Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2008

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

July 2007

Portable Symmetric Key Container
draft-ietf-keyprov-portable-symmetric-key-container-00.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.

Table of Contents

1. Introduction

With increasing use of symmetric key based authentication systems such as systems based upon 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 key based credentials from one system to another. Traditionally authentication server vendors and service providers have used proprietary formats for importing, exporting and provisioning these credentials 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 key based credentials 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 key 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 OTP provisioning protocol to different flavors of embedded tokens for OTP credentials or allow customers to import new or existing tokens in batch or single instances into a compliant system.

This draft also specifies the token attributes required for interoperability 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 credentials 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 credentials from one system to another, using some form of export and import model.

3.1.1. Credential migration by end-user

A user wants to migrate a credential 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 credential 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 credentials

Tokens are manufactured in bulk and associated credentials (key records) need to be loaded into the validation system through a file on portable media. The manufacturer provides the credentials 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.

3.1.3. Bulk migration of existing credentials

An enterprise wants to port credentials 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 credentials (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.

3.1.4. Credential upload case

User wants to activate and use a new credential against a validation system that is not aware of this credential. This credential 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 credential on a CD or a floppy in a standard format. The user can now upload via a secure online channel or import this credential into the new validation system and start using the credential.

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 credentials using some form of a online provisioning protocol.

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

A mobile device user wants to obtain an OTP credential (shared secret) 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 an OTP credential in a standard format that can be processed by the mobile device. The user can download more than one credential in a single session if the provisioning server holds multiple credentials for that user.

In a variation of the above, instead of the user's mobile phone, a credential 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 credential in a standard format that can be processed by the soft token on the PC.

3.2.2. Server to server provisioning of credentials

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

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

3.2.3. Online update of an existing authentication token credential

The end-user or the credential issuer wants to update or configure an existing credential in the authentication token and requests a replacement credential container. The container may or may not include a new secret key and may include new or updated secret key attributes such as a new counter value in HOTP credential case, a new logo, 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 key credentials 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
* Issuer logo
* Credential 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 key credentials.

R5: The format SHOULD be portable to various platforms. Furthermore, it SHOULD be computationally efficient to process.

R6: The format MUST provide appropriate level of security in terms of data encryption and data integrity.

R7: For online scenarios the format SHOULD NOT rely on transport level security (e.g., SSL/TLS) for core security requirements.

R8: The format SHOULD be extensible. It SHOULD enable extension points allowing vendors to specify additional attributes in the future.

R9: 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.

R10: 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.

R11: The format MUST support the use of pre-shared symmetric keys to ensure confidentiality of sensitive data elements.

R12: 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.

R13: 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. Symmetric Key Attributes

The symmetric key includes a number of data attributes that define the type of the key its usage and associated meta-information required during the provisioning, configuration, access or usage in the host device.

5.1. Common Attributes

5.1.1. Data (OPTIONAL)

Defines the data attributes of the symmetric key. Each is a name value pair which has both a base64 encoded value and a base 64 encoded valueDigest. The value can be encrypted. If the container has been encrypted the valueDigest MUST be populated with the digest of the unencrypted value.

This is also where the key value is held, therefore the following list of 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.

5.1.2. KeyAlgorithm (MANDATORY)

Defines the type of algorithm of the secret key and MUST be set to one of the values defined in Section 6.5. If 'OTHER' is specified an extension value MUST be set in the 'ext-KeyAlgorithm' attribute.

5.1.3. Usage (MANDATORY)

Defines the intended usage of the key and is 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

SIGN: the key will be used to generate a signature or keyed hashing 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)

Additional attributes that are specific to the usage type MAY be required. Section 6.1 describes OTP and CR specific attributes.

5.1.4. KeyId (MANDATORY)

A unique and global identifier of the symmetric key. The identifier is defined as a string of alphanumeric characters.

5.1.5. Issuer (MANDATORY)

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.

5.1.6. FriendlyName (OPTIONAL)

The user friendly name that is assigned to the secret key for easy reference. The FriendlyName is defined as a String.

5.1.7. AccessRules (OPTIONAL)

Defines a set of access rules and policies for the protection of the key on the host Device. Currently only the userPIN policy is defined. The userPIN policy specifies whether the user MUST enter a PIN (for devices with PIN input capability) in order to unlock or authenticate to the device hosting the key container. The userPIN is defined as a Boolean (TRUE or FALSE). When the user PIN is required, the policy MUST be set to TRUE. If the userPIN is NOT provided, implementations SHALL default the value to FALSE.

5.1.8. EncryptionMethod (MANDATORY when 'Data' attribute is encrypted))

Identifies the encryption algorithm and possible parameters used to protect the Secret Key data in the container and MUST be set to one

of the values defined in Section 6.2. If 'OTHER' is specified an extension value MUST be set in the 'ext-algorithm' attribute.

When the value is set to NONE, implementations SHALL ensure the privacy of the key data through other standard mechanisms e.g. transport level encryption.

When the KeyContainer contains more than one key and EncryptionMethod is different from NONE, the same encryption key MUST be used to encrypt all the key data elements in the container.

