Internet Draft
Document: draft-hallambaker-oathxkms-00.txt
Expires: August 2006

P. Hallam-Baker
VeriSign Inc.
Jeff Bohren
BMC Software
February 2006

XKMS Provisioning of OATH Shared Secret Keys

Status of this Memo

Abstract

A means of provisioning OATH shared secret OTP parameters based on the XKMS protocol is described. The techniques used may be extended to support XKMS registration of symmetric keys for other cryptographic protocols. It is hoped that publication of this note will allow others to make use of the same architectural approach as a starting point for future proposals.

This work is a joint effort by the members of OATH (Initiative for Open AuTHentication) to specify an algorithm that can be freely distributed to the technical community. The authors believe that a common and shared specification will facilitate adoption of two-factor authentication on the Internet by enabling interoperability commercial and open-source implementations.

Contents

1. Conventions used in this document

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

2. Requirements

We consider three specific use cases that are believed to be indicative of the situations in which the provisioning protocol might be employed. These use cases are intended to be illustrative rather than exhaustive. For example a token issuer that purchases a large number of tokens intending to re-initialize them before issue is likely to find that their particular situation is closer to the manufacturing use case than initialization before issue.

2.1 Use Cases

The provisioning protocol is designed to meet the needs of three principal use cases corresponding to initialization at three different stages in the token life cycle.

Direct Static Provisioning (Bulk initialization)

In most cases an OATH token will be initialized during manufacture. This is likely to be the case even if the purchaser of the token is likely to re-initialize the token on receipt. The Device Initialization Tool

obtains one or more credentials from a Credential Issuer. These are then installed in the OTP devices by the manufacturing equipment (Figure 1).

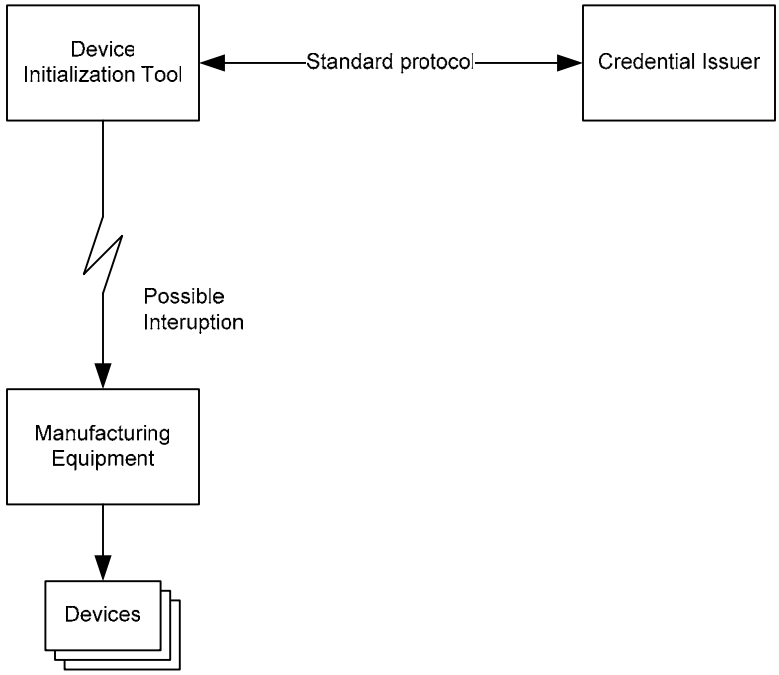

Figure 1: Bulk initialization Use Case

The requirement for interoperability and thus the scope of the standards proposal in this particular use case is limited to communications between the Device Initialization Tool and the Credential Issuer. Credential issuers require the ability to service devices produced by multiple manufacturers. Manufacturers require the ability to obtain credentials from multiple issuers at different times. Both parties want to minimize partner specific configuration.

In a manufacturing environment reliability and throughput are critical. A provisioning protocol that requires the manufacturing machinery to perform complex or lengthy calculation is unacceptable.

Manufacturing environments are frequently hostile to network communications and the ability to continue production during brief network outages is generally considered essential.

