OCRA Validation Server Profile

Version 1.0

Feb. 22, 2013

1 Overview

This document defines the technical requirements for compliance with an OCRA Validation Server profile for OATH Certification. These are described in section 2-4.

1.1 Conventions

Throughout this document, normative requirements are highlighted by use of capitalized key words as described below.

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]:

- MUST - this word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- MUST NOT - this phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
- SHOULD - this word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT - this phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY - this word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation that does not include a particular option MUST be prepared to interoperate with another implementation that does include the option, though perhaps with reduced functionality. In the same vein an implementation that does include a particular option MUST be prepared to interoperate with another implementation that does not include the option (except, of course, for the feature the option provides.)

1.2 Definitions

We follow the NIST definition about token, key and algorithm.

Token
Something a claimant possesses and controls that may be used to authenticate the claimant's identity. It is typically a secret. It can be a hardware device token or a software token, see [NIST-800-63-1].

Device
Physical device that contains one or more keys and associated cryptographic modules.

Key
Secret value used by Algorithm to generate OTP or OCRA Response

Algorithm
OATH Algorithm "urn:ietf:params:xml:ns:keyprov:pskc:[hotp|totp|ocra|pin]"

Suite
Algorithm input options to generate OTP or OCRA Response value

OTP/Response
Value generated using Algorithm, Key, Suite and DataInput

Application
One or more software applications that relies on OOTP or OCRA Response based authentication

2 Support for OCRA algorithm

The server MUST implement the OCRA algorithm according to [RFC 6287]. To be compliant the server MUST support **all** the algorithm modes described below for both challenge-response and signatures and meet all the necessary requirements.

2.1 Challenge-Response Authentication Mode

The server MUST support the Algorithm mode described in section '7.1 – One way challenge-response' of the OCRA specification [RFC 6287].

OCRA specification enables a wide variety of options. To foster interoperability, a compliant validation server

- MUST implement all the requirements described in section 2.1.1 & 2.1.2 below
- MUST implement all 13 combinations in section 2.1.7 for authentication and rules described in section 2.1.3 through 2.1.6.

2.1.1 General Requirements

a. The OCRA response value calculated MUST be based on the OCRA algorithm defined in [RFC 6287] where `OCRA = CryptoFunction(K,DataInput)`, where K is a symmetric shared secret, and DataInput is a structure that contains the concatenation of the various data input values as describe below.
b. The secret K MUST be unique for each token.
c. The validation server MUST reject a response if it is used more than once on suites that use an event Counter or Time.

2.1.2 Challenge Generation

A validation server MUST support the following options in generating challenge for Challenge-Response authentication. The specific interfaces or protocols between the validation server and the application that generate the challenge is outside the scope of this document.

Application Generated challenge Alphanumeric 6 numeric digits (QA06): The application generates a random alphanumeric 6 digits challenge and sends it to the client. The application MAY keep track of the generated challenges.

Application Generated challenge Alphanumeric 8 numeric digits (QA08): The application generates a random alphanumeric 8 digits challenge and sends it to the client. The application MAY keep track of the generated challenges.

Application Generated challenge Numeric 8 numeric digits (QN08): The application generates a random numeric 8 decimal digit challenge and sends it to the client. The application MAY keep track of the generated challenges.

Server Generated challenge Alphanumeric 6 and 8 and Numeric 8 digits: The challenge is generated by the validation server. The application will send an initial get challenge request to the validation server along with a unique handle that must identify Key/Suite. The server MAY maintain a list of pending challenges. The specific server algorithm to handle, verify and accept pending challenges is outside the scope of this document.

- The server MUST be capable of generating 6 AND 8 digits alphanumeric or 8 digits numerical challenges.
- The server MAY use a cryptographically secure pseudorandom number generator to generate the challenge.
- The server MAY implement algorithms that allow a client to verify whether the server generated challenge is correct.

2.1.3 HMAC Function

a. The server implementation MUST support both HMAC-SHA-1 and HMAC-SHA-256 for the computation.

b. The secret key size MUST be at least 20 bytes for HMAC-SHA-1 and 32 bytes for HMAC-SHA-256.

