Send comments to:
Phillip Hallam-Baker, Senior Author
401 Edgewater Place, Suite 280
Wakefield MA 01880
Tel 781 245 6996 x227
Email: pbaker@verisign.com

XML Key Management Specification (XKMS)

VeriSign
Microsoft
webMethods

Draft Version 1.0: November 27th 2000

Table Of Contents

Table of Figures

Executive Summary

This document specifies protocols for distributing and registering public keys, suitable for use in conjunction with the proposed standard for XML Signature [XML-SIG] developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) and an anticipated companion standard for XML encryption. The XML Key Management Specification (XKMS) comprises two parts -- the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS).

The X-KISS specification defines a protocol for a *Trust service* that resolves public key information contained in XML-SIG elements. The X-KISS protocol allows a client of such a service to delegate part or all of the tasks required to process `<ds:KeyInfo>` elements. A key objective of the protocol design is to minimize the complexity of application implementations by allowing them to become clients and thereby shielded from the complexity and syntax of the underlying PKI used to establish trust relationships. These may be based upon a different specification such as X.509/PKIX, SPKI or PGP.

The X-KRSS specification defines a protocol for a web service that accepts registration of public key information. Once registered, the public key may be used in conjunction with other web services including X-KISS.

Both protocols are defined in terms of structures expressed in the XML Schema Language, protocols employing the Simple Object Application Protocol (SOAP) v1.1 [SOAP] and relationships among messages defined by the Web services Definition Language v1.0 [WDSL]. Other compatible expressions are possible.

1 Introduction

1.1 Overview

This document specifies protocols for distributing and registering public keys, suitable for use in conjunction with the proposed standard for XML Signatures [XML-SIG] developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) and an anticipated companion standard for XML encryption. The XML Key Management Specification (XKMS) comprises two parts -- the XML Key Information Service Specification (X-KISS) and the XML Key Registration Service Specification (X-KRSS).

These protocols do not require any particular underlying public key infrastructure (such as X.509) but are designed to be compatible with such infrastructures.

This document comprises the following service specifications:

- XML Key Information Service Specification: A protocol to support the delegation by an application to a service of the processing of Key Information associated with an XML signature, XML encryption, or other public key. Its functions include the location of required public keys and the binding of such keys to identification information.

- XML Key Registration Service Specification: A protocol to support the registration of a key pair by a key pair holder, with the intent that the key pair subsequently be usable in conjunction with the XML Key Information Service or XML Trust Assertion Service.

Design criteria include:

- Compatibility with the XML Signature Specification (currently a standards proposal to the W3C and IETF) and with an anticipated specification on XML Encryption. However, its use is not restricted to these grammars.

- Implementation should be as simple as possible using standard XML tools while remaining consistent with the entirety of relevant specifications; the only cryptographic functions required by an application are those needed to support XML Signature or Encryption;

- Implementation should not require ASN.1 tools;

- Management of status information (e.g. "revocation") is transparent to a public-key using application in an online environment;

- Minimize client code and configuration complexity through use of standard protocols and message grammars and also by delegating tasks that may require complex configuration to web services.

To meet these design criteria, the specifications in this family are layered, separating the protocol semantics from the implementation syntax.

The message syntax presented in this document is based on XML and is designed to allow use of the Simple Object Access Protocol (SOAP) and the Web Service Definition Language (WSDL) specifications. From these, it is possible to generate APIs in common programming languages such as the C family of program languages.

It is also possible to express the messages in syntax other than XML, over protocols other than SOAP and through a definition language other than WSDL, though such expression is outside the scope of this specification except to note that SOAP and WSDL are proposals currently or potentially considered by the World Wide Web Consortium XML Protocol Activity. If XML and SOAP are not adopted by this Activity, we anticipate that the protocol would be expressible in any specification recommended by the Activity.

1.2 Definition of Terms

The following terms are used within this document with the particular meaning indicated below:

Service	An application that provides computational or informational resources on request. A service may be provided by several physical servers operating as a unit.
Web service	A service that is accessible by means of messages sent using standard web protocols, notations and naming conventions
Client	An application that makes requests of a service. The concept of 'client' is relative to a service request; an application may have the role of client for some requests and service for others.

1.3 Namepaces

For clarity, some examples of XML are not complete documents and namespace declarations may be omitted from XML fragments. In this document, certain namespace prefixes represent certain namespaces. References to XML schema defined herein use the prefix "s0" and are in the namespace

xmlns:s0=http://www.xmltrustcenter.org/xml/schema/2000-11-12-XKMS.sdl.

This specification uses the elements already defined in the XML Signature namespace. The "XML Signature namespace" is represented by the prefix "ds" and is declared as xmlns:ds=http://www.w3.org/2000/09/dsig. Unqualified elements in examples, unless otherwise noted, are also in this namespace. The "XML Signature schema" is defined in http://www.w3.org/2000/09/xmldsig, and the "`ds:KeyInfo`" element (and all of its contents) are found at http://www.w3.org/2000/09/xmldsig#sec-KeyInfo. The corresponding XML Schema definition can be found in xmldsig-core-schema.xsd.

1.4 Key Information Service Specification

X-KISS allows a client to delegate part or all of the tasks required to process XML Signature `<ds:KeyInfo>` elements to a *Trust service*. A key objective of the protocol design is to minimize the complexity of applications using XML Signature. By becoming a client of the trust service, the application is relieved of the complexity and syntax of the underlying PKI used to establish trust relationships, which may be based upon a different specification such as X.509/PKIX, SPKI or PGP.

By design, the XML Signature Specification does not mandate use of a particular trust policy. The signer of a document is not required to include any key information but may include a `<ds:KeyInfo>` element that specifies the key itself, a key name, X.509

certificate, a PGP Key Identifier etc. Alternatively, a link may be provided to a location where the full `<ds:KeyInfo>` information may be found.

The information provided by the signer may therefore be insufficient by itself to perform cryptographic verification and decide whether to trust the signing key or may not be in a format the client can use. For example:

- The Key may be specified by a name only.
- The local trust policy of the client may require additional information in order to trust the key.
- The Key may be encoded in an X.509 certificate that the client cannot parse.

In the case of an encryption operation:

- The client may not know the public key of the recipient.

1.5 Key Registration Service Specification

X-KRSS describes a protocol for registration of public key information. A client of a conforming service may request that the Registration Service bind information to a public key. The information bound may include a name, an identifier or extended attributes defined by the implementation.

The key pair to which the information is bound may be generated in advance by the client or, to support key recovery, may be generated on request by the service. The Registration protocol may also be used for subsequent recovery of a private key

The protocol provides for authentication of the applicant and, in the case that the key pair is generated by the client, Proof of Possession (POP) of the private key. A means of communicating the private key to the client is provided in the case that the private key is generated by the Registration Service.

This document specifies means of registering RSA and DSA keys and a framework for extending the protocol to support other cryptographic algorithms such as Diffie-Helleman and Elliptic Curve variants.

1.6 Tiered Service Model

Different applications require different levels of PKI service. To support this need a tiered implementation model is defined in which applications may select the precise level of processing that meets their requirements.

Tier 0 Processing of the `<ds:KeyInfo/RetrievalMethod>` element is by the application. Processing is as defined by the XML Signature specification and without assistance of a trust service.

Tier 1 Processing of the `<ds:KeyInfo>` element by the application is delegated to a service. The service returns a `<ds:KeyInfo>` element that describing a public key meeting the criteria specified by the client application. Validation of the `<ds:KeyInfo>` is performed by the client.

Tier 2 Validation Service
As in tier 1, but in addition, the service reports further information concerning the data specified in a `<ds:KeyInfo>` block.

Additional tiers could be defined in separate documents

Tier 3 Assertion Service
Establishment and management of long term trust relationships.

Tier 4 Assertion Status Service
Management of the status of assertions.

In each case, the trust service shields the client application from the complexities of the underlying PKI such as:

- Handling of complex syntax and semantics (e.g. X.509v3)
- Retrieval of information from directory and data repository infrastructure
- Revocation status verification
- Construction and processing of trust chains.

1.7 Structure of this document

The remainder of this document describes the XML Key Information Service Specification and XML Key Registration Service Specification.

Section 2: Protocol Overview.
The functional behavior of the protocol is described.

Section 3: Message Set.
The semantics of the protocol messages are defined.

Section 4: Protocol Overview.
The functional behavior of the protocol is described.

Section 5: Message Set.
The semantics of the protocol messages is defined.

Section 6: Cryptographic Algorithm support
Data formats to support use of the cryptographic algorithms RSA and DSA are defined.

1.8 Key Information Service Protocol Overview

In the XML Signature Specification, a signer may optionally include information about his public signing key ("<ds:KeyInfo>") within the signature block. This key information is designed to allow the signer to communicate "hints" to a verifier about which public key to select.

Another important property of <ds:KeyInfo> is that it may or may not be cryptographically bound to the signature itself. This allows the <ds:KeyInfo> to be substituted or supplemented without "breaking" the digital signature.

For example Alice signs a document and sends it to Bob with a <ds:KeyInfo> element that specifies only the signing Key Data. On receiving the message Bob retrieves additional information required to validate the signature and adds this information into the <ds:KeyInfo> element when he passes the document on to Carol (see Figure 1 below).

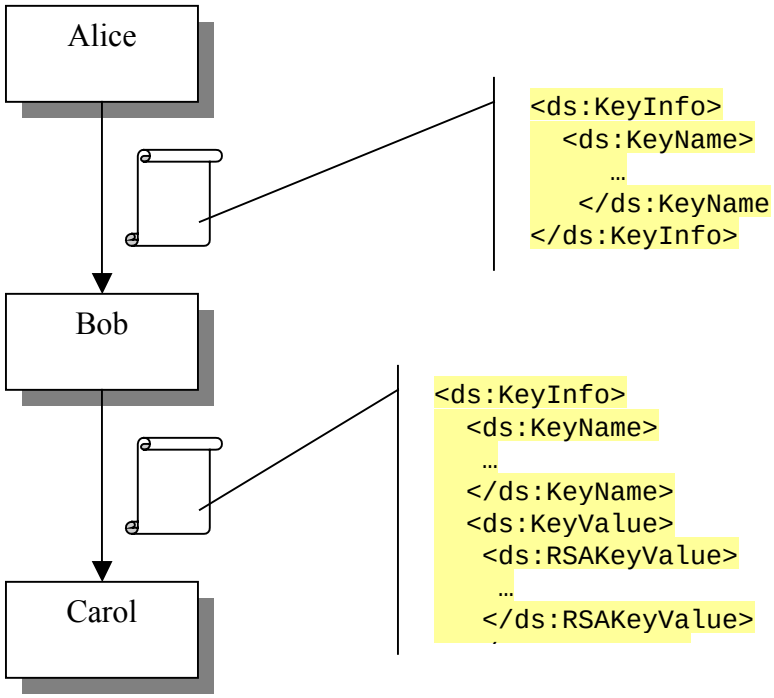

Figure 1: Substitution of the `<ds:KeyInfo>` *element as a message is passed among processors*

1.9 Tier 0 `RetrievalMethod`

A `<ds:KeyInfo>` element may include a *RetrievalMethod,* which is a means to convey information available from a remote location. The `<RetrievalMethod>` element is a feature of and is defined by the XML Signature Specification. Since it is the most basic means of resolving a ds:KeyInfo element it is described here as the 'Tier 0' Key Information service.