> ***Requirement:*** *Standards based protocol for communication between Device Initialization Tool and the Credential Issuer.*
>
> ***Requirement:*** *Bulk initialization of multiple tokens*
>
> ***Requirement:*** *Protocol must not require the manufacturing equipment to perform complex or lengthy calculation*
>
> ***Inferred Requirement:*** *Server generated keys*

Issuance to OTA Provisioning Service:

The Credential Issuer supplies OATH authentication credentials to an Over-the-Air (OTA) Provisioning Service operated by a external entity (e.g., application service provider, mobile operator, or enterprise customer), for subsequent delivery of individual credentials to end user mobile devices. The provisioning service receives stores credentials securely for future wireless download to the user device.

Requirement: *Supports bulk (pre-ordered range of tokens) and on-demand (real-time request/response) issuance.*

Requirement: *Allows token IDs to be assigned by either the credential issuer or the provisioning service.*

Administrator initialization before issue

While bulk initialization under controlled conditions during manufacture is likely to meet the security needs of most applications reliance on a pre-disclosed secret is unacceptable to some circumstances, in particular tokens issued for classified government use.

In such cases the token issuer requires the ability to remove all the secret information installed on the token during manufacture and replace it with secret keys established under conditions controlled by the issuer.

It is however in most cases impractical for the administrator to apply a physical marking to the token itself such as a serial number. It is therefore necessary for the enrolment process to communicate the token serial number to the provisioning service. Another situation in which initialization before use is required is the case where the OTP functionality is installed on a previously manufactured device as software (Figure 2).

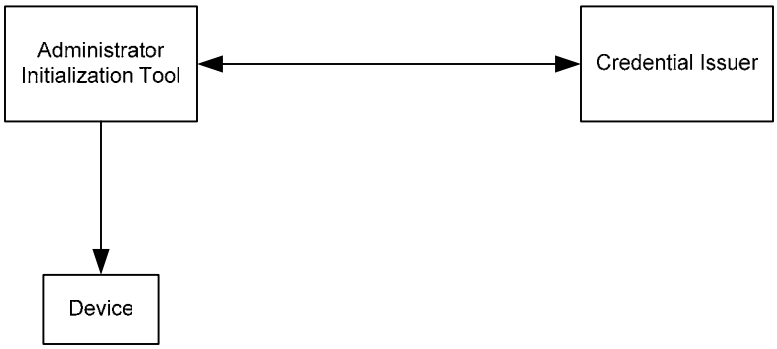

Figure 2: Initialization before issue Use Case

Requirement: *Administrator Initialization Tool to communicate a token serial number or other physical indicata to credential issuer.*

Requirement: *Administrator Initialization Tool to agree new shared secret information with the credential issuer.*

Requirement: *Administrator to be able to demonstrate that the shared secret information in the token has been generated in compliance with security practices.*

Requirement: *Credential Issuer to be able to demonstrate that the shared secret information in the token has been generated in compliance with security practices.*

Architectural Approaches

The manufacturing use case clearly points to the use of a provisioning protocol where the secret key material is generated by the provisioning service. This corresponds to 'server generated keys' in terms of XKMS.

While this approach may be appropriate in some after-issue situations it does not meet all the requirements identified in the after-issue use cases. These use cases suggest the need for the client to participate in the generation of the secret key material. This corresponds to a new mode of key generation that is at present only applicable to symmetric key protocols in which both the client and server make a verifiable contribution to the randomness of the shared secret.

2.2 Constraints

Existing open standards

Design, development and review of cryptographic security protocols is an expensive and time consuming business requiring access to limited resources. Use of existing open standards is preferred whenever possible and appropriate.

Constraint: *The protocols chosen should be based on existing open standards wherever possible*

Web Services compliant

Web Services are increasingly the platform of choice for network application protocols. In the case of a device

Constraint: *The protocols chosen should be based on the Web Services architecture*

Support low speed devices

Most of the devices used to enroll in the protocol are highly constrained offering minimal computation capability and communication bandwidth. The overhead involved in XML processing in particular is greater than these devices are typically capable of supporting.

Constraint: *It should be possible for a low speed device to participate in the protocol even if it is not capable of supporting an XML processing stack.*

PKI provisioning support

Combination tokens offer both OTP and PKI authentication. It is clearly desirable to allow provisioning of the PKI secret key and the OTP shared secret to be supported by a common infrastructure.

