OCRA Standalone Client Profile

Version 1.0

10/03/2012

1 Overview

This document defines the technical requirements for compliance with an OCRA Standalone Client profile for OATH Certification.

An OCRA standalone client is typically comprised of the following entities:

1. A token ID
2. The OCRA algorithm and related algorithm parameters
3. A way to communicate its secret key and related meta data to an OTP validation server

OATH Standalone OCRA Client Profile defines criteria for each of the above aspects of a client. In summary, the OCRA Standalone Client requires the following:

1. The token ID MUST comply with OATH Token ID Specification [OTIS]
2. The OTP algorithm MUST be as defined in 'OCRA: OATH Challenge-Response Algorithms' [RFC 6287].
3. Client application providers MUST supply a PSKC [RFC 6030] file to communicate secret keys and related meta data to a validation server

The detailed specifications for the OCRA Standalone Client profile are contained in Section 2.

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.)

2 Token ID Compliance

A compliant OCRA standalone client must use a Token ID that meets the following criteria:

a. The Token ID format SHOULD meet the specifications outlined in the OATH Token Identifier specification [OTIS].
   - OATH Manufacturer Prefix (OMP) MUST be registered at http://www.openauthentication.org/oath-id/prefixes
b. Each OCRA Standalone Client MUST be assigned a unique Token ID
c. Token ID MUST be printed on the token or MUST be able to be displayed in UI

3 Support for OCRA algorithm

The client application MUST implement the OCRA algorithm according to [RFC 6287]. To be compliant the client application MUST support one of the algorithm modes described below and meet all the necessary requirements.

3.1 Challenge-Response Authentication Mode

If the client application is designed to support challenge-response authentication, then it 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 OCRA stand alone client:

- MUST implement all the general requirements described in section 3.1.1 below
- MUST implement one option from section 3.1.2.
- MUST satisfy requirements specified in sections 3.1.3 through 3.1.6.
- MUST not support any other data inputs or options.

3.1.1 General Requirements

a. The OCRA response value calculated MUST be based on the OCRA algorithm defined [OCRA] 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 OCRA standalone client MUST be able to make generated response visible and/or accessible, for e.g. through a programmatic API.

3.1.2 Challenge/Response OCRASuite values

The following are the valid OCRASuite values. The OCRA standalone client MUST support one of the following 13 OCRASuite combinations.

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

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

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

3.1.3 HMAC Function

a. The client implementation MUST use HMAC-SHA-1 OR HMAC-SHA-256 for the computation.

b. The secret key size MUST be at least 20 bytes if HMAC-SHA-1 is used for computation. The secret key size MUST be at least 32 bytes if HMAC-SHA-256 is used for the computation.

3.1.4 Challenge Format

The format for the challenge question Q MUST be Numeric 6 digits (N06) or Numeric 8 digits (N08), Alphanumeric 6 numeric digits (QA06), or Alphanumeric 8 numeric digits (QA08).

3.1.5 Response (output) format and length

Response length MUST be 6 or 8 numeric digits.

3.1.6 PIN Policy

The OCRA standalone client MUST support one of the following 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.

3.2 Signature Mode

If the OCRA standalone client is designed to support signatures, then it MUST support one of the Algorithm modes described in section '7.3.1 – Plain Signature' of the OCRA specification or suites where a moving factor is used.

OCRA specification enables a wide variety of options. To foster interoperability, the compliant OCRA standalone client:

- MUST implement all the general requirements described in section 3.2.1 below
- MUST implement one option from section 3.2.2.
- MUST satisfy requirements specified in sections 3.2.3 through 3.2.5.
- MUST not support any other data inputs or options.

3.2.1 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. The OCRA standalone client MUST be able to make generated response visible and/or accessible.

3.2.2 Signature OCRASuite values

The following are the valid OCRASuite values. The client application MUST support one of the following 19 combinations. The list consists of "plain" signature and signing with a moving factor. Also, for each combination the client application MUST use the specified approach for Challenge (QS) generation.

