Internet Draft
David M'Raihi
VeriSign
Category:
Johan Rydell
Informational
PortWise
Document:
David Naccache
draft-mraihi-mutual-oath-hotp-variants-06.txt
ENS
Salah Machani
Diversinet
Siddharth Bajaj
VeriSign

Expires:
June 2008
December 2007

OCRA: OATH Challenge-Response Algorithms

Status of this Memo

Abstract

This document describes the OATH algorithm for challenge-response authentication and signatures. This algorithm is based on the HOTP algorithm [RFC4226] that was introduced by OATH (initiative for Open AuTHentication) [OATH] and submitted as an individual draft to the IETF last year.

OCRA: OATH Challenge Response Algorithms December 2007

Table of Contents

1. Introduction

OATH has identified several use cases and scenarios that require an asynchronous variant to accommodate users who do not want to maintain a synchronized authentication system. The commonly accepted method for this is to use a challenge-response scheme.

Such challenge response mode of authentication is widely adopted in the industry. Several vendors already offer software applications and hardware devices implementing challenge-response - but each of those uses vendor-specific proprietary algorithms. For the benefits of users we need a standardized challenge-response algorithm to allow multi-sourcing of token purchases and validation systems to facilitate the democratization of strong authentication. Additionally, this specification can also be used to create symmetric key based digital signatures. Such systems are variants of challenge-response mode where the data to be signed becomes the challenge.

2. Requirements Terminology

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 RFC 2119 [RFC2119].

3. Algorithm Requirements

This section presents the main requirements that drove this algorithm design. A lot of emphasis was placed on flexibility and usability, under the constraints and specificity of the HOTP algorithm and hardware token capabilities.

R1 - The algorithm MUST support asynchronous challenge-response based authentication.

R2 - The algorithm MUST be capable of supporting symmetric key based digital signatures. Essentially this is a variation of challenge-response where the challenge is derived from the data that needs to be signed.

R3 - The algorithm MUST be capable of supporting server-authentication, whereby the user can verify that he/she is talking to a valid server.

R4 - The algorithm SHOULD use HOTP [RFC4226] as a key building block.

R5 - The length and format for the input challenge SHOULD be configurable.

R6 - The output length and format for the response SHOULD be configurable.

R7 - The challenge MAY be generated with integrity checking (e.g., parity bits). This will allow tokens with pin pads to perform simple error checking if the user enters the value into a token.

R8 - There MUST be a unique secret (key) for each token/soft token that is shared between the token and the authentication server. The keys MUST be randomly generated or derived using some key derivation algorithm.

R9 - The algorithm MUST enable additional data attributes such as a counter, a time function or session information to be included in the computation. These data inputs MAY be used individually or all together.

4. OCRA Background

OATH introduced the HOTP algorithm as a first open, freely available building block toward hardening authentication for end-users in a variety of applications. One-time passwords are very efficient at solving specific security issues thanks to the dynamic

nature of OTP computations.

After carefully analyzing different use cases, OATH came to the conclusion that providing for extensions to the HOTP algorithms was important. A very natural extension is to introduce a challenge mode for computing HOTP values based on random questions. Equally beneficial, being able to perform mutual authentication between two parties, or short-signature computation for authenticating transaction was also identified as critical for improving the security of e-commerce applications.

4.1 HOTP Algorithm

The HOTP algorithm, as defined in [RFC4226] is based on an increasing counter value and a static symmetric key known only to the prover and verifier parties.

As a reminder:

HOTP(K,C) = Truncate(HMAC-SHA1(K,C))

Where Truncate represents the function that converts an HMAC-SHA-1 value into an HOTP value.

We refer the reader to [RFC4226] for the full description and further details on the rationale and security analysis of HOTP.

The present draft describes the different variants based on similar constructions as HOTP.

5. Definition of OCRA

OCRA is a generalization of HOTP with variable data inputs not solely based on an incremented counter and secret key values.

The definition of OCRA requires a cryptographic function, a key K and a set of DataInput parameters. This section first formally introduces the OCRA algorithm and then introduces the definitions and default values recommended for all the parameters.

In a nutshell,

OCRA = CryptoFunction(K, DataInput)

Where:

- K: a shared secret key known to both parties;
- DataInput: a structure that contains the concatenation of the various input data values. Defined in details in section 5.1;
- CryptoFunction: this is the function performing the OCRA computation from the secret key K and DataInput material; CryptoFunction is described in details in section 5.2.