5.1.9. DigestMethod (MANDATORY when Digest is present)

Identifies the algorithm and possible parameters used to generate a digest of the Secret Key data. The digest guarantees the integrity and the authenticity of the key data. The Digest algorithm MUST be set to one of the values defined in Section 6.4. If 'OTHER' is specified an extension value MUST be set in the 'ext-algorithm'

attribute.

See Section 6.1.8 for more information on Digest data value type.

5.1.10. OTP and CR specific Attributes (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).

5.1.10.1. ChallengeFormat (MANDATORY)

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

1. Format (MANDATORY)

   Defines the format of the challenge accepted by the device and MUST be one of the values defined in Section 6.6

2. 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

3. 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.

4. 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.

5.1.10.2. ResponseFormat (MANDATORY)

The ResponseFormat attribute 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 sub-attributes:

1. Format (MANDATORY)

Defines the format of the response generated by the device and MUST be one of the values defined in Section 6.6

2. 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.

3. 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.

5.1.10.3. AppProfileId (OPTIONAL)

Defines the application profile id related to attributes present on a smart card that have influence when computing a response. An example could be an EMV MasterCard CAP [CAP] application on a card that contains attributes or uses fixed 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 AppProfileId.

6. Key container XML schema definitions

The portable key container is defined by the following entities:

1. KeyContainer entity

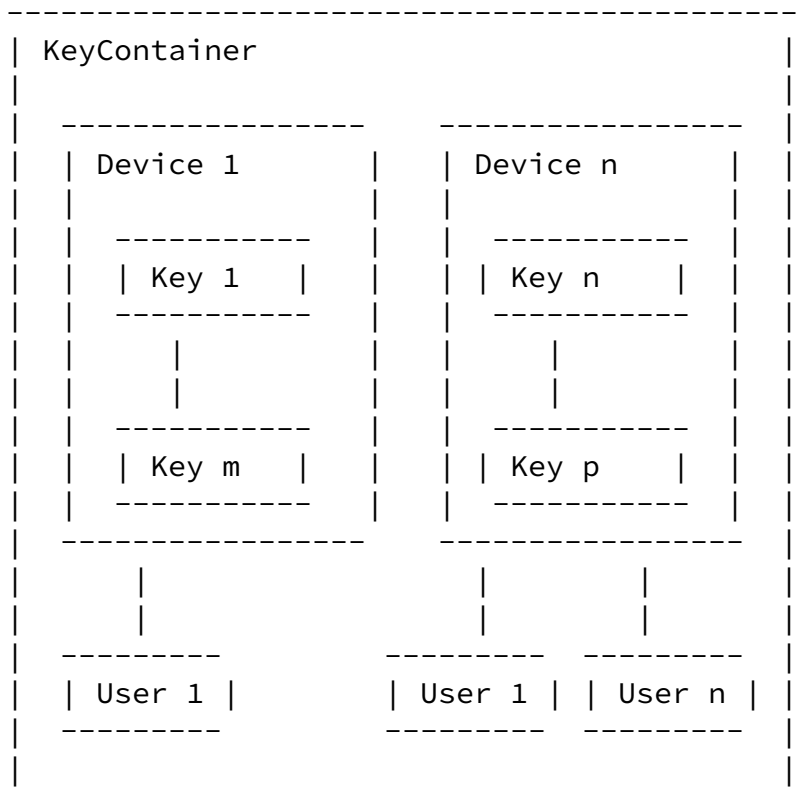
2. Device entity

3. Key entity

4. User entity

A KeyContainer MAY contain one or more Device entities. A Device MAY contain one or more Key entities and may be associated to one or more User entities.

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



6.1. XML Schema Types

The following types are defined to represent the portable key container entities and associated attributes.

6.1.1. KeyType

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

```
<complexType name="KeyType">
  <sequence>
     <element name="Issuer" type="string"/>
     <element name="Usage" type="pskc:UsageType"/>
     <element name="FriendlyName" type="string" minOccurs="0"/>
     <element name="Data" type="pskc:DataType" minOccurs="0"
     maxOccurs="unbounded"/>
     <element name="AccessRules" minOccurs="0">
        <complexType>
          <simpleContent>
          <extension base="string">
            <attribute name="userPIN" type="boolean" use="optional"
            default="false"/>
          </extension>
         </simpleContent>
        </complexType>
     </element>
     <element name="Logo" type="logo:LogoType" minOccurs="0"/>
     <element name="Expiry" type="string" minOccurs="0"/>
  </sequence>
  <attribute name="KeyId" type="string" use="required"/>
  <attribute name="KeyAlgorithm" type=
  "pskc:KeyAlgorithmType" use="required"/>
  <attribute name="ext-KeyAlgorithm" type="string"/>
</complexType>
```

The components of the KeyType have the following meanings (see

Section 5 for further information):

- <Usage> of type UsageType defines the usage of the Secret Key. The Usage attribute is described in Section 5.1.3.

- <Issuer> identifies the issuer of the Secret Key. The Issuer attribute is described in Section 5.1.5.

- <FriendlyName> is a user friendly name that is assigned to the Secret Key for easy reference.

- <Data> conveys the data attributes (e.g. the Secret Key) in name (string) value (base64 encoded) pairs. The value can be encrypted, in this case a digest of the non-encrypted data is present. The <Data> component is further described below.

- <AccessRules> Defines the rules for accessing the credential on the device e.g. a password must be provided by the user to view credential info or use the credential to generate an OTP response

- KeyId is a global identifier of the Secret Key. See Section 5.1.4.

