keyprov
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2008

P. Hoyer
ActivIdentity
M. Pei
VeriSign
S. Machani
Diversinet

February 22, 2008

Portable Symmetric Key Container
draft-ietf-keyprov-portable-symmetric-key-container-03.txt

Status of this Memo

Copyright Notice

Abstract

This document specifies a symmetric key format for transport and provisioning of symmetric keys (One Time Password (OTP) shared secrets or symmetric cryptographic keys) to different types of strong authentication devices. The standard token format enables enterprises to deploy best-of-breed solutions combining components from different vendors into the same infrastructure.

This work is a joint effort by the members of OATH (Initiative for Open AuTHentication) to specify a format 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 between commercial and open-source implementations.

Table of Contents

1. Introduction

With increasing use of symmetric key based authentication systems such as systems based one time password (OTP) and challenge response mechanisms, there is a need for vendor interoperability and a standard format for importing, exporting or provisioning symmetric keys from one system to another. Traditionally authentication server vendors and service providers have used proprietary formats for importing, exporting and provisioning these keys into their systems making it hard to use tokens from vendor A with a server from vendor B.

This Internet draft describes a standard format for serializing symmetric keys such as OTP shared secrets for system import, export or network/protocol transport. The goal is that the format will facilitate dynamic provisioning and transfer of a symmetric keys such as an OTP shared secret or an encryption key of different types. In the case of OTP shared secrets, the format will facilitate dynamic provisioning using an online provisioning protocol to different flavors of embedded tokens or allow customers to import new or existing tokens in batch or single instances into a compliant system.

This draft also specifies the key attributes required for computation such as the initial event counter used in the HOTP algorithm [HOTP]. It is also applicable for other time-based or proprietary algorithms.

To provide an analogy, in public key environments the PKCS#12 format [PKCS12] is commonly used for importing and exporting private keys and certificates between systems. In the environments outlined in

this document where OTP keys may be transported directly down to smartcards or devices with limited computing capabilities, a format with small (size in bytes) and explicit shared secret configuration attribute information is desirable, avoiding complexity of PKCS#12. For example, one would have to use opaque data within PKCS#12 to carry shared secret attributes used for OTP calculations, whereas a more explicit attribute schema definition is better for interoperability and efficiency.

2. 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 [RFC2119].

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

In the text below, OTP refers to one time password.

3. Use Cases

This section describes a comprehensive list of use cases that inspired the development of this specification. These requirements were used to derive the primary requirement that drove the design. These requirements are covered in the next section.

These use cases also help in understanding the applicability of this specification to real world situations.

3.1. Offline Use Cases

This section describes the use cases relating to offline transport of keys from one system to another, using some form of export and import model.

3.1.1. Key migration by end-user

A user wants to migrate a key from one authentication token (container) to a different authentication token. For example, the authentication tokens may be soft tokens on two different systems (computers or mobile phones). The user can export the key and related data in a standard format for import into the other authentication token.

The key protection mechanism may rely on password-based encryption for soft tokens, a pre-placed hardware-protected transfer key shared between the two systems or may also rely on asymmetric keys/ PKI if available.

3.1.2. Bulk import of new keys

Tokens are manufactured in bulk and associated keys and algorithm data need to be loaded into the validation system through a file on portable media. The manufacturer provides the keys and related data in the form of a file containing records in standard format, typically on a CD. Note that the token manufacturer and the vendor for the validation system may be the same or different.

In this case the file usually is protected by a symmetric transport key which was communicated separately outside of the file between the two parties.

Some devices will allow local PIN management (the device will have a PIN pad) hence random initial PINs set at manufacturing should be transmitted together with the respective keys they protect.

3.1.3. Bulk migration of existing keys

An enterprise wants to port keys and related data from an existing validation system A into a different validation system B. The existing validation system provides the enterprise with a functionality that enables export of keys and related data (e.g. for OTP tokens) in a standard format. Since the OTP tokens are in the standard format, the enterprise can import the token records into the

new validation system B and start using the existing tokens. Note that the vendors for the two validation systems may be the same or different.

In this case the file usually is protected by a symmetric transport key which was communicated separately outside of the file between the two validation systems.

In this case it is also important to be able to communicate the existing assignment (binding) of a device to a specific user.

3.1.4. Key upload case

User wants to activate and use a new key and related data against a validation system that is not aware of this key. This key may be embedded in the authentication token (e.g. SD card, USB drive) that the user has purchased at the local electronics retailer. Along with the authentication token, the user may get the key on a CD or a floppy in a standard format. The user can now upload via a secure online channel or import this key and related data into the new validation system and start using the key.

The key protection mechanism may rely on password-based encryption, or a pre-placed hardware-protected transfer key shared between the token manufacturer and the validation system(s) if available.

3.2. Online Use Cases

This section describes the use cases related to provisioning the keys using some form of a online provisioning protocol.

3.2.1. Online provisioning a key to end-user's authentication token

A mobile device user wants to obtain an OTP key for use with an OTP soft token on the device. The soft token client from vendor A initiates the provisioning process against a provisioning system from vendor B using a standards-based provisioning protocol such as [DSKPP]. The provisioning system delivers one or more keys in a standard format that can be processed by the mobile device. The user can download a payload that contains more than one key.

In a variation of the above, instead of the user's mobile phone, a

key is provisioned in the user's soft token application on a laptop using a network-based online protocol. As before, the provisioning system delivers an OTP key in a standard format that can be processed by the soft token on the PC.

3.2.2. Server to server provisioning of keys

Sometimes, instead of importing keys from a manufacturer using a file, an OTP validation server may download the keys using an online protocol. The keys can be downloaded in a standard format that can be processed by a validation system.

In another scenario, an OTA (over-the-air) key provisioning gateway that provisions keys to mobile phones may obtain key material from a key issuer using an online protocol. The keys are delivered in a standard format that can be processed by the OTA key provisioning gateway and subsequently sent to the end-user's mobile phone.

3.2.3. Online update of an existing authentication token key

The end-user or the key issuer wants to update or configure an existing key in the authentication token and requests a replacement key container. The container may or may not include a new key and may include new or updated key attributes such as a new counter value in HOTP key case, a modified response format or length, a new friendly name, etc.

4. Requirements

This section outlines the most relevant requirements that are the basis of this work. Several of the requirements were derived from use cases described above.

R1: The format MUST support transport of multiple types of symmetric keys and related attributes for algorithms including HOTP, other OTP, challenge-response, etc.

R2: The format MUST handle the symmetric key itself as well of attributes that are typically associated with symmetric keys. Some of these attributes may be

* Unique Key Identifier
* Issuer information
* Algorithm ID
* Algorithm mode
* Issuer Name
* Key friendly name
* Event counter value (moving factor for OTP algorithms)
* Time value

R3: The format SHOULD support both offline and online scenarios. That is it should be serializable to a file as well as it should be possible to use this format in online provisioning protocols

R4: The format SHOULD allow bulk representation of symmetric keys

R5: The format SHOULD allow bulk representation of PINs related to specific keys

R6: The format SHOULD be portable to various platforms. Furthermore, it SHOULD be computationally efficient to process.

