Identity Selector Interoperability Profile V1.0

April, 2007

Author

Arun Nanda, Microsoft Corporation

Copyright Notice

Abstract

This document is intended for developers and architects who wish to design identity systems and applications that interoperate using the Identity Selector Interoperability Profile V1.0.

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.

Table of Contents

1. Introduction

The Identity Selector Interoperability Profile V1.0 prescribes a subset of the mechanisms defined in [WS-Trust], [WS-SecurityPolicy] and [WS-MetadataExchange] to facilitate the integration of digital identity into an interoperable token issuance and consumption framework.

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

The term "relying party" (RP) means a network entity providing the desired service, and relying upon digital identity.

A "digital identity" is a set of claims made by one party about another party.

The term "identity provider" (IP) means a network entity providing the digital identity claims used by a relying party.

The term "IP/STS" refers to the security token service run by an identity provider to issue tokens.

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.

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. The information cards provide visual representations of digital identities for the end user.

This profile constrains the schema elements/extensions used by the Information Card model, and behaviors for conforming relying parties, identity providers and identity selectors.

2. Terminology and Notation

2.1. XML Namespaces

The base XML namespace URI used by the definitions in this profile is as follows:

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

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

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

Table 1 lists the XML namespaces that are used in this document. The current SOAP 1.2 namespace URI is used to provide detailed examples, not to limit the applicability of the mechanisms defined in this document to a single version of SOAP.

Table 1: Prefixes and XML namespaces used in this document

Prefix	XML Namespace	Specification(s)
S	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2 [SOAP 1.2]
xs	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]
ds	http://www.w3.org/2000/09/xmldsig#	XML Digital Signatures
ic	http://schemas.xmlsoap.org/ws/2005/05/identity	This document
wsid	http://schemas.xmlsoap.org/ws/2006/02/addressingidentity	Identity Extension for Web Services Addressing [Addressing-Ext]
wsx	http://schemas.xmlsoap.org/ws/2004/09/mex	WS-MetadataExchange [WS-MetadataExchange]
wsa	http://www.w3.org/2005/08/addressing	WS-Addressing [WS-Addressing]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-	WS-SecurityUtility

	wss-wssecurity-utility-1.0.xsd	
Wsse	http http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	WS-Security Extensions [WS-Security]
Wst	http://schemas.xmlsoap.org/ws/2005/02/trust	WS-Trust [WS-Trust]
Wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	WS-Policy [WS-Policy]
Sp	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy	WS-SecurityPolicy [WS-SecurityPolicy]

2.2. Notational Conventions

The keywords "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 [RFC 2119].

This profile uses the following syntax to describe outlines for messages and XML fragments:

- The syntax appears as an XML instance, but values in italics indicate data types instead of 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.
- An ellipsis (i.e. "...") indicates a point of extensibility that allows other child or attribute content. Additional children or attributes can be added at the indicated extension points. An identity selector MAY ignore any extensions it does not recognize.
- XML namespace prefixes (see Table 1) are used to indicate the namespace of the element being defined.

Normative text within this profile takes precedence over normative outlines, which in turn take precedence over the XML Schema descriptions.

3. 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.

3.1. 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.

3.1.1. 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>
```

3.1.2. 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>
```

3.1.3. 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]. 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 8.4.

3.2. 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.

4. 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.

4.1. 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.

4.1.1. 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> ?
  ...
</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 4.1.1.1.

/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 45 pixels high and 200 pixels wide by 150 pixels high.

/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/@{any}
This is an extensibility point to allow additional attributes to be specified.

.../ic:InformationCard/{any}
This is an extensibility point to allow additional metadata elements to be specified.

4.1.1.1. 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.

4.1.1.2. 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. 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 3.1.1 MAY be used. The `wsid:Identity` extension element [Addressing-Ext] 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 5 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.

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>
```

4.1.1.3. 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>
```

4.1.1.4. 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>
```

4.1.1.5. 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 4.3.3.

4.1.1.6. 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.

4.1.2. 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 MUST be used to sign the envelope carrying the information card is as follows:

- 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.

4.2. Identity Provider Policy

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

4.2.1. 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>
```

4.2.2. 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].

4.3. 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]. 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.

4.3.1. 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.

4.3.2. 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 3.1). 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.

