Internet Engineering Task Force (IETF)
Request for Comments: 6030
Category: Standards Track
ISSN: 2070-1721

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

October 2010

Portable Symmetric Key Container (PSKC)

Abstract

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

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6030.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 4
   1.1. Key Words 4
   1.2. Version Support 4
   1.3. Namespace Identifiers 5
        1.3.1. Defined Identifiers 5
        1.3.2. Referenced Identifiers 5
2. Terminology 6
3. Portable Key Container Entities Overview and Relationships 6
4. <KeyContainer> Element: The Basics 8
   4.1. <Key>: Embedding Keying Material and Key-Related Information 8
   4.2. Key Value Encoding 10
        4.2.1. AES Key Value Encoding 11
        4.2.2. Triple-DES Key Value Encoding 11
   4.3. Transmission of Supplementary Information 12
        4.3.1. <DeviceInfo> Element: Unique Device Identification 13
        4.3.2. <CryptoModuleInfo> Element: CryptoModule Identification 15
        4.3.3. <UserId> Element: User Identification 15
        4.3.4. <AlgorithmParameters> Element: Supplementary Information for OTP and CR Algorithms 15
   4.4. Transmission of Key Derivation Values 17
5. Key Policy 19
   5.1. PIN Algorithm Definition 23
6. Key Protection Methods 23
   6.1. Encryption Based on Pre-Shared Keys 24
        6.1.1. MAC Method 26
   6.2. Encryption Based on Passphrase-Based Keys 27
   6.3. Encryption Based on Asymmetric Keys 29

6.4. Padding of Encrypted Values for Non-Padded Encryption Algorithms 31
7. Digital Signature 31
8. Bulk Provisioning 33
9. Extensibility 35
10. PSKC Algorithm Profile 36
10.1. HOTP 36
10.2. PIN 37
11. XML Schema 38
12. IANA Considerations 44
12.1. Content-Type Registration for 'application/pskc+xml' 44
12.2. XML Schema Registration 45
12.3. URN Sub-Namespace Registration 46
12.4. PSKC Algorithm Profile Registry 46
12.5. PSKC Version Registry 47
12.6. Key Usage Registry 47
13. Security Considerations 48
13.1. PSKC Confidentiality 49
13.2. PSKC Integrity 50
13.3. PSKC Authenticity 50
14. Contributors 50
15. Acknowledgements 50
16. References 51
16.1. Normative References 51
16.2. Informative References 52
Appendix A. Use Cases 54
A.1. Online Use Cases 54
A.1.1. Transport of Keys from Server to Cryptographic Module 54
A.1.2. Transport of Keys from Cryptographic Module to Cryptographic Module 54
A.1.3. Transport of Keys from Cryptographic Module to Server 55
A.1.4. Server-to-Server Bulk Import/Export of Keys 55
A.2. Offline Use Cases 55
A.2.1. Server-to-Server Bulk Import/Export of Keys 55
Appendix B. Requirements 56

1. Introduction

With the increasing use of symmetric-key-based systems, such as encryption of data at rest or systems used for strong authentication, such as those based on One-Time Password (OTP) and Challenge/Response (CR) mechanisms, there is a need for vendor interoperability and a standard format for importing and exporting (provisioning) symmetric keys. For instance, traditionally, vendors of authentication servers and service providers have used proprietary formats for importing and exporting these keys into their systems, thus making it hard to use tokens from two different vendors.

This document defines a standardized XML-based key container, called Portable Symmetric Key Container (PSKC), for transporting symmetric keys and key-related metadata. The document also specifies the information elements that are required when the symmetric key is utilized for specific purposes, such as the initial counter in the HMAC-Based One-Time Password (HOTP) [HOTP] algorithm. It also creates an IANA registry for algorithm profiles where algorithms, their metadata and PSKC transmission profile can be recorded for a centralized, standardized reference.

1.1. Key Words

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].

1.2. Version Support

There is a provision made in the syntax for an explicit version number. Only version "1.0" is currently specified.

The numbering scheme for PSKC versions is "<major>.<minor>". The major and minor numbers MUST be treated as separate integers and each number MAY be incremented higher than a single digit. Thus, "PSKC 2.4" would be a lower version than "PSKC 2.13", which in turn would be lower than "PSKC 12.3". Leading zeros (e.g., "PSKC 6.01") MUST be ignored by recipients and MUST NOT be sent.

The major version number should be incremented only if the message format (e.g., element structure) has changed so dramatically that an older version implementation would not be able to interoperate with a newer version. The minor version number indicates new capabilities, and it MUST be ignored by an entity with a smaller minor version number but used for informational purposes by the entity with the larger minor version number.

1.3. Namespace Identifiers

This document uses Uniform Resource Identifiers (URIs) [RFC3986] to identify resources, algorithms, and semantics.

1.3.1. Defined Identifiers

The XML namespace [XMLNS] URI for Version 1.0 of PSKC is:

"urn:ietf:params:xml:ns:keyprov:pskc"

References to qualified elements in the PSKC schema defined in this specification and used in the example use the prefix "pskc" (defined as xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"). It is RECOMMENDED to use this namespace in implementations.

1.3.2. Referenced Identifiers

The PSKC syntax presented in this document relies on algorithm identifiers and elements defined in the XML Signature [XMLDSIG] namespace:

xmlns:ds="http://www.w3.org/2000/09/xmldsig#"

References to the XML Signature namespace are represented by the prefix "ds".

PSKC also relies on algorithm identifiers and elements defined in the XML Encryption [XMLENC] namespace:

xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"

References to the XML Encryption namespace are represented by the prefix "xenc".

When protecting keys in transport with passphrase-based keys, PSKC also relies on the derived key element defined in the XML Encryption Version 1.1 [XMLENC11] namespace:

xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"

References to the XML Encryption Version 1.1 namespace are represented by the prefix "xenc11".

When protecting keys in transport with passphrase-based keys, PSKC also relies on algorithm identifiers and elements defined in the PKCS #5 [PKCS5] namespace:

xmlns:pkcs5=
"http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#"

References to the PKCS #5 namespace are represented by the prefix "pkcs5".

2. Terminology

NOTE: In subsequent sections of the document, we highlight **mandatory** XML elements and attributes. Optional elements and attributes are not explicitly indicated, i.e., if it does not say mandatory, it is optional.

3. Portable Key Container Entities Overview and Relationships

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

1. KeyContainer entity - representing the container that carries a number of KeyPackage entities. A valid container MUST carry at least one KeyPackage entity.

2. KeyPackage entity - representing the package of at most one key and its related provisioning endpoint or current usage endpoint, such as a physical or virtual device and a specific CryptoModule.

3. DeviceInfo entity - representing the information about the device and criteria to identify uniquely the device.

4. CryptoModuleInfo entity - representing the information about the CryptoModule where the keys reside or to which they are provisioned.

5. Key entity - representing the key transported or provisioned.

6. Data entity - representing a list of metadata related to the key, where the element name is the name of the metadata and its associated value is either in encrypted (for example, for <Data> element <Secret>) or plaintext (for example, the <Data> element <Counter>) form.

Figure 1 shows the high-level structure of the PSKC data elements.

```
      -----------------
      | KeyContainer  |
      |---------------|
      | EncryptionKey |
      | Signature     |
      | ...           |
      -----------------
              |
              |
             /|\ 1..n
      ----------------        ----------------
      | KeyPackage   |    0..1| DeviceInfo   |
      |--------------|--------|--------------|
      |              |--      | SerialNumber |
      ----------------  |     | Manufacturer |
              |         |     | ....         |
              |         |     ----------------
             /|\ 0..1   |
      ----------------  |     --------------------
      | Key          |  | 0..1| CryptoModuleInfo |
      |--------------|   -----|------------------|
      | Id           |        | Id               |
      | Algorithm    |        |....              |
      | UserId       |        --------------------
      | Policy       |
      | ....         |
      ----------------
              |
              |
             /|\ 0..n
          ---------------------------------------- -  -
          |                     |              |
      ------------------  ----------------  -------- - -
      | Data:Secret    |  | Data:Counter |  | Data:other
      |----------------|  |--------------|  |-- - -
      | EncryptedValue |  | PlainValue   |
      | ValueMAC       |  ----------------
      ------------------
```

Figure 1: PSKC Data Elements Relationship Diagram

The following sections describe in detail all the entities and related XML schema elements and attributes.

4. <KeyContainer> Element: The Basics

In its most basic form, a PSKC document uses the top-level element <KeyContainer> and a single <KeyPackage> element to carry key information.