R7: The format MUST provide appropriate level of security in terms of data encryption and data integrity.

R8: For online scenarios the format SHOULD NOT rely on transport level security (e.g., SSL/TLS) for core security requirements.

R9: The format SHOULD be extensible. It SHOULD enable extension points allowing vendors to specify additional attributes in the future.

R10: The format SHOULD allow for distribution of key derivation data without the actual symmetric key itself. This is to support symmetric key management schemes that rely on key derivation algorithms based on a pre-placed master key. The key derivation data typically consists of a reference to the key, rather than the key value itself.

R11: The format SHOULD allow for additional lifecycle management operations such as counter resynchronization. Such processes require confidentiality between client and server, thus could use a common secure container format, without the transfer of key material.

R12: The format MUST support the use of pre-shared symmetric keys to ensure confidentiality of sensitive data elements.

R13: The format MUST support a password-based encryption (PBE) [PKCS5] scheme to ensure security of sensitive data elements. This is a widely used method for various provisioning scenarios.

R14: The format SHOULD support asymmetric encryption algorithms such as RSA to ensure end-to-end security of sensitive data elements. This is to support scenarios where a pre-set shared encryption key is difficult to use.

5. Portable Key container definition

The portable key container is based on an XML schema definition and contains the following main entities:

1. KeyContainer entity as defined in Section 5.1

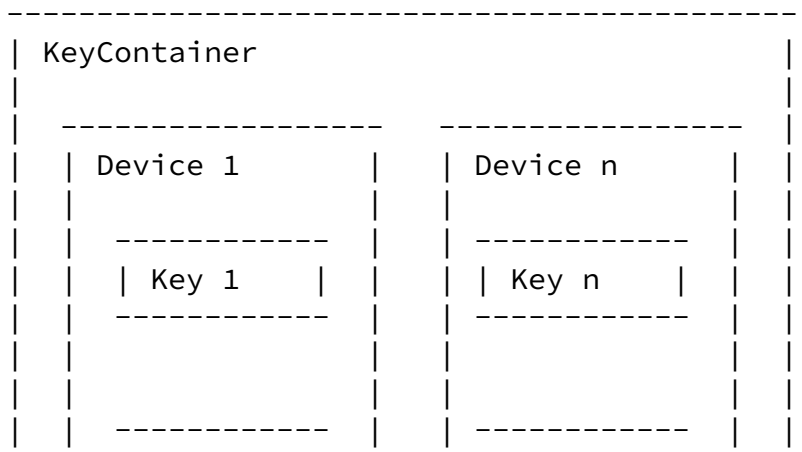
2. Device entity as defined in Section 5.2

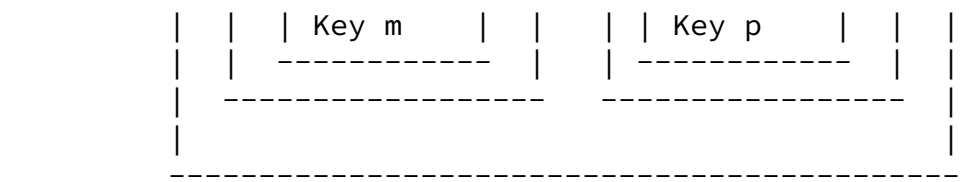
3. Key entity as defined in Section 5.3

Additionally other XML schema types have been defined and are detailed in the relevant subsections of this document

A KeyContainer MAY contain one or more Device entities. A Device MAY contain one or more Key entities

The figure below indicates a possible relationship diagram of a container.





The following section describe in detail all the entities and related XML schema elements and attributes:

5.1. KeyContainer

The KeyContainer represents the key container entity. A Container MAY contain more than one Device entity; each Device entity MAY contain more than one Key entity.

The KeyContainer is defined as follows:


```
 <xs:complexType name="KeyContainerType">
         <xs:sequence>
                 <xs:element name="EncryptionKey"
                         type="ds:KeyInfoType" minOccurs="0"/>
                         <xs:element name="MACAlgorithm"
                         type="pskc:KeyAlgorithmType" minOccurs="0"/>
                         <xs:element name="Device"
                         type="pskc:DeviceType" maxOccurs="unbounded"/>
                 <xs:element name="Signature"
                         type="ds:SignatureType" minOccurs="0"/>
         </xs:sequence>
   <xs:attribute name="Version" type="pskc:VersionType" use="required"/>
 </xs:complexType>
```

The elements of the KeyContainer have the following meanings:

- <EncryptionKey (OPTIONAL)>, Identifies the encryption key, algorithm and possible parameters used to protect the Secret Key data in the container, for profile and usage please see Section 6.1

- <MACAlgorithm (OPTIONAL)>, Identifies the algorithm used to generate a keyed digest of the the Secret Key data values when protection algorithms are used that do not have integrity checks. The digest guarantees the integrity and the authenticity of the

key data. for profile and usage please see Section 6.1.1

- <Device>, the host Device for one or more Keys as defined in Section 5.2 The KeyContainer MAY contain multiple Device data elements, allowing for bulk provisioning of keys.

- <Signature (OPTIONAL)>, the signature value of the Container. When the signature is applied to the entire container, it MUST use XML Signature methods as defined in [XMLSIG]. It MAY be omitted when application layer provisioning or transport layer provisioning protocols provide the integrity and authenticity of the payload between the sender and the recipient of the container. When required, this specification recommends using a symmetric key based signature with the same key used in the encryption of the secret key data. The signature is enveloped.

- <Version (MANDATORY)>, the version number for the portable key container format (the XML schema defined in this document).

5.2. Device

The Device represents the Device entity in the Container. A Device MAY be bound to a user and MAY contain more than one keys. It is recommended that a key is bound to one and only one Device.

The Device is defined as follows:

```
<xs:complexType name="DeviceType">
  <xs:sequence>
    <xs:element name="DeviceId" type="pskc:DeviceIdType" minOccurs="0"/>
    <xs:element name="Key" type="pskc:KeyType" maxOccurs="unbounded"/>
    <xs:element name="UserId" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The elements of the Device have the following meanings:

- <DeviceId>, a unique identifier for the device, defined in Section 5.2.1

- <Key>, represents the key entity as defined in Section 5.3

- <UserId>, optionally identifies the owner or the user of the Device TODO

5.2.1. DeviceId

The DeviceId represents the identifying criteria to uniquely identify the device that contains the associated keys. Since devices can come in different form factors such as hardware tokens, smartcards, soft tokens in a mobile phone or PC etc this type allows different criteria to be used. Combined though the criteria MUST uniquely identify the device. For example for hardware tokens the combination of SerialNo and Manufacturer will uniquely identify a device but not SerialNo alone since two different token manufacturers might issue devices with the same serial number (similar to the IssuerDN and serial number of a certificate). For keys hold on banking cards the identification of the device is often done via the Primary Account Number (PAN, the big number printed on the front of the card) and an expiry date of the card. DeviceId is an extensible type that allows all these different ways to uniquely identify a specific key containing device.

The DeviceId is defined as follows:


```
<xs:complexType name="DeviceIdType">
<xs:sequence>
    <xs:element name="Manufacturer" type="xs:string"/>
    <xs:element name="SerialNo" type="xs:string"/>
    <xs:element name="Model" type="xs:string" minOccurs="0"/>
    <xs:element name="IssueNo" type="xs:string" minOccurs="0"/>
    <xs:element name="ExpiryDate" type="xs:dateTime" minOccurs="0"/>
    <xs:element name="StartDate" type="xs:dateTime" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

The elements of DeviceId have the following meanings:

- <Manufacturer>, the manufacturer of the device.
- <SerialNo>, the serial number of the device or the PAN (primary account number) in case of EMV (Europay-MasterCard-Visa) smart cards.
- <Model>, the model of the device (e.g one-button-HOTP-token-V1)
- <IssueNo>, the issue number in case of smart cards with the same PAN, equivalent to a PSN (PAN Sequence Number).
- <ExpiryDate>, the expiry date of a device (such as the one on an EMV card,used when issue numbers are not printed on cards).
- <StartDate>, the start date of a device (such as the one on an EMV card,used when issue numbers are not printed on cards).

5.3. Key

The Key represents the key entity in the KeyContainer. The Key is defined as follows:


```
<xs:complexType name="KeyType">
  <xs:sequence>
    <xs:element name="Issuer" type="xs:string" minOccurs="0"/>
    <xs:element name="Usage" type="pskc:UsageType"/>
```

```
      <xs:element name="CardAppPersoProfileId" type="xs:string"
        minOccurs="0"/>
      <xs:element name="FriendlyName" type="xs:string" minOccurs="0"/>
      <xs:element name="Data" type="pskc:DataType" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="PINPolicy" type="pskc:PINPolicyType"
        minOccurs="0"/>
      <xs:element name="ExpiryDate" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="StartDate" type="xs:dateTime" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="KeyId" type="xs:string" use="required"/>
    <xs:attribute name="KeyAlgorithm" type="pskc:KeyAlgorithmType"
      use="required"/>
  </xs:complexType>
```

The attributes of the Key entity have the following meanings:

- KeyId (MANDATORY), a unique and global identifier of the symmetric key. The identifier is defined as a string of alphanumeric characters.

- <KeyAlgorithm (MANDATORY)>, the unique URI of the type of algorithm to use with the secret key, for profiles are described in Section 6.3

The elements of the Key entity have the following meanings:

- <Issuer (OPTIONAL)>, The key issuer name, this is normally the name of the organization that issues the key to the end user of the key. For example MyBank issuing hardware tokens to their retail banking users 'MyBank' would be the issuer. The Issuer is defined as a String.

- <Usage (MANDATORY)>, defines the intended usage of the key and related metadata as defined in Section 5.3.2

- <CardAppPersoProfileId (OPTIONAL)>, A uniquie identifier used between the sending and receiving party of the container to establish a set of constant values related to a key that are not transmitted via the container. For example a smart card application personalisation profile id related to attributes present on a smart card application that have influence when computing a response. An example could be an EMV MasterCard CAP [CAP] application on a card personalised with data for a specific

batch of cards such as:

IAF Internet authentication flag

CVN Cryptogram version number, for example (MCHIP2, MCHIP4, VISA 13, VISA14)

AIP (Application Interchange Profile), 2 bytes

TVR Terminal Verification Result, 5 bytes

CVR The card verification result

AmountOther

TransactionDate

TerminalCountryCode

TransactionCurrencyCode

AmountAuthorised

IIPB

These values are not contained within attributes in the container but are shared between the manufacturing and the validation service through this unique CardAppPersoProfileId. The CardAppPersoProfileId is defined as a String.

- <FriendlyName (OPTIONAL)>, The user friendly name that is assigned to the secret key for easy reference. The FriendlyName is defined as a String.

- <Data (OPTIONAL)>, the element carrying the data related to the key as defined in Section 5.3.1

- <PINPolicy (OPTIONAL)>, the policy of the PIN relating to the usage of this key as defined in Section 5.3.4

- <ExpiryDate (OPTIONAL)>, the expiry date of the key, it MUST not be possible to use this key after this date

- <StartDate (OPTIONAL)>, the start date of the key, it MUST not be possible to use this key before this date

5.3.1. Data (OPTIONAL)

Defines the data attributes of the symmetric key. Each is a name value pair which has either a plain value (in case of no encryption) or an encrypted value as defined in EncryptedDataType in XML Encryption.

This is also where the key value is transported, Section 7 defines a list of reserved attribute names.

Data element is defined as follows:

```
<xs:complexType name="DataType">
    <xs:sequence>
    <xs:choice>
       <xs:element name="PlainValue" type="xs:base64Binary"/>
       <xs:element name="EncryptedValue" type="xenc:EncryptedDataType"/>
    </xs:choice>
    <xs:element name="ValueMAC" type="xs:base64Binary" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="required"/>
</xs:complexType>
```

The attributes of the Data element have the following meanings:

- Name, defines the name of the name-value pair, Section 7 defines a list of reserved attribute names

The elements of the Data element have the following meanings:

- The <PlainValue> conveys an unencrypted value of the name-value pair in base 64 encoding.

- The <EncryptedValue> element in the DataType conveys an encrypted value of the name-value pair inside an EncryptedDataType as defined in XML Encryption.

- The <ValueMAC (OPTIONAL)> element in the DataType conveys a keyed MAC value of the unencrypted data for the cases where the key

protection algorithm does not support integrity checks

5.3.2. Usage (MANDATORY)

The Usage element defines the usage attribute(s) of the key entity. Usage is defined as follows:


```
  <xs:complexType name="UsageType">
      <xs:sequence>
              <xs:element name="ResponseFormat">
                      <xs:complexType>
                          <xs:attribute name="Format"
                          type="pskc:ValueFormatType" use="required"/>
                          <xs:attribute name="Length"
                          type="xs:unsignedInt" use="required"/>
                          <xs:attribute name="CheckDigits"
                          type="xs:boolean" default="false"/>
                      </xs:complexType>
              </xs:element>
              <xs:element name="ChallengeFormat" minOccurs="0">
                      <xs:complexType>
                          <xs:attribute name="Format"
                          type="pskc:ValueFormatType" use="required"/>
                          <xs:attribute name="Min"
                          type="xs:unsignedInt" use="required"/>
                          <xs:attribute name="Max"
                          type="xs:unsignedInt" use="required"/>
                          <xs:attribute name="CheckDigits"
                          type="xs:boolean" default="false"/>
                      </xs:complexType>
              </xs:element>
      </xs:sequence>
      <xs:attribute name="OTP" type="xs:boolean" default="false"/>
      <xs:attribute name="CR" type="xs:boolean" default="false"/>
      <xs:attribute name="Integrity" type="xs:boolean" default="false"/>
      <xs:attribute name="Encrypt" type="xs:boolean" default="false"/>
      <xs:attribute name="Unlock" type="xs:boolean" default="false"/>
  </xs:complexType>
```

The attributes of the Usage element define the intended usage of the

key and are a combination of one or more of the following (set to true):

- OTP, the key will be used for OTP generation
- CR, the key will be used for Challenge/Response purposes
- Encrypt, the key will be used for data encryption purposes
- Integrity, the key will be used to generate a keyed message digest for data integrity or authentication purposes.
- Unlock, the key will be used for an inverse challenge response in the case a user has locked the device by entering a wrong PIN too many times (for devices with PIN-input capability)

5.3.2.1. OTP and CR specific Usage elements (OPTIONAL)

When the key usage is set to OTP or CR, additional attributes MUST be provided to support the OTP and/or the response computation as required by the underlying algorithm and to customize or configure the outcome of the computation (format, length and usage modes).

