both retrieval methods for the same metadata addresses.

3 Identity Provider Interactions

This section defines the constructs used by an Identity Selector for interacting with an Identity Provider to obtain Information Cards, and to request and obtain Security Tokens.

3.1 Information Card

An Information Card represents a Digital Identity of a Subject that can be issued by an Identity Provider. It is an artifact containing metadata that represents the token issuance relationship between an Identity Provider and a Subject, and provides a visual representation of the Digital Identity. Multiple Digital Identities for a Subject from the same Identity Provider are represented by different Information Cards. Subjects may obtain an Information Card from an Identity Provider, and may have a collection of Information Cards from various Identity Providers.

3.1.1 Information Card Format

An Information Card is represented as a signed XML document that is issued by an Identity Provider. The XML schema for an Information Card is defined below:

Syntax:

```
<ic:InformationCard xml:lang="xs:language" ...>
  <ic:InformationCardReference> ... </ic:InformationCardReference>
  <ic:CardName> xs:string </ic:CardName> ?
  <ic:CardImage MimeType="xs:string"> xs:base64Binary </ic:CardImage> ?
  <ic:Issuer> xs:anyURI </ic:Issuer>
  <ic:TimeIssued> xs:dateTime </ic:TimeIssued>
  <ic:TimeExpires> xs:dateTime </ic:TimeExpires> ?
  <ic:TokenServiceList> ... </ic:TokenServiceList>
  <ic:SupportedTokenTypeList> ... </ic:SupportedTokenTypeList>
  <ic:SupportedClaimTypeList> ... </ic:SupportedClaimTypeList>
  <ic:RequireAppliesTo ...> ... </ic:RequireAppliesTo> ?
  <ic:PrivacyNotice ...> ... </ic:PrivacyNotice> ?
  <ic07:RequireStrongRecipientIdentity /> ?
  <ic07:IssuerInformation> ... </ic07:IssuerInformation> *
  ...
</ic:InformationCard>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:InformationCard

An Information Card issued by an Identity Provider.

/ic:InformationCard/@xml:lang

A required language identifier, using the language codes specified in [RFC 3066], in which the content of localizable elements have been localized.

/ic:InformationCard/ic:InformationCardReference

This required element provides a specific reference for the Information Card by which it can be uniquely identified within the scope of an issuer. This reference MUST be included by an Identity Selector in all token requests sent to the Identity Provider based on that Information Card. The detailed schema of this element is defined in Section 3.1.1.1.

/ic:InformationCard/ic:CardName

This optional element provides a friendly textual name for the issued Information Card. The content of this element MAY be localized in a specific language.

/ic:InformationCard/ic:CardImage

This optional element contains a base64 encoded inline image that provides a graphical image for the issued Information Card. It SHOULD contain an image within the size range of 60 pixels wide by 40 pixels high and 240 pixels wide by 160 pixels high. It is RECOMMENDED that the image have an aspect ratio of 3:2 and the image size be 120 by 80 pixels.

/ic:InformationCard/ic:CardImage/@MimeType

This required attribute provides a MIME type specifying the format of the included card image. This profile supports multiple image formats (e.g., JPEG, GIF) as enumerated in the schema for this profile.

/ic:InformationCard/ic:Issuer

This required element provides a logical name for the issuer of the Information Card. If a Relying Party specifies a token issuer by its logical name, then the content of this element MUST be used to match the required token issuer with an Information Card.

/ic:InformationCard/ic:TimeIssued

This required element provides the date and time when the Information Card was issued.

/ic:InformationCard/ic:TimeExpires

This optional element provides the date and time after which the Information Card SHOULD be treated as expired and invalid.

/ic:InformationCard/ic:TokenServiceList

This required element provides an ordered list of Security Token Service (IP/STS) endpoints, and corresponding credential descriptors (implying the required authentication mechanisms), where tokens can be requested. Each service endpoint MUST be tried in order by the Service Requester when requesting tokens.

/ic:InformationCard/ic:SupportedTokenTypeList

This required element contains the list of token types that are offered by the Identity Provider.

/ic:InformationCard/ic:SupportedClaimTypeList

This required element contains the list of claim types that are offered by the Identity Provider.

/ic:InformationCard/ic:RequireAppliesTo

This optional element indicates that token requests MUST include information identifying the Relying Party where the issued token will be used. The Relying Party information MUST be included as the content of a `wsp:AppliesTo` element in the token request.

/ic:InformationCard/ic:PrivacyNotice

This optional element provides the location of the privacy statement of the Identity Provider.

/ic:InformationCard/ic07:RequireStrongRecipientIdentity

This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, for example, an X.509v3 certificate.

/ic:InformationCard/ic07:IssuerInformation

This optional element provides information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic:InformationCard/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

.../ic:InformationCard/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:InformationCard` element of an `ic:RoamingStore` when representing the card in the Information Cards Transfer Format in Section 6.1.

3.1.1.1 Information Card Reference

Every Information Card issued by an Identity Provider MUST have a unique reference by which it can be identified within the scope of the Identity Provider. This reference is included in all token requests sent to the Identity Provider based on that Information Card.

The card reference MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:InformationCardReference>
  <ic:CardId> xs:anyURI </ic:CardId>
  <ic:CardVersion> xs:unsignedInt </ic:CardVersion>
</ic:InformationCardReference>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:InformationCardReference

A specific reference for an Information Card.

.../ic:InformationCardReference/ic:CardId

This required element provides a unique identifier in the form of a URI for the specific Information Card. The identifier provider must be able to identify the specific Information Card based on this identifier.

.../ic:InformationCardReference/ic:CardVersion

This required element provides a versioning epoch for the Information Card issuance infrastructure used by the Identity Provider. The minimum value for this field MUST be 1. Note that it is possible to include version information in CardId as it is a URI, and can have hierarchical content. However, it is specified as a separate value to allow the Identity Provider to change its issuance infrastructure, and thus its versioning epoch, independently without changing the CardId of all issued Information Cards. For example, when an Identity Provider makes a change to the supported claim types or any other policy pertaining to the issued cards, the version number allows the Identity Provider to determine if the Information Card needs to be refreshed. The version number is assumed to be monotonically increasing. If two Information Cards have the same CardId value but different CardVersion values, then the one with a higher numerical CardVersion value should be treated as being more up-to-date.

3.1.1.2 Token Service Endpoints and Authentication Mechanisms

Every Information Card issued by an Identity Provider MUST include an ordered list of IP/STS endpoints, and the corresponding credential type to be used, for requesting tokens. The list MUST be in a decreasing order of preference. Identity Selectors SHOULD attempt to use the endpoints in the order listed, using the first endpoint in the list for which the metadata is retrievable and the endpoint is reachable. For each endpoint, the required credential type implicitly determines the authentication mechanism to be used. Each credential descriptor is personalized for the user to allow an Identity Selector to automatically locate the credential once the user has selected an Information Card.

Further, each IP/STS endpoint reference in the Information Card MUST include the Security Policy metadata for that endpoint. The policy metadata MAY be specified as a metadata location within the IP/STS endpoint reference. If a metadata location URL is specified, it MUST use the [HTTPS] transport. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

The ordered list of token service endpoints MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:TokenServiceList>
 (<ic:TokenService>
   <wsa:EndpointReference> ... </wsa:EndpointReference>
   <ic:UserCredential>
    <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint> ?
    (
    <ic:UsernamePasswordCredential>...</ic:UsernamePasswordCredential> |
    <ic:KerberosV5Credential>...</ic:KerberosV5Credential> |
    <ic:X509V3Credential>...</ic:X509V3Credential> |
    <ic:SelfIssuedCredential>...</ic:SelfIssuedCredential> | ...
   )
   </ic:UserCredential>
  </ic:TokenService>) +
</ic:TokenServiceList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:TokenServiceList

This required element provides an ordered list of Security Token Service endpoints (in decreasing order of preference), and the corresponding credential types, for requesting tokens. Each service endpoint MUST be tried in order by a Service Requester.

.../ic:TokenServiceList/ic:TokenService

This required element describes a single token issuing endpoint.

.../ic:TokenServiceList/ic:TokenService/wsa:EndpointReference

This required element provides the endpoint reference for a single token issuing endpoint. For the Self-issued Identity Provider, the special address value defined in Section 2.1.1 MAY be used. The `wsid:Identity` extension element (see Section 12) for endpoint references MAY be used to include the protection token for this endpoint to secure communications with it.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential

This required element indicates the credential type to use to authenticate to the token issuing endpoint.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/ic:DisplayCredentialHint

This optional element provides a hint (string) to be displayed to the user to prompt for the correct credential (e.g. a hint to insert the right smart card). The content of this element MAY be localized in a specific language.

.../ic:TokenServiceList/ic:TokenService/ic:UserCredential/<credential descriptor>

This required element provides an unambiguous descriptor for the credential to use for authenticating to the token issuing endpoint. The schema to describe the credential is specific to each credential type. This profile defines the schema elements `ic:UsernamePasswordCredential`, `ic:KerberosV5Credential`, `ic:X509V3Credential` or `ic:SelfIssuedCredential` later in Section 4 corresponding to username/password, Kerberos v5, X.509v3 certificate and self-issued token based credential types. Other credential types MAY be introduced via the extensibility point defined in the schema within this element.

Alternatively, Identity Providers MAY publish metadata for Information Cards as WSDL documents that can be retrieved by Identity Selectors via HTTPS GET operations on URLs using the HTTPS scheme. An endpoint's metadata URL is communicated to Identity Selectors in a token service `wsx:MetadataReference` element in an Information Card using exactly the same syntax as when WS-MetadataExchange is employed to retrieve the metadata. No change occurs in the card.

The metadata documents published via HTTPS GET SHOULD contain the WSDL for the endpoint as the top-level element of the document without any SOAP or WS-MetadataExchange elements enclosing it.

Identity Providers MAY publish endpoint metadata via both the HTTPS GET and WS-MetadataExchange methods at the same metadata location. If they publish the metadata via multiple mechanisms, the metadata delivered via both mechanisms SHOULD be the same. Likewise, Identity Selectors MAY attempt to retrieve metadata via multiple mechanisms, either in sequence or in parallel.

The following example illustrates an Identity Provider with two endpoints for its IP/STS, one requiring Kerberos (higher priority) and the other requiring username/password (lower priority) as its authentication mechanism. Further, each endpoint also includes its policy metadata location as a URL using the [HTTPS] scheme.

Example:

```
<ic:TokenServiceList>
  <ic:TokenService>
    <wsa:EndpointReference>
      <wsa:Address>http://contoso.com/sts/kerb</wsa:Address>
      <wsid:Identity>
        <wsid:Spn>host/corp-sts.contoso.com</wsid:Spn>
      </wsid:Identity>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/kerb/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:KerberosV5Credential />
    </ic:UserCredential>
  </ic:TokenService>
  <ic:TokenService>
    <wsa:EndpointReference>
      <wsa:Address>http://contoso.com/sts/pwd</wsa:Address>
      <wsa:Metadata>
        <wsx:Metadata>
          <wsx:MetadataSection
              Dialect="http://schemas.xmlsoap.org/ws/2004/09/mex">
            <wsx:MetadataReference>
              <wsa:Address>https://contoso.com/sts/pwd/mex</wsa:Address>
            </wsx:MetadataReference>
          </wsx:MetadataSection>
        </wsx:Metadata>
      </wsa:Metadata>
    </wsa:EndpointReference>
    <ic:UserCredential>
      <ic:UsernamePasswordCredential>
        <ic:Username>Zoe</ic:Username>
      </ic:UsernamePasswordCredential>
    </ic:UserCredential>
  </ic:TokenService>
</ic:TokenServiceList>
```

3.1.1.3 Token Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of token types that can be issued by the Identity Provider. The set of token types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedTokenTypeList>
  <wst:TokenType> xs:anyURI </wst:TokenType> +
</ic:SupportedTokenTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedTokenTypeList

This required element contains the set of token types offered by the Identity Provider.

.../ic:SupportedTokenTypeList/wst:TokenType

This required element indicates an individual token type that is offered.

The following example illustrates an Identity Provider that offers both SAML 1.1 and SAML 2.0 tokens.

Example:

```
<ic:SupportedTokenTypeList>
  <wst:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst:TokenType>
  <wst:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</wst:TokenType>
</ic:SupportedTokenTypeList>
```

3.1.1.4 Claim Types Offered

Every Information Card issued by an Identity Provider SHOULD include an unordered list of claim types that can be issued by the Identity Provider. The set of claim types offered by the Identity Provider MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:SupportedClaimTypeList>
  (<ic:SupportedClaimType Uri="xs:anyURI">
    <ic:DisplayTag> xs:string </ic:DisplayTag> ?
    <ic:Description> xs:string </ic:Description> ?
  </ic:SupportedClaimType>) +
</ic:SupportedClaimTypeList>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SupportedClaimTypeList

This required element contains the set of claim types offered by the Identity Provider.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType

This required element indicates an individual claim type that is offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/@Uri

This required attribute provides the unique identifier (URI) of this individual claim type offered.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:DisplayTag

This optional element provides a friendly name for this individual. The content of this element MAY be localized in a specific language.

.../ic:SupportedClaimTypeList/ic:SupportedClaimType/ic:Description

This optional element provides a description of the semantics for this individual claim type. The content of this element MAY be localized in a specific language.

The following example illustrates an Identity Provider that offers two claim types.

Example:

```
<ic:SupportedClaimTypeList>
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/givenname">
    <ic:DisplayTag>Given Name</DisplayTag>
  </ic:SupportedClaimType>
  <ic:SupportedClaimType Uri=".../ws/2005/05/identity/claims/surname">
    <ic:DisplayTag>Last Name</DisplayTag>
  </ic:SupportedClaimType>
</ic:SupportedClaimTypeList>
```

3.1.1.5 Requiring Token Scope Information

An Identity Selector, by default, SHOULD NOT convey information about the Relying Party where an issued token will be used (i.e., target scope) when requesting Security Tokens. This helps safeguard user privacy. However, an Identity Provider MAY override that behavior.

Every Information Card issued by an Identity Provider MAY include a requirement that token requests must include token scope information identifying the Relying Party where the token will be used. The requirement to submit token scope information MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:RequireAppliesTo Optional="xs:boolean" /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireAppliesTo

This optional element indicates a requirement for a token requester to submit token scope information in the request. Absence of this element in an Information Card means that the token requester MUST NOT submit any token scope information.

.../ic:RequireAppliesTo/@Optional

> This optional attribute indicates whether the token scope information is mandatory or is optionally accepted by the Identity Provider. An attribute value of “true” indicates that the token scope information is not mandatory, but will be accepted by the Identity Provider if submitted. An attribute value of “false” (default) indicates that the token scope information is mandatory.

The following example illustrates the use of this element.

Example:

```
<ic:RequireAppliesTo Optional="true" />
```

If token scope information is required by an Identity Provider, an Identity Selector MUST include the Relying Party identity as the content of the `wsp:AppliesTo` element in the token request. The actual behavior of an Identity Selector vis-à-vis the possible requirements that can be expressed by the above element is specified in Section 3.3.3.

3.1.1.6 Privacy Policy Location

Every Information Card issued by an Identity Provider SHOULD include a pointer to the privacy statement of the Identity Provider. The location of the privacy statement MUST be expressed using the following schema element within an Information Card.

Syntax:

```
<ic:PrivacyNotice Version="xs:unsignedInt" /> ?
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:PrivacyNotice

> This optional element provides the location of the privacy statement of the Identity Provider.

.../ic:PrivacyNotice/@Version

> This optional attribute indicates a version number that tracks changes in the content of the privacy statement. This field MUST have a minimum value of 1 when present.

The following example illustrates the use of this element.

Example:

```
<ic:PrivacyNotice Version="1">
  http://www.contoso.com/privacynotice
</ic:PrivacyNotice>
```

An Identity Selector MUST be able to accept a privacy statement location specified as an URL using the [HTTP] scheme (as illustrated above) or the [HTTPS] scheme.

3.1.1.7 Prohibiting Use at Relying Parties Not Identified by a Cryptographically Protected Identity

Information Cards issuers MAY specify that a card MUST NOT be used at Relying Parties that do not present a cryptographically protected identity, such as an X.509v3 Certificate. This would typically be done when the issuer determines that the use of HTTP without Message Security would not provide a sufficiently secure environment for the use of the card.

Syntax:

```
<ic07:RequireStrongRecipientIdentity /> ?
```

.../ic07:RequireStrongRecipientIdentity

> This optional element informs the Identity Selector that it MUST only allow the card to be used at a Relying Party that presents a cryptographically protected identity, such as an X.509v3 certificate.

3.1.1.8 Providing Custom Data to Display with the Card

Card issuers MAY supply a set of information about the card that MAY be displayed by the Identity Selector user interface.