> ***Constraint:*** *The protocols chosen for provisioning the OTP shared secret should be capable of provisioning PKI secret keys.*

3. The XKMS Key Registration Service Specification

The use cases and constraints are best met by an extension to the XKMS Key Registration Service Specification (XKRSS) [XKMS]

XKRSS is an open standard agreed by the World Wide Web Consortium. The XKMS protocol is a Web Service designed to support provisioning of secret keys for PKI in both end user and bulk manufacture applications.

3.1 Extension to Shared Secret Keys

From an architectural point of view the problem of provisioning a shared secret is very similar to the problem of provisioning a secret key for use with an asymmetric algorithm.

- The secret key material must be transported in a form which protects against disclosure to any third party.
- At the end of the protocol the secret material must be established at the client.
- The security of protocols in which the secret key is employed will depend on the secret being generated in an appropriately secure manner, in particular the use of a sufficiently random and un-guessable seed.

There are also important differences:

- When provisioning a shared-secret there is no ‘public key’ attribute to be considered.
- At the end of the protocol the secret key material must be established at the provisioning service

3.2 Secret Key Identifier

The purpose of a PKI provisioning protocol is generally described in terms of binding a collection of attributes such as a certificate holder’s name to the public key.

In a shared-secret architecture there is no public key to bind the attributes to. There is however a surrogate that serves the same function. When an authentication service receives a request to perform an authentication against a shared-secret an identifier must be provided to inform the registry which shared secret is to be used.

A shared-secret identifier is a name: the choice of name is arbitrary and semantics of the identifier are defined exclusively by the registry. A public key is not a name: the choice is not arbitrary as it must be the mathematical compliment of the private key itself

3.3 Key Generation

The requirement to establish the shared-secret at the provisioning service has important consequences for the choice of key generation mechanism. In a PKI provisioning infrastructure the disclosure of the private key to the provisioning service is a design choice that is appropriate in some cases (e.g. escrow of encryption keys) and inappropriate in others (e.g. signature or authentication keys).

In a shared secret provisioning infrastructure the shared secret must be disclosed to the provisioning infrastructure by definition, otherwise it has not been ‘shared’.

XKMS supports two modes of key generation, at the client and at the server. Server side key generation is primarily intended for bulk issue and applications where the key is to be escrowed. If the private key is not to be escrowed by the service it is generally preferable to generate it at the client and avoid the need to transport it over the network at all.

Server Side Key Generation

Bulk issue of shared-secret keys is no different in principle to bulk issue of PKI keys. The ability to use a single protocol to support both types of provisioning is clearly a benefit to manufacturers and issuers requiring bulk re-initialization. The only additional requirement needed to support provisioning of OATH shared secrets is to define an appropriate shared secret container.

Client Side Key Generation: Not applicable

The client side key generation protocol defined in XKMS does not meet the needs of shared-secret generation described in the use cases and does not appear to offer any benefit over server generated keys when a symmetric key algorithm is involved.

Mutual Key Generation

In a mutual key generation the client and service both contribute random material to the generated key. This allows both parties to ensure that the generated key is sufficiently random and un-guessable.

The key generation mechanism defined employs a Diffie-Hellman key exchange.

- The client generates a random private key *c* and calculates the corresponding public key e^c.

- The client public key e^c is communicated to the service.

- The service generates a random private key *s* and calculates the corresponding public key e^s.
- The service calculates the shared secret $H(e^{cs})$
- The service public key e^s is communicated to the client.
- The client calculates the shared secret $H(e^{cs})$

The established shared secret contains random information contributed by both the client and the service. The shared secret will be sufficiently random provided that one of the parties generated a sufficiently random private key.

Although the Diffie-Hellman key exchange is considered to be a secure public key exchange algorithm the security of the protocol against a disclosure attack requires both parties to choose a sufficiently random private key. The use of a secure transport protocol is therefore required.

4. XKRSS Profile

4.1 Identifiers

The following identifiers defined by external sources are used in this document:

http://www.w3.org/2002/03/xkms#
: XKMS Schema

http://www.w3.org/2000/09/xmldsig#
: XML Signature Schema

http://www.w3.org/2001/04/xmlenc#DHKeyValue
: Diffie-Hellman key agreement

The following identifiers are defined and used in this document:

urn:ietf:rfc:4226
: The OATH OTP specification

The following identifiers are defined and used in the Internet draft Portable Shared Secret Container [Container]

http://www.openauthentication.org/2006/02/PSKC
: Schema used to encode the HOTP shared secret parameter values before encryption.