5.3.2.1.1. ChallengeFormat element (MANDATORY)

The ChallengeFormat element defines the characteristics of the challenge in a CR usage scenario. The Challenge element is defined by the following attributes:

- Format (MANDATORY)

  Defines the format of the challenge accepted by the device and MUST be one of the values defined in Section 5.3.3

- CheckDigit (OPTIONAL)

  Defines if the device needs to check the appended Luhn check digit contained in a provided challenge. This is only valid if the Format attribute is 'DECIMAL'. Value MUST be:

  TRUE device will check the appended Luhn check digit in a provided challenge

FALSE device will not check appended Luhn check digit in challenge

- Min (MANDATORY)

  Defines the minimum size of the challenge accepted by the device for CR mode.

  If the Format attribute is 'DECIMAL', 'HEXADECIMAL' or 'ALPHANUMERIC' this value indicates the minimum number of digits/characters.

  If the Format attribute is 'BASE64' or 'BINARY', this value indicates the minimum number of bytes of the unencoded value.

  Value MUST be:

Unsigned integer.

- Max (MANDATORY)

  Defines the maximum size of the challenge accepted by the device for CR mode.

  If the Format attribute is 'DECIMAL', 'HEXADECIMAL' or 'ALPHANUMERIC' this value indicates the maximum number of digits/characters.

  If the Format attribute is 'BASE64' or 'BINARY', this value indicates the maximum number of bytes of the unencoded value.

  Value MUST be:

Unsigned integer.

5.3.2.1.2. ResponseFormat element (MANDATORY)

The ResponseFormat element defines the characteristics of the result of a computation. This defines the format of the OTP or of the response to a challenge. The Response attribute is defined by the following attributes:

- Format (MANDATORY)

  Defines the format of the response generated by the device and MUST be one of the values defined in Section 5.3.3

- CheckDigit (OPTIONAL)

  Defines if the device needs to append a Luhn check digit to the response. This is only valid if the Format attribute is 'DECIMAL'. Value MUST be:

  TRUE device will append a Luhn check digit to the response.

  FALSE device will not append a Luhn check digit to the response.

- Length (MANDATORY)

Defines the length of the response generated by the device.

If the Format attribute is 'DECIMAL', 'HEXADECIMAL' or 'ALPHANUMERIC' this value indicates the number of digits/ characters.

If the Format attribute is 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

Value MUST be:

Unsigned integer.

5.3.3. ValueFormat

The ValueFormat element defines allowed formats for challenges or responses in OTP algorithms.

ValueFormat is defined as follows:

```
<simpleType name="ValueFormat">
  <restriction base="string">
    <enumeration value="DECIMAL"/>
    <enumeration value="HEXADECIMAL"/>
    <enumeration value="ALPHANUMERIC"/>
    <enumeration value="BASE64"/>
    <enumeration value="BINARY"/>
  </restriction>
</simpleType>
```

DECIMAL Only numerical digits

HEXADECIMAL Hexadecimal response

ALPHANUMERIC All letters and numbers (case sensitive)

BASE64 Base 64 encoded

BINARY Binary data, this is mainly used in case of connected devices

5.3.4. PINPolicy

The PINPolicy element defines a mean to define how the usage of a specific key is protected by a PIN. The PIN itself can be transmitted using the container as another Key

PINPolicy is defined as follows:

```
<xs:complexType name="PINPolicyType">
    <xs:sequence>
        <xs:element name="PINUsageMode" type="pskc:PINUsageModeType"/>
        <xs:element name="WrongPINtry" type="xs:unsignedInt"
          minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="PINKeyId" type="xs:string" use="required"/>
</xs:complexType>
```

The attributes of PINPolicy have the following meaning

- PINKeyId, the unique key Id within this container that contains the value of the PIN that protects the key

The elements of PINPolicy have the following meaning

- <PINUsageMode>, the way the PIN is used during the usage of the key as defined in Section 5.3.4.1

- <WrongPINtry>, the number of times the PIN can be entered wrongly before it MUST not be possible to use the key anymore

5.3.4.1. PINUsageMode

The PINUsageMode element defines how the PIN is used with a specific key

PINUsageMode is defined as follows:

```
<xs:complexType name="PINUsageModeType">
    <xs:choice maxOccurs="unbounded">
        <xs:element name="Local"/>
        <xs:element name="Prepend"/>
        <xs:element name="InAlgo"/>
    </xs:choice>
</xs:complexType>
```

The elements of PINPolicy have the following meaning

- <Local>, the PIN is checked locally on the device before allowing the key to be used in executing the algorithm

- <Prepend>, the PIN is prepended to the OTP or response hance it MUST be chacked by the validation server

- <InAlgo>, the PIN is used as part of the algorithm computation

6. Usage and profile of algorithms for the portable symmetric key container

This section details the use of the XML encryption and XML signature elements to protect the keys transported in the cotainer. It also profiles the number of algorithms supported by XML encryption and XML signature to a mandatory subset for interoperability.

When no algorithm is provided the values within the container are unencrypted, implementations SHALL ensure the privacy of the key data through other standard mechanisms e.g. transport level encryption.

6.1. Usage of EncryptionKey to protect keys in transit

The EncryptionKey element in the KeyContainer defines the key, algorithm and parameters used to encrypt the Secret Key data attributes in the Container. The encryption is applied on each individual Secret Key data in the Container. The encryption method MUST be the same for all Secret Key data in the container.

The following sections define specifically the different supported means to protect the keys:

6.1.1. Protecting keys using a pre-shared key via symmetric algorithms

When protecting the payload with pre-shared keys implementations SHOULD set the name of the specific pre-shared key in the KeyName element of the EncryptionKey of the KeyContainer. For example:

```
<KeyContainer Version="1.0"
  xmlns="urn:ietf:params:xml:ns:keyprov:container:1.0"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionKey>
        <ds:KeyName>PRE_SHARED_KEY</ds:KeyName>
    </EncryptionKey>
    ....
```

The following is the list of symmetric key encryption algorithm and possible parameters used to protect the Secret Key data in the container. Systems implementing PSKC MUST support the MANDATORY

algorithms detailed below.

The encryption algorithm URI can be one of the following.