5.1 DataInput Parameters

This structure is the concatenation of all the parameters used in the computation of the OCRA values, save for the secret key K.

DataInput = {C | Q | P | S | T} where:

- C is a 8-byte counter value processed high-order bit first, and MUST be synchronized between all parties;
- Q is the list of (concatenated) challenge question(s) generated by the verifier(s);the questions SHOULD be L-byte values and MUST be at least t-byte values;
- P is a SHA1-hash of PIN/password that is known to all parties during the execution of the algorithm;
- S is a string that contains information about the current session;
- T is a timestamp value in number of minutes since midnight UTC of January 1, 1970.

When computing a response, the concatenation order is always the following:

C,
OTHER-PARTY-GENERATED-CHALLENGE-QUESTION,
YOUR-GENERATED-CHALLENGE-QUESTION,
P, S and then T values.

If a value is empty (i.e. a certain input is not used in the computation) then the value is simply not represented in the string.

We always start with C to be compliant and follow the HOPT RFC when

all the other values are empty. The counter on the token or client is incremented every time a new computation is requested by the user. The server's counter value is only incremented after a successful OCRA authentication

5.2 CryptoFunction

The default CryptoFunction is HOTP-SHA1-6, i.e. the default mode of computation for OCRA is HOTP with the default 6-digit dynamic truncation and a combination of DataInput values as the message to compute the HMAC-SHA1 digest.

We denote t as the digit-length of the truncation output. For instance, if t = 6, then the output of the truncation is a 6-digit value.

We define the HOTP family of functions as an extension to HOTP:
- HOTP-H-t: these are the different possible truncated versions of HOTP, using the dynamic truncation method for extracting an HOTP value from the HMAC output;
- We will denote HOTP-H-t as the realization of an HOTP function that uses an HMAC function with the hash function H, and the dynamic truncation as described in [RFC 4226] to extract a t-digit value;
- t=0 means that no truncation is performed and the full HMAC value is used for authentication purpose.

We list the following preferred modes of computation, where * denotes the default CryptoFunction:

. HOTP-SHA1-4: HOTP with SHA-1 as the hash function for HMAC and a dynamic truncation to a 4-digit value; this mode is not recommended in the general case but can be useful when a very short authentication code is needed by an application;
. *HOTP-SHA1-6: HOTP with SHA-1 as the hash function for HMAC and a dynamic truncation to a 6-digit value;

. HOTP-SHA256-6: HOTP with SHA-256 as the hash function for HMAC and a dynamic truncation to a 6-digit value;
. HOTP-SHA512-6: HOTP with SHA-512 as the hash function for HMAC and a dynamic truncation to a 6-digit value;

This table summarizes all possible values for the CryptoFunction:

Name	HMAC Function Used	Size of Truncation (t)
HOTP-SHA1-t	HMAC-SHA1	0 (no truncation), 4-10
HOTP-SHA256-t	HMAC-SHA256	0 (no truncation), 4-10
HOTP-SHA512-t	HMAC-SHA512	0 (no truncation), 4-10

6. The OCRASuite

The following values define the OCRASuite codes used in the description of modes of operation for the OCRA algorithm.

An OCRASuite value defines an OCRA suite of operations as supported in the present draft and is represented as follows:

Algorithm:CryptoFunction:DataInput

The client and server need to agree on one or two values of OCRASuite. These values may be agreed at time of token provisioning or for more sophisticated client-server interactions these values may be negotiated for every transaction. Note that for Mutual Challenge-Response or Signature with Server Authentication modes, the client and server will need to agree on two values of OCRASuite - one for server computation and another for client computation.

Algorithm

Description: Indicates the version of OCRA algorithm.
Values: OCRA-v where v represents the version number (e.g. 1, 2 etc.). This document describes version 1 of the OCRA algorithm.

CryptoFunction

Description: Indicates the function used to compute OCRA values
Values: Permitted values are described in section 5.2

DataInput

Description: List of valid inputs for the computation; [] indicates a value is optional.
Values:
[C] | Q | [P | S | T]: Challenge-Response computation
[C] | Q | [P | T]: Plain Signature computation

Example of possible values: OCRA-1:HOTP-SHA512-8:C-Q-P means version 1 of the OCRA algorithm with HMAC-SHA512 function, truncated to an 8-digit value, using the counter, a random challenge and a hash of the PIN/Password as parameters.

7. Algorithm Modes for Authentication

In this section we describe the typical modes in which the above defined computation can be used for authentication.

7.1 One way Challenge-Response

A challenge/response is a security mechanism in which the verifier presents a question (challenge) to the prover who must provide a valid answer (response) to be authenticated.