4.2 SOAP Binding

Any of the bindings described in the document *XML Key Management Specification (XKMS 2.0) Bindings* may be employed by a compliant implementation.

The use of the following specific bindings is **RECOMMENDED**:

SOAP Binding:	SOAP 1.2, SOAP over HTTP binding
Confidentiality Binding:	SSL/TLS or WS-Security
Authentication Binding:	Payload authentication (if required)

The XKMS specification does not define a WS-Security binding as the design predates it. The use of WS-Security is preferred over TLS as it provides support for firewall infrastructure.

4.3 Authentication Binding

The choice of payload authentication or a limited use shared secret for authentication is determined by the application:

- Payload authentication using a pre-established public key is preferred for bulk issue and administrative issue. Payload authentication allows multiple registration requests to be authenticated using a single signature.

- End-users cannot be expected to have an authentication mechanism until after the registration process has been completed. In these cases the KeyBindingAuthentication mechanism is **RECOMMENDED**.

4.4 Request

Prototype Key Binding

The prototype key binding contains information that the client requests be present in the keybinding resulting from the request.

A <KeyUsage> element MUST specify authentication (signature) as an authorized use:
http://www.w3.org/2002/03/xkms#Signature

A <RespondWith> element MUST request that a Diffie-Hellman
http://www.w3.org/2001/04/xmlenc#DHKeyValue

A <UseKeyWith> element of the prototype key binding **MUST** specify the use of the OATH OTP protocol: urn:ietf:rfc:4226.

The <ds:KeyInfo> element **SHOULD** contain a <ds:KeyName> element contains an identifier for the token to be registered. This identifier need not be a physical identifier present on the token but will be the identifier used by the authentication service to identify the token in an authentication request. Administrators may make use of the <ValidityInterval> and <RevocationCodeIdentifier> elements to provide additional control over use of the token.

Additional Key Elements **MAY** be specified by including a portable key container as defined by [Container] within the <ds:KeyInfo> element

A revocation code identifier is a self authenticating information code that tells the provisioning service to revoke the corresponding key binding. Use of the revocation code does not require either authentication or a confidential channel to the provisioning service. This allows a multi-function device to provide a means of securely, reliably and verifiably terminating the validity of the key binding. The shared secret **MUST NOT** be used as the revocation code. A one way hash of the revocation code **MAY** be used.

Server Key Generation

A <RespondWith> value **MUST** specify that a private key value is requested using the identifier http://www.w3.org/2002/03/xkms#PrivateKey

Mutual Key Generation

A <RespondWith> value **MUST** specify that a request for a mutually agreed key using the identifier: urn:ietf:rfc:4226

The <ds:KeyInfo> element of the <PrototypeKeyBinding> **MUST** include a <ds:KeyValue> element that specifies the Diffie-Hellman public key parameters of the client.

The requestor SHOULD specify a random value for the <Data> element. The same value is returned in the encrypted Private key parameters in the response allowing the requestor to verify that the value of the key agreement generated at the server matches.

4.5 Response

Key Binding

The KeyBinding returned by the provisioning service must meet the same constraints as the PrototypeKeyBinding specified in the request.

Server Key Generation

If the request is successful the <PrivateKey> element MUST be present and MUST contain the encrypted OTP parameters encoded under the schema described in [Container]

Mutual Key Generation

If the request is successful the <ds:KeyInfo> element of the returned <KeyBinding> MUST include a <ds:KeyValue> element that specifies the Diffie-Hellman public key parameters newly generated by the service.

The shared secret is derived by applying the Diffie-Hellman key agreement algorithm as specified in XML Encryption. The initial counter value is 0.

The <PrivateKey> element MAY be present and MAY contain the encrypted OTP parameters encoded under the schema described in [Container]. The result of the Diffie-Hellman key agreement replaces the value specified for the <Data> element if specified.

Appendix A: Example Messages (Non normative)

This section provides non-normative examples of using the registration protocol. For clarity the XKMS transport and message level security bindings are omitted.

A.1 Server Generated Key

This example meets the requirements set out in the 'bulk initialization' use case. A client at the manufacturing site makes a request for one or more private keys to be created and bound to a specified token identifier.

