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

X-KASS: XML Key Agreement Service Specification

Phillip Hallam-Baker *VeriSign*

Draft Version 0.4: May 10th 2001

X-KASS: XML Key Agreement Service Specification

Version 0.4

Table Of Contents

Table of Figures

Executive Summary

A key exchange protocol is presented in XML syntax. The protocol comprises a single request followed by a single response and results in a shared secret being established between the two parties.

The key exchange protocol does not provide authentication in the conventional sense that both initiator and responder have determined that the other is authentic at the completion of the protocol. Only if both parties are authentic however, is the shared secret established.

The protocol is designed for use in combination with other XML protocols, including XKMS **[XKMS]** and SOAP/XML Protocol **[SOAP][WSDL]**.

1 Introduction

Public Key Infrastructure provides a secure and adaptable means of establishing a security context between two parties allowing the confidentiality and integrity of communications between them to be protected.

Public key cryptography provides greater flexibility than traditional symmetric key cryptography but requires computationally intensive calculations to be performed. For this reason public key cryptography is typically combined with symmetric key cryptography via a session key **Error! Reference source not found.**.

The XML Key Agreement Service Specification (XKASS) describes an efficient means of key agreement in which an initiator and responder establish a shared secret if and only if they hold the keying information specified in their credentials by means of a single request and a single response.

The high computational overhead of public key cryptography has been considered by some to be a prohibitive burden for many applications involving a high volume of queries against a central server. The computational overhead of executing a digital signature per transaction typically reduces throughput on a server by at least two orders of magnitude.

Use of cryptographic acceleration hardware reduces but does not eliminate the impact of using public key cryptography. Such hardware tends to be expensive, especially if it is designed to provide a high degree of protection against disclosure of the private key.

XKASS permits the key agreement operation to be separated from processing operations. This has both security advantages and operational advantages. Separating key agreement from operations allows cryptographic operations to be performed in a high security physical environment dedicated to cryptographic operations, thus eliminating the need for access by processing operations staff. Offloading cryptographic operations to separate hardware allows independent management of resources to adjust for demand. While processing requirements grow with the number of transactions, the requirement for key agreement scales with the number of active accounts.

1.1 Introduction to this Document

This document describes two Key Agreement protocols and an implementation of the protocol in XML syntax.

1.2 Structure of this document

The remainder of this document describes the Key Agreement Service Specification.

Section 2: Architecture
The Key Exchange Architecture is described

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

2 Architecture

The XKASS protocol allows an initiator to establish a shared secret with a responder that is cryptographically bound to a context that includes; a label identifying the shared secret, the credentials of both parties and additional information such as the time period in which the secret is to be used etc.

XKASS does not provide authentication of the initiator or responder, however the shared secret is only established if both parties have provided authentic credentials and an attacker cannot gain any information that would compromise either party by attempting to engage in a false exchange.

A particular feature of the XKASS architecture is that both the label and the shared secret itself are under the control of the responder. This allows the responder to encode information into the label that would allow a third party to decode it to obtain the shared secret in the manner of a Kerberos ticket [Kerberos]. This feature allows key agreement operations to be separated from data processing operations.

2.1 Data Flow

XKASS is designed for use as a single component in an XML transaction dataflow that will typically involve multiple services supported by different servers that MAY be administered by different enterprises. A typical dataflow is shown in Figure 1.

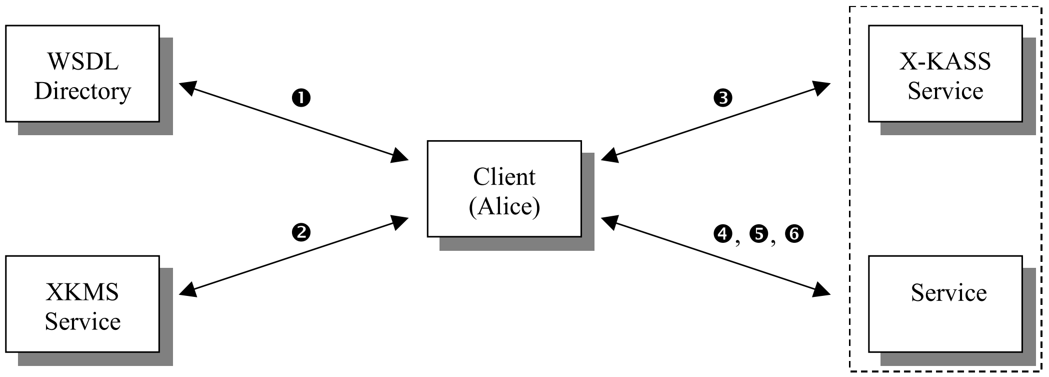

Figure 1 Data Flow of a Typical Trusted XML Transaction

The sequence of events is:

❶ The client (Alice) obtains the Web Service Description Language description of the XML service from a directory. This description specifies:

- o The XML Schema of the interface protocol.
- o The location of the service.
- o A credential C_B consisting of an XML Signature `<KeyInfo>` element specifying the public key of the XKASS service.
- o The cryptographic enhancements (signature, encryption) required.
- o The location of the XKASS key exchange service.