The following example shows a simple PSKC document. We will use it to describe the structure of the <KeyContainer> element and its child elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
    Id="exampleID1"
    xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
    <KeyPackage>
        <Key Id="12345678"
            Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Issuer-A</Issuer>
            <Data>
                <Secret>
                    <PlainValue>MTIzNA==
                    </PlainValue>
                </Secret>
            </Data>
        </Key>
    </KeyPackage>
</KeyContainer>
```

Figure 2: Basic PSKC Key Container Example

The attributes of the <KeyContainer> element have the following semantics:

'Version': The 'Version' attribute is used to identify the version of the PSKC schema version. This specification defines the initial version ("1.0") of the PSKC schema. This attribute MUST be included.

'Id': The 'Id' attribute carries a unique identifier for the container. As such, it helps to identify a specific key container in cases in which multiple containers are embedded in larger XML documents.

4.1. <Key>: Embedding Keying Material and Key-Related Information

The following attributes of the <Key> element MUST be included at a minimum:

'Id': This attribute carries a unique identifier for the symmetric key in the context of key provisioning exchanges between two parties. This means that if PSKC is used in multiple interactions between a sending and receiving party, using different containers referencing the same keys, the 'Id' attribute of <Key> MUST use the same value (e.g., after initial provisioning, if a system wants to update key metadata values in the other system, the value of the 'Id' attribute of the <Key> where the metadata is to be updated MUST be the same of the original 'Id' attribute value provisioned). The identifier is defined as a string of alphanumeric characters.

'Algorithm': This attribute contains a unique identifier for the PSKC algorithm profile. This profile associates specific semantics to the elements and attributes contained in the <Key> element. This document describes profiles for open standards algorithms in Section 10. Additional profiles are defined in the following informative document: [PSKC-ALGORITHM-PROFILES].

The <Key> element has a number of optional child elements. An initial set is described below:

<Issuer>: This element represents the name of the party that issued the key. For example, a bank "Foobar Bank, Inc." issuing hardware tokens to their retail banking users may set this element to 'Foobar Bank, Inc.'.

<FriendlyName>: A human-readable name for the secret key for easier reference. This element serves informational purposes only. This element is a language-dependent string; hence, it SHOULD have an attribute xml:lang="xx" where xx is the language identifier as specified in [RFC5646]. If no xml:lang attribute is present, implementations MUST assume the language to be English as defined by setting the attribute value to 'en' (e.g., xml:lang="en").

<AlgorithmParameters>: This element carries parameters that influence the result of the algorithmic computation, for example, response truncation and format in OTP and CR algorithms. A more detailed discussion of the element can be found in Section 4.3.4.

<Data>: This element carries data about and related to the key. The following child elements are defined for the <Data> element:

<Secret>: This element carries the value of the key itself in a binary representation. Please see Section 4.2 for more details on Key Value Encoding.

<Counter>: This element contains the event counter for event-based OTP algorithms.

<Time>: This element contains the time for time-based OTP algorithms. (If time intervals are used, this element carries the number of time intervals passed from a specific start point, normally it is algorithm dependent).

<TimeInterval>: This element carries the time interval value for time-based OTP algorithms in seconds (a typical value for this would be 30, indicating a time interval of 30 seconds).

<TimeDrift>: This element contains the device clock drift value for time-based OTP algorithms. The integer value (positive or negative drift) that indicates the number of time intervals that a validation server has established the device clock drifted after the last successful authentication. So, for example, if the last successful authentication established a device time value of 8 intervals from a specific start date but the validation server determines the time value at 9 intervals, the server SHOULD record the drift as -1.

All the elements listed above (and those defined in the future) obey a simple structure in that they MUST support child elements to convey the data value in either plaintext or encrypted format:

Plaintext: The <PlainValue> element carries a plaintext value that is typed, for example, to xs:integer.

Encrypted: The <EncryptedValue> element carries an encrypted value.

ValueMAC: The <ValueMAC> element is populated with a Message Authentication Code (MAC) generated from the encrypted value in case the encryption algorithm does not support integrity checks. The example shown in Figure 2 illustrates the usage of the <Data> element with two child elements, namely <Secret> and <Counter>. Both elements carry a plaintext value within the <PlainValue> child element.

4.2. Key Value Encoding

Two parties receiving the same key value OCTET STRING, resulting in decoding the xs:base64Binary, inside the <PlainValue> or <EncryptedValue> elements, must make use of the key in exactly the same way in order to interoperate. To ensure that, it is necessary to define a correspondence between the OCTET STRING and the notation in the standard algorithm description that defines how the key is

used. The next sections establish that correspondence for the AES algorithm [FIPS197] and the Triple Data Encryption Algorithm (TDEA or Triple DES) [SP800-67]. Unless otherwise specified for a specific algorithm, the OCTET STRING encoding MUST follow the AES Key Value Encoding.

4.2.1. AES Key Value Encoding

[FIPS197], Section 5.2, titled "Key Expansion", uses the input key as an array of bytes indexed starting at 0. The first octet of the OCTET STRING SHALL become the key byte in the AES, labeled index 0 in [FIPS197]; the succeeding octets of the OCTET STRING SHALL become key bytes in AES, in increasing index order.

Proper parsing and key load of the contents of the OCTET STRING for AES SHALL be determined by using the following value for the <PlainValue> element (binaryBase64-encoded) to generate and match the key expansion test vectors in [FIPS197], Appendix A, for AES

Cipher Key: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

```
...
 <PlainValue>K34VFiiu0qar9xWICc9PPA==</PlainValue>
...
```

4.2.2. Triple-DES Key Value Encoding

A Triple-DES key consists of three keys for the cryptographic engine (Key1, Key2, and Key3) that are each 64 bits (56 key bits and 8 parity bits); the three keys are also collectively referred to as a key bundle [SP800-67]. A key bundle may employ either two or three independent keys. When only two independent keys are employed (called two-key Triple DES), the same value is used for Key1 and Key3.

Each key in a Triple-DES key bundle is expanded into a key schedule according to a procedure defined in [SP800-67], Appendix A. That procedure numbers the bits in the key from 1 to 64, with number 1 being the leftmost, or most significant bit (MSB). The first octet of the OCTET STRING SHALL be bits 1 through 8 of Key1 with bit 1 being the MSB. The second octet of the OCTET STRING SHALL be bits 9 through 16 of Key1, and so forth, so that the trailing octet of the OCTET STRING SHALL be bits 57 through 64 of Key3 (or Key2 for two-key Triple DES).

Proper parsing and key load of the contents of the OCTET STRING for Triple DES SHALL be determined by using the following <PlainValue> element (binaryBase64-encoded) to generate and match the key expansion test vectors in [SP800-67], Appendix B, for the key bundle:

```
Key1 = 0123456789ABCDEF

Key2 = 23456789ABCDEF01

Key3 = 456789ABCDEF0123

...
 <PlainValue>ASNFZ4mrze8jRWeJq83vAUVniavN7wEj</PlainValue>
...
```

4.3. Transmission of Supplementary Information

A PSKC document can contain a number of additional information regarding device identification, cryptographic module identification, user identification, and parameters for usage with OTP and CR algorithms. The following example, see Figure 3, is used as a reference for the subsequent sub-sections.

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
    Id="exampleID1"
    xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
    <KeyPackage>
        <DeviceInfo>
            <Manufacturer>Manufacturer</Manufacturer>
            <SerialNo>987654321</SerialNo>
            <UserId>DC=example-bank,DC=net</UserId>
        </DeviceInfo>
        <CryptoModuleInfo>
            <Id>CM_ID_001</Id>
        </CryptoModuleInfo>
        <Key Id="12345678"
            Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Issuer</Issuer>
            <AlgorithmParameters>
                <ResponseFormat Length="8" Encoding="DECIMAL"/>
            </AlgorithmParameters>
            <Data>
                <Secret>
                    <PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
                    </PlainValue>
                </Secret>
                <Counter>
                    <PlainValue>0</PlainValue>
                </Counter>
            </Data>
            <UserId>UID=jsmith,DC=example-bank,DC=net</UserId>
        </Key>
    </KeyPackage>
</KeyContainer>
```

Figure 3: PSKC Key Container Example with Supplementary Data

4.3.1. <DeviceInfo> Element: Unique Device Identification

The <DeviceInfo> element uniquely identifies the device to which the <KeyPackage> is provisioned. Since devices can come in different form factors, such as hardware tokens, smart-cards, soft tokens in a mobile phone, or as a PC, this element allows different child element combinations to be used. When combined, the values of the child elements MUST uniquely identify the device. For example, for hardware tokens, the combination of <SerialNo> and <Manufacturer> elements uniquely identifies a device, but the <SerialNo> element alone is insufficient since two different token manufacturers might issue devices with the same serial number (similar to the Issuer Distinguished Name and serial number of a certificate).

The <DeviceInfo> element has the following child elements:

<Manufacturer>: This element indicates the manufacturer of the device. Values for the <Manufacturer> element MUST be taken from either [OATHMAN] prefixes (i.e., the left column) or from the IANA Private Enterprise Number Registry [IANAPENREG], using the Organization value. When the value is taken from [OATHMAN], "oath." MUST be prepended to the value (e.g., "oath.<prefix value from [OATHMAN]>"). When the value is taken from [IANAPENREG], "iana." MUST be prepended to the value (e.g., "iana.<Organization value from [IANAPENREG]>").

<SerialNo>: This element contains the serial number of the device.

<Model>: This element describes the model of the device (e.g., one-button-HOTP-token-V1).

<IssueNo>: This element contains the issue number in case there are devices with the same serial number so that they can be distinguished by different issue numbers.

<DeviceBinding>: This element allows a provisioning server to ensure that the key is going to be loaded into the device for which the key provisioning request was approved. The device is bound to the request using a device identifier, e.g., an International Mobile Equipment Identity (IMEI) for the phone, or an identifier for a class of identifiers, e.g., those for which the keys are protected by a Trusted Platform Module (TPM).

<StartDate> and <ExpiryDate>: These two elements indicate the start and end date of a device (such as the one on a payment card, used when issue numbers are not printed on cards). The date MUST be expressed as a dateTime value in "canonical representation" [W3C.REC-xmlschema-2-20041028]. Implementations SHOULD NOT rely on time resolution finer than milliseconds and MUST NOT generate time instants that specify leap seconds. Keys that reside on the device SHOULD only be used when the current date is after the <StartDate> and before the <ExpiryDate>. Note that usage enforcement of the keys with respect to the dates MAY only happen on the validation server, as some devices such as smart cards do not have an internal clock. Systems thus SHOULD NOT rely upon the device to enforce key usage date restrictions.

Depending on the device type, certain child elements of the <DeviceInfo> element MUST be included in order to uniquely identify a device. This document does not enumerate the different device types and therefore does not list the elements that are mandatory for each type of device.

4.3.2. <CryptoModuleInfo> Element: CryptoModule Identification

The <CryptoModuleInfo> element identifies the cryptographic module to which the symmetric keys are or have been provisioned. This allows the identification of the specific cases where a device MAY contain more than one crypto module (e.g., a PC hosting a TPM and a connected token).

The <CryptoModuleInfo> element has a single child element that MUST be included:

<Id>: This element carries a unique identifier for the CryptoModule and is implementation specific. As such, it helps to identify a specific CryptoModule to which the key is being or was provisioned.