For efficiency authentication and encryption are performed in this particular instance using a pre-shared key. The pre-shared key might be established through out of band configuration or by means of a standard key agreement protocol. Alternatively public key protocols might be used for authentication and encryption.

The pre-shared key used for MAC generation and encryption is: 3n9cjk4jks04jfw0934jsr09jwik4

Request

The request message is:

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="I652c3870fcdc9d8259c554ace9175846"
    Service="http://test.xmltrustcenter.org/XKMS"
    xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2002/03/xkms#PrivateKey</RespondWith>
  <PrototypeKeyBinding Id="I7d5d848c7a01682d22412b9e3ae74de7">
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:4226"
        Identifier="ALNG000DE8FA/001" />
  </PrototypeKeyBinding>
  <Authentication>
    <KeyBindingAuthentication>
      <ds:Signature>
```

```
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
       Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
          <ds:SignatureMethod
       Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
          <ds:Reference URI="#I7d5d848c7a01682d22412b9e3ae74de7">
            <ds:DigestMethod
       Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
             <ds:DigestValue>/D4ocLWp/6RxP6XkH9qn5BvqGz0=
              </ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>0l9rxrH1dSd6B5E5RYKs5yVc+FM=
          </ds:SignatureValue>
      </ds:Signature>
    </KeyBindingAuthentication>
  </Authentication>
</RegisterRequest>
```

Response

The response contains a private key encrypted under the specified private key.

The response message is:

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="Id9daf1016925c55dac5030797eb8d244"
    Service="http://test.xmltrustcenter.org/XKMS"
    ResultMajor="http://www.w3.org/2002/03/xkms#Success"
    RequestId="I652c3870fcdc9d8259c554ace9175846"
    xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="Ice130d2ae662b7644eb3f3654d929db1">
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:4226"
       Identifier="ALNG000DE8FA/001" />
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid">
      <ValidReason>http://www.w3.org/2002/03/xkms#RevocationStatus</ValidReason>
      <ValidReason>http://www.w3.org/2002/03/xkms#ValidityInterval</ValidReason>
    </Status>
  </KeyBinding>
  <PrivateKey>
    <xenc:EncryptedData>
      <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <xenc:CipherData>
        <xenc:CipherValue>Hnp23IfB9Vpt5f4A6392Lqk3+h+Y999rJhpiexi+
        xXEWokE1ntr0Z4q4u36hRy0PxAnUEovMpccfDwyJhKqooPrvOyfzX0QyGr
        9Tlk2hax/siY3vOiRWl3fLYtuJgiI/wtu/6lvWZfv6Q8x4wBfQyrTFNyUR
        eMmtBu7A/Hskk5mAGcvp4kZC/MTYJXfmvJpOiBiTm1nhoreoLDib5VKihJ
        qDFuOcju/ONMN0kWM2rBqdmSq5k+xmZhf/qMx3b7ttygSyNuoKj2GVTmyL
        ueGmiFqmQ/3Ek5oduYXo1GYPiibOz6HIyKVZRfyrtgm2lWB92bSYPS8GsR
        jloIbs+nG4GmfTcD2knbCByVV0NIwI9Qxam1EZSAGtxowtnuYheWDYRj15
        4O6LuqRNxTqr4oiQyU1Yl7x/mKTz</xenc:CipherValue>
```

```
      </xenc:CipherData>
    </xenc:EncryptedData>
  </PrivateKey>
</RegisterResult>
```

A.2 Mutual Generated Key

This example meets the requirements set out in the use cases describing registration after manufacture.

The shared secret is generated by means of the mutual key establishment mechanism. The initial request is authenticated in this particular case by means of a one time use pre-shared secret.

The pre-shared secret used to authenticate the request is 024837

Request

The client makes use of a pre-generated Diffie-Hellman parameter set. A new private key is generated randomly. In big-endian hexadecimal format this key is:

```
A Private =
    b483aec8 dc6978e6 7157c363 33e6babc a62c6082 e0d31eeb
    698432b9 f3b72d13 b544e309 223728f2 82df47d9 9885a073
    02f03990 21bf40fc 313d52ce 65d686b0 258a59f9 f847db5f
    d7d03796 f9021d7f 1fcb66c6 280aaed2 eb07cc5e 6f89e9ae
    2d4df737 69a18d59 b4dda0ff 5694d3de 7a27141c b4e6e83e
    0df5b110 bf36b274
```