2.1.4 Challenge Format

a. The Server generated challenge question Q MUST be Numeric 8 digits (N08), or Alphanumeric 6 digits (QA06), or Alphanumeric 8 digits (QA08).

2.1.5 Response (output) format and length

a. The server MUST accept response lengths of 6 AND 8 numeric digits

2.1.6 PIN Policy

a. The server MUST support all of the following three PIN/Password policies:
   i. No PIN is used in the OCRA computation
   ii. PIN is used with SHA1 being used to compute DataInput P.
   iii. PIN is used with SHA256 being used to compute DataInput P.

2.1.7 Challenge/Response OCRASuite values

Based on the above options, the following are the valid OCRASuite values. The server MUST support ALL of the following 27 combinations.

Authentication and signature with application generated challenge OCRASuite values:

1. OCRA-1:HOTP-SHA1-6: QA06
2. OCRA-1:HOTP-SHA1-8: QN08
3. OCRA-1:HOTP-SHA1-8: QA08
4. OCRA-1:HOTP-SHA256-8: QA08
5. OCRA-1:HOTP-SHA1-6: QA06-PSHA1

6. OCRA-1:HOTP-SHA1-6:C-QA06
7. OCRA-1:HOTP-SHA1-8: C-QN08
8. OCRA-1:HOTP-SHA256-8: C-QA08
9. OCRA-1:HOTP-SHA256-8: C-QA08-PSHA256

10. OCRA-1:HOTP-SHA1-6: QA06-T30S
11. OCRA-1:HOTP-SHA1-8: QN08-T30S
12. OCRA-1:HOTP-SHA1-8: QA08-T30S
13. OCRA-1:HOTP-SHA256-6: QA06-T30S
14. OCRA-1:HOTP-SHA256-8: QA08-T30S
15. OCRA-1:HOTP-SHA256-8: QA08-PSHA256-T30S

Signature OCRASuite values with transaction data generated challenge:

1. OCRA-1:HOTP-SHA1-6:QH40
2. OCRA-1:HOTP-SHA1-6:QA32
3. OCRA-1:HOTP-SHA1-8:QH40

4. OCRA-1:HOTP-SHA1-6:QA32-T30S
5. OCRA-1:HOTP-SHA1-6:QH40-T30S
6. OCRA-1:HOTP-SHA1-8:QA32-T30S
7. OCRA-1:HOTP-SHA1-8:QH40-T30S

8. OCRA-1:HOTP-SHA256-8:QA32
9. OCRA-1:HOTP-SHA256-8:QH64
10. OCRA-1:HOTP-SHA256-6:QH64-T30S
11. OCRA-1:HOTP-SHA256-8:QA32-T30S
12. OCRA-1:HOTP-SHA256-8:QH64-T30S

2.2 Signature Mode

The server MUST support the Algorithm mode described in section '7.3.1 – Plain Signature' of the OCRA specification [RFC 6287].

OCRA specification enables a wide variety of options. To foster interoperability, the compliant server

- MUST implement all the requirements described in section 2.2.1 below
- MUST implement all of the OCRASuite values in Section 2.1.7 above. The list consists of "plain" signature and signing with a time moving factor as described in section 2.2.2 through 2.2.5.
- MUST use the specified approach for Challenge (QS) generation as described in Appendix A

2.2.1 General Requirements

a. The OCRA response value calculated MUST be based on the OCRA algorithm defined [RFC 6287] where `OCRA = CryptoFunction(K, DataInput)`, where K is a symmetric shared secret, and DataInput is a structure that contains the concatenation of the various data input values as describe below.
b. The secret K MUST be unique for each token.
c. A validation server MAY validate the same signature multiple times.