4.3.3. <UserId> Element: User Identification

The <UserId> element identifies the user of a distinguished name, as defined in [RFC4514], for example, UID=jsmith,DC=example,DC=net.

Although the syntax of the user identifier is defined, there are no semantics associated with this element, i.e., there are no checks enforcing that only a specific user can use this key. As such, this element is for informational purposes only.

This element may appear in two places, namely as a child element of the <Key> element, where it indicates the user with whom the key is associated, and as a child element of the <DeviceInfo> element, where it indicates the user with whom the device is associated.

4.3.4. <AlgorithmParameters> Element: Supplementary Information for OTP and CR Algorithms

The <AlgorithmParameters> element is a child element of the <Key> element, and this document defines three child elements: <Suite>, <ChallengeFormat>, and <ResponseFormat>.

<Suite>:

The optional <Suite> element defines additional characteristics of the algorithm used, which are algorithm specific. For example, in an HMAC-based (Hashed MAC) OTP algorithm, it could designate the strength of the hash algorithm used (SHA1, SHA256, etc.). Please refer to the algorithm profile section, Section 10, for the exact semantics of the value for each algorithm profile.

<ChallengeFormat>:

The <ChallengeFormat> element defines the characteristics of the challenge in a CR usage scenario whereby the following attributes are defined:

'Encoding': This attribute, which MUST be included, defines the encoding of the challenge accepted by the device and MUST be one of the following values:

DECIMAL: Only numerical digits

HEXADECIMAL: Hexadecimal response

ALPHANUMERIC: All letters and numbers (case sensitive)

BASE64: Base-64 encoded, as defined in Section 4 of [RFC4648]

BINARY: Binary data

'CheckDigit': This attribute indicates whether a device needs to check the appended Luhn check digit, as defined in [ISOIEC7812], contained in a challenge. This is only valid if the 'Encoding' attribute is set to 'DECIMAL'. A value of TRUE indicates that the device will check the appended Luhn check digit in a provided challenge. A value of FALSE indicates that the device will not check the appended Luhn check digit in the challenge.

'Min': This attribute defines the minimum size of the challenge accepted by the device for CR mode and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the minimum number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the minimum number of bytes of the unencoded value.

'Max': This attribute defines the maximum size of the challenge accepted by the device for CR mode and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the maximum number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the maximum number of bytes of the unencoded value.

<ResponseFormat>:

The <ResponseFormat> element defines the characteristics of the result of a computation and defines the format of the OTP or the response to a challenge. For cases in which the key is a PIN value, this element contains the format of the PIN itself (e.g., DECIMAL, length 4 for a 4-digit PIN). The following attributes are defined:

'Encoding': This attribute defines the encoding of the response generated by the device, it MUST be included and MUST be one of the following values: DECIMAL, HEXADECIMAL, ALPHANUMERIC, BASE64, or BINARY.

'CheckDigit': This attribute indicates whether the device needs to append a Luhn check digit, as defined in [ISOIEC7812], to the response. This is only valid if the 'Encoding' attribute is set to 'DECIMAL'. If the value is TRUE, then the device will append a Luhn check digit to the response. If the value is FALSE, then the device will not append a Luhn check digit to the response.

'Length': This attribute defines the length of the response generated by the device and MUST be included. If the 'Encoding' attribute is set to 'DECIMAL', 'HEXADECIMAL', or ALPHANUMERIC, this value indicates the number of digits/characters. If the 'Encoding' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

4.4. Transmission of Key Derivation Values

<KeyProfileId> element, which is a child element of the <Key> element, carries a unique identifier used between the sending and receiving parties to establish a set of key attribute values that are not transmitted within the container but are agreed upon between the two parties out of band. This element will then represent the unique reference to a set of key attribute values. (For example, a smart card application personalization profile id related to specific attribute values present on a smart card application that have influence when computing a response).

For example, in the case of MasterCard's Chip Authentication Program [CAP], the sending and the receiving party would agree that KeyProfileId='1' represents a certain set of values (e.g., Internet Authentication Flag (IAF) set to a specific value). During transmission of the <KeyContainer>, these values would not be transmitted as key attributes but would only be referred to via the

<KeyProfileId> element set to the specific agreed-upon profile (in this case '1'). The receiving party can then associate all relevant key attributes contained in the profile that was agreed upon out of band with the imported keys. Often, this methodology is used between a manufacturing service, run by company A, and the validation service, run by company B, to avoid repeated transmission of the same set of key attribute values.

The <KeyReference> element contains a reference to an external key to be used with a key derivation scheme. In this case, the parent <Key> element will not contain the <Secret> subelement of <Data>, in which the key value (secret) is transported; only the reference to the external master key is transported (e.g., a PKCS #11 key label).

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0" Id="exampleID1"
     xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
    <KeyPackage>
        <DeviceInfo>
            <Manufacturer>Manufacturer</Manufacturer>
            <SerialNo>987654321</SerialNo>
        </DeviceInfo>
        <CryptoModuleInfo>
            <Id>CM_ID_001</Id>
        </CryptoModuleInfo>
        <Key Id="12345678"
         Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Issuer</Issuer>
            <AlgorithmParameters>
                <ResponseFormat Length="8" Encoding="DECIMAL"/>
            </AlgorithmParameters>
            <KeyProfileId>keyProfile1</KeyProfileId>
            <KeyReference>MasterKeyLabel
            </KeyReference>
            <Data>
                <Counter>
                    <PlainValue>0</PlainValue>
                </Counter>
            </Data>
            <Policy>
                <KeyUsage>OTP</KeyUsage>
            </Policy>
        </Key>
    </KeyPackage>
</KeyContainer>
```

Figure 4: Example of a PSKC Document Transmitting an HOTP Key via Key Derivation Values

The key value will be derived using the value of the <SerialNo> element, values agreed upon between the sending and the receiving parties and identified by the <KeyProfile> 'keyProfile1', and an externally agreed-upon key referenced by the label 'MasterKeyLabel'.

5. Key Policy

This section illustrates the functionality of the <Policy> element within PSKC, which allows a key usage and key PIN protection policy to be attached to a specific key and its related metadata. This element is a child element of the <Key> element.

If the <Policy> element contains child elements or values within elements/attributes that are not understood by the recipient of the PSKC document, then the recipient MUST assume that key usage is not permitted. This statement ensures that the lack of understanding of certain extensions does not lead to unintended key usage.

We will start our description with an example that expands the example shown in Figure 3.

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer
    Version="1.0" Id="exampleID1"
    xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
    <KeyPackage>
        <DeviceInfo>
            <Manufacturer>Manufacturer</Manufacturer>
            <SerialNo>987654321</SerialNo>
        </DeviceInfo>
        <CryptoModuleInfo>
            <Id>CM_ID_001</Id>
        </CryptoModuleInfo>
        <Key Id="12345678"
            Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Issuer</Issuer>
            <AlgorithmParameters>
                <ResponseFormat Length="8" Encoding="DECIMAL"/>
            </AlgorithmParameters>
            <Data>
                <Secret>
                    <PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
                    </PlainValue>
                </Secret>
                <Counter>
                    <PlainValue>0</PlainValue>
```

```
                    </Counter>
                </Data>
                <Policy>
                    <PINPolicy MinLength="4" MaxLength="4"
                        PINKeyId="123456781" PINEncoding="DECIMAL"
                        PINUsageMode="Local"/>
                    <KeyUsage>OTP</KeyUsage>
                </Policy>
            </Key>
        </KeyPackage>
        <KeyPackage>
            <DeviceInfo>
                <Manufacturer>Manufacturer</Manufacturer>
                <SerialNo>987654321</SerialNo>
            </DeviceInfo>
            <CryptoModuleInfo>
                <Id>CM_ID_001</Id>
            </CryptoModuleInfo>
            <Key Id="123456781"
                Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:pin">
                <Issuer>Issuer</Issuer>
                <AlgorithmParameters>
                    <ResponseFormat Length="4" Encoding="DECIMAL"/>
                </AlgorithmParameters>
                <Data>
                    <Secret>
                        <PlainValue>MTIzNA==</PlainValue>
                    </Secret>
                </Data>
            </Key>
        </KeyPackage>
    </KeyContainer>
```

Figure 5: Non-Encrypted HOTP Secret Key Protected by PIN

This document defines the following <Policy> child elements:

<StartDate> and <ExpiryDate>: These two elements denote the validity period of a key. It MUST be ensured that the key is only used between the start and the end date (inclusive). The date MUST be expressed as a dateTime value in "canonical representation" [W3C.REC-xmlschema-2-20041028]. Implementations SHOULD NOT rely on time resolution finer than milliseconds and MUST NOT generate time instants that specify leap seconds. When this element is absent, the current time is assumed as the start time.

<NumberOfTransactions>: The value in this element indicates the maximum number of times a key carried within the PSKC document can be used by an application after having received it. When this element is omitted, there is no restriction regarding the number of times a key can be used.

<KeyUsage>: The <KeyUsage> element puts constraints on the intended usage of the key. The recipient of the PSKC document MUST enforce the key usage. Currently, the following tokens are registered by this document:

OTP: The key MUST only be used for OTP generation.

CR: The key MUST only be used for Challenge/Response purposes.

Encrypt: The key MUST only be used for data encryption purposes.

Integrity: The key MUST only be used to generate a keyed message digest for data integrity or authentication purposes.

Verify: The key MUST only be used to verify a keyed message digest for data integrity or authentication purposes (this is the opposite key usage of 'Integrity').

Unlock: The key MUST only be used for an inverse Challenge/Response in the case where a user has locked the device by entering a wrong PIN too many times (for devices with PIN-input capability).

Decrypt: The key MUST only be used for data decryption purposes.

KeyWrap: The key MUST only be used for key wrap purposes.

Unwrap: The key MUST only be used for key unwrap purposes.

Derive: The key MUST only be used with a key derivation function to derive a new key (see also Section 8.2.4 of [NIST800-57]).

Generate: The key MUST only be used to generate a new key based on a random number and the previous value of the key (see also Section 8.1.5.2.1 of [NIST800-57]).