The request message is:

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterRequest xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="Ib9b0386caf44d6c6df3e4eec9f895c30"
    Service="http://test.xmltrustcenter.org/XKMS"
    xmlns="http://www.w3.org/2002/03/xkms#">
  <RespondWith>http://www.w3.org/2001/04/xmlenc#DHKeyValue</RespondWith>
  <PrototypeKeyBinding Id="I62f4453d6d4ed6bc9dd856e52d5eaa20">
    <ds:KeyInfo>
      <ds:KeyValue>
        <xenc:DHKeyValue>
          <xenc:P>4RG/7j64smZYmNL6Y4YdiMfZhJ8CCGqJOy1Qe3b9l/XO8ATv
          fXdQJ+xdR+UQ0q5BTxZWgSWL/fFzQhH3V0GmeJZ4gpIC6OTqBJBnkSm/
          ix85NUV8JLIatjjFTeAS6iCe06yTgSg1pmAC1TGLeU3MuPrAMOruzojq
          7BLhBtlV59c=</xenc:P>
          <xenc:Q>togFf3+94+GC+L+fbLWXqQC6yMAghpSnb7Zskn9T4n4n+qGS
          GOwJoLE44Gx2iQaSTDfbGxONNZnwxh1Zd+ZBwQ==</xenc:Q>
          <xenc:Generator>Rsp0nalaVmr5ggzfPMnkEbb5sdV2RDHCwKNL92tt
         24eBOT9OfjvV/5z5TFQa+QM9FyNYM+GeNfQImW37PkzNv1kjDB9IuAw1Z
         eIvEDKIyg86P2A/xOhrZ/26cwR0LMsRRSGDZlaOZsXgnMIUR7Zokjialv
         Lz4xKB7Dis5gGMYW8=
          </xenc:Generator>
          <xenc:Public>YGAcOiFj6QN15JRohMVuP3Zh0xREgoWCT7mbFToW1dO
```

```
          SNygQRnbdZzdOKvTBT1caEs8Cm/sPmjiNMiz/S3o+hdU1yDHuoVHS58W
          FI7Ns6XC7pNMQHW08L7vlhJ+K8HyABKnWFPx7Uckwj7tfmfW5i9xDEgb
          GoHM8U4mx7Hcw8tk=
          </xenc:Public>
        </xenc:DHKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:4226"
      Identifier="ALNG000DE8FA/001" />
  </PrototypeKeyBinding>
  <Authentication>
    <KeyBindingAuthentication>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
          <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
          <ds:Reference URI="#I62f4453d6d4ed6bc9dd856e52d5eaa20">
            <ds:DigestMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>vZrZ/snvEhb8OcyNKJ5577cCigQ=
               </ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>K7K6OA8j8ub3HV/fQmoHGhurLtE=
          </ds:SignatureValue>
      </ds:Signature>
    </KeyBindingAuthentication>
  </Authentication>
</RegisterRequest>
```

Response

The server private key is:

```
B Private =
    b483aec8 dc6978e6 7157c363 33e6babc a62c6082 e0d31eeb
    698432b9 f3b72d13 b544e309 223728f2 82df47d9 9885a073
    02f03990 21bf40fc 313d52ce 65d686b0 258a59f9 f847db5f
    d7d03796 f9021d7f 1fcb66c6 280aaed2 eb07cc5e 6f89e9ae
    2d4df737 69a18d59 b4dda0ff 5694d3de 7a27141c b4e6e83e
    0df5b110 bf36b274
```

The response message is:

```
<?xml version="1.0" encoding="utf-8"?>
<RegisterResult xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  Id="I9da626389715f998e75cbe1456514c89"
  Service="http://test.xmltrustcenter.org/XKMS"
  ResultMajor="http://www.w3.org/2002/03/xkms#Success"
  RequestId="Ib9b0386caf44d6c6df3e4eec9f895c30"
```