2.2.2 Signature Challenge

The validation server MUST support the following three options for computing signature-challenge.

a. Transaction data signing – Hexadecimal 40 nibbles (H40 - 20 bytes): In this mode the application compute the signature-challenge QS, using the GenerateQS_SHA1() function as described below, before performing the actual OCRA computation**.** The application MAY keep track of the generated challenges. The application will send both the challenge and response from the client to the validation server to be validated.

b. **Transaction data signing – Hexadecimal 64 nibbles (H64 - 32 bytes):** In this mode the application compute the signature-challenge QS, using the GenerateQS_SHA256() function as described below before performing the actual OCRA computation**.** The application MAY keep track of the generated challenges. The application will send both the challenge and response from the client to the validation server to be validated.

c. **Transaction data signing – Alphanumeric 32 characters (A32 - 32 bytes):** In this mode the application compute the signature-challenge QS, using the GenerateQS_NoHash() function as described below before performing the actual OCRA computation**.** The application MAY keep track of the generated challenges. The application will send both the challenge and response from the client to the validation server to be validated.

2.2.2.1 Application Generated, Alphanumeric 6 digits

In this mode the application generates the signature challenge that is 6 alphanumeric digits and sends it to the user. The exact mechanics of how the signature challenge is generated is out of the scope of this document. The user will compute a response and send it back to the application.

The application will then send both the signature challenge and response to the validation server to be verified.

2.2.2.2 Application Generated, Alphanumeric 8 digits

In this mode the application generates the signature challenge that is 8 alphanumeric digits and sends it to the user. The exact mechanics of how the signature challenge is generated is proprietary to the application. The user will compute a response and send it back to the application.

The application will then send both the signature challenge and response to the validation server to be verified.

2.2.2.3 Application Generated, Alphanumeric 32 characters (32 bytes)

In this mode the user enters individual data elements directly into the Token. The Token will first generate a signature-challenge QS using the GenerateQS_NoHash() function as described below. Then it will use the QS in the OCRA computation to generate the response value.

The application will receive the user data elements and the user response. Repeat the GenerateQS_NoHash() function on user data elements to generate the signature-challenge QS then send both the signature challenge and user response (OTP) to the validation server to be verified.

2.2.2.4 Application Computed, Hexadecimal 40 nibbles (20 bytes)

In this mode, the user enters individual data elements directly into the client or token. The client or token will first generate a signature-challenge QS using the GenerateQS_SHA1() function as described below. Then it will use the QS in the OCRA computation to generate the response value.

The application will receive the user data elements and the user response. Repeat the GenerateQS_SHA1() function on user data elements to generate the signature-challenge QS then send both the signature challenge and user response (OTP) to the validation server to be verified.

2.2.2.5 Application Computed, Hexadecimal 64 nibbles (32 bytes)

In this mode, the user enters individual data elements directly into the client or token. The client or token will first generate a signature-challenge QS using the GenerateQS_SHA256() function as described below. Then it will use the QS in the OCRA computation to generate the response value.

The application will receive the user data elements and the user response. Repeat the GenerateQS_SHA256() function on user data elements to generate the signature-challenge QS then send both the signature challenge and user response (OTP) to the validation server to be verified.

3 OTP Credential Import

The Server that is compliant with this profile MUST support import of OTP credentials in the PSKC credential transport data format as defined in [RFC 6030].

The Server MUST guarantee:

- unique Id for each Token
- unique Key Id for each Device

PSKC Key Protection & Integrity

The server MUST support at least one the following PSKC import profiles:

a. PSKC file protected with AES-128-CBC pre shared key as defined in Section 6.1 of [RFC 6030]
b. PSKC file protected with PBE encryption as defined in Section 6.2 of [RFC 6030]
   - During import the server MUST accept passwords that are between 5 and 64 characters (both inclusive) in length.

If a PSKC file contains integrity checks for the values (ValueMAC) the server MUST check the correct ValueMAC and MUST NOT import records where the ValueMAC does not match the data.

The PSKC file MUST support the HMAC-SHA1 for ValueMAC evaluation.