❷ The client validates the `<KeyInfo>` element specifying the public key of the XKASS service against a trusted XKMS service.

❸ The client performs a Key Agreement with the XKASS service authenticating itself via a credential C_A consisting of an XML Signature `<KeyInfo>` element specifying the public key of the client to establish a shared secret s_s, identified by label *L*.

❹ The client uses the shared secret to authenticate and/or encrypt the exchange with the service itself:

- o The client identifies the shared secret used by specifying the label *L*.
- o The Service authenticates *L* by means of a shared secret previously exchanged with the XKASS server.
- o The Service obtains s_S by decryption fields in *L* by means of a shared secret previously exchanged with the XKASS server.

❺, ❻ Once established, the shared secret may be used to authenticate multiple transactions with the same service.

The X-KASS exchange itself consists of a single request-response exchange. This exchange is shown in Figure 2.

❶ The Initiator A first validates the credentials C_B of the responder B.

❷ A sends the request message to B containing a challenge to the credentials C_B.

❸ B generates the response to the challenge made by A, the shared secret s_s, the context O and label L.

❹ B sends the response message to A containing a challenge to the credentials C_A and the shared secret s_s encrypted under a key that is cryptographically bound to the credentials C_A and C_B, the context O, the label L and the response to both challenges.

❺ A generates the response to the challenge made by B and uses it to recover the shared secret s_s.

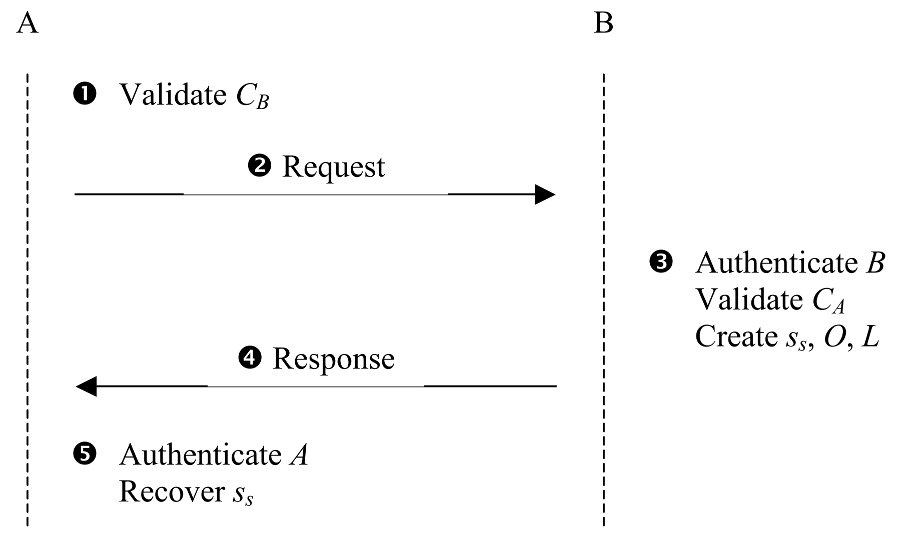

Figure 2: X-KASS Key Agreement Outline

The X-KASS protocol is designed to meet the following condition:

The initiating party A obtains the shared secret s_s from the responding party B if and only if:

- *A has access to the private key identified by the credential C_A.*
- *B has access to the private key identified by the credential C_B.*
- *The credential C_A is considered to be valid and trustworthy by B.*
- *The credential C_B is considered to be valid and trustworthy by A.*
- *The context information O and label L generated by B has been transmitted unmodified to A.*

In addition the protocol MAY provide evidence to A that demonstrates that the shared secret s_S is genuine.

Thus X-KASS does not provide mutual authentication in the traditional sense. A MAY authenticate the X-KASS service, however at the end of the protocol B does not know if the challenge was issued by the authentic party A.

The initiator is only authenticated when attempting to *use* the shared secret. Only the authentic party A can recover the correct shared secret s_S.

2.2 Key Exchange Mechanisms

XKASS currently supports two exchange mechanisms:

- Key Agreement using Encryption Credentials
- Key Agreement using Signature Credentials

The first mechanism using encryption credentials requires fewer public key operations (2 per party) than the second using signature credentials (3 per party). However the second mechanism provides perfect forward secrecy such that compromise of the credentials of either or both parties does not compromise the confidentiality of any messages previously exchanged.

The basic mechanism could be modified to support cases in which one party used an encryption credential and the other used a signature credential. Also the encryption mechanism could be modified to support perfect forward secrecy. Support for both features was rejected in favor of simplifying implementation. It is assumed that the credentials issued for use with the key agreement service would typically be issued for that purpose alone. It is therefore assumed that the issuer of the credentials will bear the limitations of the protocol in mind when issuing credentials.

