Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 29, 2013

Richer
The MITRE Corporation
W. Mills
Yahoo! Inc.
H. Tschofenig, Ed.
Nokia Siemens Networks
February 25, 2013

OAuth 2.0 Message Authentication Code (MAC) Tokens
draft-ietf-oauth-v2-http-mac-03

Abstract

This specification describes how to use MAC Tokens in HTTP requests to access OAuth 2.0 protected resources. An OAuth client willing to access a protected resource needs to demonstrate possession of a crytographic key by using it with a keyed message digest function to the request.

The document also defines a key distribution protocol for obtaining a fresh session key.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This specification describes how to use MAC Tokens in HTTP requests and responses to access protected resources via the OAuth 2.0 protocol [RFC6749]. An OAuth client willing to access a protected resource needs to demonstrate possession of a symmetric key by using it with a keyed message digest function to the request. The keyed message digest function is computed over a flexible set of parameters from the HTTP message.

The MAC Token mechanism requires the establishment of a shared symmetric key between the client and the resource server. This specification defines a three party key distribution protocol to dynamically distribute this session key from the authorization server to the client and the resource server.

The design goal for this mechanism is to support the requirements outlined in [I-D.tschofenig-oauth-security]. In particular, when a server uses this mechanism, a passive attacker will be unable to use an eavesdropped access token exchanged between the client and the resource server. In addition, this mechanism helps secure the access token against leakage when sent over a secure channel to the wrong resource server if the client provided information about the resource server it wants to interact with in the request to the authorization server.

Since a keyed message digest only provides integrity protection and data-origin authentication confidentiality protection can only be added by the usage of Transport Layer Security (TLS). This specification provides a mechanism for channel binding is included to ensure that a TLS channel is not terminated prematurely and indeed covers the entire end-to-end communication.

2. Terminology

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [RFC2119].

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [I-D.ietf-httpbis-p1-messaging]. Additionally, the following rules are included from [RFC2617]: auth-param.

Session Key:

The terms mac key, session key, and symmetric key are used interchangably and refer to the cryptographic keying material established between the client and the resource server. This temporary key used between the client and the resource server, with a lifetime limited to the lifetime of the access token. This session key is generated by the authorization server.

Authenticator:

A record containing information that can be shown to have been recently generated using the session key known only by the client and the resource server.

Message Authentication Code (MAC):

Message authentication codes (MACs) are hash functions that take two distinct inputs, a message and a secret key, and produce a ﬁxed-size output. The design goal is that it is practically infeasible to produce the same output without knowledge of the key. The terms keyed message digest functions and MACs are used interchangably.

3. Architecture

The architecture of the proposal described in this document assumes that the authorization server acts as a trusted third party that provides session keys to clients and to resource servers. These session keys are used by the client and the resource server as input to a MAC. In order to obtain the session key the client interacts with the authorization server as part of the a normal grant exchange. This is shown in an abstract way in Figure 1. Together with the access token the authorization server returns a session key (in the mac_key parameter) and several other parameters. The resource server obtains the session key via the access token. Both of these two key distribution steps are described in more detail in Section 4.

```
                        +---------------+
                       ^|               | AS-RS Key
                     // | Authorization |<*******
                    /   | Server        |       *
                  //    |               |       *
                 /      |               |       *
          (I)  //      /+---------------+       *
    Access    /      //                         *
    Token    /      /                           *
    Request//     //  (II) Access Token         *
          /      /         +Session Key (SK)    *
        //     //                               *
       /      v                                 v
     +-----------+                       +------------+
     |           |                       |            |
     |           |                       | Resource   |
     | Client    |                       | Server     |
     |           |                       |            |
     |           |                       |            |
     +-----------+                       +------------+

     ****: Out-of-Band Long-Term Key Establishment
     ----: Dynamic Session Key Distribution
```

Figure 1: Architecture: Interaction between the Client and the Authorization Server.

Once the client has obtained the necessary access token and the session key (including parameters) it can start to interact with the resource server. To demonstrate possession of the session key it computes a MAC and adds various fields to the outgoing request message. We call this structure the "Authenticator". The server evaluates the request, includes an Authenticator and returns a response back to the client. Since the access token is valid for a period of time the resource server may decide to cache it so that it does not need to be provided in every request from the client. This interaction is shown in Figure 2.

```
                      +---------------+
                      |               |
                      | Authorization |
                      | Server        |
                      |               |
                      |               |
                      +---------------+




   +-----------+  Authenticator (a)    +------------+
   |           |---------------------->|            |
   |           |  [+Access Token]      | Resource   |
   | Client    |                       | Server     |
   |           |  Authenticator (b)    |            |
   |           |<----------------------|            |
   +-----------+                       +------------+

        ^                                    ^
        |                                    |
        |                                    |
       SK                                   SK
       +param                               +param
```

Figure 2: Architecture: Interaction between the Client and the Resource Server.

4. Key Distribution

For this scheme to function a session key must be available to the client and the resource server, which is then used as a parameter in the keyed message digest function. This document describes the key distribution mechanism that uses the authorization server as a trusted third party, which ensures that the session key is transported from the authorization server to the client and the resource server.