- KeyAlgorithm defines the algorithm used with the Secret Key. The type values are defined in Section 6.5. If 'OTHER' is specified an extension value MUST be set in the 'ext-KeyAlgorithm' attribute.

- ext-KeyAlgorithm is the extension point for KeyAlgorithms not already defined Section 6.5

- Logo of type LogoType associates display logos with this Secret Key

- Expiry defines the expiry date of the Secret Key in format DD/MM/YYYY

The <Data> element is of type <DataType> and is defined as follows:

```
<complexType name="DataType">
  <sequence>
    <element name="Value" type="base64Binary"/>
```

```
      <element name="ValueDigest" type="base64Binary" minOccurs="0"/>
      <attribute name="Name" type="string" use="required"/>
    </sequence>
  </complexType>
```

The 'Name' attribute defines the name of the name-value pair, the following list of attribute names have been reserved:

SECRET: the 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.

The <Value> element in the DataType conveys the value of the name-value pair in base 64 encoding. The value MAY be encrypted or in clear text as per the EncryptionMethod data element in the KeyContainer (see Section 6.1.6 for details about KeyContainerType). When the value is encrypted, the digest value in 'ValueDigest' MUST be provided. The digest MUST be calculated on the unencrypted value and MUST use one of the Digest algorithms specified in DigestMethodType element of the KeyContainer. The MAC key for the MAC calculation should use the same key as the encryption key specified in the EncryptionMethod unless a separate MAC key is specified. When PBE method is used for encryption, a different password is recommended for the MAC key derivation. When the key data is in clear text, the KeyContainer payload signature MAY be used to check the integrity of the key octets.

6.1.2. UsageType

The UsageType defines the usage attribute of the key entity. The UsageType is defined as follows:

```
   <complexType name="UsageType">
     <sequence>
       <element name="AlgorithmIdentifier"
       type="pskc:AlgorithmIdentifierType" minOccurs="0"/>
       <element name="ResponseFormat">
         <complexType>
           <attribute name="format" type="pskc:valueFormat"
           use="required"/>
           <attribute name="length" type="unsignedInt"
           use="required"/>
           <attribute name="checkDigits" type="boolean"
           use="optional"  default="false"/>
```

```
        </complexType>
      </element>
      <element name="ChallengeFormat" minOccurs="0">
        <complexType>
          <attribute name="format" type="pskc:valueFormat"
          use="required"/>
          <attribute name="min" type="unsignedInt" use="required"/>
          <attribute name="max" type="unsignedInt" use="required"/>
          <attribute name="checkDigits" type="boolean" use="optional"
          default="false"/>
        </complexType>
      </element>
      <element name="AppProfileId" type="string" minOccurs="0"/>
    </sequence>
    <attribute name="otp" type="boolean" use="optional"
    default="false"/>
    <attribute name="cr" type="boolean" use="optional"
    default="false"/>
    <attribute name="sign" type="boolean" use="optional"
    default="false"/>
    <attribute name="encrypt" type="boolean" use="optional"
    default="false"/>
    <attribute name="unlock" type="boolean" use="optional"
    default="false"/>
  </complexType>
```

The UsageType components have the following meanings:

- <AlgorithmIdentifier> the AlgorithmIdentifier as defined in [OCRA]].

- <ChallengeFormat> hold the challenge attributes in CR based algorithm computations.

- <ResponseFormat> holds the algorithm response attributes.

- <AccessRules> holds a set of access rules and policies for the key once provisioned on the Device. Currently only the userPIN attribute is defined. The userPIN indicates whether the user MUST provide a PIN to unlock the key.

- <AppProfileId> Is the unique shared identifier for out of band shared common parameters.

6.1.3. DeviceType

The DeviceType type 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 DeviceType is defined as follows:

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

The components of the DeviceType have the following meanings:

- <DeviceId>, a unique identifier for the device, defined by the DeviceId type.

- <Key>, represents the key entity defined by the KeyType.

- <User>, optionally identifies the owner or the user of the Device, as defined by the UserType .

6.1.4. DeviceIdType

The DeviceId type 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 DeviceIdType is defined as follows:

```
<complexType name="DeviceIdType">
 <sequence>
  <element name="Manufacturer" type="string"/>
  <element name="SerialNo" type="string"/>
  <element name="Model" type="string" minOccurs="0"/>
  <element name="IssueNo" type="string" minOccurs="0"/>
  <element name="Expiry" type="string" minOccurs="0"/>
 </sequence>
</complexType>
```

The components of the DeviceId type have the following meanings:

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

6.1.5. UserType Type

The UserType represents the identifying criteria to uniquely identify the user who is bound to this device.

The UserType is defined as follows:

```
<complexType name="UserType">
  <sequence>
    <sequence>
      <element name="UserId" type="string" minOccurs="0"/>
      <element name="FirstName" type="string" minOccurs="0"/>
      <element name="LastName" minOccurs="0"/>
    </sequence>
    <element name="Org" type="string" minOccurs="0"/>
  </sequence>
</complexType>
```

The components of the UserType type have the following meanings:

- <FirstName>, user first name.
- <LastName>, user last name.
- <UserId>, user id (UID) or user name.
- <Org>, user organization name.