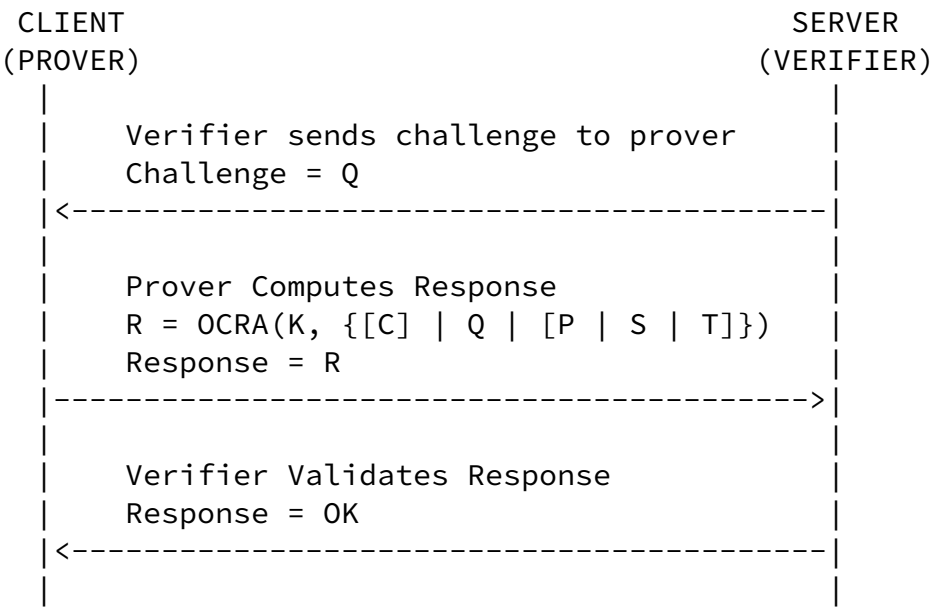
To use this algorithm for a one-way challenge-response, the verifier will communicate a challenge value (typically randomly generated) to the prover. The prover will use the challenge in the computation as described above. The prover then communicates the response to the verifier to authenticate.

Therefore in this mode, the typical data inputs will be:

C - Counter, optional.
Q - Challenge question, mandatory, supplied by the verifier.
P - Hashed version of PIN/password, optional.
S - Session information, optional
T - Timestamp, optional.

The picture below shows the messages that are exchanged between the client (prover) and the server (verifier) to complete a one-way challenge-response authentication.

We assume that the client and server have a pre-shared key K that is used for the computation.



7.2 Mutual Challenge-Response

Mutual challenge-response is a variation of one-way challenge-response where both the client and server and mutually authenticate each other.

To use this algorithm, the client will first send a random client-challenge to the server. The server computes the server-response and sends it to the client along with a server-challenge.

The client will first verify the server-response to authenticate that it is talking to a valid server. It will then compute the client-response and send it to the server to authenticate. The server verifies the client-response to complete the two-way authentication process.

In this mode there are two computations: client-response and server-response. There are two separate challenge questions, generated by both parties. We denote these challenge questions Q1 and Q2.

Typical data inputs for server-response computation will be:
C - Counter, optional.
QC - Challenge question, mandatory, supplied by the client.
QS - Challenge question, mandatory, supplied by the server.
S - Session information, optional.