- http://www.w3.org/2001/04/xmlenc#tripledes-cbc - MANDATORY

- http://www.w3.org/2001/04/xmlenc#aes128-cbc - MANDATORY
- http://www.w3.org/2001/04/xmlenc#aes192-cbc - OPTIONAL
- http://www.w3.org/2001/04/xmlenc#aes256-cbc - MANDATORY
- http://www.w3.org/2001/04/xmlenc#kw-tripledes - MANDATORY
- http://www.w3.org/2001/04/xmlenc#kw-aes128 - MANDATORY
- http://www.w3.org/2001/04/xmlenc#kw-aes256 - MANDATORY
- http://www.w3.org/2001/04/xmlenc#kw-aes512 - OPTIONAL

When algorithms without integrity checks are used (e.g. http://www.w3.org/2001/04/xmlenc#aes256-cbc) a keyed MAC value using the same key as the encryption key SHOULD be placed in the ValueMAC element of the Data element. In this case the MAC algorithm type MUST be set in the MACAlgorithm element in the key container entity as defined in Section 5.1. Implementations of PSKC MUST support the MANDATORY MAC algorithms detailed below. The MACAlgorithm URI can be one of the following:

- http://www.w3.org/2000/09/xmldsig#hmac-sha1 - MANDATORY

For example:

```
<KeyContainer Version="1.0"
  xmlns="urn:ietf:params:xml:ns:keyprov:container:1.0"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionKey>
        <ds:KeyName>PRE_SHARED_KEY</ds:KeyName>
    </EncryptionKey>
```

```
    <MACAlgorithm>http://www.w3.org/2000/09/xmldsig#hmac-sha1
    </MACAlgorithm>
    .....
```

6.1.2. Protecting keys using passphrase based encryption keys

To be able to support passphrase based encryption keys as defined in PKCS#5 the following XML representation of the PBE relates parameters have been introduced in the schema. Although the approach is extensible implementations of PSKC MUST support the KeyDerivationMethod algorithm URI of http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2.


```
<xs:complexType name="DerivedKeyType">
    <xs:sequence>
        <xs:element name="KeyDerivationMethod"
          type="pskc:KeyDerivationMethodType" minOccurs="0"/>
        <xs:element ref="xenc:ReferenceList" minOccurs="0"/>
        <xs:element name="CarriedKeyName" type="xs:string"
          minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Id" type="xs:ID" use="optional"/>
    <xs:attribute name="Type" type="xs:anyURI" use="optional"/>
    </xs:complexType>
    <xs:complexType name="KeyDerivationMethodType">
    <xs:sequence>
            <xs:any namespace="##other" minOccurs="0"
              maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
</xs:complexType>
```

The attributes of the DerivedKey have the following meanings:

- ID (OPTIONAL), the unique ID for this key
- Type (OPTIONAL), TODO

The elements of the DerivedKey have the following meanings:

- <KeyDerivationMethod>: URI of the algorithms used to derive the

key e.g. (http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2)

- <ReferenceList (OPTIONAL)>: a list of IDs of the elements that have been encrypted by this key

- <CarriedKeyName (OPTIONAL)>: friendly name of the key

When using the PKCS5 PBE algorithm (URI=http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbes2) and related parameters, the DerivedKey element MUST be used within the EncryptionKey element of the KeyContainer in exactly the form as shown below:


```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer
  xmlns="urn:ietf:params:xml:ns:keyprov:container:1.0"
  xmlns:pkcs-5=
    "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  Version="1.0">
    <EncryptionKey>
      <DerivedKey Id="#Passphrase1">
        <CarriedKeyName>Passphrase1</CarriedKeyName>
        <KeyDerivationMethod
          Algorithm=
        "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2">
          <Parameters xsi:type="pkcs-5:PBKDF2ParameterType">
            <Salt>
              <Specified>Df3dRAhjGh8=</Specified>
            </Salt>
            <IterationCount>2000</IterationCount>
            <KeyLength>16</KeyLength>
```

```
            <PRF/>
          </Parameters>
        </KeyDerivationMethod>
      </DerivedKey>
    </EncryptionKey>
....
```

6.2. Protecting keys using asymmetric algorithms

The following is the list of asymmetric key encryption algorithm and possible parameters used to protect the Secret Key data in the container. Systems implementing PSKC MUST support the MANDATORY algorithms detailed below. The encryption algorithm URI can be one of the following.

- http://www.w3.org/2001/04/xmlenc#rsa-1_5 - MANDATORY
- http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p - OPTIONAL

For example:

TODO

6.3. Profile of Key Algorithm

This section profiles the type(s) of algorithm of that can be used by the key(s) transported in the container. The following algorithm URIs are among the default support list.

- http://www.w3.org/2001/04/xmlenc#tripledes-cbc
- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes192-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc

- http://www.ietf.org/keyprov/pskc#hotp

- http://www.ietf.org/keyprov/pskc#valuecompare

6.3.1. OTP Key Algorithm Identifiers

OTP key algorithm URIs have not been defined in a commonly available standard specification. This document defines the following URIs for the known open standard OTP algorithms.

6.3.1.1. HOTP

Standard document: RFC4226

Identifier: http://www.ietf.org/keyprov/pskc#hotp

Note that the actual URL will be finalized once a URL for this document is determined.

6.3.1.2. Other OTP Algorithms

An implementation should refer to vendor supplied OTP key algorithm URIs for proprietary algorithms.

6.3.2. PIN key value compare algorithm identifier

PIN key algorithm URIs have not been defined in a commonly available standard specification. This document defines the following URIs for a straight value comaprison of the transported secret key data as when required to compare a PIN.

Identifier: http://www.ietf.org/keyprov/pskc#valuecompare

Note that the actual URL will be finalized once a URL for this

document is determined.

7. Reserved data attribute names

The following key data attribute names have been reserved:

SECRET: the shared secret key value in binary, base64 encoded

COUNTER: the event counter for event based OTP algorithms. 8 bytes unsigned integer in big endian (i.e. network byte order) form base64 encoded

TIME: the time for time based OTP algorithms. 8 bytes unsigned integer in big endian (i.e. network byte order) form base64 encoded (Number of seconds since 1970)

TIME_INTERVAL: the time interval value for time based OTP algorithms. 8 bytes unsigned integer in big endian (i.e. network byte order) form base64 encoded.

TIME_DRIFT: the device clock drift value for time based OTP algorithms. The value indicates number of seconds that the device clock may drift each day. 2 bytes unsigned integer in big endian (i.e. network byte order) form base64 encoded.