6.1.6. KeyContainerType

The KeyContainerType 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 KeyContainerType is defined as follows:

```
<complexType name="KeyContainerType">
  <sequence>
    <element name="EncryptionMethod">
      <complexType>
        <complexContent>
          <extension base="pskc:EncryptionMethodType"/>
        </complexContent>
      </complexType>
    </element>
    <element name="DigestMethod">
      <complexType>
        <complexContent>
          <extension base="pskc:DigestMethodType"/>
        </complexContent>
      </complexType>
    </element>
    <element name="Device" type="pskc:DeviceType"
    maxOccurs="unbounded"/>
    <element name="Signature" type="ds:SignatureType"
    minOccurs="0"/>
  </sequence>
  <attribute name="version" type="pskc:VersionType"
  use="required"/>
</complexType>
```

The components of the KeyContainer have the following meanings:

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

- <EncryptionMethod>, the encryption method used to protect the Key data attributes

- <DigestMethod>, the digest method used to sign the unencrypted the Secret Key data attributes

- <Device>, the host Device for one or more Keys.

- <Signature>, contains 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]. The signature is enveloped.

6.1.7. EncryptionMethodType

The EncryptionMethodType defines the 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 EncryptionMethodType is defined as follows:

```
<complexType name="EncryptionMethodType">
  <sequence>
    <element name="EncKeyLabel" minOccurs="0"/>
    <choice>
        <sequence>
          <element name="KeyInfo" type="ds:KeyInfoType" minOccurs="0"/>
          <element name="OAEPParams" type="base64Binary" minOccurs="0"/>
          <element name="HashAlgorithm"
          type="pskc:HashAlgorithmType" minOccurs="0"/>
        </sequence>
      <sequence>
        <element name="PBESalt" type="base64Binary" minOccurs="0"/>
        <element name="PBEIterationCount" type="int" minOccurs="0"/>
        <element name="IV" type="base64Binary" minOccurs="0"/>
      </sequence>
    </choice>
  </sequence>
  <attribute name="algorithm" type="pskc:EncryptionAlgorithmType"
  use="required"/>
  <attribute name="ext-algorithm" type="string"/>
</complexType>
```

The components of the EncryptionMethodType have the following meanings:

- algorithm: identifies the encryption algorithm used to protect the Secret Key data. When 'NONE' is specified, implementations MUST guarantee the privacy of the Secret Key Data through other mechanisms e.g. through transport level security. If 'OTHER' is specified an extension value MUST be set in the 'ext-algorithm' attribute. Please see EncryptionAlgorithmType for more information on supported algorithms

- <PBESalt>: conveys the Salt when [PKCS5] password-based encryption is applied.

- <PBEIterationCount>: conveys the iteration count value in [PKCS5] password-based encryption if it is different from the default value.

- <IV>: conveys the initialization vector for CBC based encryption algorithms. It is recommended for security reasons to transmit

this value out of band and treat it the same manner as the key value.

- <EncKeyLabel>: identifies a unique label for a pre-shared encryption key.

- <KeyInfo>: conveys the information of the key if an RSA algorithm has been used.

- <OAEPParams>: conveys the OAEP parameters if an RSA algorithm has been used.

- <HashAlgorithm>: conveys the digest algorithm if an RSA algorithm has been used.

6.1.8. DigestMethodType

The DigestMethodType defines the algorithm and parameters used to create the digest on the unencrypted Secret Key data in the Container. The digest is applied on each individual Secret Key data in the Container before encryption. The digest method MUST be the

same for all Secret Key data in the container. Unless a different digest key is specified it is assumed that keyed digest algorithms will use the same key as for encryption

The DigestMethodType is defined as follows:

```
<complexType name="DigestMethodType">
  <sequence>
    <element name="DigestKeyLabel" minOccurs="0"/>
  </sequence>
  <attribute name="algorithm" type="pskc:DigestAlgorithmType"
  use="required"/>
</complexType>
```

The components of the DigestMethodType have the following meanings:

- algorithm, identifies the digest algorithm used to protect the Secret Key data. Please see DigestAlgorithmType for more information on supported algorithms

- <DigestKeyLabel>: identifies a unique label for a pre-shared digest key.

6.1.9. AlgorithmIdentifierType

The AlgorithmIdentiferType defines the Algorithm identifier (AI) specified in [OCRA].

The AlgorithmIdentifierType is defined as follows:

```
<complexType name="AlgorithmIdentifierType">
    <sequence>
      <element name="Algorithm">
        <simpleType>
          <restriction base="string">
            <enumeration value="OCRA-HOTP"/>
```

```
            </restriction>
          </simpleType>
        </element>
        <element name="CryptoFunction"
        type="pskc:DigestAlgorithmType"/>
        <element name="Truncation">
          <simpleType>
            <restriction base="decimal">
              <minInclusive value="4"/>
              <maxInclusive value="10"/>
            </restriction>
          </simpleType>
        </element>
        <element name="Pin" type="boolean"/>
        <element name="Counter" type="boolean"/>
        <element name="Time" type="boolean"/>
        <element name="Session" type="boolean"/>
        <element name="Challenge" type="boolean"/>
      </sequence>
    </complexType>
```

See [OCRA] for a full description of the components of the AlgorithmIdentifierType.