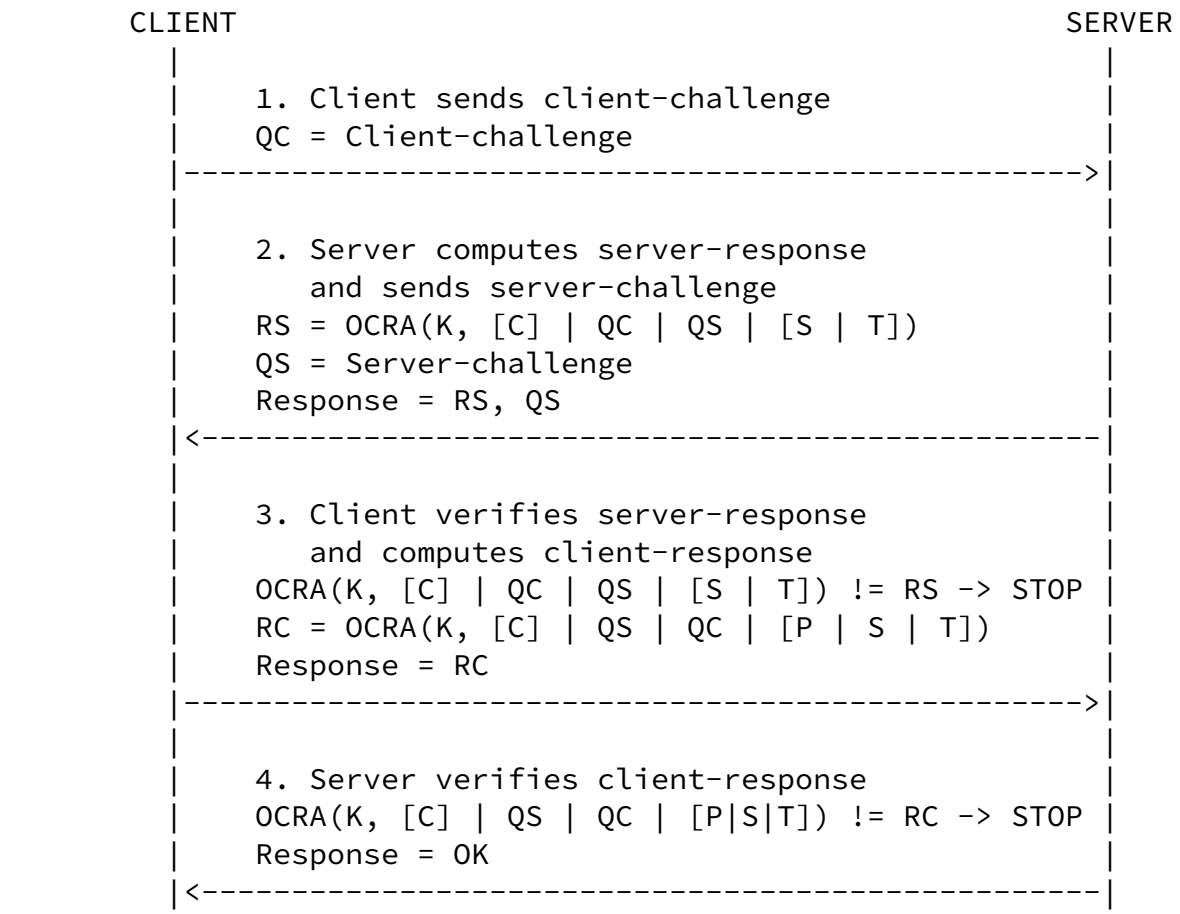
T - Timestamp, optional.

Typical data inputs for client-response computation will be:
C - Counter, optional.
QS - Challenge question, mandatory, supplied by the server.
QC - Challenge question, mandatory, supplied by the client.

P - Hashed version of PIN/password, optional.
S - Session information, optional.
T - Timestamp, optional.

The following picture shows the messages that are exchanged between the client and the server to complete a two-way mutual challenge-response authentication.

We assume that the client and server have a pre-shared key K that is used for the computation.



| |

8. Algorithm Modes for Signature

In this section we describe the typical modes in which the above defined computation can be used for digital signatures.

8.1 Plain Signature

To use this algorithm in plain signature mode, the server will communicate a signature-challenge value to the client (signer). The

signature-challenge is either the data to be signed or derived from the data to be signed using a hash function, for example.

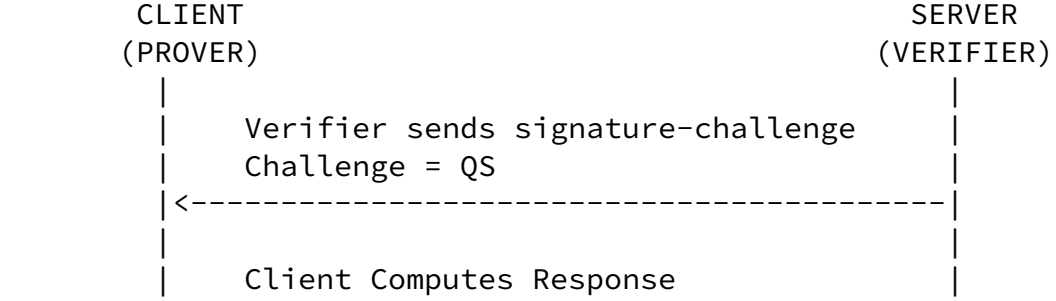
The client will use the signature-challenge in the computation as described above. The client then communicates the signature value (response) to the server to authenticate.