For example, the signer of a document may wish to refer verifiers to a chain of X.509 certificates without having to attach them. `RetrievalMethod` consists of a location, a method, and a type, which in this case, would refer to a location on the web from which the certificate chain may be retrieved.

The XML Signature Specification defines the `ds:KeyInfo RetrievalMethod` as follows:

> *A* `RetrievalMethod` *element within* `ds:KeyInfo` *is used to convey a reference to* `ds:KeyInfo` *information that is stored at another location. For example, several signatures in a document might use a key verified by an X.509v3 certificate chain appearing once in the document or remotely outside the document; each signature's* `ds:KeyInfo` *can reference this chain using a single* `RetrievalMethod` *element instead of including the entire chain with a sequence of* `X509Certificate` *elements.*

`RetrievalMethod` uses the same syntax as `Reference` except that there is no `DigestMethod` or `DigestValue` sub-element and presence of the `URI` and `Type` attributes is mandatory. The referenced data is a `ds:KeyInfo` sub-element type. The `Type` attribute, as in Section 4.3.3[of the XML Signature Specification], is a URI consisting of "http://www.w3.org/2000/07/xmldsig#" suffixed with a `ds:KeyInfo` sub-element type, such as "http://www.w3.org/2000/07/xmldsig#X509Data".

Schema Definition:

```
<element name="RetrievalMethod">
   <complexType>
      <sequence>
         <element ref="ds:Transforms" minOccurs="0"/>
      </sequence>
      <attribute name="URI" type="uriReference"/>
      <attribute name="Type" type="uriReference" use="optional"/>
   </complexType>
</element>
```

In the following example, the signer indicates a web-resident directory service (www.PkeyDir.test) where they have published information about their public key.

```
<ds:KeyInfo>
   <RetrievalMethod URI="http://www.PKeyDir.test/CheckKey" />
</ds:KeyInfo>
```

The relying party retrieves the additional Key Information by resolving the specified URL (Figure 2).

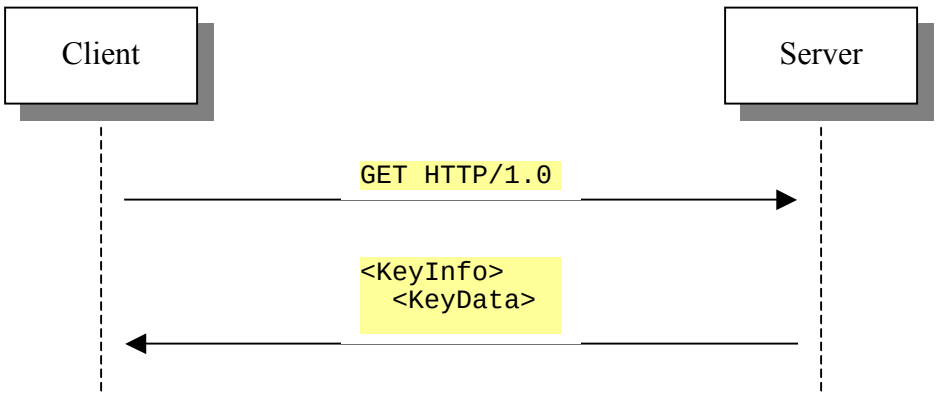

Figure 2: Tier 0 Protocol allows a `ds:KeyInfo` element to reference external data

1.10 Tier 1 `Locate` Service

Tier 1 of the protocol permits a client application to delegate processing of the `ds:KeyInfo` element to a trust service.

The Tier 1 `Locate` service resolves a `ds:KeyInfo` element but does NOT REQUIRE the service to make an assertion concerning the validity of the binding between the data in the `ds:KeyInfo` element.

The Trust service MAY resolve the `ds:KeyInfo` element using local data or MAY relay request to other servers. For example the Trust service might resolve a `RetrievalMethod` element (Figure 3) or act as a gateway to an underlying PKI based on a non-XML syntax.

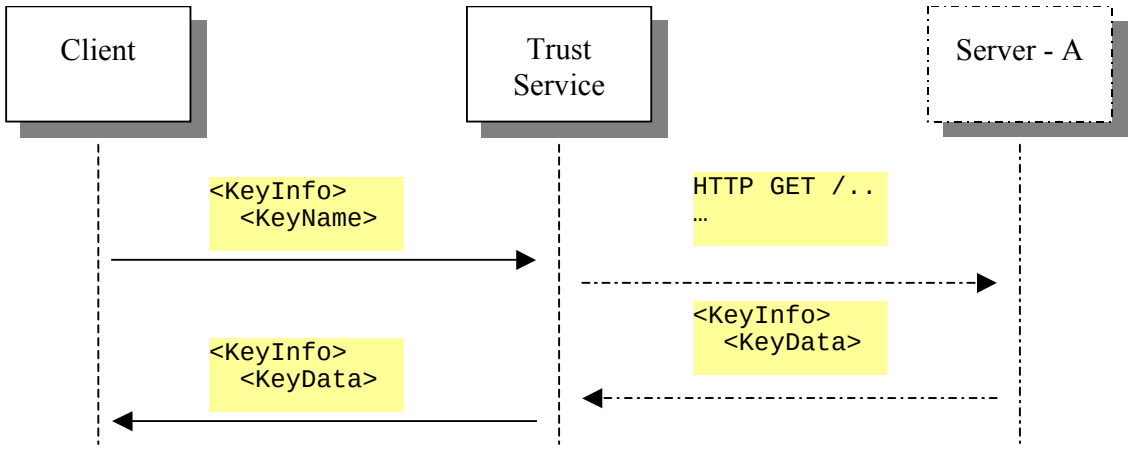

Figure 3: Tier 1 Protocol Provides Name Resolution Service

Both the request and/or the response MAY be signed, to both authenticate the sender and protect the integrity of the data being transmitted, using an XML Signature.

1.10.1 Example: Document Signature

The client receives a signed XML document. The `ds:KeyInfo` element specifies a `RetrievalMethod` for an X.509 certificate that contains the public key. The client sends the `ds:KeyInfo` element to the location service requesting that the `KeyName` and `KeyValue` elements be returned.

Request:

```
<Locate>
   <Query>
      <ds:KeyInfo>
         <ds:RetrievalMethod
           URI="http://www.PKeyDir.test/Certificates/01293122"
           Type="http://www.w3.org/2000/09/xmldsig#X509Data"/>
      </ds:KeyInfo>
   </Query>
   <Respond>
      <string>KeyName</string>
      <string>KeyValue</string>
   </Respond>
</Locate>
```

The location service resolves the `RetrievalMethod`, obtaining an X.509v3 certificate. The certificate is parsed to obtain the public key value that is returned to the client.

The location service DOES NOT report the revocation status or the trustworthiness of the certificate. The `KeyName` returned is obtained from the certificate.

Response:

```
<LocateResult>
   <result>Success</result>
   <answer>
      <ds:KeyInfo>
         <ds:KeyName>O=XMLTrustCernter.org OU="Crypto"
                     CN="Alice"</ds:KeyName>
         <ds:KeyValue>...</ds:KeyValue>
      </ds:KeyInfo>
   </answer>
</LocateResult>
```

(For readability, the contents of the KeyValue element are omitted from the example above. Full examples are shown in appendices.)

1.10.2 Example: Data Encryption

The client is attempting to send an encrypted XML document and requires the public key encryption parameters of the recipient.

Request:

```
<Locate>
   <Query>
      <ds:KeyInfo>
         <ds:KeyName>Alice Cryptographer</ds:KeyName>
      </ds:KeyInfo>
   </Query>
   <Respond>
      <string>KeyName</string>
      <string>KeyValue</string>
   </Respond>
</Locate>
```

Response:

```
<LocateResult>
   <result>Success</result>
   <answer>
      <ds:KeyInfo>
         <ds:KeyName>Alice Cryptographer</ds:KeyName>
         <ds:KeyValue>...</ds:KeyValue>
      </ds:KeyInfo>
   </answer>
</LocateResult>
```

1.11 Tier 2: Validate Service

The Tier 2 Validate Service allows all that tier one does, and in addition, the client may obtain an assertion specifying the status of the binding between the public key and other data, for example a name or a set of extended attributes. Furthermore the service represents that the status of each of the data elements returned is valid and that all are bound to the same public key. The client sends to the trust service a template containing some or all of the elements for which the status of the trust binding is required. If the information in the template is incomplete, the trust service MAY obtain additional data required from an underlying PKI Service. Once the validity of the Key Binding has been determined the Trust service returns the status result to the client (Figure 4).

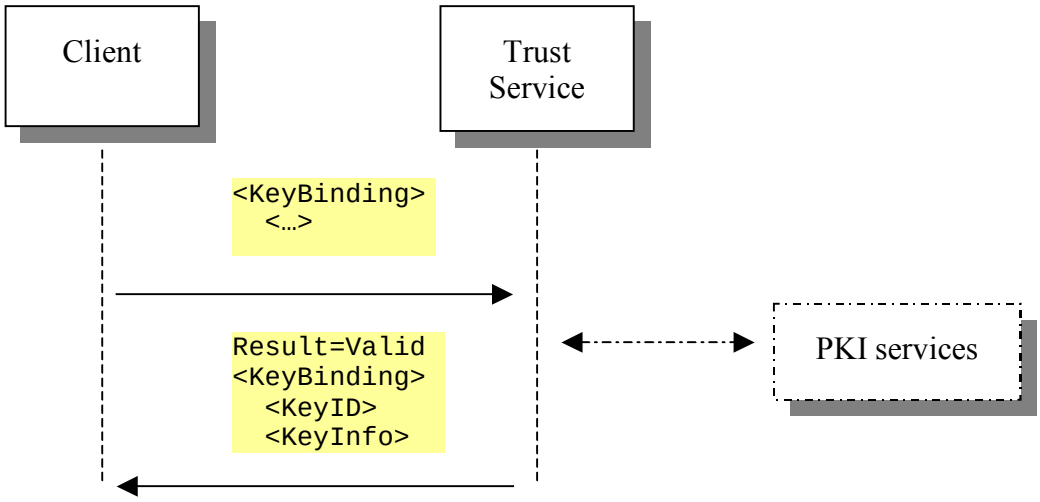

Figure 4: Tier2 Protocol Provides Key Validation Service

1.11.1 Example: Document Signature

The client of the example in section 1.10.1 has verified the document signature. The client now needs to determine whether the binding between the name and the public key is both trustworthy and valid.