For similar reasons the protocol implementing the exchange mechanism does not support a negotiation protocol to allow the parties to arrive at a mutually agreed set of cryptographic algorithms. Such frameworks add significantly to the cost of implementation but in practice provide little of identifiable benefit.

It is assumed that the client will obtain all necessary information concerning acceptable cryptographic algorithms and the service credentials as part of the service discovery process. It is expected that a future version of the Web Services Description Language **[WSDL]** would allow such parameters to be expressed.

The following table specifies the symbols used in the specification of the key exchange mechanism.

Symbol	Description

n_X	Nonce value sent by party X.
K_X, k_X	Public and private key values of party X.
s_T, s_L	Temporary and long term shared secret key values
I_X	Purported identity of party X
C_X	Credential of party X, $C_X = \{I_X, K_X\}$
L	Label which uniquely identifies the value s_X
O	Other context data, including the validity date, key usage, acceptable algorithms, etc.
$H(d)$	One way digest function on data d
$E(d, k)$	Encryption of message d with key k
$D(d, k)$	Decryption of message d with key k
$S(d, k)$	Signature of message d with key k
$M(d, k)$	Message Authentication Code of message d with key k

2.2.1 Key Exchange Using Encryption Credentials

The key exchange using encryption credentials mechanism requires one public key encryption operation and one public key decryption operation by each of the requestor and responder.

The mechanism does not provide forward secrecy. However the private encryption key of both the requestor and the responder must be compromise to compromise the established shared secret.

The mechanism is shown in diagrammatic form in Figure 3. It is assumed that the initiator of the protocol (A) begins with knowledge of the public key of the key exchange server.

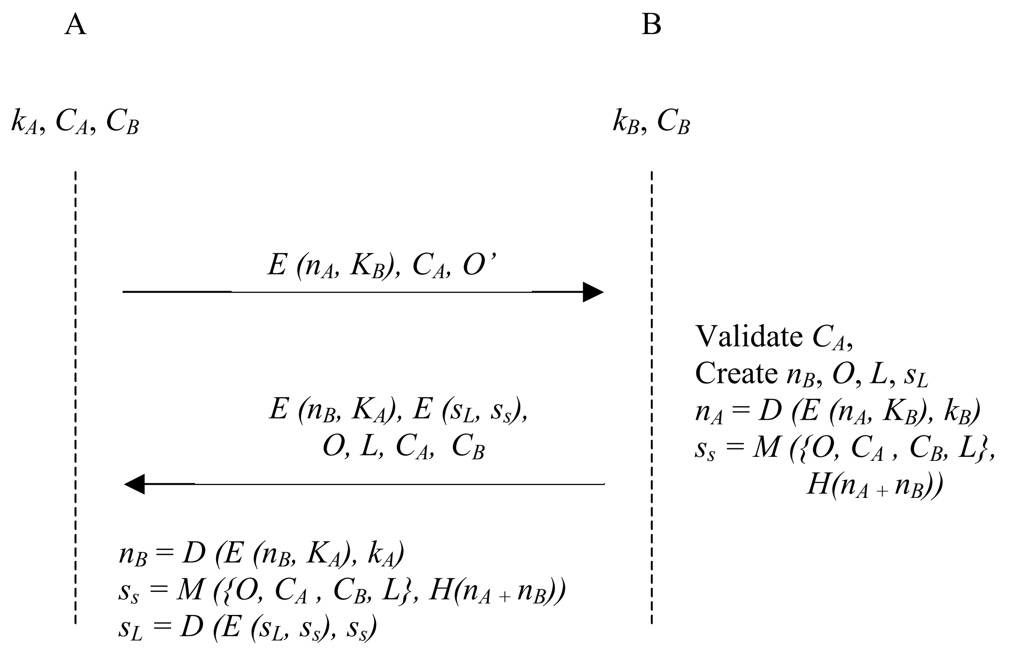
A
B
k_A, C_A, C_B
k_B, C_B
$E\ (n_A, K_B), C_A, O'$
Validate C_A,
Create n_B, O, L, s_L
$n_A = D\ (E\ (n_A, K_B), k_B)$
$s_s = M\ (\{O, C_A, C_B, L\}, H(n_A + n_B))$
$E\ (n_B, K_A), E\ (s_L, s_s), O, L, C_A, C_B$
$n_B = D\ (E\ (n_B, K_A), k_A)$
$s_s = M\ (\{O, C_A, C_B, L\}, H(n_A + n_B))$
$s_L = D\ (E\ (s_L, s_s), s_s)$

Figure 3: Key Agreement Using Encryption Credentials

The protocol parameters are defined as follows:

Message	**Parameter**	**Description**
A→ B	$E\ (n_A, K_B)$	Nonce generated by A, encrypted under the public key of B
	$C_A = \{K_A, I_A\}$	KeyInfo element specifying Public Key, Identity of A
	O'	Proposed use context
B	n_A	Calculate $n_A = D\ (E\ (n_A, K_B), k_B)$
	K_A	Validate as trustworthy
	s_s	Calculate $s_s = M\ (\{O, C_A, C_B, L\}, H(n_A + n_B))$
B → A	$E\ (n_B, K_A)$	Nonce generated by B, encrypted under the public key of A
	O	Actual use context
	$C_A = \{K_A, I_A\}$	KeyInfo element specifying Public Key, Identity of A
	$C_B = \{K_B, I_B\}$	KeyInfo element specifying Public Key, Identity of B

Message	**Parameter**	**Description**
	L	Label
	$E\ (s_L,\ s_s)$	Encrypted shared secret
A	n_B	Calculate $n_B = D\ (E\ (n_B,\ K_A),\ k_A)$
	s_s	Calculate $s_s = M\ (\{O,\ C_A\ ,\ C_B,\ L\},\ H(n_A + n_B))$
	s_L	Calculate $s_L = D\ (E\ (s_L,\ s_s),\ s_s)$

In addition B may return the following data to authenticate B to A. Note that the protocol does not provide for A to authenticate to B. Only the authentic A can recover the correct value of s_L from $E\ (s_L,\ s_s)$ however, hence only A can generate authenticated messages after the key agreement has completed.

Message	**Parameter**	**Description**
B → A	$M(E\ (s_L,\ s_s),\ s_L)$	Proof of knowledge of s_L, and hence n_A, and hence k_B
A	$M(E\ (s_L,\ s_s),\ s_L)$	Verify

2.2.2 Key Exchange Using Signature Credentials

The key exchange using signature credentials mechanism requires one public key signature operation and one public key verification operation by each of the requestor and responder.

The mechanism provides perfect forward secrecy. After the secret Diffie-Helleman key agreement parameters have been erased by both parties it is not possible to recover the established secret even if the private keys of both parties are compromised.

The mechanism is shown in diagrammatic form in Figure 4. It is assumed that the initiator of the protocol (A) begins with knowledge of the public key of the key exchange server.

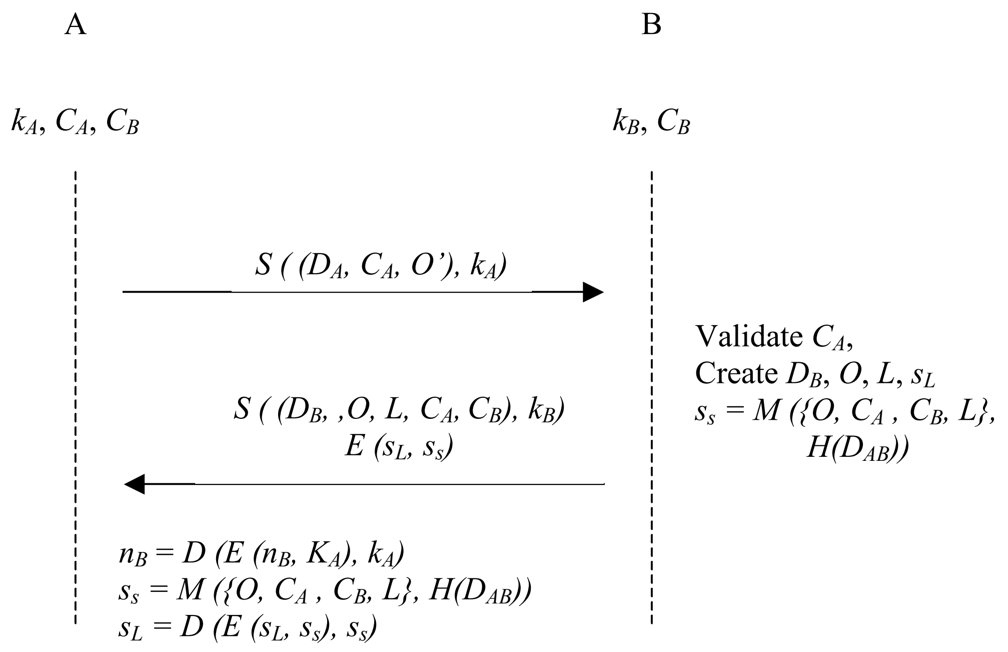
A
B
k_A, C_A, C_B
k_B, C_B
$S\ ((D_A, C_A, O'), k_A)$
Validate C_A,
Create D_B, O, L, s_L
$s_s = M (\{O, C_A , C_B, L\}, H(D_{AB}))$
$S\ ((D_B, ,O, L, C_A, C_B), k_B)$
$E\ (s_L, s_s)$
$n_B = D\ (E\ (n_B, K_A), k_A)$
$s_s = M (\{O, C_A , C_B, L\}, H(D_{AB}))$
$s_L = D\ (E\ (s_L, s_s), s_s)$

Figure 4: Key Exchange Using Signature Credentials

The protocol parameters are defined as follows:

Message	Parameter	Description
A→ B	D_A	Diffie-Helleman Key Agreement Parameters of A
	$C_A = \{P_A, I_A\}$	KeyInfo element specifying Public Key, Identity of A
	O'	Proposed use context
	$S(m, k_A)$	Message Signature m = $\{D_A, C_A, O'\}$
B	D_B, D_{AB}	Calculate remaining Diffie-Helleman Parameters
	$C_A = \{P_A, I_A\}$	Validate as trustworthy
	s_s	Calculate $s_s = M (\{O, C_A , C_B, L\}, H(D_{AB}))$
B → A	D_B	Diffie-Helleman Key Agreement Parameters of B
	O	Actual use context
	$C_A = \{P_A, I_A\}$	KeyInfo element specifying Public Key, Identity of A
	$C_B = \{P_B, I_B\}$	KeyInfo element specifying Public Key, Identity of B

Message	Parameter	Description
	L	Label
	S(m, k_B)	Message Signature m = {D_B, C_A, C_B, *L*, *O*}
	E (s_L, s_s)	Encrypted shared secret
A	D_{AB}	Calculate remaining Diffie-Helleman Parameters
	s_s	Calculate $s_s = M(\{O, C_A, C_B, L\}, H(D_{AB}))$
	s_L	Calculate $s_L = D(E(s_L, s_s), s_s)$

In addition B may return the following data to authenticate B to A. Note that the protocol does not provide for A to authenticate to B. Only the authentic A can recover the correct value of s_L from *E* (s_L, s_s) however, hence only A can generate authenticated messages after the key agreement has completed.

Message	Parameter	Description
B → A	$M(E(s_L, s_s), s_L)$	Proof of knowledge of s_L, and hence n_A, and hence k_B
A	$M(E(s_L, s_s), s_L)$	Verify

2.3 Asset/Risk/Threat (ART) Analysis

We consider the vulnerability of the protocol using an Asset/Risk/Threat analysis. In this methodology the vulnerabilities of a system are systematically examined by identifying the *assets* that might be subject to attack, then for each asset enumerating the *risks* that are intrinsic to that type of asset and the architecture specific *threats* that might realize that risk.

2.3.1 Assets

The following assets are identified:

[A-PRIV] Private Key

[A-SS] Exchanged Shared Secret

[A-UC] Use context

[A-SSL] Shared Secret Label

[A-SRV] Service
Availability of the XKASS service.

2.3.2 Risks

Having identified the assets we consider the range of *potential* harm that an attacker might attempt to cause against each asset in turn. To do this we apply a standard taxonomy of information security risks that comprises Disclosure, Integrity and Denial of Service.

A key feature of the XKASS design philosophy is to focus on the control of risks rather than piecemeal responses to specific threats. For this reason we outline the specific design features used to control each risk.

[R-DPK] Disclosure of Private Key
The private key of either party is revealed.
The protocol relies upon the security of the underlying encryption algorithms to control this risk.

[R-DPKO] Disclosure of Private Key Oracle
A weaker form of [R-DPK] in which the private key itself is not revealed but the protocol allows an attacker to gain information that requires access to the public key that could be used in another context.
The protocol uses one-way functions and masking values to ensure that no party ever applies a private key to any value that can be controlled by an external party.

Threats: [T-IMP] [T-SUB]

[R-DSS] Disclosure of Shared Secret
The shared secret is revealed.
The protocol relies upon the security of the encryption function and on the security of the one-way function to control this risk. The shared secret is an unpredictable function of secret values that are known only through the possession of the specified credential of one of the parties and the interaction through the protocol of the other party.

Threats: [T-IMP]

[R-DSSO] Disclosure of Shared Secret Oracle
A weaker form of [R-DSS] in which the shared secret itself is not revealed but the protocol allows an attacker to gain information that requires access to the shared secret that could be used in another context.
The protocol returns only one value that is a function of the shared secret, the value $M(E(s_L, s_s), s_L)$ which is used to authenticate the responder. Depending on the protocol the value $E(s_L, s_s)$ is an unpredictable function of either, n_B or D_B which are both randomly chosen by the responder. Thus an attacker cannot use the protocol as an oracle for the shared secret.

[R-IUC] Integrity of Use Context
A modified use context is accepted.
The shared secret is an unpredictable function of the use context and label. Thus a

modification of the use context or label causes an unpredictable change to the shared secret and vice versa

Threats: [T-MIM][T-REP].

[R-ISS] Integrity of Shared Secret.
A modified shared secret is accepted.
See [R-IUC]

[R-ISSL] Integrity of Shared Secret Label
A modified shared secret label is accepted.
See [R-IUC]

[R-DOS] Denial of Service
Access to the service is denied.
The protocol does not provide specific defenses against denial of service attacks.

Threats: [T-DOS]

2.3.3 Threats

2.3.3.1 [T-IMP] Impersonation

We consider the case in which an attacker attempts to impersonate each party in the protocol separately.