8. Formal Syntax

The following syntax specification uses the widely adopted XML schema format as defined by a W3C recommendation (http://www.w3.org/TR/xmlschema-0/). It is a complete syntax definition in the XML Schema Definition format (XSD)

All implementations of this standard must comply with the schema below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:pskc="urn:ietf:params:xml:ns:keyprov:container:1.0"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  targetNamespace="urn:ietf:params:xml:ns:keyprov:container:1.0"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.0">
    <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation=
      "http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/
      xmldsig-core-schema.xsd"/>
    <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
      schemaLocation="http://www.w3.org/TR/2002/
      REC-xmlenc-core-20021210/xenc-schema.xsd"/>

    <xs:complexType name="KeyContainerType">
    <xs:sequence>
        <xs:element name="EncryptionKey" type="ds:KeyInfoType"
          minOccurs="0"/>
        <xs:element name="MACAlgorithm" type="pskc:KeyAlgorithmType"
          minOccurs="0"/>
        <xs:element name="Device" type="pskc:DeviceType"
          maxOccurs="unbounded"/>
        <xs:element name="Signature" type="ds:SignatureType"
          minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Version" type="pskc:VersionType"
      use="required"/>
    </xs:complexType>
    <xs:simpleType name="VersionType" final="restriction">
            <xs:restriction base="xs:string">
```

```
                    <xs:pattern value="\d{1,2}\.\d{1,3}"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="KeyType">
        <xs:sequence>
```



```
            <xs:element name="Issuer" type="xs:string" minOccurs="0"/>
            <xs:element name="Usage" type="pskc:UsageType"/>
            <xs:element name="CardAppPersoProfileId" type="xs:string"
              minOccurs="0"/>
            <xs:element name="FriendlyName" type="xs:string"
              minOccurs="0"/>
            <xs:element name="Data" type="pskc:DataType" minOccurs="0"
              maxOccurs="unbounded"/>
            <xs:element name="PINPolicy" type="pskc:PINPolicyType"
              minOccurs="0"/>
            <xs:element name="ExpiryDate" type="xs:dateTime"
              minOccurs="0"/>
            <xs:element name="StartDate" type="xs:dateTime"
              minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="KeyId" type="xs:string" use="required"/>
        <xs:attribute name="KeyAlgorithm" type="pskc:KeyAlgorithmType"
          use="required"/>
    </xs:complexType>
    <xs:complexType name="DerivedKeyType">
        <xs:sequence>
            <xs:element name="KeyDerivationMethod"
              type="pskc:KeyDerivationMethodType" minOccurs="0"/>
            <xs:element ref="xenc:ReferenceList" minOccurs="0"/>
            <xs:element name="CarriedKeyName" type="xs:string"
              minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="Id" type="xs:ID" use="optional"/>
        <xs:attribute name="Type" type="xs:anyURI" use="optional"/>
    </xs:complexType>
    <xs:complexType name="KeyDerivationMethodType">
        <xs:sequence>
            <xs:any namespace="##other" minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
```

```
    </xs:complexType>
    <xs:complexType name="PINPolicyType">
        <xs:sequence>
            <xs:element name="PINUsageMode"
              type="pskc:PINUsageModeType"/>
            <xs:element name="WrongPINtry" type="xs:unsignedInt"
              minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="PINKeyId" type="xs:string"
          use="required"/>
    </xs:complexType>
    <xs:complexType name="PINUsageModeType">
```



```
            <xs:choice maxOccurs="unbounded">
                    <xs:element name="Local"/>
                    <xs:element name="Prepend"/>
                    <xs:element name="Embed"/>
            </xs:choice>
    </xs:complexType>
    <xs:complexType name="DeviceIdType">
        <xs:sequence>
        <xs:element name="Manufacturer" type="xs:string"/>
        <xs:element name="SerialNo" type="xs:string"/>
        <xs:element name="Model" type="xs:string" minOccurs="0"/>
        <xs:element name="IssueNo" type="xs:string" minOccurs="0"/>
        <xs:element name="ExpiryDate" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="StartDate" type="xs:dateTime" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="DeviceType">
      <xs:sequence>
        <xs:element name="DeviceId" type="pskc:DeviceIdType"
          minOccurs="0"/>
        <xs:element name="Key" type="pskc:KeyType"
          maxOccurs="unbounded"/>
        <xs:element name="UserId" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="UsageType">
        <xs:sequence>
            <xs:element name="ResponseFormat">
                <xs:complexType>
```

```
                    <xs:attribute name="Format"
                      type="pskc:ValueFormatType" use="required"/>
                    <xs:attribute name="Length" type="xs:unsignedInt"
                      use="required"/>
                    <xs:attribute name="CheckDigits" type="xs:boolean"
                      default="false"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="ChallengeFormat" minOccurs="0">
                <xs:complexType>
                    <xs:attribute name="Format"
                      type="pskc:ValueFormatType" use="required"/>
                    <xs:attribute name="Min" type="xs:unsignedInt"
                      use="required"/>
                    <xs:attribute name="Max" type="xs:unsignedInt"
                      use="required"/>
                    <xs:attribute name="CheckDigits" type="xs:boolean"
                      default="false"/>
                </xs:complexType>
```



```
            </xs:element>
            </xs:sequence>
            <xs:attribute name="OTP" type="xs:boolean" default="false"/>
            <xs:attribute name="CR" type="xs:boolean" default="false"/>
            <xs:attribute name="Integrity" type="xs:boolean"
              default="false"/>
            <xs:attribute name="Encrypt" type="xs:boolean"
              default="false"/>
            <xs:attribute name="Unlock" type="xs:boolean"
              default="false"/>
    </xs:complexType>
    <xs:complexType name="DataType">
        <xs:sequence>
            <xs:choice>
                <xs:element name="PlainValue" type="xs:base64Binary"/>
                <xs:element name="EncryptedValue"
                  type="xenc:EncryptedDataType"/>
            </xs:choice>
            <xs:element name="ValueMAC" type="xs:base64Binary"
              minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" use="required"/>
```