Request:

```
<Validate>
   <Query>
      <AssertionStatus>Valid</AssertionStatus
      <KeyID/>

      <ds:KeyInfo>
         <ds:KeyName>...</ds:KeyName>
         <ds:KeyValue>...</ds:KeyValue>
      </ds:KeyInfo>

   </Query>
   <Respond>
      <string>KeyName</string>
      <string>KeyValue</string>
   </Respond>
</Validate>
```

Response:

```
<ValidateResult>
   <Result>Success</Result>
   <Answer>
     <KeyBinding>
       <AssertionStatus>Valid</AssertionStatus>
       <KeyID><KeyIdentifier>du9cXdWZN/0=</KeyIdentifier></KeyID>

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

```
         <ValidityInterval>
            <NotBefore>2000-09-20T12:00:00</NotBefore>
            <NotAfter>2000-10-20T12:00:00</NotAfter>
          </ValidityInterval>
        </KeyBinding>
      </Answer>
   </ValidateResult>
```

1.12 Validity of the Service Response

Clients SHOULD ensure that the response from the service to a Locate or Validate operation is valid, that is that the following criteria are met.

Authenticity: That the response message was issued by a trusted Trust service

Integrity: That the response message has not been modified

Correspondence: The response from the Trust service corresponds to the request that was made to the client.

The appropriate means of validating the service response is dependent on the application. It is not necessary for the requests to be authenticated with a digital signature if the client supports some other secure means of communicating with the Trust service.

The authenticity, integrity and correspondence of the response SHOULD be ensured using one or more of the following methods:

- Authenticating the response messages using the XML Signature Specification.
- Transport layer security (e.g. SSL, TLS, WTLS)
- Packet layer security (e.g. IPSEC)

In the case that signed response messages are employed the means by which the client determines that the signing key is trustworthy is outside the scope of this specification. Possible mechanisms include:

- A root key embedded in the client application
- A trustworthy signing key exchanged using mechanisms described in the Tier 3 Trust Assertion Service Specification.
- A signing key obtained using some other retrieval mechanism such as DNSSEC, PKIX or SPKI.

2 Key Information Service Message Set

The protocol consists of pairs of messages, with an application sending a request message to a trust service and the service responding with another message.

2.1 Common Data Elements

The content and format of messages are defined using the W3C XML Schema specification. All values are encoded as element data. The XKMS specification itself uses only a restricted set of types, but element values may potentially use any type definable within XML Schemas. XKMS is compatible with the object serialization format defined within SOAP (see Appendix A) but does not use some aspects of that format. In particular, sequences of elements are expressed as sequences of elements without reference to arrays.

The following common data elements are used in the message set:

2.1.1 `ds:KeyInfo`

The `ds:KeyInfo` element is defined in the XML Signature Specification schema and that specification governs its format and use.

The `ds:KeyInfo` element communicates data using both attributes and elements. Arbitrary extension elements are permitted.

2.1.2 `ResultCode`

The enumerated type `ResultCode` is used to return result codes from each interface. It has the following possible values:

`Success`
The operation succeeded.

`NoMatch`
No match was found for the search template provided.

`Incomplete`
Only part of the information requested could be provided.

`Failure`
The operation failed for unspecified reasons.

`Refused`
The operation was refused.

`Pending`
The operation was queued for future processing.

`ResultCode` is defined as:

```
<simpleType name="ResultCode" base="string">
   <enumeration value="Success"/>
   <enumeration value="NoMatch"/>
   <enumeration value="Incomplete"/>
   <enumeration value="Failure"/>
   <enumeration value="Refused"/>
   <enumeration value="Pending"/>
```

```
    </simpleType>
```

2.1.3 AssertionStatus

The enumerated type `AssertionStatus` is used to report the status of an assertion such as a key binding. The following values are defined:

`Valid`
The binding is definitively valid.

`Invalid`
The binding is definitively invalid.

`Indeterminate`
The status of the assertion cannot be determined.

`AssertionStatus` is defined as:

```
<simpleType name="BindingStatus" base="string">
   <enumeration value="Valid"/>
   <enumeration value="Invalid"/>
   <enumeration value="Indeterminate"/>
</simpleType>
```

2.1.4 Reason

One or more strings that specify the reason(s) for a particular assertion status.

If the Trust service returns the `AssertionStatus` `Valid`, the `Reason` element lists the aspects of status that have been affirmatively verified to be `Valid`. If the service returns the `AssertionStatus` `Invalid` the Reason element lists the aspects of status that have been determined to be either `Invalid` or `Indeterminate`. If the service returns the `AssertionStatus` `Indeterminate` the Reason element lists the aspects of status that have been determined to be `Indeterminate`.

The status aspects are defined in the table below. For convenience the equivalent X509 processing steps are given:

Aspect	Description	X.509 Equivalent
`IssuerTrust`	The assertion issuer is considered to be trustworthy by the Trust service.	Certificate path anchored by trusted root successfully constructed
`Status`	The Trust service has affirmatively verified the assertion status with an authoritative source	Certificate status validated using CRL or OCSP
`ValidityInterval`	The request was made within the validity interval of the	The request was made at a time when the certificate

	assertion	chain was valid
`Signature`	Signature on signed data provided by the client in the `ds:KeyInfo` element (e.g. `X509Data` element) was successfully verified.	Certificate Signature verified

2.1.5 `Respond`

One or more strings included in the request that specify data elements to be provided in the `ds:KeyInfo` element of the response. Each string is a single identifier corresponding to a sub-element of the XML Signature Specification `ds:KeyInfo` element [XML-SIG] or the private key information defined in section 5.3.2. The XML Signature elements are described here for convenience. The normative reference is the specification [XML-SIG].

The Service SHOULD return a requested data element if it is available. The Service MAY return additional data elements that were not requested. In particular, the service MAY return data elements specified in the request with the response.

Defined identifiers include:

Identifier	**`ds:KeyInfo` Element**	**Description**
`KeyName`	`<ds:KeyName>`	Key Name
`KeyValue`	`<KeyValue>`	Public key parameters
`X509Cert`	`<X509Data>`	X509 Certificate v3 that authenticates the specified key
`X509Chain`	`<X509Data>*`	X509 Certificate v3 chain that authenticates the specified key
`X509CRL`	`<X509Data>`	X509 Certificate Revocation List v2
`OCSP`	`<X509Data>`	PKIX OCSP token that validates an X509v3 certificate that authenticates the key
`RetrievalMethod`	`<RetrievalMethod>`	Retrieval Method data
`MgmtData`	`<MgmtData>`	Management Data
`PGP`	`<PGPData>`	PGP key signing data

PGPWeb	<PGPData>*	Collection of PGP key signing data
SPKI	<SPKIData>	SPKI key signing
Private		Request that the encrypted private key be returned in the response. [Used in the X-KRSS protocol]

For example, a client that has no X.509 processing capability might perform a `Locate` operation to obtain the public key parameters and name information from a `ds:KeyInfo` element that specifies only a certificate. The `Respond` element values in this case would be "`KeyName`" and "`KeyValue`".

2.2 Locate Service

The Locate service accepts as input a `ds:KeyInfo` element that specifies a public key and returns one or more `ds:KeyInfo` elements that relate to the same public key. The `ds:KeyInfo` elements returned are specified by the `Respond` element in the request.

2.2.1 Request Message

The request message consists of the `Locate` element defined by the following schema:

```
<element name="Locate">
   <complexType>
      <sequence>
         <element name="query" type="ds:KeyInfo"/>

         <element name="respond" >
            <complexType>
               <sequence>
                  <element name="string" type="string"
                        minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>
```

The following elements are defined:

`Query`
A single complex structure containing a `ds:KeyInfo` element that specifies the public key for which additional data is requested.

`Respond`
An sequence of identifiers that specify data elements that the client requests returned in the response.

2.2.2 Response Message

The Response Message consists of a `LocateResult` element defined by the following schema:

```
<element name="LocateResult">
   <complexType>
      <sequence>
         <element name="result" type="s0:ResultCode"/>
         <element name="answer" >
            <complexType>
               <all>
                  <element name=”ds:KeyInfo” type=”ds:KeyInfo”
                       minOccurs="0" maxOccurs="unbounded"/>
               </all>
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>
```

The following elements are defined:

`Answer`
A sequence of `strings` that contain `ds:KeyInfo` elements that provide the additional information specified by the `Respond` attribute, for the public key identified by the `Query` element.

The response message returns a `ResultCode` depending on the success of the `Locate` operation as follows:

`ResultCode`	`Answer`	Description
`Success`	At least one element	The locate operation succeeded. All the information requested was available.
`NoMatch`	No elements	The locate operation succeeded but returned no matches.
`Incomplete`	At least one element	The locate operation succeeded. Some of the information requested was not available.
`Failure`	No elements	The locate operation failed.

2.2.3 Faults

When the protocol is expressed in SOAP, all `ResultCode` values except `Success`, `Incomplete` and `NoMatch` are expressed using the SOAP `Fault` element with a `faultcode` of `soap:Server`. See the [SOAP] specification for further details. The service MAY return the descriptive text set out in section 2.2.2 above.

The format of the contents returned by the service within the `detail` element is left as an implementation decision.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <soap:Body>
       <soap:Fault>
        <faultcode>soap:Server</faultcode>
        <faultstring>The locate operation failed.</faultstring>
        <detail>
        </detail>
    </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

2.3 Validate Service

The `Validate` service allows the client to query the binding between a `ds:KeyInfo` element and other data such as an identifier. The client supplies a template for the `KeyBinding` assertion requested. The template may specify either a `KeyId` or a `ds:KeyInfo` element or both. The server returns one or more `KeyBinding` assertions that meet the criteria specified in the request.

2.3.1 `ValidityInterval`

The `ValidityInterval` structure specifies limits on the validity of the assertion.