Impersonation of A Attack

Any party can cause the XKASS server to respond to a properly formatted request presenting credentials K_A of a different party. Such a party cannot discover S however since:

Such a party does not (by definition) know S_A and cannot recover n_B from $E(n_B, P_A)$.

It is not possible to recover k from $E(S, k)$ by trial and error since every possible value of S is equally valid.

Nor does the protocol provide a useful oracle for S_B or n_A since the value n_A is not returned. It is used exclusively as a keying value for a MAC.

Impersonation of B Attack

Any party may attempt to impersonate B. Such a party does not (by definition) know S_B and cannot recover n_A from $E(n_A, P_B)$.

2.3.3.2 [T-MIM] Man in the Middle

A man in the middle might attempt to discover the value of the established secret or to cause a particular value to be chosen as the shared secret by modification of messages between A and B.

Modification of any value identified as an asset that is returned by the B causes A to calculate a different value of s_L in a manner that cannot be predicted. This modification will then be detected either by inspection of the responder authentication value (if present) or when attempting to use the shared secret.

2.3.3.3 [T-REP] Replay Attack

Replaying a previous message generated by A causes B to respond with a new set of key agreement parameters. The set of parameters returned are independent of any previous key agreement and recovery of the shared secret still requires the private key held by A.

Replaying a previous response message generated by B causes A to calculate an erroneous value for the shared secret. This will then be detected either by inspection of the responder authentication value (if present) or when attempting to use the shared secret.

2.3.3.4 [T-DOS] Denial of Service

An attacker can use the protocol to request that B perform a public key operation. There is therefore a potential denial of service attack in which the attacker causes B to perform a large number of computationally intensive tasks.

This form of attack may be mitigated by giving processing priority to requests that are authenticated by a MAC keyed by a previously exchanged key. This limits the damage caused by a denial of service attack to preventing new parties performing a key agreement for the first time.

2.3.3.5 [T-SUB] Protocol Substitution

An attacker could use a message generated using the XKASS protocol as a component in another protocol. This threat is impossible to eliminate since it depends in part upon the ability of other protocols to discriminate against protocol substitution. An authentication protocol that accepted *any* signed message as a means of authenticating a principal would be vulnerable to a protocol substitution attack using messages intended for use with XKASS.

2.4 Context Binding

The shared secret is securely bound to the use context. This prevents cut and paste attacks in which the attacker modifies the label or other data that governs use of the shared secret.

2.4.1 Label

The label is a string that identifies the shared secret to another party. In many cases it is convenient to use a URI as a label although a binary string may be preferred in some applications.

The label MAY contain one or more encrypted data fields containing information such as:

- The authenticated identity of A and/or B
- The established shared secret
- The validity interval

2.4.2 Validity Interval

The validity interval specifies the time period for which the established shared secret is valid. When the time period expires the established shared secret MUST not be used.

2.4.3 Acceptable Key Usage

The established shared secret MAY be limited to specific cryptographic purposes for example; encryption, authentication and key exchange.

2.4.4 Acceptable Algorithm

The established shared secret MAY be limited to use with specific cryptographic algorithms for example 3DES or AES. Use of the same cryptographic key with multiple algorithms introduces the risk that an attacker might discover the key by breaking the weakest algorithm, allowing the data encrypted using the strongest and otherwise unbreakable algorithm to be recovered.

2.4.5 Data Limit

The established shared secret MAY be limited to encryption or authentication of a certain quantity of data, thus limiting the amount of source data available to a cryptanalyst.

3 Message Set in XML Syntax

All XKASS protocol elements are defined using XML schema **[XML-Schema1][XML-Schema2]**. For clarity unqualified elements in schema definitions are in the XML schema namespace:

xmlns="`http://www.w3.org/2000/10/XMLSchema.`"

References to the XKASS schema defined herein use the prefix “s0” and are in the namespace:

xmlns:s0="`http://www.xmltrustcenter.org/tbs/1066-12-25/`"

This namespace is also used for unqualified elements in message protocol examples.

The XKASS schema specification uses some 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/xmldsig#`"

The “XML Signature schema” is defined in **[XML-SIG-XSD]** and the `<ds:KeyInfo>` element (and all of its contents) are defined in **[XML-SIG]**§4.4.

3.1 Element `<Context>`

The `<Context>` element contains all data elements that are combined using a MAC to create the value s_S other than the MAC key. These elements are $\{C_A , C_B, L, O\}$, the credentials of both parties, the label and other context information.

Bringing together these data items into one element facilitates calculation of the MAC value that is generated using the XML Signature standard.