Syntax:

```
<ic07:IssuerInformation>
  <IssuerInformationEntry>
    <EntryName> xs:string </EntryName>
    <EntryValue> xs:string </EntryValue>
  </IssuerInformationEntry> +
</ic07:IssuerInformation>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic07:IssuerInformation

This optional element provides a set of information from the card issuer about the card that can be displayed by the Identity Selector user interface.

.../ic07:IssuerInformation/IssuerInformationEntry

This element provides one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryName

This element provides the name of one item of information about the card.

.../ic07:IssuerInformation/IssuerInformationEntry/EntryValue

This element provides the value of one item of information about the card.

The following example illustrates the use of this feature.

Example:

```
<ic07:IssuerInformation>
  <IssuerInformationEntry>
    <EntryName>Customer Service</EntryName>
    <EntryValue>+1-800-CONTOSO</EntryValue>
  </IssuerInformationEntry>
  <IssuerInformationEntry>
    <EntryName>E-mail Contact</EntryName>
    <EntryValue>cardhelp@contoso.com</EntryValue>
  </IssuerInformationEntry>
</ic07:IssuerInformation>
```

3.1.2 Issuing Information Cards

An Identity Provider can issue Information Cards to its users using any out-of-band mechanism that is mutually suitable.

In order to provide the assurance that an Information Card is indeed issued by the Identity Provider expected by the user, the Information Card MUST be carried inside a digitally signed envelope that is signed by the Identity Provider. For this, the "enveloping signature" construct (see [XMLDSIG]) MUST be used where the Information Card is included in the `ds:Object` element. The signature on the digitally signed envelope provides data origin authentication assuring the user that it came from the right Identity Provider.

The specific profile of XML digital signatures [XMLDSIG] that is RECOMMENDED for signing the envelope carrying the Information Card is as follows. Usage of other algorithms is not described.

- Use *enveloping signature* format when signing the Information Card XML document.
- Use a single `ds:Object` element within the signature to hold the `ic:InformationCard` element that represents the issued Information Card. The `ds:Object/@Id` attribute provides a

convenient way for referencing the Information Card from the `ds:SignedInfo/ds:Reference` element within the signature.

- Use RSA signing and verification with the algorithm identifier given by the URI *http://www.w3.org/2000/09/xmldsig#rsa-sha1*.
- Use exclusive canonicalization with the algorithm identifier given by the URI *http://www.w3.org/2001/10/xml-exc-c14n#*.
- Use SHA1 digest method for the data elements being signed with the algorithm identifier *http://www.w3.org/2000/09/xmldsig#sha1*.
- There MUST NOT be any other transforms used in the enveloping signature for the Information Card other than the ones listed above.
- The `ds:KeyInfo` element MUST be present in the signature carrying the signing key information in the form of an X.509 v3 certificate or a X.509 v3 certificate chain specified as one or more `ds:X509Certificate` elements within a `ds:X509Data` element.

The following example shows an enveloping signature carrying an Information Card that is signed by the Identity Provider using the format outlined above. Note that whitespace (newline and space character) is included in the example only to improve readability; they may not be present in an actual implementation.

Example:

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="#_Object_InformationCard">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue> ... </DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue> ... </SignatureValue>
  <KeyInfo>
    <X509Data>
      <X509Certificate> ... </X509Certificate>
    </X509Data>
  </KeyInfo>
  <Object Id="_Object_InformationCard">
    <ic:InformationCard
        xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity"
        xml:lang="en-us">
      [Information Card content]
    </ic:InformationCard>
  </Object>
</Signature>
```

An Identity Selector MUST verify the enveloping signature. The `ic:InformationCard` element can then be extracted and stored in the Information Card collection.

3.2 Identity Provider Policy

This section specifies additional policy elements and requirements introduced by this profile for an IP/STS policy metadata.

3.2.1 Require Information Card Provisioning

In the Information Card Model, an Identity Provider requires provisioning in the form of an Information Card issued by it which represents the provisioned identity of the user. In order to enable an Identity Selector to learn that such pre-provisioning is necessary before token requests can be made, the Identity Provider MUST provide an indication in its policy.

An Identity Provider issuing Information Cards MUST specify this provisioning requirement in its policy using the following schema element.

Syntax:

```
<ic:RequireFederatedIdentityProvisioning />
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:RequireFederatedIdentityProvisioning

This element indicates a requirement that one or more Information Cards, representing identities that can be federated, must be pre-provisioned before token requests can be made to the Identity Provider.

The following example illustrates the use of this policy element.

Example:

```
<wsp:Policy>
  ...
  <ic:RequireFederatedIdentityProvisioning />
  <sp:SymmetricBinding>
    ...
  </sp:SymmetricBinding>
  ...
</wsp:Policy>
```

3.2.2 Policy Metadata Location

In the Information Card Model, an Identity Provider MUST make the Security Policy metadata for its IP/STS endpoints available. If a metadata location is used for this purpose, the location URL MUST use the [HTTPS] scheme. An Identity Selector MAY retrieve the Security Policy it will use to communicate with the IP/STS from that metadata location using the mechanism specified in [WS-MetadataExchange].

3.3 Token Request and Response

For any given Information Card, an Identity Selector can obtain a Security Token from the IP/STS for that Information Card. Tokens MUST be requested using the “Issuance Binding” mechanism described in [WS-Trust 1.2] and [WS-Trust 1.3]. This section specifies additional constraints and extensions to the token request and response messages between the Identity Selector and the IP/STS.

The WS-Trust protocol requires that a token request be submitted by using the `wst:RequestSecurityToken` element in the request message, and that a token response be sent using the `wst:RequestSecurityTokenResponse` element in the response message. This profile refers to the “Request Security Token” message as RST and the “Request Security Token Response” message as RSTR in short.

The WS-Trust protocol allows for a token response to optionally provide multiple tokens by using the `wst:RequestSecurityTokenResponseCollection` element in the response message. This profile, however, requires that an Identity Provider MUST NOT use the `wst:RequestSecurityTokenResponseCollection` element in the response. The token response MUST consist of a single `wst:RequestSecurityTokenResponse` element.

3.3.1 Information Card Reference

When requesting a Security Token from the IP/STS, an Identity Selector MUST include the Information Card reference in the body of the RST message as a top-level element information item. The `ic:InformationCardReference` element in the Information Card, including all of its [children], [attributes] and [in-scope namespaces], MUST be copied as an immediate child of the RST element in the message as follows.

The following example illustrates the Information Card reference included in a RST message.

Example:

```
<wst:RequestSecurityToken>
  ...
  <ic:InformationCardReference>
    <ic:CardId>http://xyz.com/CardId/d795621fa01d454285f9</ic:CardId>
    <ic:CardVersion>1</ic:CardVersion>
  </ic:InformationCardReference>
  ...
</wst:RequestSecurityToken>
```

The IP/STS MAY fault with `ic:InformationCardRefreshRequired` to signal to the Service Requester that the Information Card needs to be refreshed.

3.3.2 Claims and Other Token Parameters

A Relying Party's requirements of claims and other token parameters are expressed in its policy using the `sp:RequestSecurityTokenTemplate` parameter within the `sp:IssuedToken` policy assertion (see Section 2.1). If all token parameters are acceptable to the Identity Selector, it MUST copy the content of this element (i.e. all of its [children] elements) into the body of the RST message as top-level element information items. However, if optional claims are requested by the Relying Party, requests for optional claims not selected by the user MUST NOT be copied into the RST message.

3.3.3 Token Scope

The WS-Trust protocol allows a token requester to indicate the target where the issued token will be used (i.e., token scope) by using the optional element `wsp:AppliesTo` in the RST message. By default, an Identity Selector SHOULD NOT send token scope information to the Identity Provider in token requests to protect user privacy. In other words, the element `wsp:AppliesTo` is absent in the RST message.

However, if the Identity Provider requires it (see the modes of the `ic:RequireAppliesTo` element described in Section 3.1.1.5), or if the Relying Party's token policy includes the `wsp:AppliesTo` element in the `sp:RequestSecurityTokenTemplate` parameter, then an Identity Selector MUST include token scope information in its token request as per the behavior summarized in the following table.

<RequireAppliesTo> mode in Information Card	*<AppliesTo> element present in RP policy*	*Resulting behavior of Identity Selector*
Mandatory	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Mandatory	No	Send the RP endpoint to which token will be sent as the value of <AppliesTo> in token request to IP.
Optional	Yes	Send <AppliesTo> value from RP policy in token request to IP.
Optional	No	Do not send <AppliesTo> in token request to IP.

Not present	Yes	Fail
Not present	No	Do not send <AppliesTo> in token request to IP.

The following example illustrates the token scope information included in a RST message when it is sent to the Identity Provider.

Example:

```
<wst:RequestSecurityToken>
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://ip.fabrikam.com</wsa:Address>
      <wsid:Identity>
        <ds:KeyInfo>
          <ds:X509Data>
            <ds:X509Certificate>...</ds:X509Certificate>
          </ds:X509Data>
        </ds:KeyInfo>
      </wsid:Identity>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```

3.3.4 Client Pseudonym

A private personal identifier (PPID), defined in Section 7.5.14, identifies a Subject to a Relying Party in a way such that a Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. If an Identity Provider offers the PPID claim type then it MUST generate values for the claim that have this prescribed privacy characteristic using data present in the RST request.

When a Relying Party requests a PPID claim, an Identity Selector MUST provide a Client Pseudonym value via an `ic:PPID` element in the RST request that can be used by the IP/STS as input when computing the PPID claim value in the issued token. The Client Pseudonym SHOULD be produced as described in Section 3.3.4.1. It is RECOMMENDED that the IP/STS combine this Client Pseudonym value with information specific to the entity to which the card was issued as well as a secret known only by the IP/STS and pass the combination through a cryptographically non-invertible function, such as a cryptographic hash function, to generate the PPID claim value sent in the token. Alternatively, when target scope information is sent in the token request using the `wsp:AppliesTo` element, the IP/STS MAY instead choose to use that information to generate an appropriate PPID value.

When Client Pseudonym information is included by an Identity Selector in a token request, it MUST be sent using the following schema element.

Syntax:

```
<ic:ClientPseudonym>
  <ic:PPID> xs:base64Binary </ic:PPID>
</ic:ClientPseudonym>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:ClientPseudonym

This optional top-level element contains the Client Pseudonym information item.

.../ic:ClientPseudonym/ic:PPID

This optional element contains the Client Pseudonym value that the client has submitted for use in computing the PPID claim value for the issued token. The IP/STS MAY use this value as the input (a seed) to a custom cryptographically non-invertible function, with the result used as the PPID claim value in the issued token.

The following example illustrates the Client Pseudonym information sent in a RST message.

Example:

```
<wst:RequestSecurityToken>
  <ic:ClientPseudonym>
    <ic:PPID>MIIEZzCCA9CgAwIBAgIQEmtJZc0=</ic:PPID>
  </ic:ClientPseudonym >
  ...
</wst:RequestSecurityToken>
```

When the target scope information is not sent in the token request to an IP/STS, the Identity Provider MUST NOT record any Client Pseudonym values included in the RST message. It likewise MUST NOT record the PPID claim value that it generates.

3.3.4.1 PPID

When a token request for a PPID claim is sent to an IP/STS, an Identity Selector SHOULD compute the Client Pseudonym PPID information it sends in the RST message as follows:

- Construct the *RP PPID Seed* as described in Section 7.6.1.
- Decode the base64 encoded value of the `ic:HashSalt` element of the Information Card (see Section 6.1) to obtain *SaltBytes*.
- Decode the base64 encoded value of the `ic:MasterKey` element of the Information Card (see Section 6.1) to obtain *MasterKeyBytes*.
- Hash the concatenation of *MasterKeyBytes*, *RP PPID Seed*, and *SaltBytes* using the SHA256 hash function to obtain the Client Pseudonym PPID value.

  Client Pseudonym PPID = SHA256 (MasterKeyBytes + RP PPID Seed + SaltBytes)
- Convert *Client Pseudonym PPID* to a base64 encoded string and send as the value of the `ic:PPID` element in the RST request.

3.3.5 Proof Key for Issued Token

An issued token may have a *symmetric* proof key (symmetric key token), an *asymmetric* proof key (asymmetric key token), or *no* proof key (bearer token). If no key type is specified in the Relying Party policy, then an Identity Selector SHOULD request an asymmetric key token from the IP/STS by default.

The optional `wst:KeyType` element in the RST request indicates the type of proof key desired in the issued Security Token. The IP/STS may return the proof key and/or entropy towards the proof key in the RSTR response. This section describes the behaviors for how each proof key type is requested, who contributes entropy, and how the proof key is computed and returned.

3.3.5.1 Symmetric Proof Key

When requesting a symmetric key token, an Identity Selector MUST submit entropy towards the proof key by augmenting the RST request message as follows:

- The RST SHOULD include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:

  http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey

  http://docs.oasis-open.org/ws-sx/ws-trust/200512/SymmetricKey
- The RST MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing client-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.

The size of the submitted entropy SHOULD be equal to the key size required in the Relying Party policy. If no key size is specified by the Relying Party, then an Identity Selector SHOULD request a key at least 256-bits in size, and submit an entropy of equal size to the IP/STS.

Following is a sample RST request fragment that illustrates a symmetric key token request.

Example:

```
<wst:RequestSecurityToken>
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey
  </wst:KeyType>
  <wst:KeySize>256</wst:KeySize>
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
</wst:RequestSecurityToken>
```

When processing the token request, the IP/STS MAY:

a) accept the client entropy as the sole key material for the proof key,

b) accept the client entropy as partial key material and contribute additional server-side entropy as partial key material to compute the proof key as a function of both partial key materials, or

c) reject the client-side entropy and use server-side entropy as the sole key material for the proof key.

For each of the cases above, the IP/STS MUST compute and return the proof key by augmenting the RSTR response message as follows.

For case (a) where IP/STS accepts client entropy as the sole key material:

- The RSTR MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the client-side entropy as the proof key.

For case (b) where IP/STS accepts client entropy and contributes additional server entropy:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:Entropy` element containing the server-side entropy to be used as partial key material. The entropy is conveyed as raw base64 encoded bits.
- The partial key material from the IP/STS MUST be combined (by each party) with the partial key material from the client to determine the resulting proof key.
- The RSTR MUST include a `wst:RequestedProofToken` element containing a `wst:ComputedKey` element to indicate how the proof key is to be computed. It is RECOMMENDED that an Identity Selector support the P_SHA1 computed key mechanism defined in [WS-Trust 1.2] or [WS-Trust 1.3] with the particulars below. Usage of other algorithms is not described.

ComputedKey Value	*Meaning*
http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1 or http://docs.oasis-open.org/ws-sx/ws-trust/200512/CK/PSHA1	The key is computed using P_SHA1 from the TLS specification to generate a bit stream using entropy from both sides. The exact form is: key = P_SHA1 (Entropy_{REQ}, Entropy_{RES})

Following is a sample RSTR response fragment that illustrates a token response with partial key material from the IP/STS and a computed proof key.

Example:

```
<wst:RequestSecurityTokenResponse>
  ...
  <wst:Entropy>
    <wst:BinarySecret>mQlxWxEiKOcUfnHgQpylcD7LYSkJplpE=</wst:BinarySecret>
  </wst:Entropy>
  <wst:RequestedProofToken>
    <wst:ComputedKey>
      http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1
    </wst:ComputedKey>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

For case (c) where IP/STS contributes server entropy as the sole key material:

- The RSTR MUST include a `wst:BinarySecret` element inside a `wst:RequestedProofToken` element containing the specific proof key to be used. The proof key is conveyed as raw base64 encoded bits.

Following is a sample RSTR response fragment that illustrates a token response with fully specified proof key from the IP/STS.

Example:

```
<wst:RequestSecurityTokenResponse>
  ...
  <wst:RequestedProofToken>
    <wst:BinarySecret>
      mQlxWxEiKOcUfnHgQpylcDKOcUfnHg7LYSkJplpE=
    </wst:BinarySecret>
  </wst:RequestedProofToken>
</wst:RequestSecurityTokenResponse>
```

The following table summarizes the symmetric proof key computation rules to be used by an Identity Selector:

Token Requester (Identity Selector)	*Token Issuer (IP/STS)*	*Results*
Provides entropy	Uses requester entropy as proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.
Provides entropy	Uses requester entropy and provides additional entropy of its own	<wst:Entropy> element present in RSTR containing issuer supplied entropy. <wst:RequestedProofToken> element present in RSTR containing computed key mechanism. Requestor and Issuer compute proof key by combining both entropies using the specified computed key mechanism.
Provides entropy	Uses own entropy as proof key (rejects requester entropy)	<wst:RequestedProofToken> element present in RSTR containing the proof key.

3.3.5.2 Asymmetric Proof Key

When requesting an asymmetric key token, it is RECOMMENDED that an Identity Selector generate an ephemeral RSA key pair. Usage of other algorithms is not described. The generated RSA key pair

MUST be at least 1024-bits in size for use as the proof key. It MUST submit the public key to the IP/STS by augmenting the RST request as follows:

- The RST MUST include a `wst:KeyType` element with one of the two following URI values, depending upon the version of WS-Trust being used:

  http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey

  http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey

- The RST SOAP body MUST include a `wst:UseKey` element containing the public key to be used as proof key in the returned token. The public key is present as a raw RSA key in the form of a `ds:RSAKeyValue` element inside a `ds:KeyValue` element.
- The RST SOAP security header SHOULD include a supporting signature to prove ownership of the corresponding private key. The `ds:KeyInfo` element within the signature, if present, MUST include the same public key as in the `wst:UseKey` element in the SOAP body.
- The supporting signature, if present, MUST be placed in the SOAP security header where the signature for an endorsing supporting token would be placed as per the security header layout specified in WS-SecurityPolicy.

Following is a sample RST request fragment that illustrates an asymmetric key based token request containing the public key and proof of ownership of the corresponding private key.

Example:

```
<s:Envelope ... >
  <s:Header>
    ...
    <wsse:Security>
      ...
      <ds:Signature Id="_proofSignature">
        <!-- signature proving possession of submitted proof key -->
        ...
        <!-- KeyInfo in signature contains the submitted proof key -->
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </s:Header>
  <s:Body wsu:Id="req">
    <wst:RequestSecurityToken>
      ...
      <wst:KeyType>
        http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey
      </wst:KeyType>
      <wst:UseKey Sig="#_proofSignature">
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>
              <ds:Modulus>...</ds:Modulus>
              <ds:Exponent>...</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </wst:UseKey>
    </wst:RequestSecurityToken>
  </s:Body>
```

```
</s:Envelope>
```

If a supporting signature for the submitted proof key is not present in the token request, the IP/STS MAY fail the request. If a supporting signature is present, the IP/STS MUST verify the signature and MUST ensure that the RSA key included in the `wst:UseKey` element and in the supporting signature are the same. If verification succeeds and the IP/STS accepts the submitted public key for use in the issued token, then the token response MUST NOT include a `wst:RequestedProofToken` element. The proof key is implied and an Identity Selector MUST use the public key it submitted as the proof key.

The following table summarizes the asymmetric proof key rules used by an Identity Selector:

Token Requester (Identity Selector)	*Token Issuer (IP/STS)*	*Results*
Provides ephemeral public key for use as proof key	Uses requester supplied proof key	No <wst:RequestedProofToken> element present in RSTR. Proof key is implied.