```
<complexType name="ValidityInterval">
   <sequence>
      <element name="NotBefore" type="timeInstant"/>
      <element name="NotAfter" type="timeInstant"/>
   </sequence>
</complexType>
```

Member	Type	Description
`NotBefore`	`DateTime`	Time instant at which the validity interval begins
`NotAfter`	`DateTime`	Time instant at which the validity interval has ended

The `DateTime` instant MUST fully specify the date.

The `NotBefore` and `NotAfter` elements are optional. If the value is either omitted or equal to the start of the epoch it is unspecified. If the `NotBefore` element is unspecified the assertion is valid from the start of the epoch until the `NotAfter` element. If the `NotAfter` element is unspecified the assertion is valid from the `NotBefore` element with no expiry. If neither element is specified the assertion is valid at any time.

In accordance with the XML Schemas Specification, all time instances are interpreted in Universal Coordinated Time unless they explicitly indicate a time zone.Implementations MUST NOT generate time instances that specify leap seconds.

For purposes of comparison, the time interval `NotBefore` to `NotAfter` begins at the earliest time instant compatible with the specification of `NotBefore` and *has ended* at the earliest time instant compatible with the specification of `NotAfter`

For example if the time interval specified is *day*`T12:03:02` to *day*`T12:05:12` the times `12:03:02.00` and `12:05:11.9999` are within the time interval. The time `12:05:12.0000` is outside the time interval.

2.3.2 KeyId

The `KeyId` element specifies a URI identifier for the key. The URI MAY be a name (URN), a locator (URL) or anything else permitted by the URI specification. The `KeyId` element is distinct from the `KeyName` element of `ds:KeyInfo` in that the `KeyName` element is not required to be a URI.

2.3.3 KeyUsage

The `KeyUsage` element specifies one or more intended uses of the key. If no KeyUsage is specified all uses are permitted.

```
<simpleType name="KeyUsage" base="string">
   <enumeration value="Encryption"/>
   <enumeration value="Signature"/>
   <enumeration value="Exchange"/>
</simpleType>
```

If a key usage is specified that the algorithm does not support (e.g. use of a DSA key for encryption) the element MUST be ignored.

The following identifiers are defined:

Identifier	Description
`Encryption`	The key pair may be used for encryption and decryption
`Signature`	The key pair may be used for signature and verification
`Exchange`	The key pair may be used for key exchange

2.3.4 KeyBinding

The `KeyBinding` element asserts a binding between data elements that relate to a public key including `KeyName`, `KeyID`, `KeyValue` and `X509Data`. Furthermore, the Service represents *to the client accessing the service and to that client alone* that the binding between the data elements is *valid* under whatever trust policy the service offers to that client.

```
   <complexType name="KeyBinding">
      <sequence>
         <element name="Status" type="s0:BindingStatus"/>
         <s:element name="KeyID">
             <s:complexType content="mixed">
                 <s:all>
                     <s:any/>
                 </s:all>
             </s:complexType>
         </s:element>
         <element name="KeyInfo" type="ds:KeyInfo"/>

         <element name="ValidityInterval" type="s0:ValidityInterval"/>
         <element name="KeyUsage" type="s0:KeyUsage"
              minOccurs="0" maxOccurs="unbounded"/>
         <s:element name="KeyUsage">
           <s:complexType>
             <s:all>
               <s:element name="string" type="s:string" minOccurs="0"
maxOccurs="unbounded"/>
             </s:all>
           </s:complexType>
         </s:element>
      </sequence>
   </complexType>
```

2.3.5 Request Message

The request message consists of the `Validate` element defined by the following schema:

```
   <element name="Validate">
   <s:complexType>
     <s:all>
       <s:element name="query" type="s0:KeyBinding"/>
       <s:element name="respond">
         <s:complexType>
           <s:all>
             <s:element name="string" type="s:string"
                        minOccurs="0" maxOccurs="unbounded"/>
           </s:all>
         </s:complexType>
       </s:element>
     </s:all>
   </s:complexType></element>
```

The following elements are defined:

`Query`
A single `KeyBinding` structure that is to be completed and validated.

`Respond`
A sequence of identifiers that specify data elements that the client requests be returned in the response.

2.3.6 Response Message

The Response Message consists of a `ValidateResult` element defined by the following schema:

```
<element name="ValidateResult">
   <complexType>
      <all>
         <element name="result" type="s0:ResultCode"/>
         <element name="answer" >
            <complexType>
               <sequence>
                  <element name="KeyBinding" type="s0:KeyBinding"
                       minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
      </all>
   </complexType>
</element>
```

The following elements are defined:

`Answer`
A sequence of `KeyBinding` structures that contain the results of the validation.

The response message returns a `ResultCode` depending on the success of the `Validate` operation as follows:

`ResultCode`	`Answer`	Description
`Success`	At least one element	The validate operation succeeded. all the information requested was available.
`NoMatch`	No elements	The validate operation succeeded but returned no matches.
`Incomplete`	At least one element	The validate operation succeeded. Some of the information requested was not available.
`Failure`	No elements	The validate operation failed.

Note that the `Validate` operation returns the `ResultCode` `Success` even if the `KeyBinding` assertion was found to be `Invalid` or `Indeterminate`. The `ResultCode` reflects the success or failure of the service query and not the information returned by that query.

2.3.7 Faults

In the SOAP binding, all `ResultCode` values except `Success` and `NoMatch` are expressed using the SOAP `Fault` element.

The format of additional data elements returned by the service is left as an implementation decision.

3 Key Registration Service Protocol Overview

The XML Key Registration Service Specification permits management of information that is bound to a public key pair.

The service specification supports the following operation:

`Register`
: Information is bound to a public key pair through a XKMS `KeyBinding` element. Generation of the public key pair by either the client or the server is supported.

The `Register` request does not in itself place any requirement on the Registration Service to communicate that information to any other party.

In most applications, however, a Registration Service will provide key information to other trust services such as those described in the XKMS specification or a separate underlying PKI such as PKIX.

3.1 Linkage to an Underlying PKI

Linkage to such an underlying PKI is considered to be an intrinsic property of the Registration Service rather than a parameter that the client application may negotiate. To be useful such a negotiation service would need to express more than the syntax of the credentials issued.

If necessary, Registration Services may offer links to multiple underlying PKIs through separate service address URIs. For example:

http://register.trustcenter.org/pgp
: Obtain a PGP credential

http://register.trustcenter.org/x509/public_class2
: Obtain an X.509v3 credential in the “Trust Center Public Class 2” hierarchy

http://register.trustcenter.org/private/AKEHJQ
: Obtain an X.509v3 credential in a private hierarchy, the details of which the client application does not understand

The `ds:KeyInfo` elements `X509Data`, `PGPData` and `SPKIData` MAY be used to return credentials issued in an underlying PKI. Alternatively the client may request that a `RetrievalMethod` element be returned in the response to allow retrieval of the credential to be generated (e.g. an X.509v3 certificate).

3.2 Registration

The `Register` request is used to assert a binding of information to a public key pair. Generation of the public key pair MAY be performed by either the client or the Registration service.

The Registration request message consists of a prototype of the requested assertion. The Registration Service MAY require the client to provide additional information to authenticate the request. If the public key pair is generated by the client the service MAY require the client to provide Proof of Possession of the private key.

On receipt of a registration request, the registration service verifies the authentication and POP information provided (if any). If the registration service accepts the request an assertion is registered. This assertion MAY include some, all or none of the information provided by the Template and MAY include additional information.

The Registration Service MAY return part or all of the registered assertion to the client.

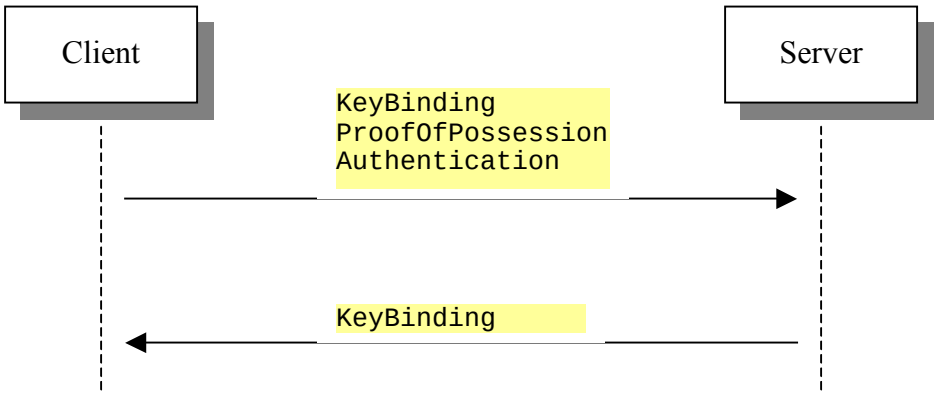

Figure 5: Registration of a KeyBinding

Example: Registration of Client Generated Key Pair

Alice requests registration of an RSA key pair for her email address `Alice@cryptographer.test`. Alice has previously received from the trust service the code "024837" with which to authenticate her request. Alice selects the pass phrase "Help I have revealed my key" to authenticate herself should it be necessary to revoke the registration at a later date.

The X-KRSS request message parameters are:

```
<Register>
   <KeyBinding>
      <AssertionStatus>Valid</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test<KeyID>
      <KeyInfo>
         <ds:KeyInfo>
            <ds:KeyValue>
               <ds:RSAKeyValue>
```

```
                        <ds:Modulus>
998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7E
                        fdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw
   </ds:Modulus>
                        <ds:Exponent>AQAB</ds:Exponent>
                     </ds:RSAKeyValue>
                  </ds:KeyValue>
                  <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
               </ds:KeyInfo>
            <KeyInfo>
         <Authentication>
            <AuthUserInfo
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-10-27-
AuthInfo.xsd">
               <ProofOfPossession>
                  <Signature  [RSA-Sign (KeyBinding, Private)] />
               </ProofOfPossession>
               <Authentication>
                  <Signature  [HMAC-SHA1 (KeyBinding, Auth)] />
               <Authentication>
               <PassPhrase>Pass</PassPhrase>
            </AuthUserInfo>
         </Authentication>
         <Respond>
            <string>KeyName<string>
            <string>KeyValue</string>
            <string>RetrievalMethod</string>
         </Respond>
   </Register>
```

Where:

Auth = HMAC-SHA1 ("024837", 0x1)
Pass = HMAC-SHA1 (HMAC-SHA1 ("helpihaverevealedmykey", 0x2), 0x3)

For clarity, the details of the signature elements are omitted.

The service accepts the registration and returns the following response:

```
   <ResultCode>Success</ResultCode>
   <Answer>
      <AssertionStatus>Valid</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test</KeyID>

      <ds:KeyInfo>
         <ds:RetrievalMethod
              URI="http://www.PKeyDir.test/Certificates/01293122"
              Type="http://www.w3.org/2000/09/xmldsig#X509Data"/>
         <ds:KeyValue>
            <ds:RSAKeyValue>

<ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Efdx
                  SXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw</ds:Modulus>
                <ds:Exponent>AQAB</ds:Exponent>
            </ds:RSAKeyValue>
         </ds:KeyValue>
         <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
      </ds:KeyInfo>

   </Answer>
```

Example: Registration of Service Generated Key Pair

The request for registration of a service generated key pair omits the public key data and requests that private key data be returned with the response.