```
  xmlns="http://www.w3.org/2002/03/xkms#">
  <KeyBinding Id="I40aa6a1795ecb38e14f2db58f3f5874d">
    <ds:KeyInfo>
      <ds:KeyValue>
        <xenc:DHKeyValue>
          <xenc:Public>YGAcOiFj6QN15JRohMVuP3Zh0xREgoWCT7mbFToW1d
          OSNygQRnbdZzdOKvTBT1caEs8Cm/sPmjiNMiz/S3o+hdU1yDHuoVHS5
          8WFI7Ns6XC7pNMQHW08L7vlhJ+K8HyABKnWFPx7Uckwj7tfmfW5i9xD
          EgbGoHM8U4mx7Hcw8tk=</xenc:Public>
        </xenc:DHKeyValue>
      </ds:KeyValue>
    </ds:KeyInfo>
    <KeyUsage>http://www.w3.org/2002/03/xkms#Signature</KeyUsage>
    <UseKeyWith Application="urn:ietf:rfc:4226"
      Identifier="ALNG000DE8FA/001" />
    <Status StatusValue="http://www.w3.org/2002/03/xkms#Valid">
      <ValidReason>http://www.w3.org/2002/03/xkms#RevocationStatus</ValidReason>
  <ValidReason>http://www.w3.org/2002/03/xkms#ValidityInterval</ValidReason>
    </Status>
  </KeyBinding>
</RegisterResult>
```

Key Generation

The mutually agreed key is:

```
Shared Key =
    295b12b1 ea56c534 a6d52b53 9d147c14 b438d161 9a2cc6c3
    7e46f99d b450b93b c44778ce 64cd99ee a4b68a74 23289571
    1ffaaadd f6483c60 d31bb8f5 80184540 7d95b171 4f485a08
    92d9f58a 7a0885c6 c811c392 d42d6d73 a3e5b023 e2928b15
    76f3bfe9 c9fc7bfc 8d8d3a2f 96e24582 59b9f273 ee40f3c4
    e2a0ba6c 835fbab8
```

The algorithm specified in [XML-ENC] is then used to derive the value for the token secret:

```
    Keying Material = KM(1) | KM(2) | ...
    KM(counter) = DigestAlg ( Shared Key | counter | EncryptionAlg | KeySize )
```

The value KA-Nonce is not used.

The string (counter | EncryptionAlg | KeySize) is:

```
    http://www.w3.org/2000/09/xmldsig#sha101urn:ietf:rfc:4226160
```

The value of the token secret is:

```
    b20d1e4ffb796685d95340b33fdd6dde695ba6a0
```

Notices

References

[Container] *Portable Shared Secret Container*, Internet Draft, To be published

[OATH] *OATH Reference Architecture Version 1.0*, Initiative for Open Authentication. 2005.

[XKMS] Phillip Hallam-Baker, Shivaram H. Mysore, *XML Key Management Specification (XKMS 2.0) Version 2.0*, W3C Recommendation 28 June 2005, http://www.w3.org/TR/2005/REC-xkms2-20050628/

[XKMS-B] Phillip Hallam-Baker, Shivaram H. Mysore, *XML Key Management Specification (XKMS 2.0) (Bindings) Version 2.0*, W3C Recommendation 28 June 2005, http://www.w3.org/TR/2005/REC-xkms2-bindings-20050628/

[XML-SIG] D. Eastlake, J. R., D. Solo, M. Bartel, J. Boyer , B. Fox , E. Simon. *XML-Signature Syntax and Processing*, W3C Recommendation, 12 February 2002. http://www.w3.org/TR/xmldsig-core/.

[XML-ENC] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, E. Simon, *XML Encryption Syntax and Processing*, W3C Recommendation, 10 December 2002, http://www.w3.org/TR/xmlenc-core/.

[WS-Security] A. Nadalin et al., Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS Standard, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.1.pdf.

[RFC 2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997

Acknowledgments

The authors would like to thank members of OATH (Initiative for Open AuTHentication) for their help during the conception and development of this document.

The authors of this document would like to emphasize the role of persons who have made a key contribution to this document:

- Siddharth Bajaj, is the Director of Advanced Products and Research at Verisign, Inc. He is also the chair of the Technology Focus Group at OATH.

- Stu Vaeth, is the Chief Security Officer at Diversinet, Inc. He is also the co-chair of the Technology Focus Group at OATH

Without their advice and valuable inputs, this document would not be the same.

Authors' Addresses

Phillip Hallam-Baker
VeriSign Inc.
pbaker@verisign.com

Jeff Bohren
BMC Software.
6526 Thoroughbred Loop, Odessa, FL
Jeff_Bohren@bmc.com