3.3.5.3 No Proof Key

When requesting a token with no proof key, an Identity Selector MUST augment the RST request message as follows:

- The RST MUST include a `wst:KeyType` element with the following URI value if [WS-Trust 1.2] is being used:

  http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey

  or the RST MUST include a wst:KeyType element with the following URI value if [WS-Trust 1.3] is being used:

  http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer

Following is a sample RST request fragment that illustrates a bearer token request.

Example:

```
<wst:RequestSecurityToken>
  ...
  <wst:KeyType>
    http://schemas.xmlsoap.org/ws/2005/05/identity/NoProofKey
  </wst:KeyType>
</wst:RequestSecurityToken>
```

When processing the token request, if the IP/STS issues a SAML v1.1 bearer token then:

- It MUST specify “urn:oasis:names:tc:SAML:1.0:cm:bearer” as the subject confirmation method in the token.
- It SHOULD include a `saml:AudienceRestrictionCondition` element restricting the token to the target site URL submitted in the token request.

3.3.6 Display Token

An Identity Selector MAY request a Display Token – a representation of the claims carried in the issued Security Token that can be displayed in an user interface – from an IP/STS as part of the token request. To request a Display Token, the following optional element MUST be included in the RST message as a top-level element information item.

Syntax:

```
<ic:RequestDisplayToken xml:lang="xs:language"? ... />
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestDisplayToken

This optional element is used to request an Identity Provider to return a Display Token corresponding to the issued token.

/ic:RequestDisplayToken/@xml:lang

This optional attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content should be localized.

An IP/STS MAY respond to a Display Token request. If it does, it MUST use the following element to return a Display Token for the issued Security Token in the RSTR message.

Syntax:

```
<ic:RequestedDisplayToken ...>
  <ic:DisplayToken xml:lang="xs:language" ... >
    [ <ic:DisplayClaim Uri="xs:anyURI" ...>
        <ic:DisplayTag> xs:string </ic:DisplayTag> ?
        <ic:Description> xs:string </ic:Description> ?
        <ic:DisplayValue> xs:string </ic:DisplayValue> ?
      </ic:DisplayClaim> ] +
    |
    [ <ic:DisplayTokenText MimeType="xs:string">
        xs:string
      </ic:DisplayTokenText> ]
    ...
  </ic:DisplayToken>
</ic:RequestedDisplayToken>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RequestedDisplayToken

This optional element is used to return a Display Token for the Security Token returned in the response.

/ic:RequestedDisplayToken/ic:DisplayToken

The returned Display Token.

/ic:RequestedDisplayToken/ic:DisplayToken/@xml:lang

This required attribute indicates a language identifier, using the language codes specified in [RFC 3066], in which the Display Token content is localized.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim

This required element indicates an individual claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/@Uri

This required attribute provides the unique identifier (URI) of the individual claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayTag

This optional element provides a friendly name for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:Description

This optional element provides a description of the semantics for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayClaim/ic:DisplayValue

This optional element provides the displayable value for the claim returned in the Security Token.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText

This element provides an alternative textual representation of the entire token as a whole when the token content is not suitable for display as individual claims.

/ic:RequestedDisplayToken/ic:DisplayToken/ic:DisplayTokenText/@MimeType

This required attribute provides a MIME type specifying the format of the Display Token content (e.g., “text/plain”).

The following example illustrates a returned Display Token corresponding to a Security Token with two claims.

Example:

```
<ic:RequestedDisplayToken>
  <ic:DisplayToken xml:lang="en-us">
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/givenname">
      <ic:DisplayTag>Given Name</ic:DisplayTag>
      <ic:DisplayValue>John</ic:DisplayValue>
    </ic:DisplayClaim>
    <ic:DisplayClaim Uri="http://.../ws/2005/05/identity/claims/surname">
      <ic:DisplayTag>Last Name</ic:DisplayTag>
      <ic:DisplayValue>Doe</ic:DisplayValue>
    </ic:DisplayClaim>
  <ic:DisplayToken>
</ic:RequestedDisplayToken>
```

3.3.7 Token References

When an IP/STS returns the token requested by an Identity Selector, it MUST also include an attached and an un-attached token reference for the issued security token using the `wst:RequestedAttachedReference` and `wst:RequestedUnattachedReference` elements, respectively, in the RSTR response message.

An Identity Selector is truly a conduit for the security tokens issued by an IP/STS and required by an RP, and it should remain agnostic of the type of the security token passing through it. Furthermore, a security token issued by an IP/STS may be encrypted directly for the RP, thus preventing visibility into the token by the Identity Selector. However, an Identity Selector (or a client application) needs to be able to use the issued security token to perform security operations (such as signature or encryption) on a message sent to an RP and thus needs a way to reference the token both when it is attached to a message and when it is not. The attached and unattached token references returned by an IP/STS in the RSTR message provide the necessary references that can be used for this purpose.

4 Authenticating to Identity Provider

The Information Card schema includes the element content necessary for an Identity Provider to express what credential the user must use in order to authenticate to the IP/STS when requesting tokens. This section defines the schema used to express the credential descriptor for each supported credential type.

4.1 Username and Password Credential

When the Identity Provider requires a *username* and *password* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential>
  <ic:UsernamePasswordCredential>
    <ic:Username> xs:string </ic:Username> ?
  </ic:UsernamePasswordCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:UsernamePasswordCredential

This element indicates that a username/password credential is needed.

.../ic:UsernamePasswordCredential/ic:Username

> This optional element provides the username part of the credential for convenience. An Identity Selector MUST prompt the user for the password. If the username is specified, then its value MUST be copied into the username token used to authenticate to the IP/STS; else an Identity Selector MUST prompt the user for the username as well.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:UsernameToken` assertion requiring a username and password value.

4.2 Kerberos v5 Credential

When the Identity Provider requires a *Kerberos v5 service ticket* for the IP/STS as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential>
  <ic:KerberosV5Credential />
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:KerberosV5Credential

> This element indicates that a Kerberos v5 credential is needed.

To enable the Service Requester to obtain a Kerberos v5 service ticket for the IP/STS, the endpoint reference of the IP/STS in the Information Card or in the metadata retrieved from it MUST include a “service principal name” identity claim (i.e. a `wsid:Spn` element) under the `wsid:Identity` tag as defined in Section 12.

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:KerberosToken` assertion requiring a Kerberos service ticket.

4.3 X.509v3 Certificate Credential

When the Identity Provider requires an *X.509 v3 certificate* for the user as the credential type, where the certificate and keys are in a hardware-based smart card or a software-based certificate, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential>
  <ic:DisplayCredentialHint> xs:string </ic:DisplayCredentialHint>
  <ic:X509V3Credential>
    <ds:X509Data>
      <wsse:KeyIdentifier
        ValueType="http://docs.oasisopen.org/wss/oasiswss-soap-
messagesecurity-1.1#ThumbPrintSHA1"
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis200401-wss-
soap-message-security-1.0#Base64Binary">
        xs:base64binary
      </wsse:KeyIdentifier>
    </ds:X509Data>
  </ic:X509V3Credential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:DisplayCredentialHint

> This optional element provides a user hint string which can be used to prompt the user, for example, to insert the appropriate smart card into the reader.

.../ic:X509Credential

This element indicates that a X.509 certificate credential is needed.

.../ic:X509V3Credential/ds:X509Data/wsse:KeyIdentifier

This element provides a key identifier for the X.509 certificate based on the SHA1 hash of the entire certificate content expressed as a "thumbprint." Note that the extensibility point in the `ds:X509Data` element is used to add `wsse:KeyIdentifier` as a child element.

Furthermore, the actual Security Policy of the IP/STS, expressed in its WSDL, MUST include the `sp:X509Token` assertion requiring an X.509v3 certificate.

4.4 Self-issued Token Credential

When the Identity Provider requires a *self-issued token* as the credential type, the following credential descriptor format MUST be used in the Information Card to specify the required credential.

Syntax:

```
<ic:UserCredential>
  <ic:SelfIssuedCredential>
    <ic:PrivatePersonalIdentifier>
      xs:base64Binary
    </ic:PrivatePersonalIdentifier>
  </ic:SelfIssuedCredential>
</ic:UserCredential>
```

The following describes the attributes and elements listed in the schema outlined above:

.../ic:SelfIssuedCredential

This element indicates that a self-issued token credential is needed.

.../ic:SelfIssuedCredential/ic:PrivatePersonalIdentifier

This required element provides the value of the PPID claim asserted in the self-issued token used previously to register with the IP/STS (see Section 7.5.14).

Furthermore, the actual Security Policy of the IP/STS (expressed in its WSDL) MUST include the `sp:IssuedToken` assertion requiring a self-issued token with exactly one claim, namely, the PPID.

5 Faults

In addition to the standard faults described in WS-Addressing, WS-Security and WS-Trust, this profile defines the following additional faults that may occur when interacting with an RP or an IP. The binding of the fault properties (listed below) to a SOAP 1.1 or SOAP 1.2 fault message is described in [WS-Addressing]. If the optional **[Detail]** property for a fault includes any specified content, then the corresponding schema fragment is included in the listing below.

5.1 Relying Party

The following faults MAY occur when submitting Security Tokens to an RP per its Security Policy.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	*S:Sender*
[Subcode]	ic:RequiredClaimMissing
[Reason]	A required claim is missing from the Security Token.
[Detail]	[URI of missing claim] `<ic:ClaimType Uri="[Claim URI]" />`

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidClaimValue
[Reason]	A claim value asserted in the Security Token is invalid.
[Detail]	[URI of invalid claim] `<ic:ClaimType Uri="[Claim URI]" />`

5.2 Identity Provider

The following faults MAY occur when requesting Security Tokens from an IP using Information Cards.

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:MissingAppliesTo
[Reason]	The request is missing Relying Party identity information.
[Detail]	(None defined.)

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InvalidProofKey
[Reason]	Invalid proof key specified in request.
[Detail]	(None defined.)

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:UnknownInformationCardReference
[Reason]	Unknown Information Card reference specified in request.

[Detail]	[Unknown Information Card reference]
	`<ic:InformationCardReference>`
	`<ic:CardId>[card ID]</ic:CardId>`
	`<ic:CardVersion>[version]</ic:CardVersion>`
	`</ic:InformationCardReference>`

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:FailedRequiredClaims
[Reason]	Could not satisfy required claims in request; construction of token failed

[Detail]	[URIs of claims that could not be satisfied]
	`<ic:ClaimType Uri="[Claim URI]" />`
	`...`

[action]	http://www.w3.org/2005/08/addressing/soap/fault
[Code]	S:Sender
[Subcode]	ic:InformationCardRefreshRequired
[Reason]	Stale Information Card reference specified in request; Information Card should be refreshed

[Detail]	[Information Card reference that needs refreshing]
	`<ic:InformationCardReference>`
	`<ic:CardId>[card ID]</ic:CardId>`
	`<ic:CardVersion>[version]</ic:CardVersion>`
	`</ic:InformationCardReference>`

5.2.1 Identity Provider Custom Error Messages

Identity Providers MAY return custom error messages to Identity Selectors via SOAP faults that can be displayed by the Identity Selector user interface. The error message MUST be communicated as an `S:Text` element within the `S:Reason` element of a SOAP fault message. Multiple `S:Text` elements MAY be returned with different `xml:lang` values and the Identity Selector SHOULD use the one matching the user's locale, if possible.

Example:

```
<s:Envelope xmlns:a="http://www.w3.org/2005/08/addressing"
xmlns:s="http://www.w3.org/2003/05/soap-envelope">
  <s:Header>
```

```
    <a:Action
s:mustUnderstand="1">http://www.w3.org/2005/08/addressing/soap/fault</a:Action
>
  </s:Header>
  <s:Body>
    <s:Fault>
      <s:Code>
        <s:Value>s:Sender</s:Value>
      </s:Code>
      <s:Reason>
        <s:Text xml:lang="en">Message in English ...</</s:Text>
        <s:Text xml:lang="es-ES">Message in the Spanish of Spain ...</s:Text>
      </s:Reason>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

6 Information Cards Transfer Format

This section defines how collections of Information Cards are transferred between Identity Selectors. The cards collection is always transferred after encrypting it with a key derived from a user specified password. Section 6.1 describes the transfer format of the collection in the clear, whereas Section 6.1.2 describes the transfer format after the necessary encryption is applied.

6.1 Pre-Encryption Transfer Format

Each Information Card in the transfer stream will contain metadata and key material maintained by the originating Identity Selector in addition to the original Information Card metadata. If an Identity Selector includes a co-resident Self-issued Identity Provider (described in Section 7), an exported self-issued card may also contain any associated claims information.

The XML schema used for the transfer format is defined below:

Syntax:

```
<ic:RoamingStore>
  <ic:RoamingInformationCard> +
    <ic:InformationCardMetaData>
      [Information Card]
      <ic:IsSelfIssued> xs:boolean </ic:IsSelfIssued>
      <ic:PinDigest> xs:base64Binary </ic:PinDigest> ?
      <ic:HashSalt> xs:base64Binary </ic:HashSalt>
      <ic:TimeLastUpdated> xs:dateTime </ic:TimeLastUpdated>
      <ic:IssuerId> xs:base64Binary </ic:IssuerId>
      <ic:IssuerName> xs:string </ic:IssuerName>
      <ic:BackgroundColor> xs:int </ic:BackgroundColor>
    </ic:InformationCardMetaData>
    <ic:InformationCardPrivateData> ?
      <ic:MasterKey> xs:base64Binary </ic:MasterKey>
      <ic:ClaimValueList> ?
        <ic:ClaimValue Uri="xs:anyURI" ...> +
          <ic:Value> xs:string </ic:Value>
        </ic:ClaimValue>
      </ic:ClaimValueList>
    </ic:InformationCardPrivateData>
    ...
  </ic:RoamingInformationCard>
  ...
</ic:RoamingStore>
```

The following describes the attributes and elements listed in the schema outlined above:

/ic:RoamingStore

The collection of Information Cards selected for transfer.

/ic:RoamingStore/ic:RoamingInformationCard (one or more)

An individual Information Card within the transfer stream.

For brevity, the prefix string “/ic:RoamingStore/ic:RoamingInformationCard” in the element names below is shortened to “...”.

.../ic:InformationCardMetaData

This required element contains the metadata for an Information Card.

.../ic:InformationCardMetaData/[Information Card]

The original content of the Information Card as issued by the Identity Provider (described in Section 3.1.1).

.../ic:InformationCardMetaData/ic:IsSelfIssued

This required element indicates if the card is self-issued (“true”) or not (“false”).

.../ic:InformationCardMetaData/ic:PinDigest

This optional element contains a digest of the user-specified PIN information if the card is PIN-protected. The digest contains the base64 encoded bytes of the SHA1 hash of the user-specified PIN represented as Unicode bytes. Usage of other algorithms is not described.

.../ic:InformationCardMetaData/ic:HashSalt

This optional element contains a random per-card entropy value used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent an Identity Provider.

.../ic:InformationCardMetaData/ic:TimeLastUpdated

This required element contains the date and time when the card was last updated.

.../ic:InformationCardMetaData/ic:IssuerId

This required element contains an identifier for the Identity Provider with which a self-issued credential descriptor in a card issued by that Identity Provider can be resolved to the correct self-issued card. The element content SHOULD be the empty string for self-issued cards.

.../ic:InformationCardMetaData/ic:IssuerName

This required element contains a friendly name of the card issuer.

.../ic:InformationCardMetaData/ic:BackgroundColor

This required element contains the background color used to display the card image.

.../ic:InformationCardMetaData/{any}

This is an extensibility point to allow additional metadata to be included.

.../ic:InformationCardPrivateData

This required element contains the private data for an Information Card.

.../ic:InformationCardPrivateData/ic:MasterKey

This required element contains a base64 encoded 256-bit random number that provides a “secret key” for the Information Card. This key is used for computing the Relying Party specific PPID claim when the card is used at a Relying Party and for computing the Client Pseudonym PPID value sent to an Identity Provider. This element is present both for self-issued and managed Information Cards.

.../ic:InformationCardPrivateData/ic:ClaimValueList

This optional element is a container for the set of claim types and their corresponding values embodied by a self-issued card.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue (one or more)

This required element is a container for an individual claim, *i.e.*, a claim type and its corresponding value.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/@Uri

This required attribute contains a URI that identifies the specific claim type.

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This required element contains the value for an individual claim type.

…/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

…/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore/ic:RoamingInformationCard` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

/ic:RoamingStore/{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them in the respective `ic:RoamingStore` element when updating information using the Information Cards Transfer Format.

6.1.1 PIN Protected Card

When an Information Card is PIN protected, in addition to storing a digest of the PIN in the card data, the master key and claim values associated with the card MUST also be encrypted with a key derived from the user-specified PIN.

It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the encryption key from the PIN. Usage of other algorithms is not described.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of PIN
Salt	16-byte random number (actual value stored along with master key)
Iteration count	1000 (actual value stored along with master key)
Key length	32 octets
Hash function	SHA-256

The encryption method and the corresponding parameters that MUST be used are summarized in the table below.

Encryption method	AES-256
Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as required by AES)

In a PIN-protected card, the encrypted content of the master key and the claim value fields are described below.

.../ic:InformationCardPrivateData/ic:MasterKey

This element MUST contain a base64 encoded byte array comprised of the encryption parameters and the encrypted master key serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Version (for internal use)	0	1
Salt used for key-derivation method	1	16
Iteration count used for key-derivation method	17	4
Initialization Vector (IV) used for encryption	21	16
Encrypted master key	37	master key length

.../ic:InformationCardPrivateData/ic:ClaimValueList/ic:ClaimValue/ic:Value

This element MUST contain a base64 encoded byte array comprised of the encrypted claim value. The encryption parameters used are taken from those serialized into the master key field and summarized in the table above.

6.1.2 Computing the ic:IssuerId

The `ic:IssuerId` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically:

- Compute *IP PPID Seed* in the same manner as *RP PPID Seed* in Section 7.6.1, except that the certificate from `ds:KeyInfo` is used, rather than the Relying Party's.

Use the *IP PPID Seed* as the `ic:IssuerId` value.

The `ic:IssuerId` value SHOULD be the empty string for self-issued cards.

6.1.3 Computing the ic:IssuerName

The `ic:IssuerName` value used for a card when representing it in the Information Cards Transfer Format SHOULD be computed as a function of the `ds:KeyInfo` field of the envelope digitally signed by the Identity Provider. Specifically, if the certificate from `ds:KeyInfo` is an extended validation (EV) certificate [EV Cert], then set `ic:IssuerName` to the Organization Name (O) field value from the certificate, otherwise set `ic:IssuerName` to the Common Name (CN) field value from the certificate.

6.1.4 Creating the ic:HashSalt

A random `ic:HashSalt` value for a card SHOULD be created by the Identity Selector when that card is created from the `ic:InformationCard` data provided by an Identity Provider.

6.2 Post-Encryption Transfer Format

The transfer stream MUST be encrypted with a key derived from a user specified password. The XML schema used for the encrypted transfer stream is defined below:

Syntax:

```
Byte-order-mark
<?xml version="1.0" encoding="utf-8"?>
<ic:EncryptedStore>
  <ic:StoreSalt> xs:base64Binary </ic:StoreSalt>
  <xenc:EncryptedData>
    <xenc:CipherData>
      <xenc:CipherValue> ... </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</ic:EncryptedStore>
...
```

The following describes the elements listed in the XML schema outlined above:

Byte-order-mark

The first three bytes in the stream containing the values {0xEF, 0xBB, 0xBF} constitutes a “byte order mark”.

/ic:EncryptedStore

The top-level container element for the encrypted transfer stream.

/ic:EncryptedStore/ic:StoreSalt

This required element contains the random salt used as a parameter for the key derivation function to derive the encryption key from a user-specified password.

/ic:EncryptedStore/xenc:EncryptedData/xenc:CipherData/xenc:CipherValue

This element contains a base64 encoded byte array containing the ciphertext corresponding to the clear text transfer stream described in Section 6.1.

@{any}

This is an extensibility point to allow additional attributes to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

{any}

This is an extensibility point to allow additional metadata elements to be specified. While an Identity Selector MAY ignore any extensions it does not recognize it SHOULD preserve those that it does not recognize and emit them when updating information using the Information Cards Transfer Format.

The remainder of this section describes the element content of the *xenc:CipherValue* element in the schema outline above. Specifically, it describes the encryption method used and the format of the encrypted content.

The following table defines two symbolic constants, namely *EncryptionKeySalt* and *IntegrityKeySalt*, and their corresponding values used by the key derivation and the encryption methods described below to encrypt the transfer stream.

EncryptionKeySalt	{ 0xd9, 0x59, 0x7b, 0x26, 0x1e, 0xd8, 0xb3, 0x44, 0x93, 0x23, 0xb3, 0x96, 0x85, 0xde, 0x95, 0xfc }
IntegrityKeySalt	{ 0xc4, 0x01, 0x7b, 0xf1, 0x6b, 0xad, 0x2f, 0x42, 0xaf, 0xf4, 0x97, 0x7d, 0x4, 0x68, 0x3, 0xdb }

The transfer stream content is encrypted with a key derived from a user-specified password. It is RECOMMENDED that the PKCS-5 based key derivation method be used with the input parameters summarized in the table below for deriving the key from the password. Usage of other algorithms is not described.

Key derivation method	PBKDF1 per [RFC 2898] (Section 5.1)
Input parameters:	
Password	UTF-8 encoded octets of user-specified password
Salt	16-byte random number (actual value stored in the *ic:StoreSalt* field)
Iteration count	1000
Key length	32 octets
Hash function	SHA-256

The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the encryption key. The order in which the values used MUST be hashed is as follows:

Encryption Key = SHA256 (EncryptionKeySalt + PKCS5-derived-key)

Further, to provide an additional integrity check at the time of import, a “hashed integrity code” MUST be computed as follows and included along with the encrypted transfer stream content.

- The PKCS-5 key derived as per the preceding table MUST be further hashed with a 16-byte salt using the SHA256 hash function, and the resulting value used as the integrity key. The order in which the values used MUST be hashed is as follows:

  Integrity Key = SHA256 (IntegrityKeySalt + PKCS5-derived-key)

- The last block of the clear text transfer stream MUST be captured and further hashed with the *integrity key (IK)* and the *initialization vector (IV)* using the SHA256 hash function, and the resulting value used as the hashed integrity code. The order in which the values used MUST be hashed is as follows:

Hashed Integrity Code = SHA256 (IV + IK + Last-block-of-clear-text)

The encryption method and the corresponding parameters that MUST be used to encrypt the transfer stream are summarized in the table below.

Encryption method	AES-256
Parameters:	
Padding	As per PKCS-7 standard
Mode	CBC
Block size	16 bytes (as required by AES)

The element content of `xenc:CipherValue` MUST be a base64 encoded byte array comprised of the initialization vector used for encryption, the hashed integrity code (as described above), and the encrypted transfer stream. It MUST be serialized as per the binary structure summarized in the table below.

Field	*Offset*	*Size (bytes)*
Initialization Vector (IV) used for encryption	0	16
Hashed integrity code	16	32
Ciphertext of transfer stream	48	Arbitrary

7 Simple Identity Provider Profile

A simple Identity Provider, called the “Self-issued Identity Provider” (SIP), is one which allows users to self-assert identity in the form of self-issued tokens. An Identity Selector MAY include a co-resident Self-issued Identity Provider that conforms to the Simple Identity Provider Profile defined in this section. This profile allows self-issued identities created within one Identity Selector to be used in another Identity Selector such that users do not have to reregister at a Relying Party when switching Identity Selectors. Because of the co-location there is data and metadata specific to an Identity Provider that need to be shareable between Identity Selectors.

7.1 Self-Issued Information Card

The `ic:Issuer` element within an Information Card provides a logical name for the issuer of the Information Card. An Information Card issued by a SIP (*i.e.*, a self-issued Information Card) MUST use the special URI below as the value of the `ic:Issuer` element in the Information Card.

URI:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

7.2 Self-Issued Token Characteristics

The self-issued tokens issued by a SIP MUST have the following characteristics:

- The token type of the issued token MUST be SAML 1.1 which MUST be identified by either of the following token type URIs:
  - *urn:oasis:names:tc:SAML:1.0:assertion*, or
  - *http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1*.
- It is RECOMMENDED that the signature key used in the issued token be a 2048-bit asymmetric RSA key which identifies the issuer. Usage of other algorithms is not described.

- The issuer of the token, indicated by the value of the saml:Issuer attribute on the saml:Assertion root element, MUST be identified by the following URI defined in Section 2.1.1 representing the issuer “self”.

```
http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self
```

- The issued token MUST contain the saml:Conditions element specifying:
  - the token validity interval using the NotBefore and NotOnOrAfter attributes, and
  - the saml:AudienceRestrictionCondition element restricting the token to a specific target scope (i.e., a specific recipient of the token).
- The saml:NameIdentifier element SHOULD NOT be used to specify the Subject of the token.
- The subject confirmation method MUST be specified as one of:
  - *urn:oasis:names:tc:SAML:1.0:cm:holder-of-key*, or
  - *urn:oasis:names:tc:SAML:1.0:cm:bearer* (for Browser based applications).
- When the subject confirmation method is “holder of key”, the subject confirmation key (also referred to as the *proof key*) MUST be included in the token in the ds:KeyInfo child element under the saml:SubjectConfirmation element. The proof key MUST be encoded in the token as follows:
  - For *symmetric* key tokens, the proof key is encrypted to the recipient of the token in the form of a xenc:EncryptedKey child element. It is RECOMMENDED that an AES key with a default size of 256 bits be used, but a different size may be specified by the Relying Party. Usage of other algorithms is not described.
  - For *asymmetric* key tokens, it is RECOMMENDED that the proof key be a public RSA key value specified as a ds:RSAKeyValue child element under the ds:KeyValue element. The default size of the key is 2048 bits. Usage of other algorithms is not described.
- The issued token MUST contain a single attribute statement (i.e., a single saml:AttributeStatement element) containing the subject confirmation data and the required claims (called *attributes* in a SAML token).
- The claim types supported by the self-issued token SHOULD include those listed in Section 7.4.
- The claims asserted in the saml:AttributeStatement element of the issued token MUST be named as follows using the claim type definitions in the XML schema file referenced in Section 7.4. For each claim represented by a saml:Attribute element,
  - the AttributeName attribute is set to the NCname of the corresponding claim type defined in the XML schema file, and
  - the AttributeNamespace attribute is set to the target namespace of the XML schema file, namely

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims
```

It is RECOMMENDED that the XML digital signature [XMLDSIG] profile used to sign a self-issued token be as follows. Usage of other algorithms is not described.

- Uses the *enveloped signature* format identified by the transform algorithm identifier “*http://www.w3.org/2000/09/xmldsig#enveloped-signature*”. The token signature contains a single ds:Reference containing a URI reference to the AssertionID attribute value of the root element of the SAML token.

- Uses the RSA signature method identified by the algorithm identifier “*http://www.w3.org/2000/09/xmldsig#rsa-sha1*”.
- Uses the exclusive canonicalization method identified by the algorithm identifier **“***http://www.w3.org/2001/10/xml-exc-c14n#*” for canonicalizing the token content as well as the signature content.
- Uses the SHA1 digest method identified by the algorithm identifier “*http://www.w3.org/2000/09/xmldsig#sha1*” for digesting the token content being signed.
- No other transforms, other than the ones listed above, are used in the enveloped signature.
- The `ds:KeyInfo` element is always present in the signature carrying the signing RSA public key in the form of a `ds:RSAKeyValue` child element.

Following is an example of a self-issued signed Security Token containing three claims.

Example:

```
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
    AssertionID="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17"
    Issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
    IssueInstant="2004-10-06T16:44:20.00Z"
    MajorVersion="1" MinorVersion="1">
  <Conditions NotBefore="2004-10-06T16:44:20.00Z"
    NotOnOrAfter="2004-10-06T16:49:20.00Z">
    <AudienceRestrictionCondition>
      <Audience>http://www.relying-party.com</Audience>
    </AudienceRestrictionCondition>
  </Conditions>
  <AttributeStatement>
    <Subject>
      <!-- Content here differs; see examples that follow -->
    </Subject>
    <Attribute AttributeName="privatpersonalidentifier"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>
        f8301dba-d8d5a904-462f0027-85dbdec0
      </AttributeValue>
    </Attribute>
    <Attribute AttributeName="givenname"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf</AttributeValue>
    </Attribute>
    <Attribute AttributeName="emailaddress"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <AttributeValue>dasf@mail.com</AttributeValue>
    </Attribute>
  </AttributeStatement>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17">
        <Transforms>
          <Transform
            Algorithm="http://.../2000/09/xmldsig#enveloped-signature"/>
          <Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        </Transforms>
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>vpnIyEi4R/S4b+1vEH4gwQ9iHsY=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>...</SignatureValue>
    <!-- token signing key -->
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>... utnQyEi8R/S4b+1vEH4gwR9ihsV ...</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</Assertion>
```

The content of the `saml:Subject` element in the self-issued token differs based on the subject confirmation method and the type of proof key used. The following examples illustrate each of the three variations of the content of this element.

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “holder of key” using a symmetric proof key.

Example:

```
<Subject>
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
      <!-- symmetric proof key encrypted to recipient -->
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <ds:KeyInfo>
          <ds:X509Data>
            <wsse:KeyIdentifier
              ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-
wss-soap-message-security-1.1#ThumbprintSHA1">
                EdFoIaAeja85201XTzjNMVWy7532jUYtrx=
            </wsse:KeyIdentifier>
          </ds:X509Data>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>
            AuFhiu72+1kaJiAuFhiu72+1kaJi=
          </xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “holder of key” using an asymmetric proof key.

Example:

```
<Subject>
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
    </ConfirmationMethod>
    <ds:KeyInfo>
      <!-- asymmetric RSA public key as proof key -->
      <KeyValue>
        <RSAKeyValue>
          <Modulus>>... FntQyKi6R/E4b+1vDH4gwS5ihsU ...</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </ds:KeyInfo>
  </SubjectConfirmation>
</Subject>
```

The following example illustrates the content of the `saml:Subject` element when subject confirmation method is “bearer” using no proof key.

Example:

```
<Subject>
  <SubjectConfirmation>
    <ConfirmationMethod>
      urn:oasis:names:tc:SAML:1.0:cm:bearer
    </ConfirmationMethod>
  </SubjectConfirmation>
</Subject>
```

7.3 Self-Issued Token Encryption

One of the goals of the Information Card Model is to ensure that any claims are exposed only to the Relying Party intended by the user. For this reason, the SIP SHOULD encrypt the self-issued token under the key of the Relying Party. This guarantees that a token intended for one Relying Party cannot be decoded by nor be meaningful to another Relying Party. As described in Section 8.3, when the Relying Party is not identified by a certificate, because no key is available for the Relying Party in this case, the token can not be encrypted, but SHOULD still be signed.

When a self-issued token is encrypted, the XML encryption [XMLENC] standard MUST be used. The encryption construct MUST use encrypting the self-issued token with a randomly generated symmetric key which in turn is encrypted to the Relying Party’s public key taken from its X.509 v3 certificate. The encrypted symmetric key MUST be placed in an `xenc:EncryptedKey` element within the `xenc:EncryptedData` element carrying the encrypted Security Token.

It is RECOMMENDED that the XML encryption [XMLENC] profile that is used for encrypting the key and the token be as follows. Usage of other algorithms is not described.

- Uses the RSA-OAEP key wrap method identified by the algorithm identifier “*http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p*” for encrypting the encryption key.
- Uses the AES256 with CBC encryption method identified by the algorithm “*http://www.w3.org/2001/04/xmlenc#aes256-cbc*” for encrypting the token. The padding method used is as per the PKCS-7 standard in which the number of octets remaining in the last block is used as the padding octet value.
- The `ds:KeyInfo` element is present in the encrypted key specifying the encryption key information in the form of a Security Token reference.

Following is an illustration of a self-issued token encrypted to a Relying Party using the encryption structure described above.

Example:

```
<xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element">
  <xenc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod
```