Token ID Compliance

The token ID MUST comply with OATH Token ID Specification [OTIS]. The token ID is commonly referred by the serial number printed in the physical device. In a PSKC transport file the token ID is typically specified in PSKC XML file by the Id attribute of <Key> element for single key token case. There are cases where a hardware device token may have multiple keys for different OTP or OCRA algorithms. All the keys within the same device token share the same externally visible Token ID. The token ID will be specified by the value of the element <SerialNo>.

A compliant PSKC file MUST specify the SerialNo attribute for the token:
<KeyContainer>/<KeyPackage>/<DeviceInfo>/<SerialNo>

The PSKC MUST specify the Key ID within the key Id attribute
<KeyContainer>/<KeyPackage>/<Key>/@Id

During import the OATH Token ID SHOULD be validated for conformance with the OATH Token Identifier Specification [OTIS]. This validation SHOULD be performed in terms of format (length and potentially characters used).

Best Practices

- If a device contains only one Key/Suite pair, the value of Id attribute <Key>/@Id SHOULD be the same as the value of <SerialNo>.
- If a Device contains more than one Key/Suite pair, the value of <Key>/@Id for each Key/Suite pair SHOULD be the value of <SerialNo> with an appending sequential number (e.g. <Key Id="ZZAA00000001#01">).

Other

a. The PSKC MUST use the Key Algorithm URI for OCRA as defined in [ALG].

b. The OCRASuite value MUST be set in the `<Suite>` element i.e. `<KeyContainer>/<KeyPackage>/<Key>/<AlgorithmParameters>/<Suite>`.
   - The validation server MUST verify that then Suite value is one of the values defined in above sections (Sections 2.1.7)
   - The PSKC SHOULD NOT contain <ChallengeFormat> element i.e. `<KeyContainer>/<KeyPackage>/<Key>/<AlgorithmParameters>/<ChallengeFormat>`. The value for the <Suite> element has indicated the challenge and response format.

c. The PSKC SHOULD NOT contain `<ResponseFormat>` element.
   - The <Suite> element has indicated the challenge and response format.

d. `<KeyContainer>/<KeyPackage>/<Key>/<Policy>/<KeyUsage>` element MAY be present and indicate the usage of the key. The value MUST one of those defined in IANA PSKC Key Usage Registry.

e. If the OCRA standalone client support a PIN/Password Policy to uses the PIN in the response computation then the `<KeyContainer>/<KeyPackage>/<Key>/<Policy>/<PINPolicy>` element MUST BE present.
   - Example when PIN is used in the computation: `<PINPolicy PINUsageMode="Algorithmic"/>`

f. If the OCRA standalone client support a PIN/Password policy that is used to enable Token computation rather than response computation, `<KeyContainer>/<KeyPackage>/<Key>/<Policy>/<PINPolicy>` element SOULD be present.

   - Example for local PIN: `<PINPolicy MinLength="4" MaxLength="4" PINKeyId="123456781" PINEncoding="DECIMAL"0 PINUsageMode="Local"/>`

4 Appendix A – Challenge Computation functions

Some implementations of the OCRA client will enable the user to input multiple discrete data values into the client directly. For example, the data values could represent a transaction that needs to be signed by the user to indicate their approval. For a funds transfer banking transaction these values could represent the account_from, account_to and the amount.

In this section, we will describe a standard functions that should be used by both the client and the server to combine the individual data elements into a single Signature-challenge, QS, that can be used for the OCRA computation as described in [RFC 6287].

4.1 GenerateQS_NoHash()

QS = GenerateQS_NoHash(Data1, Data2,…, DataN)

Step 1: Concatenate the individual Data elements into *DataString*

In the simplest form that can be entered in a Token, each input data element is assumed to be alphanumeric data. The 7-bit ASCII data pieces are concatenated without any delimiter. For example, given

Data1 = CHK1000100020 (from account ID)
Data2 = 20100208 (date)

Data3 = $10000 (amount)

we have,

DataString = Concatenate (Data1, Data2,…DataN)
DataString = CHK1000100020201002 08$10000

Note: The same order should be used to concatenate data on both the client for the signature computation and the server for signature validation. It's agreed for backward compatibility that GenerateQS_NoHash has no delimiter between fields, if delimiter is needed use GenerateQS_SHAn().