```
    </xs:complexType>
    <xs:simpleType name="KeyAlgorithmType">
        <xs:restriction base="xs:anyURI"/>
    </xs:simpleType>
    <xs:simpleType name="ValueFormatType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="DECIMAL"/>
            <xs:enumeration value="HEXADECIMAL"/>
            <xs:enumeration value="ALPHANUMERIC"/>
            <xs:enumeration value="BASE64"/>
            <xs:enumeration value="BINARY"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="KeyContainer" type="pskc:KeyContainerType"/>
</xs:schema>
```


9. Security Considerations

The portable key container carries sensitive information (e.g., cryptographic keys) and may be transported across the boundaries of one secure perimeter to another. For example, a container residing within the secure perimeter of a back-end provisioning server in a secure room may be transported across the internet to an end-user device attached to a personal computer. This means that special care must be taken to ensure the confidentiality, integrity, and authenticity of the information contained within.

9.1. Payload confidentiality

By design, the container allows two main approaches to guaranteeing the confidentiality of the information it contains while transported.

First, the container key data payload may be encrypted.

In this case no transport layer security is required. However, standard security best practices apply when selecting the strength of the cryptographic algorithm for payload encryption. Symmetric cryptographic cipher should be used - the longer the cryptographic key, the stronger the protection. At the time of this writing both 3DES and AES are recommended algorithms but 3DES may be dropped in the relatively near future. Applications concerned with algorithm longevity are advised to use AES. In cases where the exchange of encryption keys between the sender and the receiver is not possible, asymmetric encryption of the secret key payload may be employed. Similarly to symmetric key cryptography, the stronger the asymmetric key, the more secure the protection is.

If the payload is encrypted with a method that uses one of the password-based encryption methods provided above, the payload may be subjected to password dictionary attacks to break the encryption password and recover the information. Standard security best practices for selection of strong encryption passwords apply [Schneier].

Practical implementations should use PBESalt and PBEIterationCount when PBE encryption is used. Different PBESalt value per key container should be used for best protection.

The second approach to protecting the confidentiality of the payload is based on using transport layer security. The secure channel established between the source secure perimeter (the provisioning server from the example above) and the target perimeter (the device attached to the end-user computer) utilizes encryption to transport the messages that travel across. No payload encryption is required

in this mode. Secure channels that encrypt and digest each message provide an extra measure of security, especially when the signature of the payload does not encompass the entire payload.

Because of the fact that the plain text payload is protected only by the transport layer security, practical implementation must ensure protection against man-in-the-middle attacks [Schneier]. Validating the secure channel end-points is critically important for eliminating

intruders that may compromise the confidentiality of the payload.

9.2. Payload integrity

The portable symmetric key container provides a mean to guarantee the integrity of the information it contains through digital signatures. For best security practices, the digital signature of the container should encompass the entire payload. This provides assurances for the integrity of all attributes. It also allows verification of the integrity of a given payload even after the container is delivered through the communication channel to the target perimeter and channel message integrity check is no longer possible.

9.3. Payload authenticity

The digital signature of the payload is the primary way of showing its authenticity. The recipient of the container may use the public key associated with the signature to assert the authenticity of the sender by tracing it back to a preloaded public key or certificate. Note that the digital signature of the payload can be checked even after the container has been delivered through the secure channel of communication.

A weaker payload authenticity guarantee may be provided by the transport layer if it is configured to digest each message it transports. However, no authenticity verification is possible once the container is delivered at the recipient end. This approach may be useful in cases where the digital signature of the container does not encompass the entire payload.

10. Acknowledgements

The authors of this draft would like to thank the following people for their contributions and support to make this a better specification: Apostol Vassilev, Shuh Chang, Jon Martinson, Siddhart Bajaj, Stu Veath, Kevin Lewis, Philip Hallam-Baker, Hannes Tschofenig, Andrea Doherty, Magnus Nystrom, Tim Moses, and Anders Rundgren.

11. Appendix A - Example Symmetric Key Containers

All examples are syntactically correct and compatible with the XML schema in section 7.

11.1. Symmetric Key Container with a single Non-Encrypted HOTP Secret Key

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
  xmlns="urn:ietf:params:xml:ns:keyprov:container:1.0">
    <Device>
        <DeviceId>
            <Manufacturer>ACME</Manufacturer>
            <SerialNo>0755225266</SerialNo>
        </DeviceId>
        <Key KeyAlgorithm="http://www.ietf.org/keyprov/pskc#hotp"
          KeyId="0755225266">
            <Issuer>AnIssuer</Issuer>
            <Usage OTP="true">
                <ResponseFormat Length="6" Format="DECIMAL"/>
            </Usage>
            <Data Name="COUNTER">
                <PlainValue>AprkuA==</PlainValue>
            </Data>
            <Data Name="SECRET">
                <PlainValue>/4h3rOTeBegJwGpmTTq4F+RlNR0=</PlainValue>
            </Data>
            <ExpiryDate>2012-12-31T00:00:00</ExpiryDate>
        </Key>
    </Device>
</KeyContainer>
```

11.2. Symmetric Key Container with a single PIN protected Non-Encrypted HOTP Secret Key

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
  xmlns="urn:ietf:params:xml:ns:keyprov:container:1.0">
    <Device>
        <DeviceId>
            <Manufacturer>ACME</Manufacturer>
            <SerialNo>0755225266</SerialNo>
        </DeviceId>
        <Key KeyAlgorithm="http://www.ietf.org/keyprov/pskc#hotp"
          KeyId="0755225266">
            <Issuer>AnIssuer</Issuer>
            <Usage OTP="true">
                <ResponseFormat Length="6" Format="DECIMAL"/>
            </Usage>
            <Data Name="COUNTER">
                <PlainValue>AprkuA==</PlainValue>
            </Data>
            <Data Name="SECRET">
                <PlainValue>/4h3rOTeBegJwGpmTTq4F+RlNR0=</PlainValue>
            </Data>
            <PINPolicy PINKeyId="07552252661">
                <PINUsageMode>
                    <Local/>
                </PINUsageMode>
            </PINPolicy>
        </Key>
        <Key KeyId="07552252661"
          KeyAlgorithm="http://www.ietf.org/keyprov/pskc#pin">
            <Usage>
                <ResponseFormat Length="4" Format="DECIMAL"/>
            </Usage>
            <Data Name="SECRET">
                <PlainValue>MTIzNA==</PlainValue>
            </Data>
        </Key>
    </Device>
</KeyContainer>
```

11.3. Symmetric Key Container with a single AES-256-CBC Encrypted HOTP

Secret Key


```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
  xmlns="urn:ietf:params:xml:ns:keyprov:container:1.0"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionKey>
        <ds:KeyName>PRE_SHARED_KEY</ds:KeyName>
    </EncryptionKey>
    <MACAlgorithm>http://www.w3.org/2000/09/xmldsig#hmac-sha1
    </MACAlgorithm>
    <Device>
        <DeviceId>
            <Manufacturer>ACME</Manufacturer>
            <SerialNo>0755225266</SerialNo>
        </DeviceId>
        <Key KeyAlgorithm="http://www.ietf.org/keyprov/pskc#hotp"
          KeyId="0755225266">
            <Issuer>AnIssuer</Issuer>
            <Usage OTP="true">
                <ResponseFormat Length="8" Format="DECIMAL"/>
            </Usage>
            <Data Name="COUNTER">
                <PlainValue>AprkuA==</PlainValue>
            </Data>
            <Data Name="SECRET">
                <EncryptedValue>
                    <xenc:EncryptionMethod
                   Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
                    <xenc:CipherData>
                      <xenc:CipherValue>
                        kyzrWTJuhJKQHhZtf2CWbKC5H3LdfAPvKzHHQ8SdxyE=
                      </xenc:CipherValue>
```

```
                    </xenc:CipherData>
                </EncryptedValue>
                <ValueMAC>cwJI898rRpGBytTqCAsegaQqPZA=</ValueMAC>
            </Data>
        </Key>
    </Device>
</KeyContainer>
```

11.4. Symmetric Key Container with signature and a single Password-based Encrypted HOTP Secret Key

```
<?xml version="1.0" encoding="UTF-8"?>
<pskc:KeyContainer
  xmlns:pskc="urn:ietf:params:xml:ns:keyprov:container:1.0"
  xmlns:pkcs-5=
```