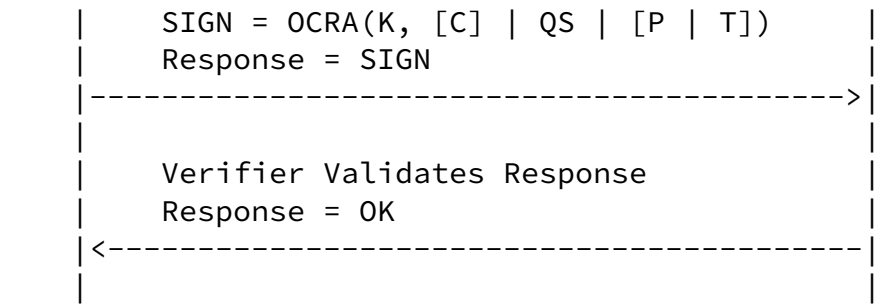
Therefore in this mode, the data inputs will be:

C - Counter, optional.
QS - Signature-challenge, mandatory, supplied by the server.
P - Hashed version of PIN/password, optional.
T - Timestamp, optional.

The picture below shows the messages that are exchanged between the client (prover) and the server (verifier) to complete a plain signature operation.

We assume that the client and server have a pre-shared key K that is used for the computation.





8.2 Signature with Server Authentication

This mode is a variation of the plain signature mode where the client can first authenticates the server before generating a digital signature.

To use this algorithm, the client will first send a random client-challenge to the server. The server computes the server-response and sends it to the client along with a signature-challenge. The client will first verify the server-response to authenticate that

it is talking to a valid server. It will then compute the signature and send it to the server.

In this mode there are two computations: client-signature and server-response.

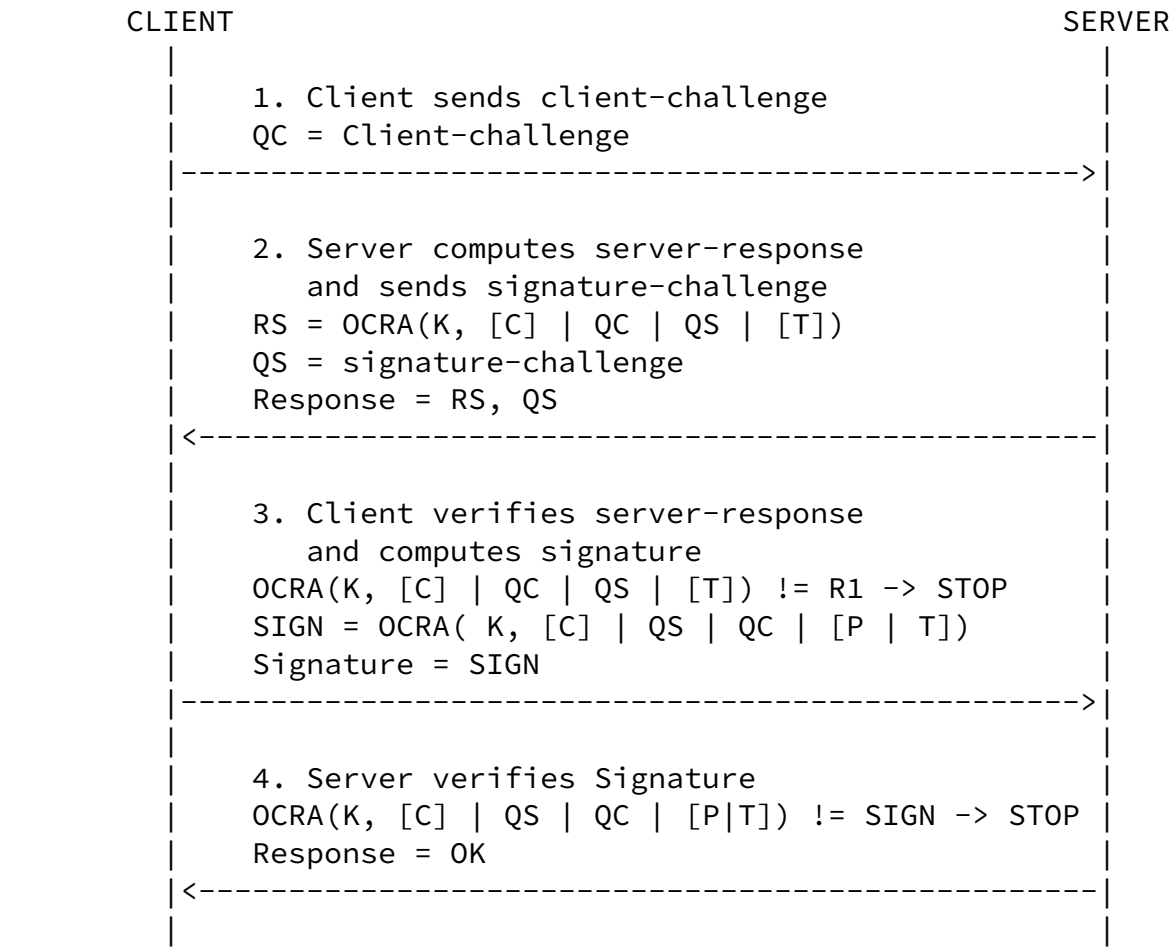
Typical data inputs for server-response computation will be:
C - Counter, optional.
QC - Challenge question, mandatory, supplied by the client.
T - Timestamp, optional.