Step 2: Pad the DataString (if required)

If the length of DataString is less than 32 characters then, pad DataString to the right with '~' character until the length is 32. For the above example,

QS = DoPadding (CHK1000100020201002 08$10000)
QS = CHK1000100020201002 08$10000~~~~~

4.2 GenerateQS_SHA1()

QS = GenerateQS_SHA1(Data1, Data2,…, DataN)

Step 1: Concatenate the individual Data elements into *DataString*
In the simplest form that can be entered in a Token, each input data element is assumed to be alphanumeric data. The 7-bit ASCII data pieces are concatenated using '~' as the delimiter. For example, given

Data1 = CHK1000100020 (from account ID)
Data2 = 20100208 (date)
Data3 = $10000 (amount)

we have

DataString = CHK1000100020~20100208~$10000

Note: The same order should be used to concatenate data on both the client for the signature computation and the server for signature validation.

Step 2: Generate *QS* from *DataString* with SHA1 hash function
The hash function MUST be SHA1. We perform this additional step so that applications can limit the exposure of actual data values to the validation server by sending the output of the hash function instead.

QS = SHA1 (DataString)

For the above example,

QS = SHA1 (CHK1000100020~20100208~$10000)
QS = 53711495d711cb2fa048f38b7512053ce53584a6

Note: The 20 byte SHA1 output MUST be encoded into 40 nibbles (20 bytes).

4.3 GenerateQS_SHA256()

QS = GenerateQS_SHA256(Data1, Data2,…, DataN)

Step 1: Concatenate the individual Data elements into *DataString*
In the simplest form that can be entered in a Token, each input data element is assumed to be alphanumeric data. The 7-bit ASCII data pieces are concatenated using '~' as the delimiter. For example, given

Data1 = CHK1000100020 (from account ID)
Data2 = 20100208 (date)
Data3 = $10000 (amount)

we have

DataString = CHK1000100020~20100208~$10000

Note: The same order should be used to concatenate data on both the client for the signature computation and the server for signature validation.

Step 2: Generate *QS* from *DataString* with SHA256 hash function
The hash function MUST be SHA256. We perform this additional step so that applications can limit the exposure of actual data values to the validation server by sending the output of the hash function instead.

QS = SHA256 (DataString)

For the above example,

QS = SHA256 (CHK1000100020~20100208~$10000)
QS = cb1f5c8a967b290443a7e2bdad2984196f376303742c248d08758142bd9de205

5 Appendix B – Sample Table

```
seed 20byte:              3132333435363738393031323334353637383930
seed 32 byte:             3132333435363738393031323334353637383930313233343536373839303132
```

```
pin:        123456
timesteps:  29412C0
counter:    0000000000000001
```