The following schema defines the `<Context>` element:

```
<complexType name="Context">
   <sequence>
      <element name="Label" type="uriReference"
                    minOccurs="1" maxOccurs="1"/>
      <element name="ValidityInterval" type="s0:ValidityInterval"/>
                    minOccurs="1" maxOccurs="1"/>
      <element name="KeyUsage" type="s0:KeyUsage"
                    minOccurs="1" maxOccurs="1"/>
      <element name="PermittedAlgorithms" type="s0:PermittedAlgorithms"
                    minOccurs="1" maxOccurs="1"/>
      <element name="PermittedProtocols" type="s0:PermittedProtocols"
                    minOccurs="1" maxOccurs="1"/>
      <element name="Requestor" type="ds:KeyInfo"
                    minOccurs="1" maxOccurs="1"/>
      <element name="Respondent" type="ds:KeyInfo"
                    minOccurs="1" maxOccurs="1"/>
   </sequence>
</complexType>
```

3.1.1 Element `<Label>` URI

The `<Label>` element contains the label assigned to the shared secret by the responder.

The following schema defines the `<Label>` element:

```
<element name="Label" type="uriReference"
              minOccurs="1" maxOccurs="1"/>
```

3.1.2 Element `<ValidityInterval>`

The `<ValidityInterval>` element contains the validity interval in which the shared secret is to be used.

The following schema defines the `<ValidityInterval>` element:

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

3.1.3 Element `<KeyUsage>` String []

The `<KeyUsage>` element specifies the range of permitted key usages. The following usages are defined:

The following schema defines the `<KeyUsage>` element:

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

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

3.1.4 Element `<PermittedAlgorithms>` URI []

The `<PermittedAlgorithms>` element if present specifies the cryptographic algorithms that are permitted for use with the established secret. If the element is not present all algorithms are permitted.

The following schema defines the `<PermittedAlgorithms>` element:

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

3.1.5 Element <PermittedProtocols> URI []

The <PermittedProtocols> element if present specifies the cryptographic protocols that are permitted for use with the established secret. If the element is not present all protocols are permitted.

The following schema defines the <PermittedProtocols> element:

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

3.1.6 </element>Element <DataLimit> Integer

The <DataLimit> element if present specifies the maximum amount of data in bytes that is to be either encrypted or authenticated using the established secret.

The following schema defines the <DataLimit> element:

```
<element name="DataLimit" type="integer"
              minOccurs="1" maxOccurs="1"/>
```

3.1.7 Elements <Requestor>, <Respondent> <ds:KeyInfo>

The <Requestor> and <Respondent> elements contain the credentials C_A , C_B of the requestor and responder respectively.

The following schema defines the <Requestor> and <Respondent> elements:

```
<element name="Requestor" type="ds:KeyInfo"
              minOccurs="1" maxOccurs="1"/>
<element name="Respondent" type="ds:KeyInfo"
              minOccurs="1" maxOccurs="1"/>
```

3.2 Cryptographic Data Elements

3.2.1 Element <EncryptedSecret>

The <EncryptedSecret> element contains the value $E\ (s_L, s_s)$ in a response in base64 encoding.

The following schema defines the <EncryptedSecret> element:

```
<element name="EncryptedSecret" type="cryptoBinary"
                    minOccurs="1" maxOccurs="1"/>
```

3.2.2 Element <ResponderAuthentication>

The <ResponderAuthentication> element contains the value $M(E\ (s_L, s_s), s_L)$ in a response in base64 encoding.

The following schema defines the `<ResponderAuthentication>` element:

```
<element name="ResponderAuthentication" type="cryptoBinary"
                    minOccurs="1" maxOccurs="1"/>
```

3.3 Key Agreement With Encryption Credentials

3.3.1 Element `<EncryptedNonce>`

The `<EncryptedNonce>` element contains the value *E* (n_A, K_B) in a request and *E* (n_B, K_A) in a response in base64 encoding.

The following schema defines the `<EncryptedNonce>` element:

```
<element name="EncryptedNonce" type="cryptoBinary"
                    minOccurs="1" maxOccurs="1"/>
```

3.3.2 Element `<EncryptedKeyAgreement>`

The `<EncryptedKeyAgreement>` element contains the values *E* (n_A, K_B), C_A, *O'* sent in a the key agreement using encryption credentials request.

The following schema defines the `<EncryptedKeyAgreement>` element:

```
<complexType name="EncryptedKeyAgreement">
   <sequence>
      <element name="Context" type="s0:Context"/>
      <element name="EncryptedNonce" type="cryptoBinary"
                    minOccurs="1" maxOccurs="1"/>
   </sequence>
</complexType>
```

3.3.3 Element `<EncryptedKeyAgreementResponse>`

The `<EncryptedKeyAgreementResponse>` element contains the values *E* (n_B, K_A), *E* (s_L, s_s), *O*, *L*, C_A, C_B sent in a the key agreement using encryption credentials request

The following schema defines the `<EncryptedKeyAgreementResponse>` element:

```
<complexType name="EncryptedKeyAgreementResponse">
   <sequence>
      <element name="Context" type="s0:Context"/>
      <element name="EncryptedNonce" type="cryptoBinary"
                    minOccurs="1" maxOccurs="1"/>
      <element name="EncryptedSecret" type="cryptoBinary"
                    minOccurs="1" maxOccurs="1"/>
   </sequence>
</complexType>
```

3.4 Key Agreement With Signature Credentials

3.4.1 Element `<DiffieHellemanPublic>`

The `<DiffieHellemanPublic>` element specifies the public Diffie-Helleman parameters.