6.2. EncryptionAlgorithmType

The EncryptionAlgorithmType defines the allowed algorithms for encrypting the Secret Key data in the Container.

The EncryptionAlgorithmType is defined as follows:


```
<simpleType name="EncryptionAlgorithmType">
  <restriction base="string">
    <enumeration value="NONE"/>
    <enumeration value="PBE-3DES112-CBC"/>
    <enumeration value="PBE-3DES168-CBC"/>
    <enumeration value="PBE-AES128-CBC"/>
    <enumeration value="PBE-AES256-CBC"/>
```

```
      <enumeration value="PBE-AES192-CBC"/>
      <enumeration value="3DES112-CBC"/>
      <enumeration value="3DES168-CBC"/>
      <enumeration value="AES128-CBC"/>
      <enumeration value="AES192-CBC"/>
      <enumeration value="AES256-CBC"/>
      <enumeration value="RSA-1_5"/>
      <enumeration value="RSA-OAEP-MGF1P"/>
      <enumeration value="OTHER"/>
    </restriction>
  </simpleType>
```

NONE when no encryption is applied on the key

PBE-3DES112-CBC when password-based encryption is applied using a 112-bit 3DES key in CBC mode

PBE-3DES168-CBC when password-based encryption is applied using a 168-bit 3DES key in CBC mode

PBE-AES128-CBC when password-based encryption is applied using a 128-bit AES key in CBC mode

PBE-AES192-CBC when password-based encryption is applied using a 192-bit AES key in CBC mode is applied.

PBE-AES256-CBC password-based encryption is applied using a 256-bit AES key in CBC mode is applied.

3DES112-CBC encryption using a pre-shared 112-bit 3DES key in CBC mode is applied.

3DES168-CBC encryption using a pre-shared 168-bit 3DES key in CBC mode is applied.

AES128-CBC encryption using a pre-shared 128-bit AES key in CBC mode is applied.

AES192-CBC encryption using a pre-shared 192-bit AES key in CBC mode is applied.

AES256-CBC encryption using a pre-shared 256-bit AES key in CBC mode is applied.

RSA-1_5 The RSAES-PKCS1-v1_5 algorithm, specified in [PKCS1], takes no explicit parameters.

RSA-OAEP-MGF1P The same algorithm as defined in section 5.4.2 RSA-OAEP in [XMLENC] It is the RSAES-OAEP-ENCRYPT algorithm, as specified in [PKCS1], it takes three parameters. The two user specified parameters are a MANDATORY message digest function and an OPTIONAL encoding octet string OAEPparams. The message digest function is indicated by the Algorithm attribute of a child ds: DigestMethod element and the mask generation function, the third parameter, is always MGF1 with SHA1 (mgf1SHA1Identifier).

OTHER extension point for not already defined algorithms in this list.

6.3. HashAlgorithmType

The HashAlgorithmType defines the allowed algorithms for generating a digest in the RSA algorithms.

The HashAlgorithmType is defined as follows:

```
<simpleType name="HashAlgorithmType">
  <restriction base="string">
    <enumeration value="SHA1"/>
    <enumeration value="SHA256"/>
    <enumeration value="SHA512"/>
  </restriction>
</simpleType>
```

SHA1 when the digest was performed using the SHA1 algorithm

SHA192 when the digest was performed using the SHA192 algorithm

SHA256 when the digest was performed using the SHA256 algorithm

6.4. DigestAlgorithmType

The DigestAlgorithmType defines the allowed algorithms for generating a digest on the unencrypted Secret Key data in the Container.

The DigestAlgorithmType is defined as follows:

```
<simpleType name="DigestAlgorithmType">
  <restriction base="string">
    <enumeration value="HMAC-SHA1"/>
    <enumeration value="HMAC-SHA256"/>
    <enumeration value="HMAC-SHA512"/>
    <enumeration value="OTHER"/>
  </restriction>
</simpleType>
```

HMAC-SHA1 when the digest was performed using the HMAC-SHA1 algorithm

HMAC-SHA192 when the digest was performed using the HMAC-SHA192 algorithm

HMAC-SHA256 when the digest was performed using the HMAC-SHA256 algorithm

OTHER extension point for not already defined algorithms in this list.

6.5. KeyAlgorithmType

The KeyAlgorithmType defines the algorithms in which the Secret Key data is used.

The KeyAlgorithmType is defined as follows:

```
   <simpleType name="KeyAlgorithmType">
     <restriction base="string">
       <enumeration value="3DES112"/>
       <enumeration value="3DES168"/>
       <enumeration value="ACTI"/>
       <enumeration value="AES128"/>
       <enumeration value="AES192"/>
       <enumeration value="AES256"/>
       <enumeration value="ANSIX9.9"/>
       <enumeration value="DES"/>
       <enumeration value="HOTP"/>
       <enumeration value="MKEYLABEL"/>
       <enumeration value="RSASECUREID"/>
       <enumeration value="VASCO"/>
       <enumeration value="OTHER"/>
     </restriction>
   </simpleType>
```

3DES112, a 112-bit 3DES key (a.k.a. two-key 3DES)

3DES168, a 168-bit parity-checked 3DES key

ACTI, algorithm family from ActivIdentity

AES128, a 128-bit AES key

AES192, a 192-bit AES key

AES256, a 256-bit AES key

ANSIX9.9, ANSI X9.9 algorithm

DES, a standard DES key

HOTP, as defined in [HOTP]

MKEYLABEL, master key abel or name when an embedded device key is used to derive the Key

RSASECUREID, SecureId algorithm family from RSA

VASCO, algorithm family from Vasco

OTHER extension point for not already defined algorithms in this list.