```
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      </xenc:EncryptionMethod
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:KeyIdentifier
            ValueType="http://docs.oasis-open.org/wss/2004/xx/oasis-2004xx-
wss-soap-message-security-1.1#ThumbprintSHA1"
            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis200401-
wss-soap-message-security-1.0#Base64Binary">
            +PYbznDaB/dlhjIfqCQ458E72wA=
          </wsse:KeyIdentifier>
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...Ukasdj8257Fjwf=</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
  <xenc:CipherData>
    <!-- Start encrypted Content
    <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
        AssertionID="urn:uuid:08301dba-d8d5-462f-85db-dec08c5e4e17" ...>
      ...
    </Assertion>
    End encrypted content -->
    <xenc:CipherValue>...aKlh4817JerpZoDofy90=</xenc:CipherValue>
  </xenc:CipherData>
</xenc:EncryptedData>
```

7.4 Self-Issued Token Signing Key

The key used to sign a self-issued token presented to a Relying Party also represents a unique identifier for the Subject of the token. In order to prevent the key from becoming a correlation identifier across relying parties, a SIP SHOULD use a different key to sign a self-issued token for each Relying Party where the card is used. In other words, the key used to sign the self-issued token is pair-wise unique for a given Information Card and RP combination. To allow self-issued identities created by a SIP within one Identity Selector to be used in another, the signing keys used by the two SIPs should be the same.

It is RECOMMENDED that the signing key be an RSA key. Usage of other algorithms is not described.

This section specifies the “processing rules” that SHOULD be used by a SIP to derive the RSA key used to sign the self-issued token for a combination of an Information Card and an RP where the card is used. Each self-issued Information Card contains a 256-bit secret random number, called the “master key” (see Section 6.1), that is used as the secret entropy in deriving the token signing RSA key. (Managed Information Cards also have a master key that is used in the Client Pseudonym PPID calculation, as per Section 3.3.4.1.)

Key derivation is done according to the ANSI X9.31 standard for key generation which starts with requiring the use of six random values denoted by X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, and X_q. The processing rules described here enunciate how to transform the master key in an Information Card into the six random inputs for the X9.31 key generation process. The actual key computation algorithm in the X9.31 standard is *not* reproduced here.

The values X_p and X_q are required to be at least 512 bits and each independently carries the full entropy of any Information Card master key of up to 512 bits in length. The values X_{p1}, X_{p2}, X_{q1}, and X_{q2} have a length of only 100 to 121 bits and therefore will be shorter than the Information Card master key and hence cannot each independently carry the full master key entropy. The details of the X9.31 protocol, however, ensure that for reasonably sized master keys, full entropy will be achieved in the generated asymmetric key pair.

7.4.1 Processing Rules

This key generation mechanism can be used to generate 1024 or 2048-bit RSA keys.

Notation: If H is an *n*-bit big-endian value, the convention H[1..p] denotes bits *1* through *p* in the value of H where $p \le n$, and bit-1 is the rightmost (least significant) bit whereas bit-n is the leftmost (most significant) bit in the value of H. Also, the convention X + Y denotes the concatenation of the big-endian bit value of X followed by the big-endian bit value of Y.

Assume that the master key for the selected Information Card (see Section 6.1) is M and the unique *RP Identifier* (derived as per Section 7.6.1) is T. The following processing rules SHOULD be used to derive the inputs for the X9.31 key generation process.

1. Define 32-bit DWORD constants C_n as follows:

   $C_n = n$, where $n = 0,1,2,...,15$

2. Compute SHA-1 hash values H_n as follows:

   If the required key size = 1024 bits, compute

   H_n = SHA1 (M + T + C_n) for $n = 0,1,2,...,9$

   If the required key size = 2048 bits, compute

   H_n = SHA1 (M + T + C_n) for $n = 0,1,2,...,15$

3. Extract the random input parameters for the X9.31 protocol as follows:

   For all key sizes, compute

   X_{p1} [112-bits long] = H_0[1..112]

   X_{p2} [112-bits long] = H_1[1..112]

   X_{q1} [112-bits long] = H_2[1..112]

   X_{q2} [112-bits long] = H_3[1..112]

   If the required key size = 1024 bits, compute

   X_p [512-bits long] = H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160]

   X_q [512-bits long] = H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160]

   If the required key size = 2048 bits, compute

   X_p [1024-bits long] = H_4[1..160] + H_5[1..160] + H_6[1..160] + H_0[129..160] +
   H_{10}[1..160] + H_{11}[1..160] + H_{12}[1..160] + H_2[129..160]

   X_q [1024-bits long] = H_7[1..160] + H_8[1..160] + H_9[1..160] + H_1[129..160] +
   H_{13}[1..160] + H_{14}[1..160] + H_{15}[1..160] + H_3[129..160]

4. The X9.31 specification (Section 4.1.2) requires that the input values X_{p1}, X_{p2}, X_{q1}, X_{q2} MUST satisfy the following conditions.

   - The large prime factors p_1, p_2, q_1, and q_2 are the first primes greater than their respective random X_{p1}, X_{p2}, X_{q1}, X_{q2} input values. They are randomly selected from the set of prime numbers between 2^{100} and 2^{120}, and each shall pass at least 27 iterations of Miller-Rabin.

   To ensure that the lower bound of 2^{100} is met, set the 101th bit of X_{p1}, X_{p2}, X_{q1}, X_{q2} to '1' (*i.e.* X_{p1}[13th byte] |= 0x10, X_{p2}[13th byte] |= 0x10, X_{q1}[13th byte] |= 0x10, X_{q2}[13th byte] |= 0x10).

5. The X9.31 specification (Section 4.1.2) requires that the input values X_p and X_q MUST satisfy the following conditions.

- If the required key size = 1024 bits, then

  $X_p \geq (\sqrt{2})(2^{511})$ and $X_q \geq (\sqrt{2})(2^{511})$

- If the required key size = 2048 bits, then

  $X_p \geq (\sqrt{2})(2^{1023})$ and $X_q \geq (\sqrt{2})(2^{1023})$

To ensure this condition is met, set the two most significant bits of X_p and X_q to '1' (*i.e.* X_p[most significant byte] |= 0xC0, X_q[most significant byte] |= 0xC0).

6. Compute 1024 or 2048-bit keys as per the X9.31 protocol using {X_{p1}, X_{p2}, X_{q1}, X_{q2}, X_p, X_q} as the random input parameters.
7. Use a 32-bit DWORD size *public exponent* value of 65537 for the generated RSA keys.

There are three conditions as follows in the X9.31 specification which, if not met, require that one or more of the input parameters must be regenerated.

- (Section 4.1.2 of X9.31) $|X_p-X_q| \geq 2^{412}$ (for 1024-bit keys) or $|X_p-X_q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q must be regenerated and q recomputed.
- (Section 4.1.2 of X9.31) $|p-q| \geq 2^{412}$ (for 1024-bit keys) or $|p-q| \geq 2^{924}$ (for 2048-bit keys). If not true, X_q must be regenerated and q recomputed.
- (Section 4.1.3 of X9.31) $d > 2^{512}$ (for 1024-bit keys) or $d > 2^{1024}$ (for 2048-bit keys). If not true, X_{q1}, X_{q2}, and X_q must be regenerated and key generation process repeated.

When it is necessary to regenerate an input parameter as necessitated by one or more of the conditions above, it is essential that the regeneration of the input parameter be deterministic to guarantee that all implementations of the key generation mechanism will produce the same results. Furthermore, input regeneration is a potentially unlimited process. In other words, it is possible that regeneration must be performed more than once. In theory, one may need to regenerate input parameters many times before a key that meets all of the requirements can be generated.

The following processing rules MUST be used for regenerating an input parameter *X* of length *n-bits* when necessary:

a. Pad the input parameter *X* on the right, assuming a big-endian representation, with *m* zero-bits where *m* is the smallest number which satisfies *((n+m) mod 128 = 0)*.

b. Encrypt the padded value with the AES-128 (**E**lectronic **C**ode **B**ook mode) algorithm using the 16-byte constant below as the encryption key:

Encryption Key	{ 0x8b, 0xe5, 0x61, 0xf5, 0xbc, 0x3e, 0x0c, 0x4e, 0x94, 0x0d, 0x0a, 0x6d, 0xdc, 0x21, 0x9d, 0xfd }

c. Use the leftmost *n-bits* of the result above as the required regenerated parameter.

If a regenerated parameter does not satisfy the necessary conditions, then repeat the 3-step process above (call it *RegenFunction*) to generate the parameter again by using the output of one iteration as input for the next iteration. In other words, if the output of the i^{th} iteration of the regeneration function above for an input parameter *X* is given by X_i then

$$X_{i+1} = RegenFunction\ (X_i)$$

7.5 Claim Types

This section specifies a set of claim (attribute) types and the corresponding URIs that is defined by this profile for some commonly used personal information. These claim types may be used by a SIP, in self-issued tokens, or by other Identity Providers. Note that, wherever possible, the claims included here reuse and refer to the attribute semantics defined in other established industry standards that deal with personal information. A SIP SHOULD support these claim types at a minimum. Other Identity Providers MAY also support these claim types when appropriate. The URIs defined here MAY be used by a Relying Party to specify required claims in its policy.

The base XML namespace URI that is used by the claim types defined here is as follows:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims
```

For convenience, an XML Schema for the claim types defined here can be found at:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims.xsd
```

7.5.1 First Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname*

Type: *xs:string*

Definition: (*givenName* in [RFC 2256]) Preferred name or first name of a Subject. According to RFC 2256: “This attribute is used to hold the part of a person’s name which is not their surname nor middle name.”

7.5.2 Last Name

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname*

Type: *xs:string*

Definition: (*sn* in [RFC 2256]) Surname or family name of a Subject. According to RFC 2256: “This is the X.500 surname attribute which contains the family name of a person.”

7.5.3 Email Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress*

Type: *xs:string*

Definition: (*mail* in inetOrgPerson) Preferred address for the “To:” field of email to be sent to the Subject, usually of the form <user>@<domain>. According to inetOrgPerson using [RFC 1274]: “This attribute type specifies an electronic mailbox attribute following the syntax specified in RFC 822.”

7.5.4 Street Address

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/streetaddress*

Type: *xs:string*

Definition: (*street* in [RFC 2256]) Street address component of a Subject’s address information. According to RFC 2256: “This attribute contains the physical address of the object to which the entry corresponds, such as an address for package delivery.” Its content is arbitrary, but typically given as a PO Box number or apartment/house number followed by a street name, *e.g.* 303 Mulberry St.

7.5.5 Locality Name or City

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/locality*

Type: *xs:string*

Definition: (*l* in [RFC 2256]) Locality component of a Subject’s address information. According to RFC 2256: “This attribute contains the name of a locality, such as a city, county or other geographic region.” *e.g.* Redmond.

7.5.6 State or Province

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/stateorprovince*

Type: *xs:string*

Definition: (*st* in [RFC 2256]) Abbreviation for state or province name of a Subject's address information. According to RFC 2256: "This attribute contains the full name of a state or province. The values should be coordinated on a national level and if well-known shortcuts exist - like the two-letter state abbreviations in the US – these abbreviations are preferred over longer full names." *e.g.* WA.

7.5.7 Postal Code

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/postalcode*

Type: *xs:string*

Definition: (*postalCode* in X.500) Postal code or zip code component of a Subject's address information. According to X.500(2001): "The postal code attribute type specifies the postal code of the named object. If this attribute value is present, it will be part of the object's postal address - zip code in USA, postal code for other countries."

7.5.8 Country

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/country*

Type: *xs:string*

Definition: (*c* in [RFC 2256]) Country of a Subject. According to RFC 2256: "This attribute contains a two-letter ISO 3166 country code."

7.5.9 Primary or Home Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/homephone*

Type: *xs:string*

Definition: (*homePhone* in inetOrgPerson) Primary or home telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a home telephone number associated with a person." Attribute values should follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.10 Secondary or Work Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/otherphone*

Type: *xs:string*

Definition: (*telephoneNumber* in X.500 Person) Secondary or work telephone number of a Subject. According to X.500(2001): "This attribute type specifies an office/campus telephone number associated with a person." Attribute values should follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.11 Mobile Telephone Number

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/mobilephone*

Type: *xs:string*

Definition: (*mobile* in inetOrgPerson) Mobile telephone number of a Subject. According to inetOrgPerson using [RFC 1274]: "This attribute type specifies a mobile telephone number associated with a person." Attribute values should follow the agreed format for international telephone numbers, *e.g.* +44 71 123 4567.

7.5.12 Date of Birth

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/dateofbirth*

Type: *xs:date*

Definition: The date of birth of a Subject in a form allowed by the *xs:date* data type.

7.5.13 Gender

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/gender*

Type: *xs:token*

Definition: Gender of a Subject that can have any of these exact string values – '0' (meaning unspecified), '1' (meaning Male) or '2' (meaning Female). Using these values allows them to be language neutral.

7.5.14 Private Personal Identifier

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/privatepersonalidentifier*

Type: *xs:base64binary*

Definition: A private personal identifier (PPID) that identifies the Subject to a Relying Party. The word "private" is used in the sense that the Subject identifier is specific to a given Relying Party and hence private to that Relying Party. A Subject's PPID at one Relying Party cannot be correlated with the Subject's PPID at another Relying Party. Typically, the PPID should be generated by an Identity Provider as a pair-wise pseudonym for a Subject for a given Relying Party. For a self-issued Information Card, the Self-issued Identity Provider in an Identity Selector system should generate a PPID for each Relying Party as a function of the card identifier and the Relying Party's identity. The processing rules and encoding of the PPID claim value is specified in Section 7.6.

Compatibility Note: Some existing Identity Selectors omit listing the PPID claim as an `ic:SupportedClaimType` from the `ic:SupportedClaimTypeList` when saving a self-issued Information Card in the Information Cards Transfer Format defined in Section 6.1, even though the PPID claim is supported by the card. This behavior is deprecated, as all supported claims should be listed. Nonetheless, Identity Selectors may choose to recognize this case and support the PPID claim for self-issued cards not explicitly listing this claim.

7.5.15 Web Page

URI: *http://schemas.xmlsoap.org/ws/2005/05/identity/claims/webpage*

Type: *xs:string*

Definition: The Web page of a Subject expressed as a URL.

7.6 The PPID Claim

The PPID claim for a Subject user represents a unique identifier for that user at a given Relying Party that is different from all identifiers for that user at any other Relying Party. In other words, the PPID is a pair-wise unique identifier for a given user identity and Relying Party combination. Since an Information Card represents a specific user identity and a Relying Party is the organization behind a Web service or site that the user interacts with, the PPID claim is logically a function of an Information Card and the organizational identity of the Relying Party.

This section describes the processing rules that SHOULD be used by a SIP to derive a PPID claim value for a combination of an Information Card and a Relying Party where it is used.

7.6.1 Relying Party Identifier and Relying Party PPID Seed

In order to derive the PPID and Signing Key as functions of the RP's organizational identity, a stable and unique identifier for the RP, called the *RP Identifier*, is needed. In the Information Card Model, the identity of a Relying Party (RP) possessing an X.509v3 certificate is presented in the form of that certificate. Therefore the organizational identity of the RP is obtained by applying a series of transformations to the identity information carried in the X.509 certificate. (See Section 8 for the specification of how to compute these values for Relying Parties not possessing a certificate.)

As specified in [RFC 2459], the subject field inside an X.509 certificate identifies the entity associated with the public key stored in the subject public key field. Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN). The DN MUST be unique for each subject entity certified by the one CA as defined by the issuer name field.

The subject field contains a DN of the form shown below:

CN=*string*, [OU=*string*, ...,] O=*string*, L=*string*, S=*string*, C=*string*

For an end-entity certificate, the values of the attribute types O (organizationName), L (localityName), S (stateOrProvinceName) and C (countryName) together uniquely identify the organization to which the end-entity identified by the certificate belongs. These attribute types are collectively referred to as the *organizational identifier attributes* here. The *RP Identifier* is constructed using these organizational identifier attributes as described below.

The *RP Identifier* value is used as an input to the Signing Key computation. A closely related value called the Relying Party PPID Seed is also computed, which is used as an input to the PPID claim and Client Pseudonym PPID computations. In many cases these are the same but in one case they differ.

There are four cases of how the *RP Identifier* and *RP PPID Seed* are constructed depending on which organizational identifier attributes the RP's certificate contains, if it is an extended validation (EV) certificate [EV Cert] with respect to the organizational identifier attributes, and if it chains to a trusted root certificate.

Case 1: RP's certificate *is* EV for organizational identifier attributes and chains to a trusted root certificate authority

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, of the following form:

  |O="*string*"|L="*string*"|S="*string*"|C="*string*"|

  The vertical bar character (ASCII 0x7C) is used as a delimiter at the start and end of the string as well as between the attribute types. Further, the string values of the individual attribute types are enclosed within double quote characters (ASCII 0x22). If an attribute type is absent in the subject field of the end-entity certificate, then the corresponding string value is the empty string (""). Following is an example *OrgIdString* per this convention.

  |O="Microsoft"|L="Redmond"|S="Washington"|C="US"|

- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using Unicode encoding (UTF-16LE with no byte order mark).
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

  RP PPID Seed = RP Identifier = SHA256 (OrgIdBytes)

Case 2: RP's certificate *is not* EV for organizational identifier attributes, has a non-empty Organization (O) value, and chains to a trusted root certificate authority

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, in the same manner as employed for Case 1 above.
- Let *QualifierString* be the string:

  |Non-EV

- Let *QualifiedOrgIdString* be the concatenation of *QualifierString* and *OrgIdString*.

  QualifiedOrgIdString = QualifierString + OrgIdString

- Encode all the characters in *QualifiedOrgIdString* into a sequence of bytes, call it *QualifiedOrgIdBytes*, using Unicode encoding (UTF-16LE with no byte order mark).

- Hash *QualifiedOrgIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier*.

  RP Identifier = SHA256 (QualifiedOrgIdBytes)

- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using Unicode encoding (UTF-16LE with no byte order mark).
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as the *Relying Party PPID Seed*.

  RP PPID Seed = SHA256 (OrgIdBytes)

<u>Case 3</u>: RP's certificate has an empty or no Organization (O) value and has an empty or no Common Name (CN) or does not chain to a trusted root certificate authority

- Take the subject public key in the end-entity certificate, call it *PublicKey*, as a byte array.
- Hash *PublicKey* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

  RP PPID Seed = RP Identifier = SHA256 (PublicKey)

<u>Case 4</u>: RP's certificate has an empty or no Organization (O) value but has a non-empty Common Name (CN) value and chains to a trusted root certificate authority

- Convert the Common Name attribute value in the end-entity certificate into a string, call it *CnIdString*, of the following form:

  |CN="*string*"|

  Following is an example *CnIdString* per this convention:

  |CN="login.live.com"|

- Encode all the characters in *CnIdString* into a sequence of bytes, call it *CnIdBytes*, using Unicode encoding (UTF-16LE with no byte order mark).
- Hash *CnIdBytes* using the SHA256 hash function, and use the resulting value as the *RP Identifier* and *RP PPID Seed*.

  RP PPID Seed = RP Identifier = SHA256 (CnIdBytes)

7.6.2 PPID

The PPID value SHOULD be produced as follows using the card identifier and the *RP PPID Seed* (specified in Section 7.6.1):

- Encode the value of the `ic:CardId` element of the Information Card into a sequence of bytes, call it *CardIdBytes*, using Unicode encoding.
- Hash *CardIdBytes* using the SHA256 hash function to obtain the canonical card identifier *CanonicalCardId*.

  CanonicalCardId = SHA256 (CardIdBytes)

- Hash the concatenation of *RP PPID Seed* and *CanonicalCardId* using the SHA256 hash function to obtain the PPID.

  PPID = SHA256 (RP PPID Seed + CanonicalCardId)

7.6.3 Friendly Identifier

The PPID provides an RP-specific identifier for a Subject that is suitable for programmatic processing, but is not a user-friendly identifier. The simple transformation rules specified in this section MAY be used by a SIP, or any other Identity Provider supporting the PPID claim, to create a friendly identifier for use within a Display Token accompanying a Security Token carrying the PPID claim.

The Friendly Identifier has the following characteristics:

- It is encoded as a 10-character alphanumeric string of the form “AAA-AAAA-AAA” grouped into three groups separated by the ‘hyphen’ character (*e.g.*, the string “6QR-97A4-WR5”). Note that the hyphens are used for punctuation only.
- The encoding alphabet does NOT use the numbers ‘0’ and ‘1’, and the letters ‘O’ and ‘I’ to avoid confusion stemming from the similar glyphs used for these numbers and characters. This leaves 8 digits and 24 letters – a total of 32 alphanumeric symbols – as the alphabet for the encoding.

The processing rules used for deriving a Friendly Identifier from a PPID are as follows:

- The PPID value is conveyed as a base64 encoded string inside tokens. Start with the base64 decoded PPID value as input.
- Hash the PPID value using the SHA1 hash function to obtain a hashed identifier.

  HashId = SHA1 (PPID)
- Let the Friendly Identifier be the string “$A_0\ A_1\ A_2– A_3\ A_4\ A_5\ A_6– A_7\ A_8\ A_9$” where each A_i is an alphanumeric character from the encoding alphabet described above.
- For *i := 0 to 9*, each A_i is determined as below:
  - Take the i^{th} octet of *HashId* (denoted as *HashId[i]*)
  - Find *RawValue* = *HashId[i] % 32* (where % is the remainder operation)
  - A_i = *EncodedSymbol* obtained by mapping *RawValue* to *EncodedSymbol* using the table below

Raw Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded Symbol	Q	L	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Raw Value	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Encoded Symbol	G	H	J	K	M	N	P	R	S	T	U	V	W	X	Y	Z

8 Relying Parties without Certificates

While Relying Parties are typically identified by presenting a cryptographically protected identity, such as an X.509v3 certificate, the Information Card Model is also applicable in situations in which no Relying Party certificate is available. This section specifies how Information Cards are used at Relying Parties with no certificate: specifically, Web sites using the [HTTP] scheme. Also see `ic07:RequireStrongRecipientIdentity` in Section 3.1.1.7 for a means whereby card issuers can prohibit the use of cards at Relying Parties not identified by a certificate.

8.1 Relying Party Identifier and Relying Party PPID Seed

The Relying Party Identifier and Relying Party PPID Seed values for Relying Parties without certificates are computed in this manner:

- Set the string *OrgIdString* to be the fully qualified DNS host name in lowercase characters specified in the URI of the Relying Party, or if a numeric IP address was used, then the canonical string representation of the IP address of the server.
- Encode all the characters in *OrgIdString* into a sequence of bytes, call it *OrgIdBytes*, using the Unicode encoding UTF-16LE with no byte order mark.
- Hash *OrgIdBytes* using the SHA256 hash function, and use the resulting value as both the *RP Identifier* and the *RP PPID Seed*.

The *RP Identifier* and *RP PPID Seed* are then used in the same manner as for Relying Parties identified by certificates when computing PPID claim and Client Pseudonym PPID values.

8.2 AppliesTo Information

Under the circumstances described in Section 3.3.3 that the RP endpoint to which the token will be sent is supplied as the `wsp:AppliesTo` value to the IP, when the RP possesses no certificate, the URL of the RP is supplied as that `wsp:AppliesTo` value.

Example:

```
<wst:RequestSecurityToken>
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:Address>http://login.contoso.com</wsa:Address>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  ...
</wst:RequestSecurityToken>
```

8.3 Token Signing and Encryption

When the Relying Party is not identified by a certificate, tokens sent from the Self-issued Identity Provider are not encrypted, although they are still signed in the manner described in Section 7.2. Tokens generated by Identity Providers for Relying Parties not identified by a certificate are also typically not encrypted, as no encryption key is available. However, the token may still be encrypted if the Identity Provider has a pre-existing relationship with the Relying Party and they have mutually agreed on the use of a known encryption key. The token should still typically be signed, even when not encrypted.

9 Using WS-SecurityPolicy 1.2 and WS-Trust 1.3

Software implementing the Information Card Model SHOULD utilize the OASIS standard versions of WS-SecurityPolicy and WS-Trust – [WS-SecurityPolicy 1.2] and [WS-Trust 1.3] and MAY utilize the previous draft versions – [WS-SecurityPolicy 1.1] and [WS-Trust 1.2]. This section describes the differences between the old and standard versions of these protocols that may affect software implementing the Information Card Model.

9.1 Overview of Differences

The following changes between the protocol versions affect software implementing this specification:

- **Namespace changes:**
  http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702 replaces
  http://schemas.xmlsoap.org/ws/2005/07/securitypolicy.
  http://docs.oasis-open.org/ws-sx/ws-trust/200512 replaces
  http://schemas.xmlsoap.org/ws/2005/02/trust.
- **Use of RequestSecurityTokenResponseCollection:** A `wst:RequestSecurityTokenResponseCollection` element encloses the `wst:RequestSecurityTokenResponse` when WS-Trust 1.3 is used.

- **Use of SecondaryParameters:** An Identity Selector sends some information received from the Relying Party to the Identity Provider in a `wst:SecondaryParameters` element.
- **Bearer Token Request Syntax:** The new `wst:KeyType` value http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer is used to request a bearer token.

9.2 Identity Selector Differences

Identity Selectors MUST determine the WS-Trust versions used by Identity Provider STSs and Relying Party STSs using their Security Policy.

Identity Selectors supporting WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType may be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

When constructing an RST for an Identity Provider using WS-Trust 1.3, the Identity Selector SHOULD send parameters received from the Relying Party in a `wst13:SecondaryParameters` element within the `wst13:RequestSecurityToken`, with these exceptions:

- The user chooses not to send optional claims. In this scenario, no SecondaryParameters element is sent in order to hide this user decision.
- No `wsp:AppliesTo` is being sent in the RST. In this scenario, no `wst13:SecondaryParameters` element is sent so that the Identity Provider does not obtain any identifying information about the Relying Party.

Example:

```
<wst13:RequestSecurityToken Context="ProcessRequestSecurityToken">
  <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst13:RequestType>
  <wsid:InformationCardReference
xmlns:wsid="http://schemas.xmlsoap.org/ws/2005/05/identity">
  ...
  </wsid:InformationCardReference>
  <wst13:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
  ...
  </wst13:Claims>
  <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/SymmetricKey</wst13:KeyType>
  <wst13:SecondaryParameters>
    <wst13:RequestType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue</wst13:RequestType>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:KeyType>http://docs.oasis-open.org/ws-sx/ws-
trust/200512/SymmetricKey</wst13:KeyType>
    <wst13:KeyWrapAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-oaep-
mgf1p</wst13:KeyWrapAlgorithm>
    ...
  </wst13:SecondaryParameters>
</wst13:RequestSecurityToken>
```

The `wst13:RequestSecurityTokenResponse` constructed must be enclosed within a `wst13:RequestSecurityTokenResponseCollection` element.

Example:

```
<wst13:RequestSecurityTokenResponseCollection>
  <wst13:RequestSecurityTokenResponse>
    <wst13:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst13:TokenType>
    <wst13:RequestedSecurityToken> ... </wst13:RequestedSecurityToken>
    ...
  </wst13:RequestSecurityTokenResponse>
</wst13:RequestSecurityTokenResponseCollection>
```

9.3 Security Token Service Differences

To utilize WS-Trust 1.3, an Identity Provider STS and Relying Party STSs MUST express their Security Policy using WS-SecurityPolicy 1.2.

STSs using WS-Trust 1.3 MUST understand the new WS-Trust 1.3 elements and syntax such as `wst13:RequestSecurityTokenResponseCollection` and new URIs such as http://docs.oasis-open.org/ws-sx/wstrust/200512/Bearer. They MUST also understand that typical properties of an RST like Claims and KeyType may be either a direct child of the top level `wst13:RequestSecurityToken` element or contained within a `wst13:SecondaryParameters` element in the RST.

10 Browser Behavior with Information Cards

This section explains the steps that a Web browser takes when using an Information Card to authenticate to a Web site. Two cases are described. The basic case is where the Web site provides all the Relying Party functionality via HTML extensions transported over HTTPS. The second case is where the Relying Party employs a Relying Party Security Token Service (STS), which it references via HTML extensions transported over HTTPS.

10.1 Basic Protocol Flow when using an Information Card at a Web Site

This section explains the protocol flow when using an Information Card to authenticate at a Web site where no Relying Party STS is employed.

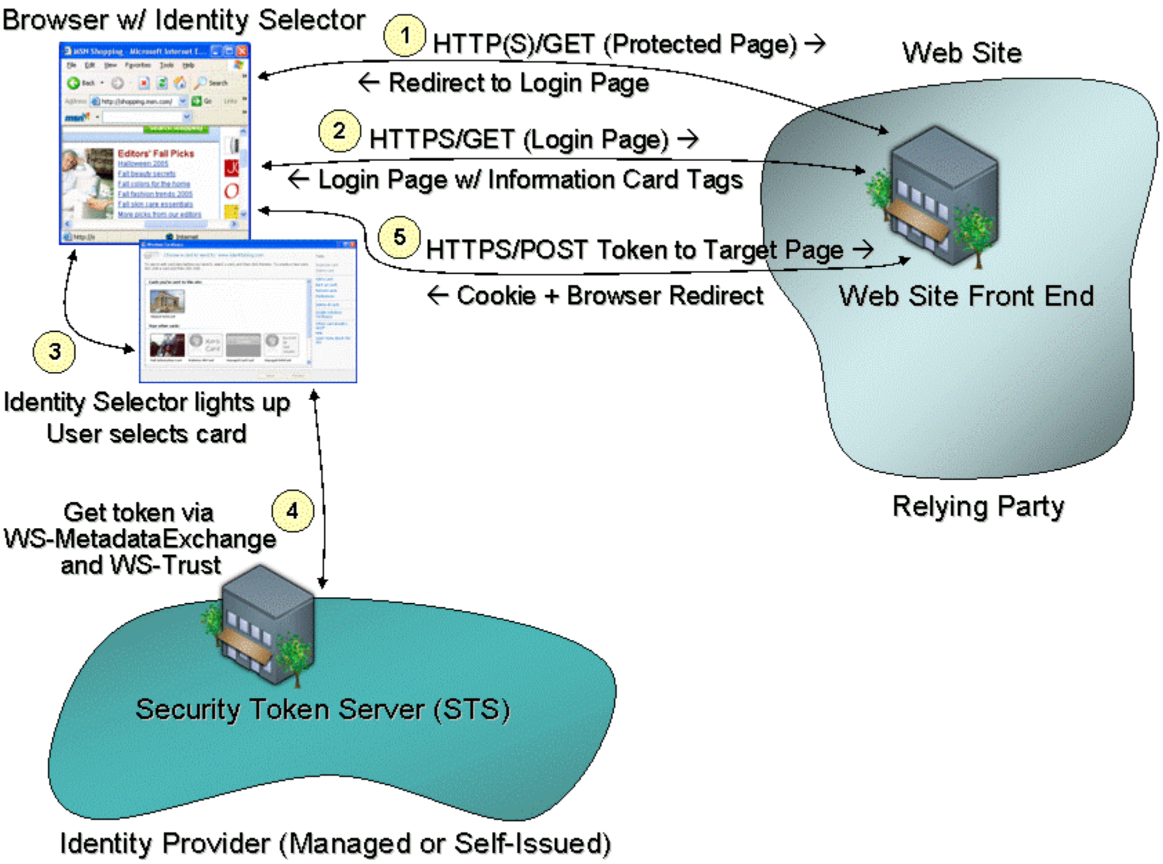
Browser w/ Identity Selector
1
HTTP(S)/GET (Protected Page) →
← Redirect to Login Page
Web Site
2
HTTPS/GET (Login Page) →
← Login Page w/ Information Card Tags
5
HTTPS/POST Token to Target Page →
← Cookie + Browser Redirect
Web Site Front End
3
Identity Selector lights up
User selects card
Relying Party
Get token via
WS-MetadataExchange
and WS-Trust
4
Security Token Server (STS)
Identity Provider (Managed or Self-Issued)

Figure 1. Basic protocol flow when using an Information Card to authenticate at a Web site

Figure 1 gives an example of the basic protocol flow when an Information Card is used to authenticate at a Web site that employs no Relying Party STS. Steps 1, 2, and 5 are essentially the same as a typical forms-based login today: (1) The user navigates to a protected page that requires authentication. (2) The site redirects the browser to a login page, which presents a Web form. (5) The browser posts the Web form that includes the login credentials supplied by the user back to the login page. The site then validates the contents of the form including the user credentials, typically writes a client-side browser cookie to the client for the protected page domain, and redirects the browser back to the protected page.

The key difference between this scenario and today’s site login scenarios is that the login page returned to the browser in step (2) contains an HTML tag that allows the user to choose to use an Information Card to authenticate to the site. When the user selects this tag, the browser invokes an Identity Selector, which implements the Information Card user experience and protocols, and triggers steps (3) through (5).

In Step (3), the browser Information Card support code invokes the Identity Selector, passing it parameter values supplied by the Information Card HTML tag supplied by the site in Step (2). The user then uses the Identity Selector to choose an Information Card, which represents a Digital Identity that can be used to authenticate at that site. Step (4) retrieves a Security Token that represents the Digital Identity selected by the user from the STS at the Identity Provider for that identity.

In Step (5), the browser posts the token obtained back to the Web site using a HTTPS/POST. The Web site validates the token, completing the user’s Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

It is worth noting that this cookie is likely to be *exactly the same cookie* as the site would have written back had the user authenticated via other means, such as a forms-based login using username/password. This is one of the ways that the goal of “minimal impact on Web sites” is achieved. Other than its authentication subsystem, the bulk of a Web site’s code can remain completely unaware that Information Card-based authentication is even utilized. It just uses the same kinds of cookies as always.

10.2 Protocol Flow with Relying Party STS

In the previous scenario, the Web site communicated with the client Identity Selector using only the HTML extensions enabling Information Card use, transported over the normal browser HTTPS channel. In this scenario, the Web site also employs a Relying Party STS to do part of the work of authenticating the user, passing the result of that authentication on to the login page via HTTPS POST.

There are several reasons that a site might factor its solution this way. One is that the same Relying Party STS can be used to do the authentication work for both browser-based applications and smart client applications that are using Web services. Second, it allows the bulk of the authentication work to be done on servers dedicated to this purpose, rather than on the Web site front-end servers. Finally, this means that the front-end servers can accept site-specific tokens, rather than the potentially more general or more complicated authentication tokens issued by the Identity Providers.

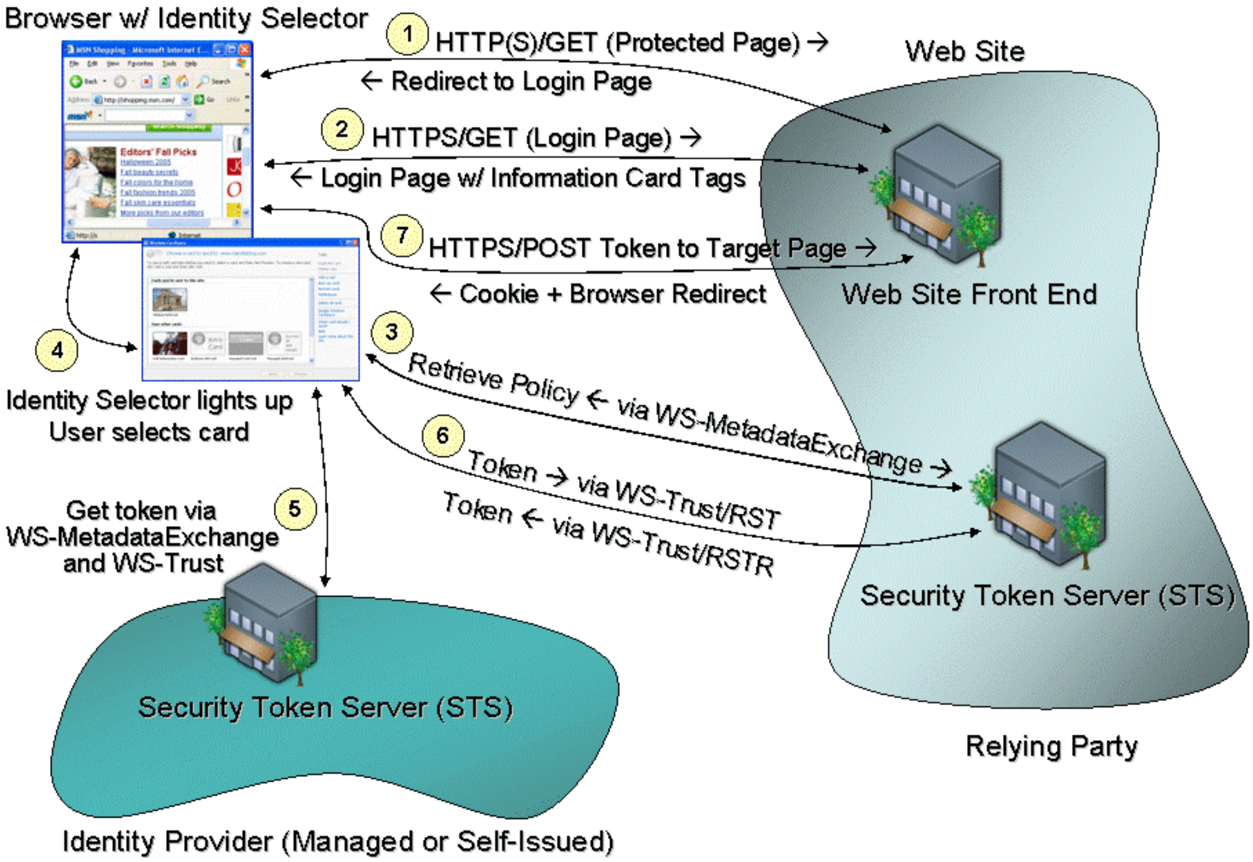
Browser w/ Identity Selector
1
HTTP(S)/GET (Protected Page) →
← Redirect to Login Page
Web Site
2
HTTPS/GET (Login Page) →
← Login Page w/ Information Card Tags
7
HTTPS/POST Token to Target Page →
← Cookie + Browser Redirect
Web Site Front End
4
3
Retrieve Policy ← via WS-MetadataExchange →
Identity Selector lights up
User selects card
6
Token → via WS-Trust/RST
Token ← via WS-Trust/RSTR
Get token via
WS-MetadataExchange
and WS-Trust
5
Security Token Server (STS)
Security Token Server (STS)
Relying Party
Identity Provider (Managed or Self-Issued)

Figure 2. Protocol flow when using an Information Card to authenticate at a Web site, where the Web site employs a Relying Party STS

This scenario is similar to the previous one, with the addition of steps (3) and (6). The differences start with the Information Card information supplied to the browser by the Web site in Step (2). In the previous scenario, the site encoded its WS-SecurityPolicy information using Information Card HTML extensions and supplied them to the Information Card-extended browser directly. In this scenario, the site uses different Information Card HTML extensions in the Step (2) reply to specify which Relying Party STS should be contacted to obtain the WS-SecurityPolicy information.

In Step (3), the Identity Selector contacts the Relying Party STS specified by the Web site and obtains its WS-SecurityPolicy information via WS-MetadataExchange. In Step (4) the Identity Selector user interface is shown and the user selects an Information Card, which represents a Digital Identity to use at the site. In Step (5), the Identity Provider is contacted to obtain a Security Token for the selected Digital Identity. In Step (6), the Security Token is sent to the Web site's Relying Party STS to authenticate the user and a site-specific authentication token is returned to the Identity Selector. Finally, in Step (7), the browser posts the token obtained in Step (6) back to the Web site using HTTPS/POST. The Web site validates the token, completing the user's Information Card-based authentication to the Web site. Following authentication, the Web site typically then writes a client-side browser cookie and redirects the browser back to the protected page.

10.3 User Perspective and Examples

The Information Card user experience at Web sites is intended to be intuitive and natural enough that users' perspective on it will simply be "That's how you log in". Today, Web sites that require authentication typically ask the user to supply a username and password at login time. With Information Cards, they instead ask users to choose an Information Card. Some sites will choose to accept only

Information Cards whereas others will give users the choice of Information Cards or other forms of authentication.

A site that accepts Information Cards typically has a login screen that contains button with a label such as "**Sign in with an Information Card**" or "**Log in using an Information Card**". Upon clicking this button, the user is presented with a choice of his Information Cards that are accepted at the site, and is asked to choose one. Once a card is selected and submitted to the site, the user is logged in and continues using the site, just as they would after submitting a username and password to a site.

Sites that accept both Information Cards and other forms of authentication present users with both an Information Card login choice and whatever other choices the site supports. For instance, a site login screen might display both "**Sign in with your username and password**" and "**Sign in with an Information Card**" buttons.

10.4 Browser Perspective

Very little additional support is required from today's Web browsers to also support Information Cards. The main addition is that they must recognize special HTML and/or XHTML tags for invoking the Identity Selector, pass encoded parameters on to the Identity Selector on the platform, and POST back the token resulting from the user's choice of an Information Card.

10.5 Web Site Perspective

Web sites that employ Information Card-based authentication must support two new pieces of functionality: adding HTML or XHTML tags to their login page to request an Information Card-based login and code to log the user into the site using the POSTed credentials. In response to the Information Card-based login, the Web site typically writes the same client-side browser cookie that it would have if the login had occurred via username/password authentication or other mechanisms, and issue the same browser redirects. Thus, other than the code directly involved with user authentication, the bulk of a Web site can remain unchanged and oblivious to the site's acceptance of Information Cards as a means of authentication.

11 Invoking an Identity Selector from a Web Page

11.1 Syntax Alternatives: OBJECT and XHTML tags

HTML extensions are used to signal to the browser when to invoke the Identity Selector. However, not all HTML extensions are supported by all browsers, and some commonly supported HTML extensions are disabled in browser high security configurations. For example, while the OBJECT tag is widely supported, it is also disabled by high security settings on some browsers, including Internet Explorer.

An alternative is to use an XHTML syntax that is not disabled by changing browser security settings. However, not all browsers provide full support for XHTML.

To address this situation, two HTML extension formats are specified. Browsers may support one or both of the extension formats.

11.1.1 OBJECT Syntax Examples

An example of the OBJECT syntax is as follows:

```
<html>
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg'/>
    <form name="ctl00" id="ctl00" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <center>
        <img src='infocard_56x39.png' onClick='ctl00.submit()'/>
        <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
      </center>
      <OBJECT type="application/x-informationCard" name="xmlToken">
        <PARAM Name="tokenType" Value="urn:oasis:names:tc:SAML:1.0:assertion">
        <PARAM Name="issuer" Value=
            "http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self">
        <PARAM Name="requiredClaims" Value=
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname">
      </OBJECT>
    </form>
  </body>
</html>
```

This is an example of a page that requests that the user log in using an Information Card. The key portion of this page is the OBJECT of type “`application/x-informationCard`”. Once a card is selected by the user, the resulting Security Token is included in the resulting POST as the xmlToken value of the form. Appendix A shows a sample POST resulting from using a login page similar to the preceding one. If the user cancels the authentication request, the resulting POST contains an empty xmlToken value.

Parameters of the Information Card OBJECT are used to encode the required WS-SecurityPolicy information in HTML. In this example, the Relying Party is requesting a SAML 1.0 token from a Self-issued Identity Provider, supplying the required claims “`emailaddress`”, “`givenname`”, and “`surname`”. This example uses the basic protocol described in Section 2.1 (without employing a Relying Party STS).

A second example of the OBJECT syntax is as follows:

```
<html>
```

```
  <body>
    <form name="ctl01" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser-STS/login.aspx"
        id="ctl01" onSubmit="fnGetCard();">
      <img src='infocard_56x39.png' onClick='ctl01.submit()'/>
      <input type="submit" name="InfoCardSignin" value="Log in"
          id="InfoCardSignin" />
      <OBJECT type="application/x-informationCard" name="xmlToken"
          ID="oCard" />
    </form>
    <script type="text/javascript">
    <!--
      function fnGetCard(){
        oCard.issuer = "http://www.fabrikam.com/sts";
        oCard.issuerPolicy = "https://www.fabrikam.com/sts/mex";
        oCard.tokenType = "urn:fabricam:custom-token-type";
      }
    //-->
    </script>
  </body>
</html>
```

This example uses the enhanced protocol described in Section 2.3, which employs a Relying Party STS. Note that in this case, the “issuer” points to a Relying Party STS. The “issuerPolicy” points to an endpoint where the Security Policy of the STS (expressed via WS-SecurityPolicy) is to be obtained using WS-MetadataExchange. Also, note that the “tokenType” parameter requests a custom token type defined by the site for its own purposes. The “tokenType” parameter could have been omitted as well, provided that the Web site is capable of understanding all token types issued by the specified STS or if the STS has prior knowledge about the token type to issue for the Web site.

The object parameters can be set in normal script code. This is equivalent to setting them using the “PARAM” declarations in the previous example.

11.1.2 XHTML Syntax Example

An example of the XHTML syntax is as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity">
  <head>
    <title>Welcome to Fabrikam</title>
  </head>
  <body>
    <img src='fabrikam.jpg'/>
    <form name="ctl00" id="ctl00" method="post"
        action="https://www.fabrikam.com/InfoCard-Browser/Main.aspx">
      <ic:informationCard name='xmlToken'
          style='behavior:url(#default#informationCard)'
          issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
          tokenType="urn:oasis:names:tc:SAML:1.0:assertion">
        <ic:add claimType=
       "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
            optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname"
            optional="false" />
        <ic:add claimType=
         "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname"
            optional="false" />
      </ic:informationCard>
      <center>
        <input type="submit" name="InfoCardSignin" value="Log in"
            id="InfoCardSignin" />
      </center>
    </form>
  </body>
</html>
```

11.2 Identity Selector Invocation Parameters

The parameters to the OBJECT and XHTML Information Card objects are used to encode information in HTML that is otherwise supplied as WS-SecurityPolicy information via WS-MetadataExchange when an Identity Selector is used in a Web services context.

11.2.1 issuer (optional)

This parameter specifies the URL of the STS from which to obtain a token. If omitted, no specific STS is requested. The special value "`http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self`" specifies that the token should come from a Self-issued Identity Provider.

11.2.2 issuerPolicy (optional)

This parameter specifies the URL of an endpoint from which the STS's WS-SecurityPolicy can be retrieved using WS-MetadataExchange. This endpoint must use HTTPS.

11.2.3 tokenType (optional)

This parameter specifies the type of the token to be requested from the STS as a URI. This parameter can be omitted if the STS and the Web site front-end have a mutual understanding about what token type will be provided or if the Web site is willing to accept any token type.

11.2.4 requiredClaims (optional)

This parameter specifies the types of claims that must be supplied by the identity. If omitted, there are no required claims. The value of `requiredClaims` is a space-separated list of URIs, each specifying a required claim type.

11.2.5 optionalClaims (optional)

This parameter specifies the types of optional claims that may be supplied by the identity. If omitted, there are no optional claims. The value of `optionalClaims` is a space-separated list of URIs, each specifying a claim type that can be optionally submitted.

11.2.6 privacyUrl (optional)

This parameter specifies the URL of the human-readable Privacy Policy of the site, if provided.

11.2.7 privacyVersion (optional)

This parameter specifies the Privacy Policy version. This must be a value greater than 0 if a privacyUrl is specified. If this value changes, the UI notifies the user and allows them review the change to the Privacy Policy.

11.3 Data Types for Use with Scripting

The object used in the Information Card HTML extensions has the following type signature, allowing it to be used by normal scripting code:

```
interface IInformationCardSigninHelper
{
  string issuer;              // URI specifying token issuer
  string issuerPolicy;        // MetadataExchange endpoint of issuer
```

```
    string tokenType;           // URI specifying type of token to be requested
    string [] requiredClaims;   // Array of required claims
    string [] optionalClaims;   // Array of optional claims
    string privacyUrl;          // URL of the Privacy Policy of the site
    string privacyVersion;      // Version number of the Privacy Policy
    boolean isInstalled;        // True when an Identity Selector is available
                                // to the browser
  }
```

11.4 Detecting and Utilizing an Information Card-enabled Browser

Web sites may choose to detect browser and Identity Selector support for Information Cards and modify their login page contents depending upon whether Information Card support is present, and which of the OBJECT and/or XHTML syntaxes are supported by the browser and supported by the Web site. This allows Information Card capabilities to be shown when available to the user, and to be not displayed otherwise.

Detecting an Information Card-enabled browser may require detecting specific browser and Identity Selector versions and being aware of the nature of their Information Card support.

11.5 Behavior within Frames

When the object tag is specified in an embedded frame, the certificate of the frame is compared to that of the root frame. For this configuration to work, the scheme, domain, and security zone (for example https, microsoft.com, and Intranet) of the URL of the embedded frame must be the same as that of the root frame. If they do not match, the object tag should not be acted upon. This prevents a form of cross-site scripting attacks.

11.6 Invocation Using the Document Object Model (DOM)

In addition to being invokable using static HTML tags and script code, Identity Selectors can be invoked from script injected into the page using the Document Object Model [DOM]. Invocation from dynamically generated script allows the Web site's requirements to be set dynamically.

11.7 Auditing, Non-Auditing, and Auditing-Optional Cards

- **Auditing Card:** When a managed card with an `ic:RequireAppliesTo` element and no `Optional` attribute or `Optional=false` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.
- **Non-Auditing Card:** When a managed card with no `ic:RequireAppliesTo` element is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains no `wsp:AppliesTo` element.
- **Auditing-Optional Card:** When a managed card with an `ic:RequireAppliesTo` element with `Optional=true` attribute is used at a Web site, the Request Security Token (RST) sent to the Identity Provider contains a `wsp:AppliesTo` element.

12 Endpoint Reference wsid:Identity Property

This section adds the `wsid:Identity` property to an Endpoint Reference [WS-Addressing] and leverages extensibility of the `wsa:EndpointReferenceType` schema to include a `wsid:Identity` element as described below:

```
<wsa:EndpointReference>
  ...
  <wsid:Identity>...Claim...</wsid:Identity>
  ...
</wsa:EndpointReference>
```

The Identity element inside `wsa:EndpointReference` can hold any of the claims defined in Section 12.2 below.

12.1 Default Value

If an EndpointReference does not contain an Identity element, a DNS Name claim can be assumed by extracting the hostname from the Address URI.

If the URI does not have a hostname, it does not have an implicit identity value and can not be verified by the mechanisms defined in this document.

12.2 Identity Representation

12.2.1 DNS Name

The DNS Name claim implies that the remote principal is trusted to speak for that DNS name. For instance the DNS Name claim could specify “fabrikam.com”. When challenged, the endpoint contacted must be able to prove its right to speak for “fabrikam.com”. The service could prove its right by proving ownership of a certificate containing a reference to fabrikam.com and signed by a trusted Certificate Authority (e.g. VeriSign). The following element of type xs:string can be used to represent a DNS Name claim within a `wsid:Identity` element.

```
<wsid:Dns>fabrikam.com</wsid:Dns>
```

12.2.2 Service Principal Name

The SPN claim implies that the remote principal is trusted to speak for that SPN, a mechanism common in intranet domains. Its format is <serviceClass>/<host>. For example, the SPN for a generic service running on “server1.fabrikam.com” would be “host/server1.fabrikam.com”. The client could confidentially speak to the service and verify replies back from the service by obtaining a Kerberos ticket from the realm’s domain controller. The following element of type xs:string can be used to represent an SPN claim within a `wsid:Identity` element.

```
<wsid:Spn>host/hrweb</wsid:Spn>
```

12.2.3 User Principal Name

The UPN claim implies that the remote principal is a particular user in a domain. Its format is: <user>@<domain>. For example, the UPN for a user “someone” at a domain “example.com” would be “someone@example.com”. A service could prove its UPN by providing the password for the user associated with “someone@example.com”. The following element of type xs:string can be used to represent a UPN claim within a `wsid:Identity` element.

```
<wsid:Upn>someone@example.com</wsid:Upn>
```

12.2.4 KeyInfo

This identity value is similar to the previous three, but rather than describing an attribute of the target, this mechanism describes a reference (embedded or external) to key material associated with the target. This allows confirmation of the target trust identity through encryption. These values can also be used to compare authenticated identities similar to the basic trust identity values by comparing the hash of the specified trust identity value with a hash of the authenticated identity of the service. The ds:KeyInfo element defined in [XML Signature] can be used.

```
<ds:KeyInfo>...</ds:KeyInfo>
```

12.2.4.1 Example specifying an RSA Public Key

The PublicKey claim states the public key of the remote principal. A service could prove its ownership of the key by signing some data with the private key.

```
<wsid:Identity>
  <ds:KeyInfo>
    <ds:RSAKeyValue>
      <ds:Modulus>xA7SEU+e0yQH5...</ds:Modulus>
      <ds:Exponent>AQAB</ds:Exponent>
    </ds:RSAKeyValue>
  </ds:KeyInfo>
</wsid:Identity>
```

12.2.4.2 Example specifying an X509 Certificate

This example shows a certificate of the remote principal being used as the identity value.

```
<wsid:Identity>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate>MIICXTCCA...</ds:X509Certificate>
    </ds:X509Data>
  </ds:KeyInfo>
</wsid:Identity>
```

12.2.5 Security Token

A security token can be an identity value representing claims about the identity of an endpoint. E.g.:

```
<wsid:Identity>
      <wsse:BinarySecurityToken ValueType="...#X509v3">
            <!--base64 encoded value of the X509 certificate-->
      </wsse:BinarySecurityToken>
</wsid:Identity>
```

12.2.6 Security Token Reference

Similarly to ds:KeyInfo, wsse:SecurityTokenReference element can be used within wsid:Identity element to reference a token representing collection of claims about the identity of an endpoint. E.g.:

```
<wsid:Identity>
      <wsse:SecurityTokenReference>
           <wsse:KeyIdentifier ValueType="...#ThumbprintSHA1">
                <!-- thumbprint of the X509 certificate  -->
           </wsse:KeyIdentifier>
      </wsse:SecurityTokenReference>
</wsid:Identity>
```

13 Security Considerations

13.1 Protection of Information Cards by Identity Selectors

It is recommended that Identity Selectors encrypt or otherwise secure the Information Card data held by them to help protect cards from being stolen and then used by an attacker. This is particularly important for self-issued Information Cards, where possession of the unencrypted contents of a card could enable an attacker to gain access to Relying Parties accounts associated with that card.

13.2 Relying Parties Without Certificates

Because claims sent to relying parties without certificates are not encrypted, it is recommended that sensitive claims not be released to these relying parties. Identity Providers holding sensitive user data that can be released as claim values are encouraged to issue cards containing an `ic07:RequireStrongRecipientIdentity` element to prevent transmission of sensitive claim values over an unencrypted channel.

13.3 Endpoint References

It is recommended that Endpoint Reference elements be signed to prevent tampering.

An Endpoint Reference should not be accepted unless it is signed and have an associated security token to specify the signer has the right to “speak for” the endpoint. That is, the relying party should not use an endpoint reference unless the endpoint reference is signed and presented with sufficient credentials to pass the relying parties acceptance criteria.

It is recommended that an endpoint reference be encrypted when it contains claims and other sensitive information.

When included in a SOAP message, endpoint references are recommended to be protected using the mechanisms described in WS-Security [WS-Security]

14 Conformance

An implementation conforms to this specification if it satisfies all of the MUST or REQUIRED level requirements defined within this specification for the portions of the specification implemented by that implementation. Furthermore, when an implementation supports functionality in which there is a RECOMMENDED algorithm or set of parameter choices, conforming implementations MUST support the RECOMMENDED algorithm and parameter choices. A SOAP Node MUST NOT use the XML namespace identifiers for this specification (listed in Section 1.2) within SOAP Envelopes unless it is compliant with this specification.

This specification references a number of other specifications. In order to comply with this specification, an implementation MUST implement the portions of referenced specifications necessary to comply with the required provisions of the portions of this specification that it implements. Additionally, the implementation of the portions of the referenced specifications that are specifically cited in this specification MUST comply with the rules for those portions as established in the referenced specification.

Additionally, normative text within this specification takes precedence over normative outlines (as described in Section 1.1), which in turn take precedence over the XML Schema [XML Schema Part 1, Part 2] and WSDL [WSDL 1.1] descriptions. That is, the normative text in this specification further constrains the schemas and/or WSDL that are part of this specification; and this specification contains further constraints on the elements defined in referenced schemas.

If an OPTIONAL message is not supported, then the implementation SHOULD Fault just as it would for any other unrecognized/unsupported message. If an OPTIONAL message is supported, then the implementation MUST satisfy all of the MUST and REQUIRED sections of the message.