The element MAY also be repeated to allow several key usages to be expressed. When this element is absent, no key usage constraint is assumed, i.e., the key MAY be utilized for every usage.

<PINPolicy>: The <PINPolicy> element allows policy about the PIN usage to be associated with the key. The following attributes are specified:

'PINKeyId': This attribute carries the unique 'Id' attribute vale of the <Key> element held within this <KeyContainer> that contains the value of the PIN that protects the key.

'PINUsageMode': This mandatory attribute indicates the way the PIN is used during the usage of the key. The following values are defined:

Local: This value indicates that the PIN is checked locally on the device before allowing the key to be used in executing the algorithm.

Prepend: This value indicates that the PIN is prepended to the algorithm response; hence, it MUST be checked by the party validating the response.

Append: This value indicates that the PIN is appended to the algorithm response; hence, it MUST be checked by the party validating the response.

Algorithmic: This value indicates that the PIN is used as part of the algorithm computation.

'MaxFailedAttempts': This attribute indicates the maximum number of times the PIN may be entered wrongly before it MUST NOT be possible to use the key anymore (typical reasonable values are in the positive integer range of at least 2 and no more than 10).

'MinLength': This attribute indicates the minimum length of a PIN that can be set to protect the associated key. It MUST NOT be possible to set a PIN shorter than this value. If the 'PINFormat' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the number of digits/characters. If the 'PINFormat' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

'MaxLength': This attribute indicates the maximum length of a PIN that can be set to protect this key. It MUST NOT be possible to set a PIN longer than this value. If the 'PINFormat' attribute is set to 'DECIMAL', 'HEXADECIMAL', or 'ALPHANUMERIC', this value indicates the number of digits/

characters. If the 'PINFormat' attribute is set to 'BASE64' or 'BINARY', this value indicates the number of bytes of the unencoded value.

'PINEncoding': This attribute indicates the encoding of the PIN and MUST be one of the values: DECIMAL, HEXADECIMAL, ALPHANUMERIC, BASE64, or BINARY.

If the 'PinUsageMode' attribute is set to 'Local', then the device MUST enforce the restriction indicated in the 'MaxFailedAttempts', 'MinLength', 'MaxLength', and 'PINEncoding' attributes; otherwise, it MUST be enforced on the server side.

5.1. PIN Algorithm Definition

The PIN algorithm is defined as:

boolean = comparePIN(K,P)

Where:

'K' is the stored symmetric credential (PIN) in binary format.

'P' is the proposed PIN to be compared in binary format.

The function comparePIN is a straight octet comparison of K and P. Such a comparison MUST yield a value of TRUE (credentials matched) when the octet length of K is the same as the octet length of P and all octets comprising K are the same as the octets comprising P.

6. Key Protection Methods

With the functionality described in the previous sections, information related to keys had to be transmitted in cleartext. With the help of the <EncryptionKey> element, which is a child element of the <KeyContainer> element, it is possible to encrypt keys and associated information. The level of encryption is applied to the value of individual elements and the applied encryption algorithm MUST be the same for all encrypted elements. Keys are protected using the following methods: pre-shared keys, passphrase-based keys, and asymmetric keys. When encryption algorithms are used that make use of Initialization Vectors (IVs), for example, AES-128-CBC, a random IV value MUST be generated for each value to be encrypted and it MUST be prepended to the resulting encrypted value as specified in [XMLENC].

6.1. Encryption Based on Pre-Shared Keys

Figure 6 shows an example that illustrates the encryption of the content of the <Secret> element using AES-128-CBC and PKCS #5 Padding. The plaintext value of <Secret> is '3132333435363738393031323334353637383930'. The name of the pre-shared secret is "Pre-shared-key", as set in the <KeyName> element (which is a child element of the <EncryptionKey> element). The value of the encryption key used is '12345678901234567890123456789012'.

The IV for the MAC key is '11223344556677889900112233445566', and the IV for the HOTP key is '000102030405060708090a0b0c0d0e0f'.

As AES-128-CBC does not provide integrity checks, a keyed MAC is applied to the encrypted value using a MAC key and a MAC algorithm as declared in the <MACMethod> element (in our example, "http://www.w3.org/2000/09/xmldsig#hmac-sha1" is used as the algorithm and the value of the MAC key is randomly generated, in our case '11223344556677889900112233445566778899 00', and encrypted with the above encryption key). The result of the keyed-MAC computation is placed in the <ValueMAC> child element of <Secret>.

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
    xmlns="urn:ietf:params:xml:ns:keyprov:pskc"
    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>
    <MACMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
        <MACKey>
            <xenc:EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
            <xenc:CipherData>
                <xenc:CipherValue>
    ESIzRFVmd4iZABEiM0RVZgKn6WjLaTC1sbeBMSvIhRejN9vJa2BOlSaMrR7I5wSX
                </xenc:CipherValue>
            </xenc:CipherData>
        </MACKey>
    </MACMethod>
    <KeyPackage>
        <DeviceInfo>
            <Manufacturer>Manufacturer</Manufacturer>
            <SerialNo>987654321</SerialNo>
        </DeviceInfo>
        <CryptoModuleInfo>
```

```
            <Id>CM_ID_001</Id>
        </CryptoModuleInfo>
        <Key Id="12345678"
            Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Issuer</Issuer>
            <AlgorithmParameters>
                <ResponseFormat Length="8" Encoding="DECIMAL"/>
            </AlgorithmParameters>
            <Data>
                <Secret>
                    <EncryptedValue>
                        <xenc:EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                        <xenc:CipherData>
                            <xenc:CipherValue>
    AAECAwQFBgcICQoLDA0OD+cIHItlB3Wra1DUpxVvOx2lef1VmNPCMl8jwZqIUqGv
                            </xenc:CipherValue>
                        </xenc:CipherData>
                    </EncryptedValue>
                    <ValueMAC>Su+NvtQfmvfJzF6bmQiJqoLRExc=
                    </ValueMAC>
                </Secret>
                <Counter>
                    <PlainValue>0</PlainValue>
                </Counter>
            </Data>
        </Key>
    </KeyPackage>
</KeyContainer>
```

Figure 6: AES-128-CBC Encrypted Pre-Shared Secret Key with HMAC-SHA1

When protecting the payload with pre-shared keys, implementations MUST set the name of the specific pre-shared key in the <KeyName> element inside the <EncryptionKey> element. When the encryption method uses a CBC mode that requires an explicit initialization vector (IV), the IV MUST be passed by prepending it to the encrypted value.

For systems implementing PSKC, it is RECOMMENDED to support AES-128-CBC (with the URI of http://www.w3.org/2001/04/xmlenc#aes128-cbc) and KW-AES128 (with the URI of http://www.w3.org/2001/04/xmlenc#kw-aes128). Please note that KW-AES128 requires that the key to be protected must be a multiple of 8 bytes in length. Hence, if keys of a different length have to be protected, then the usage of the key-wrap algorithm with padding, as described in [RFC5649] is RECOMMENDED. Some of the encryption algorithms that can optionally be implemented are:

Algorithm	Uniform Resource Locator (URL)
AES192-CBC	http://www.w3.org/2001/04/xmlenc#aes192-cbc
AES256-CBC	http://www.w3.org/2001/04/xmlenc#aes256-cbc
TripleDES-CBC	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
Camellia128	http://www.w3.org/2001/04/xmldsig-more#camellia128
Camellia192	http://www.w3.org/2001/04/xmldsig-more#camellia192
Camellia256	http://www.w3.org/2001/04/xmldsig-more#camellia256
KW-AES128	http://www.w3.org/2001/04/xmlenc#kw-aes128
KW-AES192	http://www.w3.org/2001/04/xmlenc#kw-aes192
KW-AES256	http://www.w3.org/2001/04/xmlenc#kw-aes256
KW-TripleDES	http://www.w3.org/2001/04/xmlenc#kw-tripledes
KW-Camellia128	http://www.w3.org/2001/04/xmldsig-more#kw-camellia128
KW-Camellia192	http://www.w3.org/2001/04/xmldsig-more#kw-camellia192
KW-Camellia256	http://www.w3.org/2001/04/xmldsig-more#kw-camellia256

6.1.1. MAC Method

When algorithms without integrity checks are used, such as AES-128-CBC, a keyed-MAC value MUST be placed in the <ValueMAC> element of the <Data> element. In this case, the MAC algorithm type MUST be set in the <MACMethod> element of the <KeyContainer> element. The MAC key MUST be a randomly generated key by the sender, be pre-agreed upon between the receiver and the sender, or be set by the application protocol that carries the PSKC document. It is RECOMMENDED that the sender generate a random MAC key. When the sender generates such a random MAC key, the MAC key material MUST be encrypted with the same encryption key specified in <EncryptionKey> element of the key container. The encryption method and encrypted value MUST be set in the <EncryptionMethod> element and the <CipherData> element, respectively, of the <MACKey> element in the <MACMethod> element. The <MACKeyReference> element of the <MACMethod> element MAY be used to indicate a pre-shared MAC key or a provisioning protocol derived MAC key. For systems implementing PSKC, it is RECOMMENDED to implement the HMAC-SHA1 (with the URI of 'http://www.w3.org/2000/09/xmldsig#hmac-sha1'). Some of the MAC algorithms that can optionally be implemented are:

Algorithm	Uniform Resource Locator (URL)
HMAC-SHA224	http://www.w3.org/2001/04/xmldsig-more#hmac-sha224
HMAC-SHA256	http://www.w3.org/2001/04/xmldsig-more#hmac-sha256
HMAC-SHA384	http://www.w3.org/2001/04/xmldsig-more#hmac-sha384
HMAC-SHA512	http://www.w3.org/2001/04/xmldsig-more#hmac-sha512

6.2. Encryption Based on Passphrase-Based Keys

Figure 7 shows an example that illustrates the encryption of the content of the <Secret> element using passphrase-based key derivation (PBKDF2) to derive the encryption key as defined in [PKCS5]. When using passphrase-based key derivation, the <DerivedKey> element defined in XML Encryption Version 1.1 [XMLENC11] MUST be used to specify the passphrased-based key. A <DerivedKey> element is set as the child element of <EncryptionKey> element of the key container.

The <DerivedKey> element is used to specify the key derivation function and related parameters. The encryption algorithm, in this example, AES-128-CBC (URI 'http://www.w3.org/2001/04/xmlenc#aes128-cbc'), MUST be set in the 'Algorithm' attribute of <EncryptionMethod> element used inside the encrypted data elements.

When PBKDF2 is used, the 'Algorithm' attribute of the <xenc11:KeyDerivationMethod> element MUST be set to the URI 'http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2'. The <xenc11:KeyDerivationMethod> element MUST include the <PBKDF2-params> child element to indicate the PBKDF2 parameters, such as salt and iteration count.

When the encryption method uses a CBC mode that uses an explicit initialization vector (IV) other than a derived one, the IV MUST be passed by prepending it to the encrypted value.

In the example below, the following data is used.

Password: qwerty

Salt: 0x123eff3c4a72129c

Iteration Count: 1000

MAC Key: 0xbdaab8d648e850d25a3289364f7d7eaaf53ce581

OTP Secret: 12345678901234567890

The derived encryption key is "0x651e63cd57008476af1ff6422cd02e41". The initialization vector (IV) is "0xa13be8f92db69ec992d99fd1b5ca05f0". This key is also used to encrypt the randomly chosen MAC key. A different IV can be used, say "0xd864d39cbc0cdc8e1ee483b9164b9fa0", in the example. The encryption with algorithm "AES-128-CBC" follows the specification defined in [XMLENC].