Typical data inputs for client-signature computation will be:
C - Counter, optional.
QS - Signature-challenge, mandatory, supplied by the server.
P - Hashed version of PIN/password, optional.
T - Timestamp, optional.

The picture below shows the messages that are exchanged between the client and the server to complete a signature with server authentication transaction.

We assume that the client and server have a pre-shared key K that

is used for the computation.



9. Security Considerations

Any algorithm is only as secure as the application and the authentication protocols that implement it. Therefore, this section discusses the critical security requirements that our choice of algorithm imposes on the authentication protocol and validation software.

9.1 Security Analysis of the OCRA algorithm

The security and strength of this algorithm depends on the properties of the underlying building block HOTP, which is a construction based on HMAC [RFC2104] using SHA-1 as the hash function.

The conclusion of the security analysis detailed in [RFC4226] is that, for all practical purposes, the outputs of the dynamic truncation on distinct counter inputs are uniformly and independently distributed strings.

The analysis demonstrates that the best possible attack against the HOTP function is the brute force attack.

9.2 Implementation Considerations

S1 - In the authentication mode, the client MUST support two-factor authentication, i.e., the communication and verification of something you know (secret code such as a Password, Pass phrase, PIN code, etc.) and something you have (token). The secret code is known only to the user and usually entered with the Response value for authentication purpose (two-factor authentication). Alternatively, instead of sending something you know to the server, the client may use a hash of the Password or PIN code in the computation itself, thus implicitly enabling two-factor authentication.

S2 - The keys for HOTP can be of any length equal or longer than L bytes, where L is the byte-length of the CryptoFunction output. Keys longer than L bytes are acceptable; they are first hashed using the supported hash function, e.g. SHA-1, to become usable. Nevertheless, the extra length would not significantly increase the cryptographic strength of OCRA, provided the randomness of the original key material is sufficient.

S3 - Keys need to be chosen at random or using a cryptographically strong pseudo-random generator properly seeded with a random value. We RECOMMEND following the recommendations in [RFC1750] for all pseudo-random and random generations. The pseudo-random numbers

used for generating the keys SHOULD successfully pass the randomness test specified in [CN].

S4 - On the client side, the keys MUST be embedded in a tamper resistant device or securely implemented in a software application. Additionally, by embedding the keys in a hardware device, you also have the advantage of improving the flexibility (mobility).

S5 - For authentication computations, the challenge value MUST be randomly generated and SHALL NOT be re-used. We RECOMMEND following the recommendations in [RFC1750] for all pseudo-random and random generations.

S6 - All the communications SHOULD take place over a secure channel e.g. SSL/TLS, IPsec connections.

S7 - The OCRA algorithm when used in mutual authentication mode or in signature with server authentication mode SHOULD use dual key mode - i.e. there are two keys that are shared between the client and the server. One shared key is used to generate the server response on the server side and to verify it on the client side. The other key is used to create the response or signature on the client side and to verify the same on the server side.

S8 - We recommend that implementations MAY use the session information, S as an additional input in the computation. For example, S could be the session identifier from the TLS session. This will enable you to counter certain types of man-in-the-middle attacks. However, this will introduce the additional dependency that first of all the prover needs to have access to the session identifier to compute the response and the verifier will need access to the session identifier to verify the response.

S9 - In the signature mode, whenever the counter or time (defined as optional elements) are not used in the computation, there might be a risk of replay attack and the implementers should carefully consider this issue in the light of their specific application requirements and security guidelines.