SUITE	QUESTION	RESPONSE
OCRA-1:HOTP-SHA1-6:QA06	857430	545082
OCRA-1:HOTP-SHA1-6:QA06-PSHA1	009384	978358
OCRA-1:HOTP-SHA1-6:C-QA06	233957	723678
OCRA-1:HOTP-SHA1-6:QA06-T30S	089711	622816
OCRA-1:HOTP-SHA1-8:QN08	10023847	54563391
OCRA-1:HOTP-SHA1-8:QA08	92847623	73426915
OCRA-1:HOTP-SHA1-8:C-QN08	89562072	90368606
OCRA-1:HOTP-SHA1-8:QN08-T30S	14232161	84829193
OCRA-1:HOTP-SHA256-6:QA06-T30S	204887	841709
OCRA-1:HOTP-SHA256-8:QA08	07895909	15548300
OCRA-1:HOTP-SHA256-8:C-QA08	09688411	63248426
OCRA-1:HOTP-SHA256-8:C-QA08-PSHA256	78563753	74391716
OCRA-1:HOTP-SHA256-8:QA08-T30S	00946734	99056818
OCRA-1:HOTP-SHA256-8:QA08-PSHA256-T30S	95627347	02379282
OCRA-1:HOTP-SHA1-6:QH40	0eb3bd52bf54db8f598f72e4c5193ea6b2f206a9	187219
OCRA-1:HOTP-SHA1-6:QA32	this is my first question~~~~~~~	902518
OCRA-1:HOTP-SHA1-8:QH40	604a3334a2f7c6d2d3a3adc64c0cc04d28dbc491	55000943
OCRA-1:HOTP-SHA256-8:QA32	this is my second question~~~~~~	12328097
OCRA-1:HOTP-SHA256-8:QH64	10abe2517b814aef25b1bd1783697feda76d2f4ed7c0e40ca02e010f0f80a74a	09574772
OCRA-1:HOTP-SHA1-6:QA32-T30S	this is my third question~~~~~~~	858997
OCRA-1:HOTP-SHA1-6:QH40-T30S	7f59dde57eef672e7aa252a11c7a0d284d9a11a4	071342
OCRA-1:HOTP-SHA1-8:QA32-T30S	this is my fourth question~~~~~~	40139015
OCRA-1:HOTP-SHA1-8:QH40-T30S	41816b6d46c6f79fc0900a63ee463d3ff4f6e909	36604759
OCRA-1:HOTP-SHA256-6:QH64-T30S	e1ac6dccc5d808cd13786ca2fb0a3f6696edc4163c52c151b1dbe9562a73d55e	174788
OCRA-1:HOTP-SHA256-8:QA32-T30S	this is my fifth question~~~~~~~	55093048
OCRA-1:HOTP-SHA256-8:QH64-T30S	d75deda145b7797e744cee173ac83514449db25d1663062c245b89943548a55d	47313920

6 Appendix D – PSKC sample file

6.1 Normal single key token

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

<pskc:KeyContainer xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
xmlns:pkcs5="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-
0#" Id="ExampleID" Version="1.0">
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ0000000000</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp"
Id="ZZ0000000000">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>HMAC-SHA1</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</pskc:PlainValue>
                </pskc:Secret>
                <pskc:Counter>
                    <pskc:PlainValue>0</pskc:PlainValue>
```

```
                </pskc:Counter>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ1000000000</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:totp"
Id="ZZ1000000000">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>HMAC-SHA1</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</pskc:PlainValue>
                </pskc:Secret>
                <pskc:Time>
                    <pskc:PlainValue>0</pskc:PlainValue>
                </pskc:Time>
                <pskc:TimeInterval>
                    <pskc:PlainValue>30</pskc:PlainValue>
                </pskc:TimeInterval>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ1100000001</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:totp"
Id="ZZ1100000001">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>HMAC-SHA256</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTM=</pskc:Pla
inValue>
                </pskc:Secret>
                <pskc:Time>
                    <pskc:PlainValue>0</pskc:PlainValue>
                </pskc:Time>
                <pskc:TimeInterval>
                    <pskc:PlainValue>30</pskc:PlainValue>
                </pskc:TimeInterval>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
```

```
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ1110000000</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:ocra"
Id="ZZ1110000000">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>OCRA-1:HOTP-SHA1-6:C-QA08</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</pskc:PlainValue>
                </pskc:Secret>
                <pskc:Counter>
                    <pskc:PlainValue>0</pskc:PlainValue>
                </pskc:Counter>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ1110000001</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:ocra"
Id="ZZ1110000001">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>OCRA-1:HOTP-SHA1-6:QA08</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</pskc:PlainValue>
                </pskc:Secret>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ1110000002</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:ocra"
Id="ZZ1110000002">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>OCRA-1:HOTP-SHA1-6:C-QA08-
PSHA1</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
```