6.6. valueFormat

The valueFormat defines allowed formats for challenges or responses in the OTP algorithms.

The 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

6.7. Data elements

6.7.1. KeyContainer

The KeyContainer data element is defined as:

```
<element name="KeyContainer" type="pskc:KeyContainerType"/>
```

The KeyContainer data element is of type KeyContainerType defined in Section 6.1.6.

The EncryptionMethod data element in the KeyContainer defines the encryption algorithm used to protect the Key data. In a multi-key KeyContainer, the same encryption method and the same encryption key MUST be used for all key data elements.

The KeyContainer data element MAY contain multiple Device data elements, allowing for bulk provisioning of keys.

The Signature data element is of type <ds:Signature> as defined in [XMLSIG] and MAY be omitted in the KeyContainer data element 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.

7. 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 implentations of this standard must comply with the schema below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:pskc="urn:ietf:params:xml:ns:keyprov:container"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:logo="urn:ietf:params:xml:ns:keyprov:logo"
```

```
targetNamespace="urn:ietf:params:xml:ns:keyprov:container"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/
xmldsig-core-schema.xsd"/>
<import namespace="urn:ietf:params:xml:ns:keyprov:logo"
schemaLocation="oath_logotype_v1.0.xsd"/>
<complexType name="KeyContainerType">
<sequence>
<element name="EncryptionMethod">
<complexType>
<complexContent>
<extension base="pskc:EncryptionMethodType"/>
</complexContent>
</complexType>
</element>
<element name="DigestMethod">
<complexType>
<complexContent>
<extension base="pskc:DigestMethodType"/>
</complexContent>
</complexType>
</element>
<element name="Device" type="pskc:DeviceType"
maxOccurs="unbounded"/>
<element name="Signature" type="ds:SignatureType"
minOccurs="0"/>
</sequence>
<attribute name="version" type="pskc:VersionType"
use="required"/>
</complexType>
  <complexType name="AlgorithmIdentifierType">
    <sequence>
```



```
      <element name="Algorithm">
        <simpleType>
          <restriction base="string">
            <enumeration value="OCRA-HOTP"/>
          </restriction>
        </simpleType>
      </element>
      <element name="CryptoFunction"
```

```
        type="pskc:DigestAlgorithmType"/>
        <element name="Truncation">
          <simpleType>
            <restriction base="decimal">
              <minInclusive value="4"/>
              <maxInclusive value="10"/>
            </restriction>
          </simpleType>
        </element>
        <element name="Pin"
        type="boolean"/>
        <element name="Counter"
        type="boolean"/>
        <element name="Time"
        type="boolean"/>
        <element name="Session"
        type="boolean"/>
        <element name="Challenge"
        type="boolean"/>
      </sequence>
    </complexType>
  <complexType name="KeyType">
  <sequence>
  <element name="Issuer" type="string"/>
  <element name="Usage" type="pskc:UsageType"/>
  <element name="FriendlyName" type="string"
  minOccurs="0"/>
  <element name="Data" type="pskc:DataType"
  minOccurs="0"/>
  <element name="AccessRules" minOccurs="0">
  <complexType>
  <simpleContent>
  <extension base="string">
  <attribute name="userPIN" type="boolean"
  use="optional" default="false"/>
  </extension>
  </simpleContent>
  </complexType>
  </element>
  <element name="Logo" type="logo:LogoType"
```



```
  minOccurs="0"/>
```

```
<element name="Expiry" type="string" minOccurs="0"/>
</sequence>
<attribute name="KeyId" type="string" use="required"/>
<attribute name="KeyAlgorithm"
type="pskc:KeyAlgorithmType" use="required"/>
<attribute name="ext-KeyAlgorithm" type="string"/>
</complexType>
<complexType name="DeviceIdType">
<sequence>
<element name="Manufacturer" type="string"/>
<element name="SerialNo" type="string"/>
<element name="Model" type="string" minOccurs="0"/>
<element name="IssueNo" type="string" minOccurs="0"/>
<element name="Expiry" type="string" minOccurs="0"/>
</sequence>
</complexType>
<complexType name="DeviceType">
<sequence>
<element name="DeviceId" type="pskc:DeviceIdType"
minOccurs="0"/>
<element name="Key" type="pskc:KeyType"
maxOccurs="unbounded"/>
<element name="User" type="pskc:UserType"
minOccurs="0"/>
</sequence>
</complexType>
<complexType name="UserType">
<sequence>
<sequence>
<element name="UserId" type="string" minOccurs="0"/>
<element name="FirstName" type="string" minOccurs="0"/>
<element name="LastName" minOccurs="0"/>
</sequence>
<element name="Org" type="string" minOccurs="0"/>
</sequence>
</complexType>
<complexType name="UsageType">
<sequence>
<element name="AlgorithmIdentifier"
type="pskc:AlgorithmIdentifierType" minOccurs="0"/>
<element name="ResponseFormat">
<complexType>
<attribute name="format" type="pskc:valueFormat"
use="required"/>
<attribute name="length" type="unsignedInt" use="required"/>
<attribute name="checkDigits" type="boolean" use="optional"
default="false"/>
```