S10 - We also RECOMMEND storing the shared secrets securely in the validation system, and more specifically encrypting the shared secrets using tamper-resistant hardware encryption and exposing them only when required: for example, the shared secret is decrypted when needed to verify an HOTP value, and re-encrypted immediately to limit exposure in the RAM for a short period of time. The data store holding the shared secrets MUST be in a secure area, to avoid as much as possible direct attack on the validation system and secrets database.

Particularly, access to the shared secrets should be limited to

programs and processes required by the validation system only. We will not elaborate on the different security mechanisms to put in place, but obviously, the protection of shared secrets is of the uttermost importance.

10. IANA Considerations

This document has no actions for IANA.

11. Conclusion

This draft introduced several variants of HOTP for challenge-response based authentication and short signature-like computations.

The OCRASuite provides for an easy integration and support of different flavors within an authentication and validation system.

Finally, OCRA should enable cross-authentication both in connected and off-line modes, with the support of different response sizes and mode of operations.

12. Acknowledgements

We would like to thank Philip Hoyer, Jonathan Tuliani, Shuh Chang, Stu Vaeth, Jon Martinsson, Jeff Burstein, Frederik Mennes, Oanh Hoang, Mingliang Pei and Enrique Rodriguez for their comments and suggestions to improve this draft document.

13. References

13.1 Normative

[RFC2104] M. Bellare, R. Canetti and H. Krawczyk, "HMAC: Keyed-Hashing for Message Authentication", IETF Network Working Group, RFC 2104, February 1997.

[RFC1750] D. Eastlake, 3rd., S. Crocker and J. Schiller, "Randomness Recommendations for Security", IETF Network Working Group, RFC 1750, December 2004.

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

[RFC3668] S. Bradner, "Intellectual Property Rights in IETF Technology", BCP 79, RFC 3668, February 2004.

[RFC4226] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache and O. Ranen, "HOTP: An HMAC-based One Time Password Algorithm", IETF Network Working Group, RFC 4226, December 2005.

13.2 Informative

[BCK] M. Bellare, R. Canetti and H. Krawczyk, "Keyed Hash Functions and Message Authentication", Proceedings of Crypto'96, LNCS Vol. 1109, pp. 1-15.

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

[CN] J.S. Coron and D. Naccache, "An accurate evaluation of Maurer's universal test" by Jean-Sebastien Coron and David Naccache In Selected Areas in Cryptography (SAC '98), vol. 1556 of Lecture Notes in Computer Science, S. Tavares and H. Meijer, Eds., pp. 57-71, Springer-Verlag, 1999

Appendix A: Source Code

```
import java.lang.reflect.UndeclaredThrowableException;
import java.security.GeneralSecurityException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

/**
 * This an example implementation of the OATH OCRA algorithm.
 * Visit www.openauthentication.org for more information.
 *
 * @author Johan Rydell, PortWise
 */
public class OCRA {
   private OCRA() {}

   /**
    * This method uses the JCE to provide the crypto
```

```
     * algorithm.
     * HMAC computes a Hashed Message Authentication Code with the
     * crypto hash algorithm as a parameter.
```



```
     *
     * @param crypto     the crypto algorithm
     *                   (HmacSHA1, HmacSHA256, HmacSHA512)
     * @param keyBytes   the bytes to use for the HMAC key
     * @param text       the message or text to be authenticated.
     */
    public static byte[] hmac_sha1(String crypto,
                                   byte[] keyBytes,
                                   byte[] text)
    {
       try {
          Mac hmac;
          hmac = Mac.getInstance(crypto);
          SecretKeySpec macKey =
             new SecretKeySpec(keyBytes, "RAW");
          hmac.init(macKey);
          return hmac.doFinal(text);
       } catch (GeneralSecurityException gse) {
          throw new UndeclaredThrowableException(gse);
       }
    }

    private static final int[] DIGITS_POWER
    // 0 1  2   3    4     5      6       7        8
    = {1,10,100,1000,10000,100000,1000000,10000000,100000000};

    /**
     * This method generates an OCRA HOTP value for the given
     * set of parameters.
     *
     * @param crypto              the crypto algorithm
     * @param key                 the shared secret
     * @param movingFactor        the counter that changes
     *                             on a per use basis
     * @param question            the challenge question
     * @param password            a password that can be used
     * @param sessionInformation    Static information that
     *                            identifies the current session
     * @param timeStamp           a value that reflects a time
```

```
     * @param codeDigits          number of digits in the OTP
     *
     * @return A numeric String in base 10 that includes
     * {@link truncationDigits} digits
     */
    static public String generateOTP(String crypto,
          String key,
          String movingFactor,
          String question,
          String password,
```



```
          String sessionInformation,
          String timeStamp,
          int codeDigits)
    {
       String result = null;
       String messageStr =
          question + password +
          sessionInformation + timeStamp ;
       byte[] msg;

       // Using the counter
       if (0 < movingFactor.length()){
          // First 8 bytes are for the movingFactor
          // Complient with RFC 4226
          messageStr = "00000000" + messageStr;
          msg = messageStr.getBytes();
          long mFactor = Long.decode(movingFactor);
          for (int i = 7; i >= 0; i--) {
             msg[i] = (byte) (mFactor & 0xff);
             mFactor >>= 8;
          }
       }else
          msg = messageStr.getBytes();

       // compute hmac hash
       byte[] hash = hmac_sha1(crypto, key.getBytes(), msg);

       // put selected bytes into result int
       int offset = hash[hash.length - 1] & 0xf;

       int binary =
          ((hash[offset] & 0x7f) << 24) |
```

```
            ((hash[offset + 1] & 0xff) << 16) |
           ((hash[offset + 2] & 0xff) << 8) |
            (hash[offset + 3] & 0xff);

      int otp = binary % DIGITS_POWER[codeDigits];

      result = Integer.toString(otp);
      while (result.length() < codeDigits) {
         result = "0" + result;
      }
      return result;
   }
}
```