```
<?xml version="1.0" encoding="UTF-8"?>
<pskc:KeyContainer
  xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
  xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
  xmlns:pkcs5=
  "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" Version="1.0">
    <pskc:EncryptionKey>
        <xenc11:DerivedKey>
            <xenc11:KeyDerivationMethod
              Algorithm=
 "http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-0#pbkdf2">
                <pkcs5:PBKDF2-params>
                    <Salt>
                        <Specified>Ej7/PEpyEpw=</Specified>
                    </Salt>
                    <IterationCount>1000</IterationCount>
                    <KeyLength>16</KeyLength>
                    <PRF/>
                </pkcs5:PBKDF2-params>
            </xenc11:KeyDerivationMethod>
            <xenc:ReferenceList>
                <xenc:DataReference URI="#ED"/>
            </xenc:ReferenceList>
            <xenc11:MasterKeyName>My Password 1</xenc11:MasterKeyName>
        </xenc11:DerivedKey>
    </pskc:EncryptionKey>
    <pskc:MACMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1">
        <pskc:MACKey>
            <xenc:EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
            <xenc:CipherData>
                <xenc:CipherValue>
2GTTnLwM3I4e5IO5FkufoOEiOhNj91fhKRQBtBJYluUDsPOLTfUvoU2dStyOwYZx
                </xenc:CipherValue>
            </xenc:CipherData>
        </pskc:MACKey>
    </pskc:MACMethod>
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>TokenVendorAcme</pskc:Manufacturer>
            <pskc:SerialNo>987654321</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:CryptoModuleInfo>
            <pskc:Id>CM_ID_001</pskc:Id>
        </pskc:CryptoModuleInfo>
        <pskc:Key Algorithm=
```

```
          "urn:ietf:params:xml:ns:keyprov:pskc:hotp" Id="123456">
              <pskc:Issuer>Example-Issuer</pskc:Issuer>
              <pskc:AlgorithmParameters>
                  <pskc:ResponseFormat Length="8" Encoding="DECIMAL"/>
              </pskc:AlgorithmParameters>
              <pskc:Data>
                  <pskc:Secret>
                  <pskc:EncryptedValue Id="ED">
                      <xenc:EncryptionMethod
                          Algorithm=
  "http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
                          <xenc:CipherData>
                              <xenc:CipherValue>
        oTvo+S22nsmS2Z/RtcoF8Hfh+jzMe0RkiafpoDpnoZTjPYZu6V+A4aEn032yCr4f
                          </xenc:CipherValue>
                      </xenc:CipherData>
                      </pskc:EncryptedValue>
                      <pskc:ValueMAC>LP6xMvjtypbfT9PdkJhBZ+D6O4w=
                      </pskc:ValueMAC>
                  </pskc:Secret>
              </pskc:Data>
          </pskc:Key>
      </pskc:KeyPackage>
  </pskc:KeyContainer>
```

Figure 7: Example of a PSKC Document Using Encryption Based on Passphrase-Based Keys

6.3. Encryption Based on Asymmetric Keys

When using asymmetric keys to encrypt child elements of the <Data> element, information about the certificate being used MUST be stated in the <X509Data> element, which is a child element of the <EncryptionKey> element. The encryption algorithm MUST be indicated in the 'Algorithm' attribute of the <EncryptionMethod> element. In the example shown in Figure 8, the algorithm is set to 'http://www.w3.org/2001/04/xmlenc#rsa_1_5'.

```
<?xml version="1.0" encoding="UTF-8" ?>
<KeyContainer
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    id="KC0001"
    Version="1.0">
    <EncryptionKey>
        <ds:X509Data>
```

```
<ds:X509Certificate>MIIB5zCCAVCgAwIBAgIESZp/vDANBgkqhkiG9w0BAQUFADA4M
Q0wCwYDVQQKEwRJRVRGMRMwEQYDVQQLEwpLZXlQcm92IFdHMRIwEAYDVQQDEwlQU0tDIF
Rlc3QwHhcNMDkwMjE3MDkxMzMyWhcNMTEwMjE3MDkxMzMyWjA4MQ0wCwYDVQQKEwRJRVR
GMRMwEQYDVQQLEwpLZXlQcm92IFdHMRIwEAYDVQQDEwlQU0tDIFRlc3QwgZ8wDQYJKoZI
hvcNAQEBBQADgY0AMIGJAoGBALCWLDa2ItYJ6su80hd1gL4cggQYdyyKK17btt/aS6Q/e
DsKjsPyFIODsxeKVV/uA3wLT4jQJM5euKJXkDajzGGOy92+ypfzTX4zDJMkh61SZwlHNJ
xBKilAM5aW7C+BQ0RvCxvdYtzx2LTdB+X/KMEBA7uIYxLfXH2Mnub3WIh1AgMBAAEwDQY
JKoZIhvcNAQEFBQADgYEAe875m84sYUJ8qPeZ+NG7REgTvlHTmoCdoByU0LBBLotUKuqf
rnRuXJRMeZXaaEGmzY1kLonVjQGzjAkU4dJ+RPmiDlYuHLZS41Pg6VMwY+03lhk6I5A/w
4rnqdkmwZX/NgXg06alnc2pBsXWhL4O7nk0S2ZrLMsQZ6HcsXgdmHo=
</ds:X509Certificate>
        </ds:X509Data>
    </EncryptionKey>
    <KeyPackage>
        <DeviceInfo>
            <Manufacturer>TokenVendorAcme</Manufacturer>
            <SerialNo>987654321</SerialNo>
        </DeviceInfo>
        <Key
            Id="MBK000000001"
            Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Example-Issuer</Issuer>
            <AlgorithmParameters>
                <ResponseFormat Length="6" Encoding="DECIMAL"/>
            </AlgorithmParameters>
            <Data>
                <Secret>
                    <EncryptedValue>
                        <xenc:EncryptionMethod
             Algorithm="http://www.w3.org/2001/04/xmlenc#rsa_1_5"/>
                        <xenc:CipherData>
<xenc:CipherValue>hJ+fvpoMPMO9BYpK2rdyQYGIxiATYHTHC7e/sPLKYo5/r1v+4
xTYG3gJolCWuVMydJ7Ta0GaiBPHcWa8ctCVYmHKfSz5fdeV5nqbZApe6dofTqhRwZK6
Yx4ufevi91cjN2vBpSxYafvN3c3+xIgk0EnTV4iVPRCR0rBwyfFrPc4=
</xenc:CipherValue>
                        </xenc:CipherData>
                    </EncryptedValue>
                </Secret>
                <Counter>
                    <PlainValue>0</PlainValue>
                </Counter>
            </Data>
        </Key>
    </KeyPackage>
</KeyContainer>
```


Figure 8: Example of a PSKC Document Using Encryption Based on Asymmetric Keys

For systems implementing PSKC, it is RECOMMENDED to implement the RSA-1.5 algorithm, identified by the URI 'http://www.w3.org/2001/04/xmlenc#rsa-1_5'.

Some of the asymmetric encryption algorithms that can optionally be implemented are:

Algorithm	Uniform Resource Locator (URL)
RSA-OAEP-MGF1P	http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p

6.4. Padding of Encrypted Values for Non-Padded Encryption Algorithms

Padding of encrypted values (for example, the key secret value) is required when key protection algorithms are used that do not support embedded padding and the value to be encrypted is not a multiple of the encryption algorithm cipher block length.

For example, when transmitting an HOTP key (20 bytes long) protected with the AES algorithm in CBC mode (8-byte block cipher), padding is required since its length is not a multiple of the 8-byte block length.

In these cases, for systems implementing PSKC, it is RECOMMENDED to pad the value before encryption using PKCS #5 padding as described in [PKCS5].

7. Digital Signature

PSKC allows a digital signature to be added to the XML document, as a child element of the <KeyContainer> element. The description of the XML digital signature can be found in [XMLDSIG].

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer
    xmlns="urn:ietf:params:xml:ns:keyprov:pskc"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Version="1.0">
    <KeyPackage>
        <DeviceInfo>
            <Manufacturer>TokenVendorAcme</Manufacturer>
            <SerialNo>0755225266</SerialNo>
        </DeviceInfo>
        <Key Id="123"
        Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Example-Issuer</Issuer>
            <AlgorithmParameters>
```

```
                    <ResponseFormat Length="6" Encoding="DECIMAL"/>
                </AlgorithmParameters>
                <Data>
                    <Secret>
                        <PlainValue>
                            MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
                        </PlainValue>
                    </Secret>
                    <Counter>
                        <PlainValue>0</PlainValue>
                    </Counter>
                </Data>
            </Key>
        </KeyPackage>
        <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="#Device">
                    <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=Example.com,C=US
                        </ds:X509IssuerName>
                        <ds:X509SerialNumber>
                            12345678
                        </ds:X509SerialNumber>
                    </ds:X509IssuerSerial>
                </ds:X509Data>
            </ds:KeyInfo>
        </Signature>
    </KeyContainer>
```

Figure 9: Digital Signature Example

8. Bulk Provisioning

The functionality of bulk provisioning can be accomplished by repeating the <KeyPackage> element multiple times within the <KeyContainer> element, indicating that multiple keys are provided to different devices or cryptographic modules. The <EncryptionKey> element then applies to all <KeyPackage> elements. When provisioning multiple keys to the same device, the <KeyPackage> element is repeated, but the enclosed <DeviceInfo> element will contain the same sub-elements that uniquely identify the single device (for example, the keys for the device identified by SerialNo=’9999999’ in the example below).