4.3.3. 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 4.1.1.5), 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>
```

4.3.4. Client pseudonym

A private personal identifier (PPID), defined in Section 8.5.14, 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 the target scope information is sent in the token request using the `wsp:AppliesTo` element, that information can be used by the IP/STS to generate the appropriate PPID value. When token scope information is not sent, an identity selector SHOULD specify the PPID value it would like to be used in the issued token by using the `ic:PPID` element in the RST request. This may be produced as described in Section 4.3.4.1. The IP/STS MAY use this value as is or as an input seed to a custom function to derive a value for the PPID claim.

When PPID 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 PPID information item.

.../ic:ClientPseudonym/ic:PPID
This optional element contains the PPID that the client has submitted for use in the issued token. The IP/STS MAY use this value as the input (a seed) to a custom function and the result used in the issued token.

The following example illustrates the PPID 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 the PPID value or any other client pseudonym values included in the RST message. It MUST NOT record the PPID claim value that it generates.

4.3.4.1. PPID

When token scope information is not sent in a token request to an IP/STS that supports the PPID claim, an Identity Selector SHOULD compute the PPID information it sends in the RST message as follows:

- Construct the *RP identifier* as described in Section 8.6.1.
- Decode the base64 encoded value of the `ic:HashSalt` element of the information card (see Section 7.1) to obtain *SaltBytes*.
- Decode the base64 encoded value of the `ic:MasterKey` element of the information card (see Section 7.1) to obtain *MasterKeyBytes*.
- Hash the *RP identifier* with *MasterKeyBytes* and *SaltBytes* using the SHA256 hash function to obtain the client pseudonym PPID value.

Client Pseudonym PPID = SHA256 (MasterKeyBytes + RP identifier + SaltBytes)

- Convert *Client Pseudonym PPID* to a base64 encoded string and send as the value of the `ic:PPID` element in the RST request.

4.3.5. 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.

4.3.5.1. 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 the following URI value.

  http://schemas.xmlsoap.org/ws/2005/02/trust/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. An identity selector MUST support the P_SHA1 computed key mechanism defined in [WS-Trust] with the particulars below

ComputedKey Value	*Meaning*
http://schemas.xmlsoap.org/ws/2005/02/trust/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.

4.3.5.2. Asymmetric proof key

When requesting an asymmetric key token, an identity selector MUST generate an ephemeral RSA key pair 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 the following URI value.

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

4.3.5.3. 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.

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

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 an <AudienceRestrictionCondition> element restricting the token to the target site URL submitted in the token request.

4.3.6. 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>
```

5. 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.

5.1. 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.

5.2. 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 [Addressing-Ext].

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

5.3. 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 a X.509v3 certificate.

5.4. 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 8.5.14).

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.

6. 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 a 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.

6.1. Relying Party

The following faults MAY occur when submitting security tokens to a 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]" />`

6.2. 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>

7. 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 7.1 describes the transfer format of the collection in the clear, whereas Section 7.2 describes the transfer format after the necessary encryption is applied.

7.1. Pre-Encryption Transfer Format

Each information card in the transfer stream may contain metadata 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 8), an exported self-issued card may also contain key material as well as any associated claims information. This information is referred to as the "information card private data". For managed information cards, the private data is absent as that data resides at the managed identity provider.

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 4.1.1).

.../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.

.../ic:InformationCardMetaData/ic:HashSalt
This optional element contains a random per-card entropy used for computing the relying party specific PPID claim when the card is used at the relying party. This element is deprecated since the "MasterKey" element below can serve the same purpose.

.../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 using 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 may be empty.

.../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 optional element contains the private data for a self-issued information card. This element is absent for a managed information card. So the following elements are only present for a self-issued 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.

.../ic:InformationCardPrivateData/ic:ClaimValueList
This optional element is a container for the set of claim types and their corresponding values embodied by the 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.

7.1.1. 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.

The PKCS-5 based key derivation method MUST be used with the input parameters summarized in the table below for deriving the encryption key from the pin.

Key derivation method	PBKDF1 per [RFC2898] (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.

7.2. 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 7.1.

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. The PKCS-5 based key derivation method MUST be used with the input parameters summarized in the table below for deriving the key from the password.

Key derivation method	PBKDF1 per [RFC2898] (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

8. 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.

8.1. 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
```

8.2. 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*.
- The signature key used in the issued token MUST be a 2048-bit asymmetric RSA key which identifies the issuer.

- 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 3.1.1 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. The default size of the key is 256 bits, but a different size may be specified by the relying party.
  - For *asymmetric* key tokens, the proof key is a public RSA key value specified as a `ds:RSAKeyValue` child element under `ds:KeyValue` element. The default size of the key is 2048 bits.
- 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 8.4.
- 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 8.4. 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`

The XML digital signature [XMLDSIG] profile used to sign a self-issued token MUST be as follows:

- 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>
```

8.3. 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.

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.

The XML encryption [XMLENC] profile that MUST be used for encrypting the key and the token is as follows:

- 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>
```

8.4. Self-Issued Token Signing Key

The RSA 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 RSA 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.

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 a RP where the card is used. Each self-issued information card contains a 256-bit secret random number, called the "master key" (see Section 7.1), that is used as the secret entropy in deriving the token signing RSA key.

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.

8.4.1. 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 $p \leq n$, 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 7.1) is M and the unique RP identifier (derived as per Section 8.6.1) 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

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

8.5. 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
```

8.5.1. 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."

8.5.2. 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."

8.5.3. 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."

8.5.4. 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.

8.5.5. 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.