```
   <KeyBinding>
      <AssertionStatus>Valid</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test</KeyID>
      <KeyInfo>
         <ds:KeyInfo>
            <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
         </ds:KeyInfo>
      </KeyInfo>
   </KeyBinding>
   <Authentication>
      <AuthServerInfo
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-10-27-
AuthInfo.xsd">
         <Authentication>
            <Signature  [HMAC-SHA1 (KeyBinding, Auth)] />
         <Authentication>
         <PassPhrase>Pass</PassPhrase>
      </AuthServerInfo>
   <Authentication>
   <Respond>
      <string>KeyName</string>
      <string>KeyValue</string>
      <string>Private</string>
   </Respond>
```

Where

Auth = HMAC-SHA1 ("`024837`", 0x1)
Pass = HMAC-SHA1 (HMAC-SHA1 ("`helpihaverevealedmykey`", 0x2), 0x3)

The response includes both the public key data and the encrypted private key:

```
   <ResultCode>Success</ResultCode>
   <Answer>
      <AssertionStatus>Valid</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test<KeyID>

      <ds:KeyInfo>
         <ds:KeyValue>
            <ds:RSAKeyValue>

<ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Efdx
                SXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw</ds:Modulus>
               <ds:Exponent>AQAB</ds:Exponent>
            </ds:RSAKeyValue>
         </ds:KeyValue>
         <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
      </ds:KeyInfo>

   <Private> Base64 ( 3DES ( RSAPrivate, Enc)) </Private>
```

Where:

Enc = HMAC-SHA1 ("`024837`", 0x4)
RSAPrivate =

```
   <RSAKeyPair>
      <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidr
      uAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw </ds:Modulus>
      <PublicExponent>AQAB</PublicExponent>
      <PrivateExponent>whatever</PrivateExponent>
      <P>whatever</P>
      <Q>whatever</Q>
   </RSAKeyPair>
```

3.3 Revocation

A Registration service MAY permit clients to revoke previously issued assertions. A revocation request is made in the same manner as the initial registration of a key except that:

- The status of the `KeyBinding` or `KeyAssertion` template is `Invalid`.
- If the Registration service has no record of the assertion the result code `NotFound` is returned.

Example: Revocation

For some reason Alice requests the Registration Service revoke the binding for her public key. Alice authenticates herself using by signing her request with the corresponding private key. Alice could have used the pass phrase she established during registration instead.

The parameters of the request message are:

```
   <KeyBinding>
      <AssertionStatus>Invalid</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test</KeyId>

      <ds:KeyInfo>
         <ds:KeyValue>
            <ds:RSAKeyValue>

<ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Efdx
                  SXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw</ds:Modulus>
               <ds:Exponent>AQAB</ds:Exponent>
            </ds:RSAKeyValue>
         </ds:KeyValue>
         <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
      </ds:KeyInfo>

   </KeyBinding>
   <Authentication>
      <AuthUserInfo
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-10-27-
AuthInfo.xsd">
         <ProofOfPossession>
            <Signature  [RSA-Sign (KeyBinding, Private)] />
         </ProofOfPossession>
      </AuthUserInfo>
   </Authentication>
   <Respond>
      <string>KeyName</string>
      <string>KeyValue</string>
   </Respond>
```

The service responds that the key binding has been revoked:

```
   <ResultCode>Success</ResultCode>
   <Answer>
      <AssertionStatus>Invalid</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test</KeyID>

      <ds:KeyInfo>
         <ds:KeyValue>
            <ds:RSAKeyValue>

<ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Efdx
                  SXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw</ds:Modulus>
               <ds:Exponent>AQAB</ds:Exponent>
            </ds:RSAKeyValue>
         </ds:KeyValue>
         <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
      </ds:KeyInfo>

   </Answer>
```

3.4 Key Recovery

A Registration service MAY permit clients to request key recovery. A key recovery request is made in the same manner as the initial registration of a key except that:

- The key recovery service is likely to require time to respond to the recovery request and MAY return a `ResultCode` of Pending.
- If the Registration service has no record of the assertion the result code `NotFound` is returned.

Example: Key Recovery

Alice has forgotten the private key she registered earlier. She first contacts the administrator of the key recovery service using an out-of-band authentication procedure determined by site policy. The key recovery administrator issues to Alice (using an out of band method) the key recovery authorization code “A8C8S H93HU C9H29 8Y43U H9J3 I23”.

The request parameters for the key recovery are:

```
   <KeyBinding>
      <AssertionStatus>Indeterminate</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test</KeyId>

      <ds:KeyInfo>
         <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
      </ds:KeyInfo>

   </KeyBinding>
   <Authentication>
      <AuthUserInfo
xmlns=“http://www.xmltrustcenter.org/xml/schema/2000-10-27-AuthInfo.xsd”
         <PassPhrase>Auth</PassPhrase>
      </AuthUserInfo>
   </Authentication>
   <Respond>
```

```
      <string>KeyName</string>
      <string>KeyValue</string>
      <string>Private</string>
   </Respond>
```

Where

Auth = HMAC-SHA1 ("a8c8sh93huc9h298y43uh9j3i23", 0x1)

The registration service policy is to revoke a private key whenever key recovery is performed. The service returns the revoked key binding and the private key parameters:

```
   <Answer>
      <AssertionStatus>Invalid</AssertionStatus>
      <KeyID>mailto:Alice@cryptographer.test</KeyID>

      <ds:KeyInfo>
         <ds:KeyValue>
            <ds:RSAKeyValue>

<ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7Efdx
                  SXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw</ds:Modulus>
               <ds:Exponent>AQAB</ds:Exponent>
            </ds:RSAKeyValue>
         </ds:KeyValue>
         <ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
      </ds:KeyInfo>

   </Answer>
   <Private> Base64 ( 3DES ( RSAPrivate, Enc)) </Private>
```

Where:

Enc = HMAC-SHA1 ("a8c8sh93huc9h298y43uh9j3i23", 0x4)
RSAPrivate = "

```
   <RSAKeyPair>
      <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidr
      uAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw</ds:Modulus>
      <PublicExponent>AQAB</PublicExponent>
      <PrivateExponent>whatever</PrivateExponent>
      <P>whatever</P>
      <Q>whatever</Q>
   </RSAKeyPair>"
```

3.5 Request Authentication

The Service SHOULD ensure that all requests are valid.

Authenticity: The request message originated from the specified party.

Integrity: The request message has not been modified.

Possession: If a public key is specified in a registration request, proof that the request is authorized by a party that has access to the corresponding private key.

Registration services set their own authentication policy. This specification defines an authentication mechanism that employs a shared secret established out of band between the client and the Registration Service.

Services SHOULD require that clients demonstrate Proof of Possession of the private key components of a public key if a request is made to register a valid assertion bound to that public key.

Services SHOULD accept Proof of Possession of the private key component of a public key to effect revocation of any assertion bound to that key.

4 Key Registration Service Message Set

The protocol operations consist of a remote procedure call that consists of a single request message sent by the client to the Registration Service followed by a single response message sent by the server to the client.

4.1 Common Syntax

In addition to the common syntax elements defined in section **Error! Reference source not found.** the following additional parameters are defined:

4.1.1 `Respond`

A sequence of strings included in the request that specify data elements to be provided in the `ds:KeyInfo` element of the response. This element is specified in Section 2.1.5 above

The following additional response code is defined:

Identifier	`ds:KeyInfo` Element	Description
`Private`	-	Request that the encrypted private key be returned in the response.

4.2 Registration

The Request message specifies a `KeyBinding` element that provides a template for the key binding to be registered.

The `KeyBinding` element may contain only partial information, a key without a name or a name without a key. In this case, the client is requesting that the Registration Service provide the additional information required to complete the binding.

For example, the client may not specify the public key parameters because the public and private key pair is to be generated by the Registration Service.

4.2.1 Authentication

The Authentication element contains data that authenticates the request. The form of the authentication data depends upon:

- The means of authentication used;
- The public key algorithm used; and
- The party that generates the key pair (client or service).

The information MAY include a proof of possession for a public key that is registered and MAY include information that authenticates the request through a cryptographic binding to the `KeyBinding` element.

4.2.2 Request Message

The request message consists of the `Register` element defined by the following schema:

```
<element name="Register">
   <complexType>
      <all>
         <element name="Template" type="s0:KeyBinding"/>
         <element name="Authentication">
      </all>
<AuthUserInfo/>
<all>
</element>
         <element name="Respond" >
           <complexType >
              <sequence>
                 <element name="string" type=“string"
                          minOccurs="0" maxOccurs="unbounded"/>
              </sequence>
           </complexType>
         </element>
      </sequence>
  </complexType>
</element>
```

The following elements are defined:

`Template`
A single `KeyBinding` structure that specifies elements that the client requests be registered.

`Authentication`
An XML document node that provides information that authenticates the request.

`Respond`
A sequence of identifiers that specify data elements that the client requests be returned in the response.

4.2.3 Response Message

The Response Message consists of a `RegisterResult` element defined by the following schema:

```
<element name="RegisterResult">
   <complexType>
      <sequence>
         <element name="result" type="s0:ResultCode"/>
         <element name="Answer" >
            <complexType>
               <all>
                  <element name="KeyBinding" type="s0:KeyBinding"
                      minOccurs="0" maxOccurs="unbounded"/>
               </all>
            </complexType>
         </element>
         <element name="Private" type="string" />
      </sequence>
   </complexType>
</element>
```

The following elements are defined:

`KeyBinding`
If present specifies the key binding that was registered by the service

`Private`
Additional information provided by the server that MAY provide values for private key parameters generated by the Registration Service

Both the `KeyBinding` and the `Private` data elements are optional.

4.2.4 Faults

In the SOAP binding the `ResultCode` values `NoMatch` and `Failure` are expressed using the SOAP `Fault` element. The service MAY return the descriptive text set out in section 2.2.2 above.

The format of additional data elements returned by the service is left as an implementation decision.

5 Cryptographic Algorithm Specific Parameters

5.1 Use of Symmetric Keying Data

It is frequently necessary or desirable to use symmetric authentication data (i.e. a one time use PIN or pass phrase) to authenticate registration request messages. In particular a private key cannot be used for authentication until the corresponding public key has been registered.

In addition it is desirable that private key parameters generated or recovered by the registration service be returned encrypted. It is convenient to use symmetric data for this purpose.

Since human users are the most demanding in terms of interface requirements the handling of symmetric key data is designed for the needs of clients supporting human users directly. Symmetric keying data is typically issued to a human user in the form of a text string. The authentication data itself MAY be randomly generated and represent an underlying numeric value, or MAY be a password or phrase. In either case it is most convenient to present the value to the human user as a string of characters in a character set the particular user understands.

- All shared string values are encoded as XML
- All space and control characters are removed.
- All upper case characters in the Latin-1 alphabet (A-Z) are converted to lower case.
- No other characters, including accented characters are converted

Keying material is derived from the shared string using a MAC function. Different MAC keying values are used according to the use of the symmetric key derived as follows:

Value	Application
0x1	Authentication
0x2	Encoding of Pass Phrase – Pass 1
0x3	Encoding of Pass Phrase – Pass 2
0x4	Encryption of private key data

If the output of the MAC function provides more keying material than is required for a cryptographic operation (i.e. encryption, MAC), the lowest significant bits are used.

If the output of the MAC function provides less keying material than is required the value is padded with zero bits in the most significant bits.

5.1.1 `Authentication`

The Proof of Possession element contains a XML Signature element. The signature scope is the `KeyBinding` template using the public key that is to be registered. The private key component of the public key contained within the `KeyBinding` is used to generate the signature.

5.1.2 PassPhrase

The PassPhrase element contains a MAC value encoded as a base64 string.

On initial registration the PassPhrase value is obtained by first performing the MAC calculation on the pass phrase value, then performing a second MAC calculation on the result.

To prove knowledge of the pass phrase in a subsequent revocation request the PassPhrase value is obtained by performing the MAC calculation on the pass phrase value.

5.1.3 ProofOfPossession

The Proof of Possession element contains a XML Signature element. The signature scope is the KeyBinding template using the public key that is to be registered. The private key component of the public key contained within the KeyBinding is used to generate the signature.

5.2 Registration of User-Generated RSA or DSA Keys

If an RSA or DSA key pair generated by the user is to be registered, the registration service MAY be required by its registration policy to ensure that:

- The party that made the request has possession of the specified private key
- The party that made the request is authorized to assert the specified binding to a public key.

The Authentication element has the following form:

```
<element name="AuthUserInfo">
   <complexType>
      <all>
         <element name="ProofOfPossession" type="s1:Signature" />
         <element name="Authentication" type="s1:Signature" />
         <element name="Passphrase" type="string" />
      </all>
   </complexType>
</element>
```

The Authentication, PassPhrase and ProofofPossession elements are as defined in section 5.1.1 above.

5.3 Registration of Service-Generated RSA Keys

If the RSA key pair is generated by the registration service the registration MAY be required by its registration policy to ensure that:

- The party that made the request is authorized to assert the specified binding to a public key.

In addition the Registration Service MUST communicate the private key parameters to the user. The Registration Service SHOULD ensure that the confidentiality of the private key is protected.

```
<element name="AuthServerInfo">
   <complexType>
      <all>
         <element name="Authentication" type="s1:Signature" />
         <element name="Passphrase" type="string" />
      </all>
   </complexType>
</element>
```

The `Authentication` and `PassPhrase` elements are as defined in section 5.1.1 above.

Registration of service-generated DSA keys is not supported. A DSA key can only be used for signature. Key recovery of signature keys has only limited application. The principal reason to perform server generation of key pairs is to support Key Recovery.

5.3.1 Encoding of RSA Private Key Parameters

The service MAY return the RSA public and private key parameters to the client.

The public and private parameters for the RSA algorithm are generated from the parameters *p* and *q*. Although private key operations may be performed using the private modulus alone knowledge of the generator parameters permits optimizations such as the Chinese Remainder Theorem to be applied. Accordingly the private key element permits these to be specified.

The XML schema for this structure is:

```
<element name='RSAKeyPair'>
   <complexType content='elementOnly'>
     <all>
      <element name='Modulus' type='ds:CryptoBinary'
             minOccurs='1' maxOccurs='1'/>
      <element name='PublicExponent' type='ds:CryptoBinary'
             minOccurs='1' maxOccurs='1'/>
       <element name='PrivateExponent' type='ds:CryptoBinary'
             minOccurs='1' maxOccurs='1'/>
       <element name='P' type='ds:CryptoBinary'
             minOccurs='0' maxOccurs='1'/>
       <element name='Q' type='ds:CryptoBinary'
             minOccurs='0' maxOccurs='1'/>
       <element name='DP' type='ds:CryptoBinary'
             minOccurs='0' maxOccurs='1'/>
       <element name='DQ' type='ds:CryptoBinary'
             minOccurs='0' maxOccurs='1'/>
       <element name='QINV' type='ds:CryptoBinary'
             minOccurs='0' maxOccurs='1'/>
    </all>
   </complexType>
</element>
```

5.3.2 Encryption of Private Key Parameters

The use of the XML Encryption standard for this purpose is anticipated.

Until the XML Encryption standard is available, the following syntax MAY be used to wrap the private key:

```
<element name='Encrypt'>
   <complexType content='elementOnly'>
     <all>
      <element name='KeyAlg' type='ds:String'
             minOccurs='1' maxOccurs='1'/>
      <element name='EncryptionAlg' type='ds:String'
             minOccurs='1' maxOccurs='1'/>
      <element name='IV' type='ds:CryptoBinary'
             minOccurs='1' maxOccurs='1'/>
      <element name='EncryptedData' type='ds:CryptoBinary'
             minOccurs='1' maxOccurs='1'/>
     </all>
   </complexType>
</element>
```

Where the elements have the following meaning:

ID	Type	Description
`KeyAlg`	`String`	Means of deriving the encryption key from the shared authentication parameter. Possible value `HMAC-SHA1`
`EncryptionAlg`	`String`	Symmetric cipher algorithm. Possible values to include `3DES-CBC`, `AES`
`IV`	`CryptoBinary`	Initialization vector for the symmetric cipher
`EncryptedData`	`CryptoBinary`	The encrypted data itself

6 Authors

Warwick Ford, Phillip Hallam-Baker (Verisign), Barbara Fox, Blair Dillaway, Brian LaMacchia (Microsoft), Jeremy Epstein, Joe Lapp (webMethods)

The authors also acknowledge the contributions of David Solo (CitiGroup), Mack Hicks (Bank of America), Andrew Layman (Microsoft).

Appendix A Web Service Contract

A.1 Protocol Interface

The protocol interface may be indicated informally (not normatively) using syntax based on the C programming language. Note, however, that this indication serves only as a brief introduction to the interactions. They are defined formally and normatively in the next appendix and the definition is in terms of protocol elements and formats, not procedure invocations.

```
public enum ResultCode {
   Success,
   NoMatch,
   Incomplete,
   Failure,
   Refused,
   Pending
   }

public enum AssertionStatus {
   Valid,
   Invalid,
   Indeterminate
   }

public struct ValidityInterval {
   public DateTime              NotBefore;
   public DateTime              NotAfter;
   } ;

public struct KeyBinding {
   public AssertionStatus       Status;
   public XmlElement            KeyID;
   public XmlElement            KeyInfo;
   public string []             KeyUsage;
   public ValidityInterval      ValidityInterval;
   } ;

public interface ITier1Service {
   ResultCode Locate(
      XmlElement                Query,
      string[]                 Respond,
      out XmlElement[]             Answer);
   }

public interface ITier2Service {
   ResultCode Validate (
      KeyBinding                Query,
      string []                 Respond,
      out KeyBinding []         Answer);
   }

public interface RegistrationService {
   ResultCode Register (
      KeyBinding             Template,
      XmlElement             Authentication,
      string                 Respond,
      out KeyBinding []      Answer,
      out XmlElement         Private);
   }
```

```
   Where, XmlElement is a type that represents an valid XML Element
value.  For example, this could be a serialized string representation or
an XML DOM node and would depend upon the tools being used.
```

A.2 Schemas and Web Service Definition

The following Web Service Definition Language definitions are *normative*. The Service, and associated Port, elements define a specific implementation and are exemplary only. In this case, a service located at http://service.xmltrustcenter.org/Test/KeySrvc.asmx.

```
<?xml version="1.0"?>
<definitions xmlns:s="http://www.w3.org/1999/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ds=”http://www.w3.org/2000/09/xmldsig#”
xmlns:s0="http://www.xmltrustcenter.org/xml/schema/2000-11-12-XKMS.sdl"
targetNamespace="http://www.xmltrustcenter.org/xml/schema/2000-11-12-
XKMS.sdl" xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema targetNamespace"http://www.xmltrustcenter.org/xml/schema/2000-
11-12-XKMS.sdl"  attributeFormDefault="qualified"
elementFormDefault="qualified">
      <s:import namespace=”http://www.w3.org/2000/09/xmldsig#”
                schemaLocation=”http://www.w3.org/TR/2000/CR-xmldsig-core-
20001031/xmldsig-core-schema.xsd” />
      <s:element name="Register">
        <s:complexType>
          <s:all>
            <s:element name="Template" type="s0:KeyBinding"/>
            <s:element name="Authentication">
              <s:complexType content="mixed">
                <s:all>
                  <s:AuthUserInfo/>
                </s:all>
              </s:complexType>
            </s:element>
            <s:element name="Respond">
              <s:complexType>
                <s:all>
                  <s:element name="string" type="s:string" minOccurs="0"
maxOccurs="unbounded"/>
                </s:all>
              </s:complexType>
            </s:element>
          </s:all>
        </s:complexType>
      </s:element>
      <s:complexType name="KeyBinding">
        <s:all>
          <s:element name="Status" type="s0:AssertionStatus"/>
          <s:element name="KeyID">
            <s:complexType content="mixed">
              <s:all>
                <s:any/>
```

```
            </s:all>
          </s:complexType>
        </s:element>
        <s:element name="KeyInfo" type="ds:KeyInfo"/>

        <s:element name="KeyUsage">
          <s:complexType>
            <s:all>
              <s:element name="string" type="s:string" minOccurs="0"
maxOccurs="unbounded"/>
            </s:all>
          </s:complexType>
        </s:element>
        <s:element name="ValidityInterval" type="s0:ValidityInterval"/>
      </s:all>
    </s:complexType>
    <s:simpleType name="AssertionStatus" base="s:string">
      <s:enumeration value="Valid"/>
      <s:enumeration value="Invalid"/>
      <s:enumeration value="Indeterminate"/>
    </s:simpleType>
    <s:complexType name="ValidityInterval">
      <s:all>
        <s:element name="NotBefore" type="s:timeInstant"/>
        <s:element name="NotAfter" type="s:timeInstant"/>
      </s:all>
    </s:complexType>
    <s:element name="RegisterResult">
      <s:complexType>
        <s:all>
          <s:element name="result" type="s0:ResultCode"/>
          <s:element name="Answer">
            <s:complexType>
              <s:all>
                <s:element name="KeyBinding" type="s0:KeyBinding"
minOccurs="0" maxOccurs="unbounded"/>
              </s:all>
            </s:complexType>
          </s:element>
          <s:element name="Private">
            <s:complexType content="mixed">
              <s:all>
                <s:any/>
              </s:all>
            </s:complexType>
          </s:element>
        </s:all>
      </s:complexType>
    </s:element>
    <s:simpleType name="ResultCode" base="s:string">
      <s:enumeration value="Success"/>
      <s:enumeration value="NoMatch"/>
      <s:enumeration value="Incomplete"/>
      <s:enumeration value="Failure"/>
      <s:enumeration value="Refused"/>
      <s:enumeration value="Pending"/>
    </s:simpleType>
```