Appendix B: Test Vectors

Plain challenge response

OCRA-HOTP-SHA1-8-Q

```
K = 12345678901234567890      Q = 10000000      OCRA = 57953866
K = 12345678901234567890      Q = 10000001      OCRA = 15772773
K = 12345678901234567890      Q = 10000002      OCRA = 68105940
```

OCRA-HOTP-SHA256-8-Q

```
K = 12345678901234567890      Q = 10000000      OCRA = 79730854
K = 12345678901234567890      Q = 10000001      OCRA = 22925447
K = 12345678901234567890      Q = 10000002      OCRA = 15947867
```

OCRA-HOTP-SHA512-8-Q

```
K = 12345678901234567890      Q = 10000000      OCRA = 68325835
K = 12345678901234567890      Q = 10000001      OCRA = 53995836
K = 12345678901234567890      Q = 10000002      OCRA = 89008345
```

Mutual challenge response

```
==========================

OCRA-HOTP-SHA512-8-Q
--------------------
(From server) K = 12345678901234567890
Q1 = 11111110     Q2 = 22222220     OCRA = 70933163
(From client) K = 12345678901234567890
Q1 = 11111110     Q2 = 22222220     OCRA = 63875222

(From server) K = 12345678901234567890
Q1 = 11111111     Q2 = 22222221     OCRA = 08364053
(From client) K = 12345678901234567890
Q1 = 11111111     Q2 = 22222221     OCRA = 91844292

(From server) K = 12345678901234567890
Q1 = 11111112     Q2 = 22222222     OCRA = 70960179
(From client) K = 12345678901234567890
Q1 = 11111112     Q2 = 22222222     OCRA = 75789938
```



```
Plain signature
===============

OCRA-HOTP-SHA512-8-Q
--------------------
K = 12345678901234567890      Q (value) = 00010000
OCRA (signature) = 13175449
K = 12345678901234567890      Q (value) = 00011000
OCRA (signature) = 41866883
K = 12345678901234567890      Q (value) = 00012000
OCRA (signature) = 82912137
```

14. Authors' Addresses

Primary point of contact (for sending comments and question):

David M'Raihi

VeriSign, Inc.
685 E. Middlefield Road Phone: 1-650-426-3832
Mountain View, CA 94043 USA Email: dmraihi@verisign.com

Other Authors' contact information:

Johan Rydell
Portwise, Inc.
275 Hawthorne Ave, Suite 119 Phone: 1-650-515-3569
Palo Alto, CA 94301 USA Email: johan.rydell@portwise.com

David Naccache
ENS, DI
45 rue d'Ulm Phone: +33 6 16 59 83 49
75005, Paris France Email: david.naccache@ens.fr

Salah Machani
Diversinet Corp.
2225 Sheppard Avenue East
Suite 1801
Toronto, Ontario M2J 5C2 Phone: 1-416-756-2324 Ext. 321
Canada Email: smachani@diversinet.com

Siddharth Bajaj
VeriSign, Inc.
487 E. Middlefield Road Phone: 1-650-426-3458
Mountain View, CA 94043 USA Email: sbajaj@verisign.com

15. Full Copyright Statement

IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

16. Intellectual Property

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

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

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