Signature with application generated challenge:

A1. OCRA-1:HOTP-SHA1-6: QA06
A2.OCRA-1:HOTP-SHA1-8: QA08
A3.OCRA-1:HOTP-SHA256-8: QA08
A4.OCRA-1:HOTP-SHA1-6: QA06-T30S
A5.OCRA-1:HOTP-SHA1-8: QA08-T30S
A6.OCRA-1:HOTP-SHA256-6: QA06-T30S
A7.OCRA-1:HOTP-SHA256-8: QA08-T30S

Transaction signing with transaction data generated challenge:

S1. OCRA-1:HOTP-SHA1-6: QH40
S2. OCRA-1:HOTP-SHA1-6: QA32

S3. OCRA-1:HOTP-SHA1-8: QH40
S4. OCRA-1:HOTP-SHA256-8: QA32
S5. OCRA-1:HOTP-SHA256-8: QH64
S6. OCRA-1:HOTP-SHA1-6: QA32-T30S
S7. OCRA-1:HOTP-SHA1-6: QH40-T30S
S8. OCRA-1:HOTP-SHA1-8: QA32-T30S
S9. OCRA-1:HOTP-SHA1-8: QH40-T30S
S10. OCRA-1:HOTP-SHA256-6: QH64-T30S
S11. OCRA-1:HOTP-SHA256-8: QA32-T30S
S12. OCRA-1:HOTP-SHA256-8: QH64-T30S

3.2.3 HMAC Function

a. The OCRA standalone client implementation MUST use HMAC-SHA-1 OR HMAC-SHA-256 for the computation.

b. The secret key size MUST be at least 20 bytes if HMAC-SHA-1 is used for computation. The secret key size MUST be at least 32 bytes if HMAC-SHA-256 is used for the computation.

3.2.4 Signature Challenge

The OCRA standalone client MUST support one of the following options for the signature-challenge:

a. **Application generated - Alphanumeric 6 digits (QA06) :** In this scenario, the challenge is provided to the user from the application such as a banking application. The client MUST provide an interface for getting the signature-challenge, either programmatic or a user-interface.

b. **Application generated - Alphanumeric 8 digits (QA08):** In this scenario, the challenge is provided to the user from the application such as a banking application. The client MUST provide an interface for getting the signature-challenge, either programmatic or a user-interface.

c. Transaction data signing – Hexadecimal 40 nibbles (H40): In this scenario, the client MUST provide an interface where the user can provide one or more data input values. It then performs an additional processing step to compute the signature-challenge QS, using the GenerateQS_SHA1() function as described below, before performing the actual OCRA computation.

d. Transaction data signing – Hexadecimal 64 nibbles (H64): In this scenario, the client MUST provide an interface where the user can provide one or more data input values. It then performs an additional processing step to compute the signature-challenge QS, using the GenerateQS_SHA256() function as described below before performing the actual OCRA computation.

e. **Transaction data signing – Alphanumeric 32 characters (A32):** In this scenario, the client MUST provide an interface where the user can provide one or more data input values. It then performs an additional processing step to compute the signature-challenge QS, using the GenerateQS_NoHash() function as described below before performing the actual OCRA computation.

3.2.5 Response (output) format and length

Response length MUST be 6 OR 8 numeric digits.

4 OTP Credential Transport

The OCRA Standalone Client provider MUST make the OTP credentials available in the PSKC credential transport data format as defined in [RFC 6030]. Further the PSKC MUST meet all the requirements described below.

PSKC Key Protection & Integrity

The PSKC MUST use one of the following key protection methods and algorithms to protect the OCRA credential values in the PSKC `<KeyContainer>` element:

a. Pre-shared AES key for key encryption. The key encryption algorithm MUST be AES-128-CBC as defined in Section 6.1 of [RFC 6030]
b. PSKC file protected with PBE encryption as defined in Section 6.2 of [PSKC]. The password length is up to 64 characters.

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`

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 value MUST be one of the values specified in Section 3.

   The PSKC SHOULD not contain the `<ChallengeFormat>` element. The value for the <Suite> element has indicated challenge and response format.

c. The PSKC SHOULD NOT contain `<ResponseFormat>` element. The value for the <Suite> element has indicated 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"/>`

5 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].

5.1 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 client device, 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 = CHK1000100020201002080$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 (CHK1000100020201002080$10000)
QS = CHK1000100020201002080$10000~~~~~

5.2 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 client device, 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.

5.3 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 client device, 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
```

Note: The 32 byte SHA256 output MUST be encoded into 64 nibbles.

6 References

[RFC 6287] OCRA: OATH Challenge-Response Algorithms
http://tools.ietf.org/html/rfc6287

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