Figure 10 shows an example utilizing these capabilities.

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyContainer Version="1.0"
    xmlns="urn:ietf:params:xml:ns:keyprov:pskc">
    <KeyPackage>
        <DeviceInfo>
            <Manufacturer>TokenVendorAcme</Manufacturer>
            <SerialNo>654321</SerialNo>
        </DeviceInfo>
        <Key Id="1"
        Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
            <Issuer>Issuer</Issuer>
            <AlgorithmParameters>
                <ResponseFormat Length="8" Encoding="DECIMAL"/>
            </AlgorithmParameters>
            <Data>
                <Secret>
                    <PlainValue>
                        MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
                    </PlainValue>
                </Secret>
                <Counter>
                    <PlainValue>0</PlainValue>
                </Counter>
            </Data>
            <Policy>
                <StartDate>2006-05-01T00:00:00Z</StartDate>
                <ExpiryDate>2006-05-31T00:00:00Z</ExpiryDate>
            </Policy>
        </Key>
    </KeyPackage>
```

```
<KeyPackage>
    <DeviceInfo>
        <Manufacturer>TokenVendorAcme</Manufacturer>
        <SerialNo>123456</SerialNo>
    </DeviceInfo>
    <Key Id="2"
    Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
        <Issuer>Issuer</Issuer>
        <AlgorithmParameters>
            <ResponseFormat Length="8" Encoding="DECIMAL"/>
        </AlgorithmParameters>
        <Data>
            <Secret>
                <PlainValue>
                    MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
                </PlainValue>
            </Secret>
            <Counter>
                <PlainValue>0</PlainValue>
            </Counter>
        </Data>
        <Policy>
            <StartDate>2006-05-01T00:00:00Z</StartDate>
            <ExpiryDate>2006-05-31T00:00:00Z</ExpiryDate>
        </Policy>
    </Key>
</KeyPackage>
<KeyPackage>
    <DeviceInfo>
        <Manufacturer>TokenVendorAcme</Manufacturer>
        <SerialNo>9999999</SerialNo>
    </DeviceInfo>
    <Key Id="3"
    Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
        <Issuer>Issuer</Issuer>
        <AlgorithmParameters>
            <ResponseFormat Length="8" Encoding="DECIMAL"/>
        </AlgorithmParameters>
        <Data>
            <Secret>
                <PlainValue>
                    MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
                </PlainValue>
            </Secret>
            <Counter>
                <PlainValue>0</PlainValue>
            </Counter>
        </Data>
```

```
                <Policy>
                    <StartDate>2006-03-01T00:00:00Z</StartDate>
                    <ExpiryDate>2006-03-31T00:00:00Z</ExpiryDate>
                </Policy>
            </Key>
        </KeyPackage>
        <KeyPackage>
            <DeviceInfo>
                <Manufacturer>TokenVendorAcme</Manufacturer>
                <SerialNo>9999999</SerialNo>
            </DeviceInfo>
            <Key Id="4"
            Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp">
                <Issuer>Issuer</Issuer>
                <AlgorithmParameters>
                    <ResponseFormat Length="8" Encoding="DECIMAL"/>
                </AlgorithmParameters>
                <Data>
                    <Secret>
                        <PlainValue>
                            MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=
                        </PlainValue>
                    </Secret>
                    <Counter>
                        <PlainValue>0</PlainValue>
                    </Counter>
                </Data>
                <Policy>
                    <StartDate>2006-04-01T00:00:00Z</StartDate>
                    <ExpiryDate>2006-04-30T00:00:00Z</ExpiryDate>
                </Policy>
            </Key>
        </KeyPackage>
    </KeyContainer>
```

Figure 10: Bulk Provisioning Example

9. Extensibility

This section lists a few common extension points provided by PSKC:

New PSKC Version: Whenever it is necessary to define a new version of this document, a new version number has to be allocated to refer to the new specification. The version number is carried inside the 'Version' attribute, as described in Section 4, the numbering scheme MUST follow Section 1.2, and rules for extensibility are defined in Section 12.

New XML Elements: The usage of the XML schema and the available extension points allows new XML elements to be added. Depending on the type of XML element, different ways for extensibility are offered. In some places, the <Extensions> element can be used and elsewhere the "<xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>" XML extension point is utilized.

New XML Attributes: The XML schema allows new XML attributes to be added where XML extension points have been defined (see "<xs: anyAttribute namespace="##other"/>" in Section 11).

New PSKC Algorithm Profiles: This document defines two PSKC algorithm profiles, see Section 10. The following informational document describes additional profiles [PSKC-ALGORITHM-PROFILES]. Further PSKC algorithm profiles can be registered as described in Section 12.4.

Algorithm URIs: Section 6 defines how keys and related data can be protected. A number of algorithms can be used. New algorithms can be used by pointing to a new algorithm URI.

Policy: Section 5 defines policies that can be attached to a key and keying-related data. The <Policy> element is one such item that allows implementers to restrict the use of the key to certain functions, such as "OTP usage only". Further values may be registered as described in Section 12.

10. PSKC Algorithm Profile

10.1. HOTP

Common Name: HOTP

Class: OTP

URI: urn:ietf:params:xml:ns:keyprov:pskc:hotp

Algorithm Definition: [HOTP]

Identifier Definition: (this RFC)

Registrant Contact: IESG

Deprecated: FALSE

Profiling:

The <KeyPackage> element MUST be present and the <ResponseFormat> element, which is a child element of the <AlgorithmParameters> element, MUST be used to indicate the OTP length and the value format.

The <Counter> element (see Section 4.1) MUST be provided as metadata for the key.

The following additional constraints apply:

+ The value of the <Secret> element MUST contain key material with a length of at least 16 octets (128 bits), if it is present.

+ The <ResponseFormat> element MUST have the 'Format' attribute set to "DECIMAL", and the 'Length' attribute MUST indicate a length value between 6 and 9 (inclusive).

+ The <PINPolicy> element MAY be present, but the 'PINUsageMode' attribute cannot be set to "Algorithmic".

An example can be found in Figure 3.

10.2. PIN

Common Name: PIN

Class: Symmetric static credential comparison

URI: urn:ietf:params:xml:ns:keyprov:pskc:pin

Algorithm Definition: (this RFC) Section 5.1

Identifier Definition (this RFC)

Registrant Contact: IESG

Deprecated: FALSE

Profiling:

The <Usage> element MAY be present, but no attribute of the <Usage> element is required. The <ResponseFormat> element MAY be used to indicate the PIN value format.

The <Secret> element (see Section 4.1) MUST be provided.

See the example in Figure 5

11. XML Schema

This section defines the XML schema for PSKC.

```
<?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:pskc"
     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:pskc"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified">
     <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:import namespace="http://www.w3.org/XML/1998/namespace"/>
     <xs:complexType name="KeyContainerType">
          <xs:sequence>
               <xs:element name="EncryptionKey"
                    type="ds:KeyInfoType" minOccurs="0"/>
               <xs:element name="MACMethod"
                    type="pskc:MACMethodType" minOccurs="0"/>
               <xs:element name="KeyPackage"
                    type="pskc:KeyPackageType" maxOccurs="unbounded"/>
               <xs:element name="Signature"
                    type="ds:SignatureType" minOccurs="0"/>
               <xs:element name="Extensions"
                    type="pskc:ExtensionsType"
                    minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="Version"
               type="pskc:VersionType" use="required"/>
          <xs:attribute name="Id"
               type="xs:ID" use="optional"/>
     </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="AlgorithmParameters"
                    type="pskc:AlgorithmParametersType"
                    minOccurs="0"/>
               <xs:element name="KeyProfileId"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="KeyReference"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="FriendlyName"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="Data"
                    type="pskc:KeyDataType" minOccurs="0"/>
               <xs:element name="UserId"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="Policy"
                    type="pskc:PolicyType" minOccurs="0"/>
               <xs:element name="Extensions"
                    type="pskc:ExtensionsType" minOccurs="0"
                    maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="Id"
               type="xs:string" use="required"/>
          <xs:attribute name="Algorithm"
               type="pskc:KeyAlgorithmType" use="optional"/>
     </xs:complexType>
     <xs:complexType name="PolicyType">
          <xs:sequence>
               <xs:element name="StartDate"
                    type="xs:dateTime" minOccurs="0"/>
               <xs:element name="ExpiryDate"
                    type="xs:dateTime" minOccurs="0"/>
               <xs:element name="PINPolicy"
                    type="pskc:PINPolicyType" minOccurs="0"/>
               <xs:element name="KeyUsage"
                    type="pskc:KeyUsageType"
                    minOccurs="0" maxOccurs="unbounded"/>
               <xs:element name="NumberOfTransactions"
                    type="xs:nonNegativeInteger" minOccurs="0"/>
               <xs:any namespace="##other"
                    minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
     </xs:complexType>
     <xs:complexType name="KeyDataType">
          <xs:sequence>
```