```
      <s:element name="Validate">
        <s:complexType>
          <s:all>
            <s:element name="query" type="s0:KeyBinding"/>
            <s:element name="respond">
              <s:complexType>
                <s:all>
                  <s:element name="string" type="s:string" minOccurs="0"
maxOccurs="unbounded"/>
                </s:all>
              </s:complexType>
            </s:element>
          </s:all>
        </s:complexType>
      </s:element>
      <s:element name="ValidateResult">
        <s:complexType>
          <s:all>
            <s:element name="result" type="s0:ResultCode"/>
            <s:element name="answer">
              <s:complexType>
                <s:all>
                  <s:element name="KeyBinding" type="s0:KeyBinding"
minOccurs="0" maxOccurs="unbounded"/>
                </s:all>
              </s:complexType>
            </s:element>
          </s:all>
        </s:complexType>
      </s:element>
      <s:element name="Locate">
        <s:complexType>
          <s:all>
            <s:element name="query" type="ds:KeyInfo"/>
              <s:complexType content="mixed">

            <s:element name="respond">
              <s:complexType>
                <s:all>
                  <s:element name="string" type="s:string" minOccurs="0"
maxOccurs="unbounded"/>
                </s:all>
              </s:complexType>
            </s:element>
          </s:all>
        </s:complexType>
      </s:element>
      <s:element name="LocateResult">
        <s:complexType>
          <s:all>
            <s:element name="result" type="s0:ResultCode"/>
            <s:element name="answer">
              <s:complexType>
                <s:all>
                  < element name=”ds:KeyInfo” type=”ds:KeyInfo” minOccurs="0"
maxOccurs="unbounded"/>
                </s:all>
```

```
              </s:complexType>
            </s:element>
          </s:all>
        </s:complexType>
      </s:element>
    </s:schema>
  </types>
  <message name="RegisterSoapIn">
    <part name="parameters" element="s0:Register"/>
  </message>
  <message name="RegisterSoapOut">
    <part name="parameters" element="s0:RegisterResult"/>
  </message>
  <message name="ValidateSoapIn">
    <part name="parameters" element="s0:Validate"/>
  </message>
  <message name="ValidateSoapOut">
    <part name="parameters" element="s0:ValidateResult"/>
  </message>
  <message name="LocateSoapIn">
    <part name="parameters" element="s0:Locate"/>
  </message>
  <message name="LocateSoapOut">
    <part name="parameters" element="s0:LocateResult"/>
  </message>
  <portType name="KeyServiceSoap">
    <operation name="Register">
      <input message="s0:RegisterSoapIn"/>
      <output message="s0:RegisterSoapOut"/>
    </operation>
    <operation name="Validate">
      <input message="s0:ValidateSoapIn"/>
      <output message="s0:ValidateSoapOut"/>
    </operation>
    <operation name="Locate">
      <input message="s0:LocateSoapIn"/>
      <output message="s0:LocateSoapOut"/>
    </operation>
  </portType>
  <portType name="KeyServiceHttpPost"/>
  <portType name="KeyServiceHttpGet"/>
  <binding name="KeyServiceSoap" type="s0:KeyServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
    <operation name="Register">
      <soap:operation soapAction="http://service.xmltrustcenter.org/Register"
style="document"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="Validate">
      <soap:operation
soapAction=="http://service.xmltrustcenter.org/Validate" style="document"/>
```

```
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="Locate">
      <soap:operation soapAction=="http://service.xmltrustcenter.org/Locate"
style="document"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <binding name="KeyServiceHttpPost" type="s0:KeyServiceHttpPost">
    <http:binding verb="POST"/>
  </binding>
  <binding name="KeyServiceHttpGet" type="s0:KeyServiceHttpGet">
    <http:binding verb="GET"/>
  </binding>
  <service name="KeyService">
    <port name="KeyServiceSoap" binding="s0:KeyServiceSoap">
      <soap:address
location=="http://service.xmltrustcenter.org/Test/KeySrvc.asmx"/>
    </port>
    <port name="KeyServiceHttpPost" binding="s0:KeyServiceHttpPost">
      <http:address
location=="http://service.xmltrustcenter.org/Test/KeySrvc.asmx"/>
    </port>
    <port name="KeyServiceHttpGet" binding="s0:KeyServiceHttpGet">
      <http:address
location=="http://service.xmltrustcenter.org/Test/KeySrvc.asmx"/>
    </port>
  </service>
</definitions>
```

A.3 Authentication Schema

The following schema defines the authentication data:

```
   <schema
targetNamespace="http://www.xmltrustcenter.org/xml/schema/2000-10-27-
AuthInfo.xsd”
      xmlns="http://www.w3.org/1999/XMLSchema"
      xmlns:ds=”http://www.w3.org/2000/09/xmldsig”>
      <element name="AuthUserInfo">
         <complexType>
            <sequence>
               <element name="ProofOfPossession" type="s1:Signature"
                    />
               <element name="Authentication" type="s1:Signature"
                    />
```

```
               <element name="Passphrase" type="string"
                    />
            </sequence>
         </complexType>
      </element>
      <element name="AuthServerInfo">
         <complexType>
            <sequence>
               <element name="Authentication" type="s1:Signature"
                    />
               <element name="Passphrase" type="string"
                    />
            </sequence>
         </complexType>
      </element>
      <element name='RSAKeyPair'>
         <complexType content='elementOnly'>
            <sequence>
                <element name='Modulus' type='ds:CryptoBinary'
                       minOccurs='1' maxOccurs='1'/>
                <element name='PublicExponent' type='ds:CryptoBinary'
                       minOccurs='1' maxOccurs='1'/>
                 <element name='PrivateExponent' type='ds:CryptoBinary'
                       minOccurs='1' maxOccurs='1'/>
                 <element name='P' type='ds:CryptoBinary'
                       minOccurs='0' maxOccurs='1'/>
                 <element name='Q' type='ds:CryptoBinary'
                       minOccurs='0' maxOccurs='1'/>
             </sequence>
         </complexType>
      </element>
   </schema>
```

A.4 Encryption Envelope

```
   <schema
targetNamespace="http://www.xmltrustcenter.org/xml/schema/2000-11-12-
XKMS.sdl/EncInfo.xsd"
      xmlns="http://www.w3.org/1999/XMLSchema"
      xmlns:ds="http://www.w3.org/2000/09/xmldsig">
      <element name='Encrypt'>
         <complexType content='elementOnly'>
            <sequence>
                <element name='KeyAlg' type='ds:String'
                       minOccurs='1' maxOccurs='1'/>
                <element name='EncryptionAlg' type='ds:String'
                       minOccurs='1' maxOccurs='1'/>
                <element name='IV' type='ds:CryptoBinary'
                       minOccurs='1' maxOccurs='1'/>
                <element name='EncryptedData' type='ds:CryptoBinary'
                       minOccurs='1' maxOccurs='1'/>
            </sequence>
         </complexType>
   </element>
   <Schema>
```

Appendix B Sample Protocol Exchanges

B.1 Tier 1 Example 1

This example shows the formatting of the X-KISS request and response for the first example in section 1.10 above.

Client Request

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <soap:Body>
       <Locate xmlns="http://www.xmltrustcenter.org/xml/schema/2000-11-
12-XKMS.sdl">
   <query><ds:KeyInfo><ds:KeyValue><ds:RSAKeyValue><ds:Modulus>998/T2PUN
8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ
/l6xw==</ds:Modulus><ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds
:KeyValue></ds:KeyInfo></query>
         <Respond>KeyName</Respond>
      </Locate>
     </soap:Body>
   </soap:Envelope>
```

Server Response

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <soap:Body>
       <LocateResult
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-11-12-XKMS.sdl">
         <Result>Success</Result>
         <Answer>
   <ds:KeyInfo><ds:KeyValue><ds:RSAKeyValue><ds:Modulus>998/T2PUN8HQlnhf
9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw==
</ds:Modulus><ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds:KeyVal
ue><ds:KeyName>Account 1823945 Key 3</ds:KeyName></ds:KeyInfo>
        </Answer>
       </LocateResult>
     </soap:Body>
   </soap:Envelope>
```

B.2 Tier 1 Example 2

This example shows the formatting of the X-KISS request and response for the second example in section 1.10 above.

Client Request

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <soap:Body>
       <Locate xmlns="http://www.xmltrustcenter.org/xml/schema/2000-11-
12-XKMS.sdl">
         <query><ds:KeyInfo><ds:KeyName>Alice
Cryptographer</ds:KeyName></ds:KeyInfo></query>
         <Respond>KeyValue</Respond>
       </Locate>
     </soap:Body>
```

```
   </soap:Envelope>
```

Server Response

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <soap:Body>
       <LocateResult
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-11-12-XKMS.sdl">
         <Result>Success</Result>
         <Answer>
<ds:KeyInfo><ds:KeyValue><ds:RSAKeyValue><ds:Modulus>998/T2PUN8HQlnhf9YI
KdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw==</d
s:Modulus><ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds:KeyValue>
<ds:KeyName>Alice Cryptographer</ds:KeyName></ds:KeyInfo>
         </Answer>
       </LocateResult>
     </soap:Body>
   </soap:Envelope>
```

B.3 Tier 2

This example shows the formatting of the X-KISS request and response for the example in section 1.10.1 above.

Client Request

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <soap:Body>
       <Validate xmlns="http://www.xmltrustcenter.org/xml/schema/2000-
11-12-XKMS.sdl">
         <Query>
           <AssertionStatus>Valid</AssertionStatus>
           <KeyID/>

   <ds:KeyInfo><ds:KeyValue><ds:RSAKeyValue><ds:Modulus>998/T2PUN8HQlnhf
9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw==
</ds:Modulus><ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds:KeyVal
ue><ds:KeyName>Account 1823945 Key 3</ds:KeyName></ds:KeyInfo>

         </Query>
         <Respond>KeyName</Respond>
         <Respond>KeyValue</Respond>
      </Validate>
     </soap:Body>
   </soap:Envelope>
```

Server Response

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
```

```
     <soap:Body>
       <ValidateResult
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-11-12-XKMS.sdl">
         <Result>Success</Result>
         <Answer soapenc:arrayType="KeyBinding[1]">
           <KeyBinding>
             <AssertionStatus>Valid</AssertionStatus>
             <KeyID><KeyIdentifier>du9cXdWZN/0=</KeyIdentifier></KeyID>

   <ds:KeyInfo><ds:KeyValue><ds:RSAKeyValue><ds:Modulus>998/T2PUN8HQlnhf
9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw==
</ds:Modulus><ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds:KeyVal
ue><ds:KeyName>Account 1823945 Key 3</ds:KeyName></ds:KeyInfo>