A. HTTPS POST Sample Contents

The contents of an HTTPS POST generated by a page like the first example in Section 4.1.1 follow:

```
POST /test/s/TokenPage.aspx HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 6478
Content-Type: application/x-www-form-urlencoded
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-sh
ockwave-flash, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Host: calebb-tst
Referer: https://localhost/test/s/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30)
UA-CPU: x86

InfoCardSignin=Log+in&xmlToken=%3Cenc%3AEncryptedData+Type%3D%22http%3A%2F%2F
www.w3.org%2F2001%2F04%2Fxmlenc%23Element%22+xmlns%3Aenc%3D%22http%3A%2F%2Fww
w.w3.org%2F2001%2F04%2Fxmlenc%23%22%3E%3Cenc%3AEncryptionMethod+Algorithm%3D%
22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23aes256-cbc%22+%2F%3E%3CKeyIn
fo+xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2F09%2Fxmldsig%23%22%3E%3Ce%3AEn
cryptedKey+xmlns%3Ae%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23%22%
3E%3Ce%3AEncryptionMethod+Algorithm%3D%22http%3A%2F%2Fwww.w3.org%2F2001%2F04%
2Fxmlenc%23rsa-oaep-mgf1p%22%3E%3CDigestMethod+Algorithm%3D%22http%3A%2F%2Fww
w.w3.org%2F2000%2F09%2Fxmldsig%23sha1%22+%2F%3E%3C%2Fe%3AEncryptionMethod%3E%
3CKeyInfo%3E%3Co%3ASecurityTokenReference+xmlns%3Ao%3D%22http%3A%2F%2Fdocs.oa
sis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-wssecurity-secext-1.0.xsd%2
2%3E%3Co%3AKeyIdentifier+ValueType%3D%22http%3A%2F%2Fdocs.oasis-open.org%2Fws
s%2Foasis-wss-soap-message-security-1.1%23ThumbprintSHA1%22+EncodingType%3D%2
2http%3A%2F%2Fdocs.oasis-open.org%2Fwss%2F2004%2F01%2Foasis-200401-wss-soap-m
essage-security-1.0%23Base64Binary%22%3E%2BPYbznDaB%2FdlhjIfqCQ458E72wA%3D%3C
%2Fo%3AKeyIdentifier%3E%3C%2Fo%3ASecurityTokenReference%3E%3C%2FKeyInfo%3E%3C
e%3ACipherData%3E%3Ce%3ACipherValue%3EEq9UhAJ8C9K5l4Mr3qmgX0XnyL1ChKs2PqMj0Sk
6snw%2FIRNtXqLzmgbj2Vd3vFA4Vx1hileSTyqc1kAsskqpqBc4bMHT61w1f0NxU10HDor0DlNVcV
Dm%2FAfLcyLqEP%2Boh05B%2B5ntVIJzL8Ro3typF0eoSm3S6UnINOHIjHaVWyg%3D%3C%2Fe%3AC
ipherValue%3E%3C%2Fe%3ACipherData%3E%3C%2Fe%3AEncryptedKey%3E%3C%2FKeyInfo%3E
%3Cenc%3ACipherData%3E%3Cenc%3ACipherValue%3ErBvpZydiyDzJtzl1%2FjUFX9XAzO1mOR
q0ypPLjh%2FBagXcfZeYwWD57v4Jvn1QwGajadcDASCisazswn1skdkwgmd4IUWJpPMRH7es9zY0U
vnS4ccsakgDcmscq3pDYTrxbSBfhdvrzjDiHC2XCtowOveoHeB51C5N8UAbff18IxCNtkWO8y3wLH
VGdvwaDOSakK%2FK%2Fv1UgXIc51%2FtYvjeFGeGbbSNxo8DTqeDnAMQ%2B4Y%2B1aUGhI%2FtbSr
EyJECkDgtztcxhrumbupKO%2BogWKUTTpSt851xjOFxAMiVaPZ%2FAm8V8H3ZLsR087sX%2FJ%2Bn
bRqze%2BfbdUwimN5pNoJDdMnF%2BEDLass1dPsvhL4EXzuIp5deGBaqAIoaOMEUW7ssuh1PtwkEM
eqwlOzOhu%2FHtwP1qh3D02U59MtyQnJMD5UwIwO7sZJl6%2BPg6Zp9HHtKKUMnkguvFmhyXS4BFS
ZVxPl18i%2B0MLO1um5dejEFd4nwGO%2FmNw6yEI8DdGVjXcYOT6JhPz9rHNh9%2F%2FOj5snJfL6
j2sg0EvIYoRs%2BhT4sdHZ95tGAiwMwT6cFOXbAQZUbYTr1ZOC6XPsfL2CFwiTM3mI%2Blco4Hc%2
F7IakIA8jwAJdtnd2mGuV67ZbY1mzibM1LUApixZj59El83ixctSQbV7iyywQ4IYN2CAq%2BCLMdl
R%2BDHfgEe8O3IVaGBDUEcd2MYimEiA7Yw3NIDrC14SbLzNvU702HpVJMeYv9q6S9xIVGApSrARsw
RFXyMbkMDp5WIQaJEXon7qLcsZONpdlX9bCcmaiikdpxmCeyS638te%2FhGBLmYJSQ0stf7BhA6E0
kwDRgdwsAa88bODiWHek0vDhAN4HlXFZ%2BCxp53L9Mmvy%2FCAOI%2B9OkPL2yxS22yjWQxom%2F
yZuawsK98JHVShsIVmmbKvRM6xJwvHDSzuBAOlQKS%2FMHcFZn8vHZR4lMhm5nL3F%2B%2BumMKh0
vMuKk6JiCqG9OEj996bVIIkLzESU5Z5vT6I1Kr9Brdx8ckDElipdH3x54WVfaItHJTYU%2BsxIR1T
25fi9k%2FOc%2FMX7Q%2B6NSDs4nGqkn4rzqpez9BUWNZw7caVOrDeao85f%2FiDCGymtl0A3JaSZ
dTKfzHLGmUfSkCAlVeisdvB6R7uBw8tR%2BZlgLIGS28wppFlnUYvSK7DnPrzId%2BGfHwLfL6WA%
2FEzBMMgppb5Vi%2BauHq%2BHxpCamlkrcUkzagbwNkGV8TfafkqUvRwJbxRwNVPI%2F%2Fxs%2Fp
Lcu1dh6eKcmU00%2FNx0zNOScd9XoeEU3zsV78PgvPIBT4EDugdv4bMR6dExXvZBl%2F84b1gOMhK
ZRplF8t6EAc4LCct01ht7VOVNz25NtP27ct9QPrDJc%2FoxihT4Df6NV3l4vlTnu%2B%2BzVB%2BH
JAxNkiO9gx3uLUJM9XEZCDzZKihaBk2y%2F3RhsJpABVneUd%2B3sCRbQXhgKYNBHZyRAUGpMDLhL
qpjoF9x%2FNvUujQ5DBLJafxxzNVshG52jRz%2BikhCNhJDDbeA5MQ8Q7QsYcKDC0DBFsewtWaA%2
```

```
FsKxl3JU6hyTotnFS%2FoS2EzbOSvn25qZuBERsZ3w%2B5WMkRzfQadyIYOSv2Df1YoljubDKy1l9
St%2FbCIBgXbVIZKYtQ%2BLyepxxFjrN7cWo2aYFnB6YLurg4USJwhXzcGcvA3%2BR5dRT6Fr37U6
OcHc%2Fz2MaZmn1cQWiDGNxHtRVxEvirBc1x47hWfSRjrKzf3orL5LzgMlYc7Iwclw2rbeWljCqOb
oV3d71ez%2FvNz1pxEMi4w8yUAQL8p%2FRCZ%2BpzvsgORu4RWKWiSwbl7AN0J3jiWShyZgDmxd2O
DDYffXjNiuH1mQWnDTkJX1ig88mqjhOYJEal0W6L0ErwrRIy29tOiAvXZANC8kA1HexulH0e38x8E
IOaVaJtNz9mqrnmnp4GdZ38txV%2BCUeWHOZaHLF4xkdtRxMAu%2FbzQ03YmUOhgxqkTfNzV6Ymne
v2nv5VsyQGJaQsNjb0M4yOe6kX2qNTwKBN2%2Bp%2Fz3f15i8KuGCgBcfP%2BP9xBizBeo7FbFtyo
2pfFhzBPmZeSOJ6kEbF1yQKHYQAT5iZ4SyTIfqqmwGxsQpWMstx3qJF8aW8WFzU1qXcC1LmgClg19
rx9NYFaQshX4f729B9Ue5MX7gTrMgwAnlXty9BsoP7nzGbr3HSXy8pR%2BimuAFW3c2NaQSbjSH5Z
FOr7PZdLHsNVJzFIsaufAwr0CAEtvlPJUt7%2B%2FE5MQsMsVqMoXFmefgdxbvY1Ue6MX1wtuJYY1
PAX7MHTyRUR3RfJDO054EoflVTwNE1fmocUXUh5rtFFuzy2T%2F2Y6pLAARXzo8uslAuH67VkuXv%
2BEMc7e3ogbf5%2BROsgJirZS6qkcYpfEUwqHiQYLnSIP4bt%2BWI5j1bxs7yzcSCkNZ2rd%2FHWr
A41AyGMfYzqxfGcrOaxHsds3JUcByB5Zw17W58GBC32Iusqa69BFTPagEapM0Fb5CbTqXnWTNNB5J
t40BVZvLv3u5oy%2BBRaMKXZhwnbT2WUTp0Ebsn17xvte52B%2BLMlSWJn96Nl5thd%2Ft1D7PlWA
sUvpJAd0UHPizCkY8VIhcXTrsSyEwer2J2I9TQTUosmssFjoP8Lx9qMfXo0eGVmneV8kVBtu4J7N1
QmWfV%2B%2FK8vGbCwW3Gm%2FEUlOO4ZbbK39y0JgNQ7fshxHr5Hdtd%2F6S%2FQkb6NPVDwn7Srh
Y0diWujXz5QlIYBSN7vDfMun3yF%2BGbmMExZ8MkOthuYkgMS9qiFoJGUXGyELsJfxbzdcRE9iyJn
p88L4%2BCtcO3l2JxIhMAgxOZx42RfAiDV1Gbpa4f%2F0urmWQ2VK7uZ%2FlViVrGAJ2kpH0EfwYE
Mb2YYT8FFjogqEpDSJX48BLIh1TE4nMbqQVG1cksCGDc0XyGKaF5Z7Ikw493Xz0JQ0BZvaf2Kceb7
MUZlsU1DSHcQQ9X%2Bxu9RcgUePJEe9BgCMpZ5Kr6r43qyk79noBSgrsSkDhT5sg%2Fc20RHQB8OX
%2BC4r3XGQFWF2m2j0xTc%2Boy14xqUmSB2qJtuWGOXDJspejDRP1GIfFnqDFdqSO3%2FkV9AC5Ee
39iJGv8I%2B5nErtQao645bCytn4B2bJah8R2fXLs8Dd4%2BC2ykxVrLxTUmJaGqd2RK%2F6t1E47
l%2B90Vp4WEzC0CFXXt9XNqdVjo2bZsXbfKQgO2zT2q2qCsgwbxVzIF5y39R%2BrkSkX16uuz3q6w
n3I5RI9M8Hn3DCzzv6Ms4rYxYuiqxaIcb7DgjI2fk1bdyiiRjSxzpCHpK6CWjBD8DPQYdkqGr%2Bs
oWeSvHvPLMSDxEPzwlnaxysRXzKphHUeUa2CCqcpagux2mbKkwHSXemX9I3V3AhPePp5XI5eCRiy3
D4%2BcBXOydie94Nz9DIhW749hPiVD9CioAgyqgAzFwCxEEUCXKTzu9xXX4DXg9b3CUfGzwERtY7x
TGT2y%2F9i7r5Xs0lrKi9ftws4JO5v%2Be3WuAEtWv0w%2FVKCl1WwTbV9xtx%2B4RZQ3%2Fewvv%
2F0GqiiSrhiVBGuCDaQs7stwqfkF3vFgGXmmODGTIkIxvYm2fzcEfq4A6LRp5RkYyJyUTF87c56tn
Qa%2Bo3xeiX5WRJybpabrRou09vyWLdlkhcUaBElGWB7iYUJ9bCltByEdNZnuDV%2FXlfnmDARKp8
RVN028czIk57wQMuizgWrM6S9Ku20noDmLgbT554UBf7FnjRWOb%2FF9OJuPpUcARBPrfuqTcOsBq
tZr7AJl3zz%2F53mpyn9rgzw5gBLgkvrdbciabJOAacccTDEB5kEzCLuprC3SlVedhgY%2BMQ5%2F
xgN%2Faf3TtJiBKFvb1V37BlbXXGosnPFcoH8I0XbqW5FSsxmcnpg48poJcB7j5eHq7Y%2F01RLb4
iMmzNap4%2BFg2F3LrwOI0Wk7ueIjgFd5KJ1iTda1ivGU%2Fchr9aTNpM5HiLb2fDW0pZ%2FFBJcI
XxpT9eNY%2FpVj5pnTW2ubpPnBulPOQTLCi1EOxbl33wnhUIfnGiVWJdrls2j3GWgqOnrYUbP%2FX
tNJqIucnMYGqPbcGIF2QRuiwD%2FiTRMvCRCmdCsYE%2FaXjOMhskX7KYC%2B9iG%2FT1wQRbfHSK
WD%2Fpv450OVDsfc1Adq6FCr1LesDNTew%2FF8Z3SiHnWS76OVsNM2SB%2FhMP67iu5UWVkb3%2FQ
qCN0aosOPs2QX0XBCZFmN6p3FhFnXPbAbaGz9y6KzUiUxC03U0fZcToKl4y%2Bw0P4IvxpjVt4t8b
84Q9hiBxd5xu1%2BRE973a%2FyIWO%2Fit1MdUSmxWakxWuGxDnQxwkNCN7ekL%2FQ%2B6FItm86b
w9cc%2FMiI7q2fK7y7YAzM3tmamhF1%2FWJNj1lH0vh%2BhNehJlLlb4Z%2F9ZtxMWV4LVTyrFaF1
zyCEqcKUTk0jc%2FXDwyKZc%2FSV9EOoPk2fVnmzs3WkA74GB%2BWtjdvQjSmnJYtPkMNsikHw%2B
RyB1hTkYbn3iQ6BUiJ0v97j7MVZHxCa1KS3t2gx8H7ts6Tfy5il89xVUdiZwfj0w06g199qlAqUMZ
EWxh0%3D%3C%2Fenc%3ACipherValue%3E%3C%2Fenc%3ACipherData%3E%3C%2Fenc%3AEncryp
tedData%3E
```

An un-escaped and reformatted version of the preceding xmlToken value, with the encrypted value elided, is as follows:

```
<enc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element" xmlns:enc=
"http://www.w3.org/2001/04/xmlenc#">
<enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"
/>
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<e:EncryptedKey xmlns:e="http://www.w3.org/2001/04/xmlenc#">
<e:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1
p">
<DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
</e:EncryptionMethod>
<KeyInfo>
<o:SecurityTokenReference xmlns:o="http://docs.oasis-open.org/wss/2004/01/oas
is-200401-wss-wssecurity-secext-1.0.xsd">
<o:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-soap-mes
sage-security-1.1#ThumbprintSHA1" EncodingType="http://docs.oasis-open.org/ws
s/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary">
+PYbznDaB/dlhjIfqCQ458E72wA=
</o:KeyIdentifier>
</o:SecurityTokenReference>
</KeyInfo>
<e:CipherData>
<e:CipherValue>
Eq9UhAJ8C9K5l4Mr3qmgX0XnyL1ChKs2PqMj0Sk6snw/IRNtXqLzmgbj2Vd3vFA4Vx1hileSTyqc1
kAsskqpqBc4bMHT61w1f0NxU10HDor0DlNVcVDm/AfLcyLqEP+oh05B+5ntVIJzL8Ro3typF0eoSm
3S6UnINOHIjHaVWyg=
</e:CipherValue>
</e:CipherData>
</e:EncryptedKey>
</KeyInfo>
<enc:CipherData>
<enc:CipherValue>
...=
</enc:CipherValue>
</enc:CipherData>
</enc:EncryptedData>
```

B. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Original Authors of the initial contributions:

Arun Nanda, Microsoft Corporation
Michael B. Jones, Microsoft Corporation
Jan Alexander, Microsoft
Giovanni Della-Libera, Microsoft
Martin Gudgin, Microsoft
Kirill Gavrylyuk, Microsoft
Tomasz Janczuk, Microsoft
Michael McIntosh, IBM
Anthony Nadalin, IBM
Bruce Rich, IBM
Doug Walter, Microsoft

Participants:

John Bradley, Individual
Norman Brickman, Mitre Corporation
Jeffrey Broberg, CA
Scott Cantor Internet2
Ruchith Fernando, WSO2
Marc Goodner, Microsoft Corporation (Chair)
Patrick Harding, Ping Identity
Andrew Hodgkinson, Novell
Mario Ivkovic, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael B. Jones, Microsoft Corporation (Editor)
Mike Kirkwood, Polka Networks
Herbert Leitold, A-SIT, Zentrum für Sichere Informationstechnologie - Austria
Michael McIntosh, IBM (Editor)
Dale Olds, Novell
Anthony Nadalin, IBM (Chair)
Drummond Reed, Cordance
Bruce Rich ,IBM
Darran Rolls, SailPoint Technologies
Prabath Siriwardena, WSO2

C. Revision History

Revision	Date	Editor	Changes Made
cd-02	2009-02-19	Michael B. Jones	Added conformance statement about RECOMMENDED algorithms. Updated IP/STS PPID calculation recommendation. Corrected `ic:IssuerId` computation description.
ed-06	2009-02-16	Michael B. Jones	Changed crypto algorithm language statements from MUST to RECOMMENDED, to potentially allow alternative algorithms to be used in the future. Usage of other algorithms is not described.
ed-05	2009-02-04	Michael B. Jones	Document capability to retrieve metadata with HTTPS GET. First drafts of complete Security Considerations and Conformance sections.
ed-04	2009-01-12	Michael B. Jones	Use OASIS-format namespace URI. Added clarifications about the use of ClientPseudonym in PPID computations, support for the `ic:IssuerId` element, and self-issued card support for the PPID claim. Added participant names.
cd-01	2008-11-10	Michael B. Jones	Created first committee draft from ed-03.
ed-03	2008-11-07	Michael B. Jones	Rationalized namespace prefixes and references. Clarified that cards have a 3:2 aspect ratio and a recommended size of 120x80.
ed-02	2008-10-24	Michael B. Jones	Rationalized content from the different input documents.
ed-01	2008-10-15	Marc Goodner	Initial conversion of input documents to OASIS format.