```
                <xs:element name="Secret"
                     type="pskc:binaryDataType" minOccurs="0"/>
                <xs:element name="Counter"
                     type="pskc:longDataType" minOccurs="0"/>
                <xs:element name="Time"
                     type="pskc:intDataType" minOccurs="0"/>
                <xs:element name="TimeInterval"
                     type="pskc:intDataType" minOccurs="0"/>
                <xs:element name="TimeDrift"
                     type="pskc:intDataType" minOccurs="0"/>
                <xs:any namespace="##other"
                     processContents="lax"
                     minOccurs="0" maxOccurs="unbounded"/>
           </xs:sequence>
      </xs:complexType>
      <xs:complexType name="binaryDataType">
           <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:complexType>
      <xs:complexType name="intDataType">
           <xs:sequence>
                <xs:choice>
                     <xs:element name="PlainValue" type="xs:int"/>
                     <xs:element name="EncryptedValue"
                          type="xenc:EncryptedDataType"/>
                </xs:choice>
                <xs:element name="ValueMAC"
                     type="xs:base64Binary" minOccurs="0"/>
           </xs:sequence>
      </xs:complexType>
      <xs:complexType name="stringDataType">
           <xs:sequence>
                <xs:choice>
                     <xs:element name="PlainValue" type="xs:string"/>
                     <xs:element name="EncryptedValue"
                          type="xenc:EncryptedDataType"/>
                </xs:choice>
                <xs:element name="ValueMAC"
                     type="xs:base64Binary" minOccurs="0"/>
           </xs:sequence>
```

```
     </xs:complexType>
     <xs:complexType name="longDataType">
          <xs:sequence>
               <xs:choice>
                    <xs:element name="PlainValue" type="xs:long"/>
                    <xs:element name="EncryptedValue"
                         type="xenc:EncryptedDataType"/>
               </xs:choice>
               <xs:element name="ValueMAC"
                    type="xs:base64Binary" minOccurs="0"/>
          </xs:sequence>
     </xs:complexType>
     <xs:complexType name="PINPolicyType">
          <xs:attribute name="PINKeyId"
               type="xs:string" use="optional"/>
          <xs:attribute name="PINUsageMode"
               type="pskc:PINUsageModeType"/>
          <xs:attribute name="MaxFailedAttempts"
               type="xs:unsignedInt" use="optional"/>
          <xs:attribute name="MinLength"
               type="xs:unsignedInt" use="optional"/>
          <xs:attribute name="MaxLength"
               type="xs:unsignedInt" use="optional"/>
          <xs:attribute name="PINEncoding"
               type="pskc:ValueFormatType" use="optional"/>
          <xs:anyAttribute namespace="##other"/>
     </xs:complexType>
     <xs:simpleType name="PINUsageModeType">
          <xs:restriction base="xs:string">
               <xs:enumeration value="Local"/>
               <xs:enumeration value="Prepend"/>
               <xs:enumeration value="Append"/>
               <xs:enumeration value="Algorithmic"/>
          </xs:restriction>
     </xs:simpleType>
     <xs:simpleType name="KeyUsageType">
          <xs:restriction base="xs:string">
               <xs:enumeration value="OTP"/>
               <xs:enumeration value="CR"/>
               <xs:enumeration value="Encrypt"/>
               <xs:enumeration value="Integrity"/>
               <xs:enumeration value="Verify"/>
               <xs:enumeration value="Unlock"/>
               <xs:enumeration value="Decrypt"/>
               <xs:enumeration value="KeyWrap"/>
               <xs:enumeration value="Unwrap"/>
               <xs:enumeration value="Derive"/>
               <xs:enumeration value="Generate"/>
```

```
          </xs:restriction>
     </xs:simpleType>
     <xs:complexType name="DeviceInfoType">
          <xs:sequence>
               <xs:element name="Manufacturer"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="SerialNo"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="Model"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="IssueNo"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="DeviceBinding"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="StartDate"
                    type="xs:dateTime" minOccurs="0"/>
               <xs:element name="ExpiryDate"
                    type="xs:dateTime" minOccurs="0"/>
               <xs:element name="UserId"
                    type="xs:string" minOccurs="0"/>
               <xs:element name="Extensions"
                    type="pskc:ExtensionsType" minOccurs="0"
                    maxOccurs="unbounded"/>
          </xs:sequence>
     </xs:complexType>
     <xs:complexType name="CryptoModuleInfoType">
          <xs:sequence>
               <xs:element name="Id" type="xs:string"/>
               <xs:element name="Extensions"
                    type="pskc:ExtensionsType" minOccurs="0"
                    maxOccurs="unbounded"/>
          </xs:sequence>
     </xs:complexType>
     <xs:complexType name="KeyPackageType">
          <xs:sequence>
               <xs:element name="DeviceInfo"
                    type="pskc:DeviceInfoType" minOccurs="0"/>
               <xs:element name="CryptoModuleInfo"
                    type="pskc:CryptoModuleInfoType" minOccurs="0"/>
               <xs:element name="Key"
                    type="pskc:KeyType" minOccurs="0"/>
               <xs:element name="Extensions"
                    type="pskc:ExtensionsType" minOccurs="0"
                    maxOccurs="unbounded"/>
          </xs:sequence>
     </xs:complexType>
     <xs:complexType name="AlgorithmParametersType">
          <xs:choice>
```

```
                <xs:element name="Suite" type="xs:string" minOccurs="0"/>
                <xs:element name="ChallengeFormat" minOccurs="0">
                     <xs:complexType>
                          <xs:attribute name="Encoding"
                               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:element name="ResponseFormat" minOccurs="0">
                     <xs:complexType>
                          <xs:attribute name="Encoding"
                               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="Extensions"
                     type="pskc:ExtensionsType" minOccurs="0"
                     maxOccurs="unbounded"/>
           </xs:choice>
      </xs:complexType>
      <xs:complexType name="ExtensionsType">
           <xs:sequence>
                <xs:any namespace="##other"
                     processContents="lax" maxOccurs="unbounded"/>
           </xs:sequence>
           <xs:attribute name="definition"
                type="xs:anyURI" use="optional"/>
      </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:complexType name="MACMethodType">
           <xs:sequence>
                  <xs:choice>
                        <xs:element name="MACKey"
               type="xenc:EncryptedDataType" minOccurs="0"/>
                        <xs:element name="MACKeyReference"
                                type="xs:string" minOccurs="0"/>
                        </xs:choice>
                        <xs:any namespace="##other"
           processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:attribute name="Algorithm" type="xs:anyURI" use="required"/>
        </xs:complexType>
     <xs:element name="KeyContainer"
          type="pskc:KeyContainerType"/>
</xs:schema>
```

12. IANA Considerations

12.1. Content-Type Registration for ’application/pskc+xml’

This specification contains the registration of a new media type according to the procedures of RFC 4288 [RFC4288] and guidelines in RFC 3023 [RFC3023].

MIME media type name: application

MIME subtype name: pskc+xml

Required parameters: There is no required parameter.

Optional parameters: charset

Indicates the character encoding of enclosed XML.

Encoding considerations: Uses XML, which can employ 8-bit characters, depending on the character encoding used. See RFC 3023 [RFC3023], Section 3.2.

Security considerations: Please refer to Section 13 of RFC 6030.

Interoperability considerations: None

Published specification: RFC 6030.

Applications which use this media type: This media type is being used as a symmetric key container 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. As such, it is used for key provisioning systems.

Additional information:

Magic Number: None

File Extension: .pskcxml

Macintosh file type code: 'TEXT'

Personal and email address to contact for further information: Philip Hoyer, Philip.Hoyer@actividentity.com

Intended usage: LIMITED USE

Restrictions on usage: None

Author: This specification is a work item of the IETF KEYPROV working group, with mailing list address <keyprov@ietf.org>.

Change controller: The IESG <iesg@ietf.org>

12.2. XML Schema Registration

This section registers an XML schema as per the guidelines in [RFC3688].

URI: urn:ietf:params:xml:schema:keyprov:pskc

Registrant Contact: IETF KEYPROV Working Group, Philip Hoyer (Philip.Hoyer@actividentity.com).

XML Schema: The XML schema to be registered is contained in Section 11. Its first line is

```
<?xml version="1.0" encoding="UTF-8"?>
```

and its last line is

```
</xs:schema>
```

12.3. URN Sub-Namespace Registration

This section registers a new XML namespace, "urn:ietf:params:xml:ns:keyprov:pskc", per the guidelines in [RFC3688].

URI: urn:ietf:params:xml:ns:keyprov:pskc

Registrant Contact: IETF KEYPROV Working Group, Philip Hoyer (Philip.Hoyer@actividentity.com).

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
  "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
        content="text/html;charset=iso-8859-1"/>
  <title>PSKC Namespace</title>
</head>
<body>
  <h1>Namespace for PSKC</h1>
  <h2>urn:ietf:params:xml:ns:keyprov:pskc</h2>
<p>See <a href="http://www.rfc-editor.org/rfc/rfc6030.txt">
 RFC 6030</a>.</p>
</body>
</html>
END
```

12.4. PSKC Algorithm Profile Registry

IANA has created a registry for PSKC algorithm profiles in accordance with the principles set out in RFC 5226 [RFC5226].

As part of this registry, IANA maintains the following information:

Common Name: The name by which the PSKC algorithm profile is generally referred.

Class: The type of PSKC algorithm profile registry entry being created, such as encryption, Message Authentication Code (MAC), One-Time Password (OTP), Digest.

URI: The URI to be used to identify the profile.

Identifier Definition: IANA will add a pointer to the specification containing information about the PSKC algorithm profile registration.

Algorithm Definition: A reference to the stable document in which the algorithm being used with the PSKC is defined.

Registrant Contact: Contact information about the party submitting the registration request.