             <ValidityInterval>
               <NotBefore>2000-09-20T12:00:00</NotBefore>
               <NotAfter>2000-10-20T12:00:00</NotAfter>
             </ValidityInterval>
           </KeyBinding>
         </Answer>
       </ValidateResult>
     </soap:Body>
   </soap:Envelope>
```

B.4 Registration of Client Generated Key Pair

Request Message

```
   <?xml version="1.0"?>
    <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <soap:Body>
        <Register xmlns="http://www.xmltrustcenter.org/xml/schema/2000-
11-12-XKMS.sdl">
          <Template>
            <AssertionStatus>Valid</AssertionStatus>
            <KeyID>mailto:Alice@cryptographer.test</KeyID>

               <ds:KeyInfo>

<ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
               </ds:KeyInfo>

            <ValidityInterval>
              <NotBefore>2000-09-20T12:00:00</NotBefore>
              <NotAfter>2001-09-20T12:00:00</NotAfter>
            </ValidityInterval>
          </Template>
          <Authentication>
            <RSAAuthUserInfo
xmlns='http://schema.tbs.test/RSAUserInfo.xsd'>
              <ProofOfPossession>
                <Signature>2PUN8HQlnhf9YI</Signature>
              </ProofOfPossession>
              <Authentication>
                <Signature>EfdxSXAidruAszN</Signature>
              </Authentication>
              <PassPhrase>qfarJIsfcVKLo</PassPhrase>
            </RSAAuthUserInfo>
          </Authentication>
```

```
          <Respond>
            <string>KeyName</string>
            <string>KeyValue</string>
          </Respond>
        </Register>
      </soap:Body>
    </soap:Envelope>
```

Server Response

```
   <?xml version="1.0"?>
    <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <soap:Body>
        <RegisterResult
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-11-12-XKMS.sdl">
          <Result>Success</Result>
          <Answer soapenc:arrayType=”KeyBinding[1]”>
            <KeyBinding>
              <AssertionStatus>Valid</AssertionStatus>
              <KeyID>mailto:Alice@cryptographer.test</KeyID>

                <ds:KeyInfo>
                  <ds:KeyValue>
                    <ds:RSAKeyValue>
   <ds:Modulus>998/T2PUN8HQlnhf9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidruAsz
NqBoOqfarJIsfcVKLob1hGnQ/l6xw==</ds:Modulus>
                      <ds:Exponent>AQAB</ds:Exponent>
                    </ds:RSAKeyValue>
                  </ds:KeyValue>

<ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName>
                </ds:KeyInfo>

              <ValidityInterval>
                <NotBefore>2000-09-20T12:00:00</NotBefore>
                <NotAfter>2001-09-20T12:00:00</NotAfter>
              </ValidityInterval>
            </KeyBinding>
          </Answer>
          <Private/>
        </RegisterResult>
      </soap:Body>
    </soap:Envelope>
```

B.5 Registration of server generated key

Request

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <soap:Body>
        <Register xmlns="http://www.xmltrustcenter.org/xml/schema/2000-
11-12-XKMS.sdl">
          <Template>
            <AssertionStatus>Valid</AssertionStatus>
```

```
            <KeyID>mailto:Alice@cryptographer.test</KeyID>

   <ds:KeyInfo><ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName><
/ds:KeyInfo>

            <ValidityInterval>
              <NotBefore>2000-09-20T12:00:00</NotBefore>
              <NotAfter>2001-09-20T12:00:00</NotAfter>
            </ValidityInterval>
          </Template>
          <Authentication><RSAAuthServerInfo

xmlns='http://schema.tbs.test/RSAServerInfo.xsd'><Authentication><Signat
ure>EfdxSXAidruAszN</Signature></Authentication><PassPhrase>qfarJIsfcVKL
o</PassPhrase></RSAAuthUserInfo></Authentication>
          <Respond>KeyName</Respond>
          <Respond>KeyValue</Respond>
          <Respond>Private</Respond>
        </Register>
      </soap:Body>
    </soap:Envelope>
```

Service Response

```
   <?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <soap:Body>
        <RegisterResult
xmlns="http://www.xmltrustcenter.org/xml/schema/2000-11-12-XKMS.sdl">
          <result>Success</result>
          <Answer soapenc:arrayType=”KeyBinding[1]”>
            <KeyBinding>
              <AssertionStatus>Valid</AssertionStatus>
              <KeyID>mailto:Alice@cryptographer.test</KeyID>

   <ds:KeyInfo><ds:KeyValue><ds:RSAKeyValue><ds:Modulus>998/T2PUN8HQlnhf
9YIKdMHHGM7HkJwA56UD0a1oYq7EfdxSXAidruAszNqBoOqfarJIsfcVKLob1hGnQ/l6xw==
</ds:Modulus><ds:Exponent>AQAB</ds:Exponent></ds:RSAKeyValue></ds:KeyVal
ue><ds:KeyName>mailto:Alice@cryptographer.test</ds:KeyName></ds:KeyInfo>

              <ValidityInterval>
                <NotBefore>2000-09-20T12:00:00</NotBefore>
                <NotAfter>2001-09-20T12:00:00</NotAfter>
              </ValidityInterval>
            </KeyBinding>
          </Answer>
          <Private>IsfcVKLob1hGnQ/l6xw</Private>
        </RegisterResult>
      </soap:Body>
   </soap:Envelope>
```

Appendix C Immediate Binding

This appendix describes a means of expressing X-KISS and X-KRSS messages without reference to the SOAP protocol or an equivalent. This is an optional feature, implementations may implement this but are considered fully compliant with XKMS if they do not.

Messages may be layered on a MIME compliant transport protocol such as HTTP or MIME using the following header descriptor.

```
   Content-Type: text/xml; charset="utf-8"
```

In HTTP the POST method is used to exchange a request and response.

The collected XML schema specification is:

```
   <schema
      xmlns:s0="http://www.xmltrustcenter.org/xml/schema/2000-11-12-
XKMS.xsd"
      targetNamespace="http://www.xmltrustcenter.org/xml/schema/2000-11-
12-XKMS.xsd"
      attributeFormDefault="qualified" elementFormDefault="qualified">
      <element name="Register">
         <complexType>
            <sequence>
               <element name="Template" type="s0:KeyBinding"/>
               <element name="Authentication" type="string" />
               <element name="Respond" >
                 <complexType >
                    <sequence>
                        <element name="string" type="string"
                             minOccurs="0" maxOccurs="unbounded"/>
                    </sequence>
                 </complexType>
               </element>
            </sequence>
        </complexType>
      </element>

      <complexType name="KeyBinding">
         <sequence>
            <element name="AssertionStatus" type="s0:BindingStatus"/>
            <element name="KeyID" type="string" />
            <element name="KeyInfo" type="ds:KeyInfo"/>
            <element name="ValidityInterval"
type="s0:ValidityInterval"/>
            <element name="KeyUsages" >
               <complexType >
                  <sequence>
                     <element name="KeyUsage" type="s:string"
                           minOccurs="0" maxOccurs="unbounded"/>
                  <sequence>
               <complexType>
            </element>
         </sequence>
      </complexType>

      <simpleType name="BindingStatus" base="string">
         <enumeration value="Valid"/>
```

```
      <enumeration value="Invalid"/>
      <enumeration value="Indeterminate"/>
   </simpleType>

   <simpleType name="KeyUsage" base="string">
      <enumeration value="Encryption"/>
      <enumeration value="Signature"/>
      <enumeration value="Exchange"/>
   </simpleType>

   <complexType name="ValidityInterval">
      <sequence>
         <element name="NotBefore" type=“timeInstant"/>
         <element name="NotAfter" type=“timeInstant"/>
      </sequence>
   </complexType>

   <element name="RegisterResult">
      <complexType>
         <sequence>
            <element name="result" type="s0:ResultCode"/>
            <element name="Answer" >
               <complexType>
                  <sequence>
                     <element name="KeyBinding" type="s0:KeyBinding"
                         minOccurs="0" maxOccurs="unbounded"/>
                  </sequence>
               </complexType>
            </element>
            <element name="Private" type=“string" />
         </sequence>
      </complexType>
   </element>

   <simpleType name="ResultCode" base=“string">
      <enumeration value="Success"/>
      <enumeration value="NoMatch"/>
      <enumeration value="Incomplete"/>
      <enumeration value="Failure"/>
      <enumeration value="Refused"/>
      <enumeration value="Pending"/>
   </simpleType>

   <element name="Validate">
      <complexType>
         <sequence>
            <element name="query" type="s0:KeyBinding"/>
            <element name="respond" >
               <complexType>
                  <sequence>
                     <element name="string" type=“string"
                           minOccurs="0" maxOccurs="unbounded"/>
                  </sequence>
               </complexType>
            </element>
         </sequence>
      </complexType>
   </element>

   <element name="ValidateResult">
      <complexType>
         <sequence>
            <element name="result" type="s0:ResultCode"/>
            <element name="answer" >
```

```
                  <complexType>
                     <sequence>
                        <element name="KeyBinding" type="s0:KeyBinding"
                             minOccurs="0" maxOccurs="unbounded"/>
                     </sequence>
                  </complexType>
               </element>
            </sequence>
         </complexType>
      </element>

      <element name="Locate">
         <complexType>
            <sequence>
               <element name="query" type="ds:KeyInfo"/>

               <element name="respond" >
                  <complexType>
                     <sequence>
                        <element name="string" type=“string"
                             minOccurs="0" maxOccurs="unbounded"/>
                     </sequence>
                  </complexType>
               </element>
            </sequence>
         </complexType>
      </element>

      <element name="LocateResult">
         <complexType>
            <sequence>
               <element name="result" type="s0:ResultCode"/>
               <element name="answer" >
                  <complexType>
                     <all>
                        < element name=”ds:KeyInfo” type=”ds:KeyInfo”
minOccurs="0" maxOccurs="unbounded"/>
                     </all>
                  </complexType>
               </element>
            </sequence>
         </complexType>
      </element>
   </schema>
```

Appendix D References

[SOAP] D. Box, D Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S Thatte, D. Winer. *Simple Object Access Protocol (SOAP) 1.1*, W3C Note 08 May 2000, http://www.w3.org/TR/SOAP

[WDSL] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, *Web services Description Language (WSDL) 1.0* September 25, 2000, http://msdn.microsoft.com/xml/general/wsdl.asp

[XML-SIG] D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-Signature Syntax and Processing*, World Wide Web Consortium. http://www.w3.org/TR/xmldsig-core/

[XML-Schema1] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn. *XML Schema Part 1: Structures*, W3C Working Draft 22 September 2000, http://www.w3.org/TR/xmlschema-1/

[XML-Schema2] P. V. Biron, A. Malhotra, *XML Schema Part 2: Datatypes*; W3C Working Draft 22 September 2000, http://www.w3.org/TR/xmlschema-2/

Appendix E Legal Notices