```
   </complexType>
   </element>
   <element name="ChallengeFormat" minOccurs="0">
   <complexType>
   <attribute name="format" type="pskc:valueFormat"
   use="required"/>
   <attribute name="min" type="unsignedInt" use="required"/>
   <attribute name="max" type="unsignedInt" use="required"/>
   <attribute name="checkDigits" type="boolean" use="optional"
   default="false"/>
   </complexType>
   </element>
   <element name="Time" type="unsignedLong" minOccurs="0"/>
   <element name="AppProfileId" type="string" minOccurs="0"/>
   </sequence>
   <attribute name="otp" type="boolean" use="optional"
   default="false"/>
   <attribute name="cr" type="boolean" use="optional"
   default="false"/>
   <attribute name="sign" type="boolean" use="optional"
   default="false"/>
   <attribute name="encrypt" type="boolean" use="optional"
   default="false"/>
   <attribute name="unlock" type="boolean" use="optional"
   default="false"/>
   </complexType>
   <complexType name="AttributeType">
   <simpleContent>
   <extension base="string">
   <attribute name="name" type="string" use="required"/>
   </extension>
   </simpleContent>
   </complexType>
   <complexType name="EncryptionMethodType">
   <sequence>
   <element name="EncKeyLabel" minOccurs="0"/>
   <choice>
           <sequence>
             <element name="KeyInfo"
             type="ds:KeyInfoType" minOccurs="0"/>
             <element name="OAEPParams"
             type="base64Binary" minOccurs="0"/>
             <element name="HashAlgorithm"
             type="pskc:HashAlgorithmType" minOccurs="0"/>
           </sequence>
   <sequence>
```

```
   <element name="PBESalt" type="base64Binary"
   minOccurs="0"/>
```



```
   <element name="PBEIterationCount" type="int"
   minOccurs="0"/>
   <element name="IV" type="base64Binary" minOccurs="0"/>
   </sequence>
   </choice>
   </sequence>
   <attribute name="algorithm"
   type="pskc:EncryptionAlgorithmType" use="required"/>
   </complexType>
   <complexType name="DigestMethodType">
   <sequence>
   <element name="DigestKeyLabel" minOccurs="0"/>
   </sequence>
   <attribute name="algorithm"
   type="pskc:DigestAlgorithmType" use="required"/>
   <attribute name="ext-algorithm" type="string"/>
   </complexType>
   <simpleType name="EncryptionAlgorithmType">
   <restriction base="string">
   <enumeration value="NONE"/>
   <enumeration value="PBE-3DES112-CBC"/>
   <enumeration value="PBE-3DES168-CBC"/>
   <enumeration value="PBE-AES128-CBC"/>
   <enumeration value="PBE-AES256-CBC"/>
   <enumeration value="PBE-AES192-CBC"/>
   <enumeration value="3DES112-CBC"/>
   <enumeration value="3DES168-CBC"/>
   <enumeration value="AES128-CBC"/>
   <enumeration value="AES192-CBC"/>
   <enumeration value="AES256-CBC"/>
   <enumeration value="RSA-1_5"/>
   <enumeration value="RSA-OAEP-MGF1P"/>
   <enumeration value="OTHER"/>
   </restriction>
   </simpleType>
   <simpleType name="DigestAlgorithmType">
   <restriction base="string">
   <enumeration value="HMAC-SHA1"/>
   <enumeration value="HMAC-SHA256"/>
```

```
<enumeration value="HMAC-SHA512"/>
<enumeration value="OTHER"/>
</restriction>
</simpleType>
  <simpleType name="HashAlgorithmType">
    <restriction base="string">
      <enumeration value="SHA1"/>
      <enumeration value="SHA256"/>
      <enumeration value="SHA512"/>
```



```
    </restriction>
  </simpleType>
<simpleType name="KeyAlgorithmType">
<restriction base="string">
<enumeration value="3DES112"/>
<enumeration value="3DES168"/>
<enumeration value="ACTI"/>
<enumeration value="AES128"/>
<enumeration value="AES192"/>
<enumeration value="AES256"/>
<enumeration value="ANSIX9.9"/>
<enumeration value="DES"/>
<enumeration value="HOTP"/>
<enumeration value="MKEYLABEL"/>
<enumeration value="RSASECUREID"/>
<enumeration value="VASCO"/>
<enumeration value="OTHER"/>
</restriction>
</simpleType>
<simpleType name="valueFormat">
<restriction base="string">
<enumeration value="DECIMAL"/>
<enumeration value="HEXADECIMAL"/>
<enumeration value="ALPHANUMERIC"/>
<enumeration value="BASE64"/>
<enumeration value="BINARY"/>
</restriction>
</simpleType>
<simpleType name="VersionType" final="restriction">
<restriction base="string">
<pattern value="\d{1,9}\.\d{0,9}"/>
</restriction>
```