Deprecated: TRUE if this entry has been deprecated based on expert approval and SHOULD not be used in any new implementations. Otherwise, FALSE.

PSKC Profiling: Information about PSKC XML elements and attributes being used (or not) with this specific profile of PSKC.

PSKC algorithm profile identifier registrations are to be subject to Specification Required as per RFC 5226 [RFC5226]. Updates can be provided based on expert approval only. Based on expert approval, it is possible to mark entries as "deprecated". A designated expert will be appointed by the IESG.

IANA has added two initial values to the registry based on the algorithm profiles described in Section 10.

12.5. PSKC Version Registry

IANA has created a registry for PSKC version numbers. The registry has the following structure:

PSKC Version	Specification
1.0	RFC 6030

Standards action is required to define new versions of PSKC. It is not envisioned to deprecate, delete, or modify existing PSKC versions.

12.6. Key Usage Registry

IANA has created a registry for key usage. A description of the <KeyUsage> element can be found in Section 5.

As part of this registry IANA will maintain the following information:

Key Usage: The identifier of the Key Usage.

Specification: IANA will add a pointer to the specification containing information about the semantics of a new Key Usage registration.

Deprecated: TRUE if this entry has been deprecated based on expert approval and SHOULD not be used in any new implementations. Otherwise, FALSE.

IANA has added these initial values to the registry:

Key Usage	Specification	Deprecated
OTP	[Section 5 of this document]	FALSE
CR	[Section 5 of this document]	FALSE
Encrypt	[Section 5 of this document]	FALSE
Integrity	[Section 5 of this document]	FALSE
Verify	[Section 5 of this document]	FALSE
Unlock	[Section 5 of this document]	FALSE
Decrypt	[Section 5 of this document]	FALSE
KeyWrap	[Section 5 of this document]	FALSE
Unwrap	[Section 5 of this document]	FALSE
Derive	[Section 5 of this document]	FALSE
Generate	[Section 5 of this document]	FALSE

Key Usage Registry registrations are to be subject to Specification Required as per RFC 5226 [RFC5226]. Expert Review is required to define new Key Usage values. Updates can be provided based on expert approval only. Based on expert approval, it is possible to mark entries as "deprecated". A designated expert will be appointed by the IESG.

13. Security Considerations

The portable symmetric key container (PSKC) 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.

13.1. PSKC 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 key data payload encryption. A symmetric cryptographic cipher SHOULD be used -- the longer the cryptographic key, the stronger the protection. Please see Section 6.1 for recommendations of key data payload protection using symmetric cryptographic ciphers. In cases where the exchange of key encryption keys between the sender and the receiver is not possible, asymmetric encryption of the key data payload may be employed, see Section 6.3. Similar to symmetric key cryptography, the stronger the asymmetric key, the more secure the protection.

If the key data payload is encrypted with a method that uses one of the password-based encryption methods (PBE methods) detailed in Section 6.2, the key data 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.

Additionally, it is strongly RECOMMENDED that practical implementations use PBESalt and PBEIterationCount when PBE encryption is used. A different PBESalt value per PSKC SHOULD be used for best protection.

The second approach to protecting the confidentiality of the key data is based on using lower-layer security mechanisms (e.g., [TLS], [IPsec]). The secure connection 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 protect the messages that travel across that connection. No key data payload encryption is required in this mode. Secure connections that encrypt and digest each message provide an extra measure of security.

Because of the fact that the plaintext PSKC is protected only by the transport layer security, practical implementation MUST ensure protection against man-in-the-middle attacks. Authenticating the secure channel endpoints is critically important for eliminating intruders that may compromise the confidentiality of the PSKC.

13.2. PSKC Integrity

The PSKC provides means to guarantee the integrity of the information it contains through the use of digital signatures. It is RECOMMENDED that for best security practices, the digital signature of the container encompasses the entire PSKC. This provides assurances for the integrity of all attributes. It also allows verification of the integrity of a given PSKC even after the container is delivered through the communication channel to the target perimeter and channel message integrity check is no longer possible.

13.3. PSKC Authenticity

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

Authenticity guarantee may be provided by [TLS] or [IPsec]. However, no authenticity verification is possible once the container is delivered at the recipient end. Since the TLS endpoints could differ from the key provisioning endpoints, this solution is weaker than the previous solution that relies on a digital signature of the PSKC.

14. Contributors

We would like Hannes Tschofenig for his text contributions to this document.

15. Acknowledgements

The authors of this document would like to thank the following people for their feedback: Apostol Vassilev, Shuh Chang, Jon Martinson, Siddhart Bajaj, Stu Vaeth, Kevin Lewis, Philip Hallam-Baker, Andrea Doherty, Magnus Nystrom, Tim Moses, Anders Rundgren, Sean Turner, and especially Robert Philpott.

We would like to thank Sean Turner for his review in January 2009. We would also like to thank Anders Rundgren for triggering the discussion regarding to the selection of encryption algorithms (KW-AES-128 vs. AES-128-CBC) and his input on the keyed message digest computation.

This work is based on earlier work by the members of OATH (Initiative for Open AuTHentication), see [OATH], to specify a format that can be freely distributed to the technical community.

16. References

16.1. Normative References

[FIPS197] National Institute of Standards, "FIPS Pub 197: Advanced Encryption Standard (AES)", November 2001.

[HOTP] M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., and O. Ranen, "HOTP: An HMAC-Based One-Time Password Algorithm", RFC 4226, December 2005.

[IANAPENREG] IANA, "Private Enterprise Numbers", <http://www.iana.org>.

[ISOIEC7812] ISO, "ISO/IEC 7812-1:2006 Identification cards -- Identification of issuers -- Part 1: Numbering system", October 2006, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39698>.

[OATHMAN] OATH, "List of OATH Manufacturer Prefixes (omp)", April 2009, <http://www.openauthentication.org/oath-id/prefixes/>.

[PKCS5] RSA Laboratories, "PKCS #5: Password-Based Cryptography Standard", Version 2.0, March 1999, <http://www.rsasecurity.com/rsalabs/pkcs/>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[RFC4514] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, June 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, September 2009.

[SP800-67]
National Institute of Standards, "NIST Special Publication 800-67 Version 1.1: Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher", NIST Special Publication 800-67, May 2008.

[W3C.REC-xmlschema-2-20041028]
Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[XMLDSIG] Solo, D., Reagle, J., and D. Eastlake, "XML-Signature Syntax and Processing", World Wide Web Consortium FirstEdition REC-xmldsig-core-20020212, February 2002, <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212>.

[XMLENC] Eastlake, D., "XML Encryption Syntax and Processing.", W3C Recommendation, December 2002, <http://www.w3.org/TR/xmlenc-core/>.

[XMLENC11]
Reagle, J. and D. Eastlake, "XML Encryption Syntax and Processing Version 1.1", World Wide Web Consortium WD WD-xmlenc-core1-20090730, July 2009, <http://www.w3.org/TR/2009/WD-xmlenc-core1-20090730>.

16.2. Informative References

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

[IPsec] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[NIST800-57]
Barker, E., Barker, W., Burr, W., Polk, W., and M. Smid, "NIST Special Publication 800-57, Recommendation for Key Management Part 1: General (Revised)", NIST Special Publication 800-57, March 2007.

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

[PSKC-ALGORITHM-PROFILES]
Hoyer, P., Pei, M., Machani, S., and A. Doherty, "Additional Portable Symmetric Key Container (PSKC) Algorithm Profiles", Work in Progress, May 2010.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[XMLNS] Hollander, D., Bray, T., and A. Layman, "Namespaces in XML", World Wide Web Consortium FirstEdition REC-xml-names-19990114, January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114>.

Appendix A. 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.

A.1. Online Use Cases

This section describes the use cases related to provisioning the keys using an online provisioning protocol.

A.1.1. Transport of Keys from Server to Cryptographic Module

For example, a mobile device user wants to obtain a symmetric key for use with a cryptographic module on the device. The cryptographic module from vendor A initiates the provisioning process against a provisioning system from vendor B using a standards-based provisioning protocol. The provisioning entity delivers one or more keys in a standard format that can be processed by the mobile device.

For example, 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 a key in a standard format that can be processed by the soft token on the PC.

For example, the end user or the key issuer wants to update or configure an existing key in the cryptographic module 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.

A.1.2. Transport of Keys from Cryptographic Module to Cryptographic Module

For example, a user wants to transport a key from one cryptographic module to another. There may be two cryptographic modules, one on a computer and one on a mobile phone, and the user wants to transport a key from the computer to the mobile phone. The user can export the key and related data in a standard format for input into the other cryptographic module.

A.1.3. Transport of Keys from Cryptographic Module to Server

For example, a 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 cryptographic module (e.g., a Secure Digital (SD) card, USB drive) that the user has purchased at the local electronics retailer. Along with the cryptographic module, 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.

A.1.4. Server-to-Server Bulk Import/Export of Keys

From time to time, a key management system may be required to import or export keys in bulk from one entity to another.

For example, instead of importing keys from a manufacturer using a file, a 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.

For example, in a variation of the above, an Over-The-Air (OTA) 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 key provisioning gateway and subsequently sent to the mobile phone of the end user.

A.2. 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.

A.2.1. Server-to-Server Bulk Import/Export of Keys

For example, cryptographic modules, such as OTP authentication tokens, may have their symmetric keys initialized during the manufacturing process in bulk, requiring copies of the keys and algorithm data to be loaded into the authentication 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. Some crypto modules 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.

For example, 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 authentication 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.

Appendix B. 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 the 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 an appropriate level of security in terms of data encryption and data integrity.

R8: For online scenarios, the format SHOULD NOT rely on transport layer security (e.g., Secure Socket Layer/Transport Layer Security (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 life cycle 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 key encryption key is difficult to use.

Authors' Addresses

Philip Hoyer
ActivIdentity, Inc.
117 Waterloo Road
London, SE1 8UL
UK

Phone: +44 (0) 20 7960 0220
EMail: phoyer@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