The following schema defines the `<DiffieHellemanPublic`> element:

```
<complexType name="DiffieHellemanPublic">
   <sequence>
      <element name="Modulus" type="cryptoBinary"
                  minOccurs="1" maxOccurs="1"/>
      <element name="Public" type="cryptoBinary"
                  minOccurs="1" maxOccurs="1"/>
   </sequence>
</complexType>
```

3.4.2 Element `<SignedKeyAgreementData>`

The `<SignedKeyAgreementData>` specifies the data that is signed to create the value *S ((D_A, C_A, O'), k_A)* in a request and *S ((D_B, ,O, L, C_A, C_B), k_B)* in a response.

The following schema defines the `<SignedKeyAgreementData>` element:

```
<complexType name="SignedKeyAgreementData">
   <sequence>
      <element name="Context" type="s0:Context"/>
      <element name="Context" type="s0:DiffieHellemanPublic"/>
   </sequence>
</complexType>
```

3.4.3 Element `<SignedKeyAgreement>`

The `<SignedKeyAgreement>` element contains the value *S ((D_A, C_A, O'), k_A)* sent in a the key agreement using signature credentials request.

The following schema defines the <SignedKeyAgreement> element:

```
<complexType name="SignedKeyAgreement">
   <sequence>
      <element name="Context" type="s0:SignedKeyAgreementData"/>
      <element name="Context" type="ds:Signature"/>
   </sequence>
</complexType>
```

3.4.4 Element `<SignedKeyAgreementResponse>`

The <SignedKeyAgreementResponse> element contains the values *S ((D_B, ,O, L, C_A, C_B), k_B)*, *E (s_L, s_s)* sent in a the key agreement using signature credentials request

The following schema defines the <SignedKeyAgreementResponse> element:

```
<complexType name="SignedKeyAgreement">
   <sequence>
      <element name="SignedKeyAgreementData"
```

```
               type="s0:SignedKeyAgreementData"/>
      <element name="Signature" type="ds:Signature"/>
      <element name="EncryptedSecret" type="s0:EncryptedSecret"/>
   </sequence>
</complexType>
```

Appendix A Algorithm and Protocol Identifiers

A.1 Message Digest

[These values as defined in XML Signature]

SHA-1

A.2 Message Authentication Code

[These values as defined in XML Signature]

HMAC-SHA1

A.3 Symmetric Encryption

[These values as defined in XML Encryption]

AES

3DES

A.4 Public Key Signature

[These values as defined in XML Signature]

RSA

DSA

A.5 Public Key Encryption

[These values as defined in XML Encryption]

RSA

A.6 XML Canonicalization

[These values as defined in XML Signature]

Canonical-XML

A.7 Cryptographic Protocols

At present there is no standard identifier for IETF cryptographic protocols other than the RFC number. Many protocols are specified in multiple documents. We use the http retrieval locator for the principal RFC specifying the protocol in the IETF repository as the identifier for the corresponding cryptographic protocol.

The following identifiers are defined

IPSEC
http://www.ietf.org/rfc/rfc2401.txt (v1.0)

DNSSEC
http://www.ietf.org/rfc/rfc2535.txt (tsig)
http://www.ietf.org/rfc/rfc2931.txt (sig(0))

S/MIME
http://www.ietf.org/rfc/rfc2630.txt (CMS)
http://www.ietf.org/rfc/rfc2311.txt (v2.0)
http://www.ietf.org/rfc/rfc2633.txt (v3.0)

SSL 3.0 / TLS
http://www.ietf.org/rfc/rfc2246.txt (TLS)

Kerberos
http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-revisions-08.txt (v5.0)

Appendix B Examples

B.1 Key Exchange Using RSA Encryption Credentials

Request

Response

B.2 Key Exchange Using RSA Signature Credentials

Request

Response

B.3 Key Exchange Using DSA Signature Credentials

Request

Response

Appendix C References

[Schneier] B. Schneier, *Applied Cryptography 2nd Edition*

[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

[WSDL] 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

[XKMS] et al. *XML Key Management Specification 1.1*, W3C Note XX XXX 2001

[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-SIG-XSD] XML Signature Schema available from http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/xmldsig-core-schema.xsd.

[XML-Enc] *XML Encryption Specification*, In development.

[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/2000/WD-xmlschema-1-20000922/, latest draft at 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/2000/WD-xmlschema-2-20000922/, latest draft at http://www.w3.org/TR/xmlschema-2/

Appendix D Legal Notices

Copyright

Intellectual Property Statement

Disclaimer