```
</simpleType>
<element name="KeyContainer"
type="pskc:KeyContainerType"/>
<complexType name="DataType">
<sequence>
<element name="Value" type="base64Binary"/>
<element name="ValueDigest"
type="base64Binary" minOccurs="0"/>
</sequence>
</complexType>
</schema>
```


8. 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.

8.1. 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 credential record 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.

8.2. Payload integrity

The portable symmetric key container provides a means 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.

8.3. 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.

9. Acknowledgements

This work initiated from a joint effort by the members of OATH (Initiative for Open AuTHentication). 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, Jon Martinson, Siddhart Bajaj, Stu Veath, Kevin Lewis, and Andrea Doherty.

10. Appendix A - Example Symmetric Key Containers

All examples are syntactically correct and compatible with the XML schema in section 7. However, <Signature>, Key <Value> and Key

<ValueDigest> data values are fictitious

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

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer
xmlns="urn:ietf:params:xml:ns:keyprov:container"
xmlns:logo="urn:ietf:params:xml:ns:keyprov:logo"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:keyprov:container
keyprov_pskc_schema_v1.1.xsd" version="1.1">
  <EncryptionMethod algorithm="NONE"/>
  <DigestMethod algorithm="HMAC-SHA1"></DigestMethod>
  <Device>
    <DeviceId>
      <Manufacturer>Token Manufacturer</Manufacturer>
      <SerialNo>98765432187</SerialNo>
      <Expiry>01/01/2008</Expiry>
    </DeviceId>
    <Key KeyAlgorithm="HOTP"  KeyId="98765432187">
      <Issuer>Credential Issuer</Issuer>
      <Usage>
       <ResponseFormat format="DECIMAL" length="6"/>
      </Usage>
      <FriendlyName>MyFirstToken</FriendlyName>
      <Data Name="SECRET">
        <Value>WldjTHZwRm9YTkhBRytseDMrUnc=</Value>
        <ValueDigest>WldjTHZwRm9YTkhBRytseDM=</ValueDigest>
      </Data>
      <Data Name="COUNTER">
        <Value>WldjTHZwRm9YTkhBRytseDMrUnc=</Value>
        <ValueDigest>WldjTHZwRm9YTkhBRytseDM=</ValueDigest>
      </Data>
    </Key>
  </Device>
</KeyContainer>
```

10.2. Symmetric Key Container with a single Password-based Encrypted HOTP Secret Key

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer
xmlns="urn:ietf:params:xml:ns:keyprov:container"
xmlns:logo="urn:ietf:params:xml:ns:keyprov:logo"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:keyprov:container
.\keyprov_pskc_schema_v1.1.xsd" version="1.1">
  <EncryptionMethod algorithm="PBE-3DES112-CBC">
    <PBESalt>y6TzckeLRQw=</PBESalt>
    <PBEIterationCount>999</PBEIterationCount>
  </EncryptionMethod>
  <DigestMethod algorithm="HMAC-SHA1"></DigestMethod>
  <Device>
    <DeviceId>
      <Manufacturer>Token Manufacturer</Manufacturer>
      <SerialNo>98765432187</SerialNo>
      <Expiry>01/01/2008</Expiry>
    </DeviceId>
  <Key KeyAlgorithm="HOTP"  KeyId="77654321870">
    <Issuer>Credential Issuer</Issuer>
    <Usage>
      <ResponseFormat format="DECIMAL" length="6"/>
    </Usage>
    <FriendlyName>MySecondToken</FriendlyName>
      <Data Name="SECRET">
<Value>7JHUyp3azOkqJENSsh6b2vxXzwGBYypzJxEr+ikQAa229KV/BgZhGA==</Value>
      <ValueDigest>WldjTHZwRm9YTkhBRytseDMrUnc=</ValueDigest>
      </Data>
      <Data Name="COUNTER">
<Value>7JHUyp3azOkqJENSsh6b2vxXzwGBYypzJxEr+ikQAa229KV/BgZhGA==</Value>
      <ValueDigest>WldjTHZwRm9YTkhBRytseDMrUnc=</ValueDigest>
      </Data>
    </Key>
  </Device>
</KeyContainer>
```

11. Normative References

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

[DSKPP] Doherty, A., Pei, M., Nystroem, M., and S. Machani, "Dynamic Symmetric Key Provisioning Protocol", Internet Draft Informational, URL: http://tools.ietf.org/id/draft-ietf-keyprov-dskpp-00.txt, July 2007.

[HOTP] MRaihi, D., Bellare, M., Hoornaert, F., Naccache, D., and O. Ranen, "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.

[OCRA] MRaihi, D., Rydell, J., Machani, S., and S. Bajaj, "OCRA: OATH Challenge Response Algorithm", Internet Draft Informational, URL: http://www.ietf.org/internet-drafts/draft-mraihi-mutual-oath-hotp-variants-05.txt, December 2005.

[PKCS1] Kaliski, B. and J. Staddon, "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] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[Schneier] Schneier, B., "Secrets and Lies: Digitial Security in a Networked World", Wiley Computer Publishing, ISBN 0-8493-8253-7, 2000.

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

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

Authors' Addresses

Philip Hoyer
ActivIdenity, 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

Shuh Chang
Gemalto Inc.
9442 Capital of Texas Hwy. North
Suite 400, Arboretum Plaza II
Austin, Texas 78759
USA

Phone: +1 512 257 3859
Email: shuh.chang@gemalto.com

Full Copyright Statement

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at 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).