```
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</pskc:PlainValue>
                </pskc:Secret>
                <pskc:Counter>
                    <pskc:PlainValue>0</pskc:PlainValue>
                </pskc:Counter>
            </pskc:Data>
            <pskc:Policy>
                <pskc:PINPolicy PINEncoding="BINARY" MaxLength="20"
MinLength="20" PINUsageMode="Algorithmic" PINKeyId="ZZ1120000002"/>
            </pskc:Policy>
        </pskc:Key>
    </pskc:KeyPackage>
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ1120000002</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:pin"
Id="ZZ1120000002">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>fEqNCco3Yq9h5ZUglD3CZJT4lBs=</pskc:PlainValue>
                </pskc:Secret>
            </pskc:Data>
            <pskc:Policy>
                <pskc:PINPolicy PINEncoding="BINARY" MaxLength="20"
MinLength="20" PINUsageMode="Local" PINKeyId="ZZ1130000002"/>
            </pskc:Policy>
        </pskc:Key>
    </pskc:KeyPackage>
   <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ1130000002</pskc:SerialNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:pin"
Id="ZZ1130000002">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>fEqNCco3Yq9h5ZUglD3CZJT4lBs=</pskc:PlainValue>
                </pskc:Secret>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
</pskc:KeyContainer>
```

6.2 Multiple key hardware device token case

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

```
<pskc:KeyContainer xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
xmlns:pkcs5="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5v2-
0#" Id="ExampleID" Version="1.0">
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ2000000000</pskc:SerialNo>
            <pskc:IssueNo>01</pskc:IssueNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:hotp"
Id="ZZ2000000000#01">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>HMAC-SHA1</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</pskc:PlainValue>
                </pskc:Secret>
                <pskc:Counter>
                    <pskc:PlainValue>0</pskc:PlainValue>
                </pskc:Counter>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ2000000000</pskc:SerialNo>
            <pskc:IssueNo>02</pskc:IssueNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:totp"
Id="ZZ2000000000#02">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>HMAC-SHA1</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=</pskc:PlainValue>
                </pskc:Secret>
                <pskc:Time>
                    <pskc:PlainValue>0</pskc:PlainValue>
                </pskc:Time>
                <pskc:TimeInterval>
                    <pskc:PlainValue>30</pskc:PlainValue>
                </pskc:TimeInterval>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
```

```
    <pskc:KeyPackage>
        <pskc:DeviceInfo>
            <pskc:Manufacturer>Manufacturer</pskc:Manufacturer>
            <pskc:SerialNo>ZZ2000000000</pskc:SerialNo>
            <pskc:IssueNo>03</pskc:IssueNo>
        </pskc:DeviceInfo>
        <pskc:Key Algorithm="urn:ietf:params:xml:ns:keyprov:pskc:totp"
Id="ZZ2000000000#03">
            <pskc:Issuer>Issuer0</pskc:Issuer>
            <pskc:AlgorithmParameters>
                <pskc:Suite>HMAC-SHA256</pskc:Suite>
                <pskc:ResponseFormat Length="6" Encoding="DECIMAL"/>
            </pskc:AlgorithmParameters>
            <pskc:Data>
                <pskc:Secret>

<pskc:PlainValue>MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTM=</pskc:Pla
inValue>
                </pskc:Secret>
                <pskc:Time>
                    <pskc:PlainValue>0</pskc:PlainValue>
                </pskc:Time>
                <pskc:TimeInterval>
                    <pskc:PlainValue>30</pskc:PlainValue>
                </pskc:TimeInterval>
            </pskc:Data>
        </pskc:Key>
    </pskc:KeyPackage>
</pskc:KeyContainer>
```

7 References

[OCRA] OCRA: OATH Challenge-Response Algorithms
http://tools.ietf.org/html/rfc6287

[RFC 6030] Portable Symmetric Key Container
http://tools.ietf.org/search/rfc6030

[OTIS] OATH Token Identifier specification
http://www.openauthentication.org/oath-id/

[ALG] Additional PSKC Algorithm Profiles
http://tools.ietf.org/id/draft-hoyer-keyprov-pskc-algorithm-profiles-01.txt

[NIST-800-63-1] NIST Special Publication 800-63-1,
csrc.**nist**.gov/publications/**nist**pubs/**800-63-1**/SP-**800-63-1**.pdf, Dec. 2011