4.1. Session Key Transport to Client

Authorization servers issue MAC Tokens based on requests from clients. The request MUST include the audience parameter defined in [I-D.tschofenig-oauth-audience], which indicates the resource server the client wants to interact with. This specification assumes use of

the 'Authorization Code' grant. If the request is processed successfully by the authorization server it MUST return at least the following parameters to the client:

kid

The name of the key (key id), which is an identifier generated by the resource server. It is RECOMMENDED that the authorization server generates this key id by computing a hash over the access_token, for example using SHA-1, and to encode it in a base64 format.

access_token

The OAuth 2.0 access token.

mac_key

The session key generated by the authorization server. Note that the lifetime of the session key is equal to the lifetime of the access token.

mac_algorithm

The MAC algorithm used to calculate the request MAC. The value MUST be one of "hmac-sha-1", "hmac-sha-256", or a registered extension algorithm name as described in Section 9.2. The authorization server is assumed to know the set of algorithms supported by the client and the resource server. It selects an algorithm that meets the security policies and is supported by both nodes.

For example:


```
  HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store

  {
    "access_token":
"eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDK0hTMjU2In0.
pwaFh7yJPivLjjPkzC-GeAyHuy7AinGcS51AZ7TXnwkC80Ow1aW47kcT_UV54ubo
nONbeArwOVuR7shveXnwPmucwrk_3OCcHrCbE1HR-Jfme2mF_WR3zUMcwqmU0RlH
kwx9txo_sKRasjlXc8RYP-evLCmT1XRXKjtY5l44Gnh0A84hGvVfMxMfCWXh38hi
2h8JMjQHGQ3mivVui5lbf-zzb3qXXxNO1ZYoWgs5tP1-T54QYc9Bi9wodFPWNPKB
kY-BgewG-Vmc59JqFeprk1O08qhKQeOGCWc0WPC_n_LIpGWH6spRm7KGuYdgDMkQ
bd4uuB0uPPLx_euVCdrVrA.
AxY8DCtDaGlsbGljb3RoZQ.
7MI2lRCaoyYx1HclVXkr8DhmDoikTmOp3IdEmm4qgBThFkqFqOs3ivXLJTku4M0f
laMAbGG_X6K8_B-0E-7ak-Olm_-_V03oBUUGTAc-F0A.
OwWNxnC-BMEie-GkFHzVWiNiaV3zUHf6fCOGTwbRckU",
    "token_type":"mac",
    "expires_in":3600,
    "refresh_token":"8xLOxBtZp8",
    "kid":"22BIjxU93h/IgwEb4zCRu5WF37s=",
    "mac_key":"adijq39jdlaska9asud",
    "mac_algorithm":"hmac-sha-256"
  }
```


4.2. Session Key Transport to Resource Server

The transport of the mac_key from the authorization server to the resource server is accomplished by conveying the encrypting mac_key inside the access token. At the time of writing only one standardized format for carrying the access token is defined: the JSON Web Token (JWT) [I-D.ietf-oauth-json-web-token]. Note that the header of the JSON Web Encryption (JWE) structure [I-D.ietf-jose-json-web-encryption], which is a JWT with encrypted content, MUST contain a key id (kid) in the header to allow the resource server to select the appropriate keying material for decryption. This keying material is a symmetric or an asymmetric long-term key established between the resource server and the authorization server, as shown in Figure 1 as AS-RS key. The establishment of this long-term key is outside the scope of this specification.

This document defines two new claims to be carried in the JWT: mac_key, kid. These two parameters match the content of the mac_key

and the kid conveyed to the client, as shown in Section 4.1.

kid

The name of the key (key id), which is an identifier generated by the resource server.

mac_key

The session key generated by the authorization server.

This example shows a JWT claim set without header and without encryption:

```
{"iss":"authorization-server.example.com",
"exp":1300819380,
"kid":"22BIjxU93h/IgwEb4zCRu5WF37s=",
"mac_key":"adijq39jdlaska9asud",
"aud":"apps.example.com"
}
```

QUESTIONS: An alternative to the use of a JWT to convey the access token with the encrypted mac_key is use the token introspect [I-D.richer-oauth-introspection]. What mechanism should be described? What should be mandatory to implement?
QUESTIONS: The above description assumes that the entire access token is encrypted but it would be possible to only encrypt the session key and to only apply integrity protection to other fields. Is this desireable?

5. The Authenticator

To access a protected resource the client must be in the possession of a valid set of session key provided by the authorization server. The client constructs the authenticator, as described in Section 5.1.