8.5.6. 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.

8.5.7. 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."

8.5.8. 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."

8.5.9. 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.

8.5.10. 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.

8.5.11. 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.

8.5.12. 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.

8.5.13. 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.

8.5.14. 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 8.6.

8.5.15. 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.

8.6. 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.

8.6.1. Relying party identifier

In order to derive PPID as a function 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) is presented in the form of an X.509v3 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.

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.

There are three cases of how the RP identifier is constructed depending on whether the RP's certificate has any organizational identifier attributes in it, and if it is an extended validation (EV) certificate with respect to the organizational identifier attributes.

Case 1: RP's certificate *is* EV for organizational identifier attributes

- 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.

  RP identifier = SHA256 (OrgIdBytes)

Case 2: RP's certificate *is not* EV for organizational identifier attributes

- Convert the organizational identifier attributes in the end-entity certificate into a string, call it *OrgIdString*, as for Case 1 above.
- Starting with the immediate parent of the end-entity certificate in the certification path and moving backwards to the root CA certificate, convert the subject field in the certificate into a string, call it *CertPathString*, of the following form:

  |ChainElement="*string representation of subject DN*"

  The vertical bar character (ASCII 0x7C) is used as a delimiter at the start of the string. The string representation of the subject DN in the certificate is enclosed within double quote characters (ASCII 0x22).

  The string representation of the subject DN in the certificate is as per RFC 1779 with the following deviations. In the string representation for each RDN in the subject DN:

  - The encodings of adjoining RDNs are separated by a comma followed by a space (,).
  - The AttributeType OID is converted to its X.500 key name. If an OID does not have a corresponding X.500 name, it is encoded as the prefix string "OID." followed by the dotted-decimal encoding of the OID.
  - The AttributeValue is quoted within double quote characters (ASCII 0x22) if it is empty, or contains leading or trailing white space, or contains one of the following characters:
    - Comma (,)

- Plus sign (+)
- Equal sign (=)
- Inch mark (")
- Backslash followed by the letter n (\n)
- Less than sign (<)
- Greater than sign (>)
- Number sign (#)
- Semicolon (;)

- Concatenate the individual *CertPathString* values with the *OrgIdString* value to obtain the string value *QualifiedOrgIdString* such that the *CertPathString* for the root CA is the left-most component and the *OrgIdString* is the right-most component.

  Following is an example *QualifiedOrgIdString* per this convention.

  ```
  |ChainElement="OU=Contoso Trust, Inc., O=Contoso Corporation, C=US"
  |ChainElement="OU=Contoso Internet Authority"
  |O="Microsoft"|L="Redmond"|S="Washington"|C="US"|
  ```

  Note that newline characters are included in the example above only to improve readability; they are not part of the actual constructed string.

- 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)

Case 3: RP's certificate does not have any organizational identifier attributes

- 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.

  RP identifier = SHA256 (PublicKey)

8.6.2. PPID

The PPID value SHOULD be produced as follows using the card identifier and the RP identifier (specified in Section 8.6.1):

- 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 *RP identifier* with the *CanonicalCardId* using the SHA256 hash function to obtain the PPID.

  PPID = SHA256 (RP identifier + CanonicalCardId)

8.6.3. Friendly identifier

The PPID provides a 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

9. References

[HTTP]
R. Fielding et al, "IETF RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", June 1999.

[HTTPS]
E. Rescorla, "RFC 2818: HTTP over TLS", May 2000.

[RFC 2119]
S. Bradner, "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", March 1997.

[RFC 2898]
B. Kaliski, "PKCS #5: Password-Based Cryptography Specification, Version 2.0", September 2000.

[RFC 3066]
H. Alvestrand, "Tags for the Identification of Languages", January 2001.

[SOAP 1.2]
M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework", June 2003.

[WS-Addressing]
M. Gudgin et al, "Web Services Addressing 1.0 – Core", August 2005.

[Addressing-Ext]
Document to be published in the near future.

[WS-MetadataExchange]
"Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1" August 2006.

[WS-Security]
A. Natalin et al, "Web Services Security: SOAP Message Security 1.0", May 2004.

[WS-Policy]
"Web Services Policy Framework (WS-Policy), Version 1.2", March 2006.

[WS-SecurityPolicy]
"Web Services Security Policy Language (WS-SecurityPolicy)", July 2005.

[WS-Trust]
"Web Services Trust Language (WS-Trust)", February 2005.

[XMLDSIG]
Eastlake III, D., Reagle, J., and Solo, D., "XML-Signature Syntax and Processing", March 2002.

[XMLENC]
Imamura, T., Dillaway, B., and Simon, E., "XML Encryption Syntax and Processing", August 2002.

[XML Schema, Part 1]
H. Thompson et al, "XML Schema Part 1: Structures", May 2001.

[XML Schema, Part 2]
P. Biron et al, "XML Schema Part 2: Datatypes", May 2001.