```
    "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  Version="1.0">
    <pskc:EncryptionKey>
      <pskc:DerivedKey Id="#Passphrase1">
        <pskc:CarriedKeyName>Passphrase1</pskc:CarriedKeyName>
        <pskc:KeyDerivationMethod
          Algorithm=
     "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2">
          <pkcs-5:Parameters xsi:type="pkcs-5:PBKDF2ParameterType">
            <Salt>
              <Specified>Df3dRAhjGh8=</Specified>
            </Salt>
            <IterationCount>2000</IterationCount>
            <KeyLength>16</KeyLength>
            <PRF/>
          </pkcs-5:Parameters>
        </pskc:KeyDerivationMethod>
        <xenc:ReferenceList>
          <xenc:DataReference URI="#ED"/>
        </xenc:ReferenceList>
      </pskc:DerivedKey>
    </pskc:EncryptionKey>
```

```
   <pskc:Device>
     <pskc:DeviceId>
       <pskc:Manufacturer>ACME</pskc:Manufacturer>
       <pskc:SerialNo>0755225266</pskc:SerialNo>
     </pskc:DeviceId>
         <pskc:Key KeyAlgorithm="http://www.ietf.org/keyprov/pskc#hotp"
           KeyId="0755225266">
             <pskc:Issuer>AnIssuer</pskc:Issuer>
             <pskc:Usage OTP="true">
                 <pskc:ResponseFormat Length="6" Format="DECIMAL"/>
             </pskc:Usage>
             <pskc:Data Name="COUNTER">
                 <pskc:PlainValue>AprkuA==</pskc:PlainValue>
             </pskc:Data>
             <pskc:Data Name="SECRET">
                 <pskc:EncryptedValue Id="ED">
                   <xenc:EncryptionMethod Algorithm=
                   "http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
                   <xenc:CipherData>
                     <xenc:CipherValue>rf4dx3rvEPO0vKtKL14NbeVu8nk=
                     </xenc:CipherValue>
                   </xenc:CipherData>
                 </pskc:EncryptedValue>
```



```
             </pskc:Data>
         </pskc:Key>
   </pskc:Device>
   <pskc:Signature>
     <ds:SignedInfo>
       <ds:CanonicalizationMethod
         Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
       <ds:SignatureMethod
         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
       <ds:Reference URI="">
         <ds:DigestMethod Algorithm=
           "http://www.w3.org/2000/09/xmldsig#sha1"/>
         <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
       </ds:Reference>
     </ds:SignedInfo>
     <ds:SignatureValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:SignatureValue>
     <ds:KeyInfo>
       <ds:X509Data>
```

```
          <ds:X509IssuerSerial>
            <ds:X509IssuerName>CN=KeyProvisioning'R'Us.com,C=US
            </ds:X509IssuerName>
            <ds:X509SerialNumber>12345678</ds:X509SerialNumber>
          </ds:X509IssuerSerial>
        </ds:X509Data>
      </ds:KeyInfo>
    </pskc:Signature>
  </pskc:KeyContainer>
```

11.5. Symmetric Key Container with a single RSA 1.5 Encrypted HOTP Secret Key


```
<?xml version="1.0" encoding="UTF-8"?>
<pskc:KeyContainer Version="1.0"
  xmlns:pskc="urn:ietf:params:xml:ns:keyprov:container:1.0"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <pskc:EncryptionKey>
        <ds:X509Data>
            <ds:X509Certificate>miib</ds:X509Certificate>
        </ds:X509Data>
    </pskc:EncryptionKey>
    <pskc:Device>
```

```
        <pskc:DeviceId>
            <pskc:Manufacturer>ACME</pskc:Manufacturer>
            <pskc:SerialNo>0755225266</pskc:SerialNo>
        </pskc:DeviceId>
        <pskc:Key KeyAlgorithm="http://www.ietf.org/keyprov/pskc#hotp"
                  KeyId="0755225266">
            <pskc:Issuer>AnIssuer</pskc:Issuer>
            <pskc:Usage OTP="true">
                <pskc:ResponseFormat Length="8" Format="DECIMAL"/>
            </pskc:Usage>
            <pskc:Data Name="COUNTER">
                <pskc:PlainValue>AprkuA==</pskc:PlainValue>
            </pskc:Data>
            <pskc:Data Name="SECRET">
              <pskc:EncryptedValue Id="ED">
                <xenc:EncryptionMethod
                  Algorithm="http://www.w3.org/2001/04/xmlenc#rsa_1_5"/>
                <xenc:CipherData>
                  <xenc:CipherValue>rf4dx3rvEPO0vKtKL14NbeVu8nk=
                  </xenc:CipherValue>
                </xenc:CipherData>
              </pskc:EncryptedValue>
            </pskc:Data>
        </pskc:Key>
    </pskc:Device>
</pskc:KeyContainer>
```


12. Normative References

[CAP] MasterCard International, "Chip Authentication Program Functional Architecture", September 2004.

[DSKPP] "Dynamic Symmetric Key Provisioning Protocol", Internet Draft Informational, URL: http://tools.ietf.org/wg/keyprov/draft-doherty-keyprov-dskpp-00.txt, June 2007.

[HOTP] MRaihi, D., "HOTP: An HMAC-Based One Time Password Algorithm", RFC 4226, URL: http://rfc.sunsite.dk/rfc/rfc4226.html, December 2005.

[OATH] "Initiative for Open AuTHentication", URL: http://www.openauthentication.org.

[OATHRefArch] OATH, "Reference Architecture", URL: http://www.openauthentication.org, Version 1.0, 2005.

[OCRA] MRaihi, D., "OCRA: OATH Challenge Response Algorithm", Internet Draft Informational, URL: http://www.ietf.org/internet-drafts/draft-mraihi-mutual-oath-hotp-variants-01.txt, December 2005.

[PKCS1] Kaliski, B., "RFC 2437: PKCS #1: RSA Cryptography Specifications Version 2.0.", URL: http://www.ietf.org/rfc/rfc2437.txt, Version: 2.0, October 1998.

[PKCS12] RSA Laboratories, "PKCS #12: Personal Information Exchange Syntax Standard", Version 1.0, URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-12/.

[PKCS5] RSA Laboratories, "PKCS #5: Password-Based Cryptography Standard", Version 2.0, URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5/, March 1999.

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.

[Schneier] Schneier, B., "Secrets and Lies: Digitial Security in a Networked World", Wiley Computer Publishing, ISBN 0-8493-

8253-7, 2000.

[XMLENC] Eastlake, D., "XML Encryption Syntax and Processing.", URL: http://www.w3.org/TR/xmlenc-core/, December 2002.

[XMLSIG] Eastlake, D., "XML-Signature Syntax and Processing", URL: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/, W3C Recommendation, February 2002.

Authors' Addresses

Philip Hoyer
ActivIdentity, Inc.
109 Borough High Street
London, SE1 1NL
UK

Phone: +44 (0) 20 7744 6455
Email: Philip.Hoyer@actividentity.com

Mingliang Pei
VeriSign, Inc.
487 E. Middlefield Road
Mountain View, CA 94043
USA

Phone: +1 650 426 5173
Email: mpei@verisign.com

Salah Machani
Diversinet, Inc.
2225 Sheppard Avenue East
Suite 1801
Toronto, Ontario M2J 5C2
Canada

Phone: +1 416 756 2324 Ext. 321
Email: smachani@diversinet.com

Full Copyright Statement

Intellectual Property

http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).