5.1. The Authenticator

The client constructs the authenticator and adds the resulting fields to the HTTP request using the "Authorization" request header field. The "Authorization" request header field uses the framework defined by [RFC2617]. To include the authenticator in a subsequent response from the authorization server to the client the WWW-Authenticate header is used. For further exchanges a new, yet-to-be-defined

header will be used.

```
authenticator  = "MAC" 1*SP #params

params         = id / ts / seq-nr / access_token / mac / h / cb

kid            = "kid" "=" string-value
ts             = "ts" "=" ( <"> timestamp <"> ) / timestamp
seq-nr         = "seq-nr" "=" string-value
access_token   = "access_token" "=" b64token
mac            = "mac" "=" string-value
cb             = "cb" "=" token
h              = "h" "=" h-tag
h-tag          = %x68 [FWS] "=" [FWS] hdr-name
                 *( [FWS] ":" [FWS] hdr-name )
hdr-name       = token

timestamp      = 1*DIGIT
string-value   = ( <"> plain-string <"> ) / plain-string
plain-string   = 1*( %x20-21 / %x23-5B / %x5D-7E )

b64token       = 1*( ALPHA / DIGIT /
                     "-" / "." / "_" / "~" / "+" / "/" ) *"="
```

The header attributes are set as follows:

kid

REQUIRED. The key identifier.

ts

REQUIRED. The timestamp. The value MUST be a positive integer set by the client when making each request to the number of milliseconds since 1 January 1970.

The JavaScript getTime() function or the Java System.currentTimeMillis() function, for example, produce such a timestamp.

seq-nr

OPTIONAL. This optional field includes the initial sequence number to be used by the messages exchange between the client and the server when the replay protection provided by the

timestamp is not sufficient enough replay protection. This field specifies the initial sequence number for messages from the client to the server. When included in the response message, the initial sequence number is that for messages from the server to the client. Sequence numbers fall in the range 0 through 2^64 - 1 and wrap to zero following the value 2^64 - 1.

The initial sequence number SHOULD be random and uniformly distributed across the full space of possible sequence numbers, so that it cannot be guessed by an attacker and so that it and the successive sequence numbers do not repeat other sequences. In the event that more than 2^64 messages are to be generated in a series of messages, rekeying MUST be performed before sequence numbers are reused. Rekeying requires a new access token to be requested.

access_token

CONDITIONAL. The access_token MUST be included in the first request from the client to the server but MUST NOT be included in a subsequent response and in a further protocol exchange.

mac

REQUIRED. The result of the keyed message digest computation, as described in Section 5.3.

cb

OPTIONAL. This field carries the channel binding value from RFC 5929 [RFC5929] in the following format: cb= channel-binding-type ":" channel-binding-content. RFC 5929 offers two types of channel bindings for TLS. First, there is the 'tls-server-end-point' channel binding, which uses a hash of the TLS server's certificate as it appears, octet for octet, in the server's Certificate message. The second channel binding is 'tls-unique', which uses the first TLS Finished message sent (note: the Finished struct, not the TLS record layer message containing it) in the most recent TLS handshake of the TLS connection being bound to. As an example, the cb field may contain cb=tls-unique:9382c93673d814579ed1610d3

h

OPTIONAL. This field contains a colon-separated list of header field names that identify the header fields presented to the keyed message digest algorithm. If the 'h' header field is absent then the following value is set by default: h="host". The field MUST contain the complete list of header fields in

the order presented to the keyed message digest algorithm. The field MAY contain names of header fields that do not exist at the time of computing the keyed message digest; nonexistent header fields do not contribute to the keyed message digest computation (that is, they are treated as the null input, including the header field name, the separating colon, the header field value, and any CRLF terminator). By including header fields that do not actually exist in the keyed message digest computation, the client can allow the resource server to detect insertion of those header fields by intermediaries. However, since the client cannot possibly know what header fields might be defined in the future, this mechanism cannot be used to prevent the addition of any possible unknown header fields. The field MAY contain multiple instances of a header field name, meaning multiple occurrences of the corresponding header field are included in the header hash. The field MUST NOT include the mac header field. Folding whitespace (FWS) MAY be included on either side of the colon separator. Header field names MUST be compared against actual header field names in a case-insensitive manner. This list MUST NOT be empty. See Section 8 for a discussion of choosing header fields.

Attributes MUST NOT appear more than once. Attribute values are limited to a subset of ASCII, which does not require escaping, as defined by the plain-string ABNF.

5.2. MAC Input String

An HTTP message can either be a request from client to server or a response from server to client. Syntactically, the two types of message differ only in the start-line, which is either a request-line (for requests) or a status-line (for responses).

Two parameters serve as input to a keyed message digest function: a key and an input string. Depending on the communication direction either the request-line or the status-line is used as the first value followed by the HTTP header fields listed in the 'h' parameter. Then, the timestamp field and the seq-nr field (if present) is concatenated.

As an example, consider the HTTP request with the new line separator character represented by "\n" for editorial purposes only. The h parameter is set to h=host, the kid is 314906b0-7c55, and the timstamp is 1361471629.

```
POST /request?b5=%3D%253D&a3=a&c%40=&a2=r%20b&c2&a3=2+q HTTP/1.1
Host: example.com
```

```
Hello World!
```

The resulting string is:

```
POST /request?b5=%3D%253D&a3=a&c%40=&a2=r%20b&c2&a3=2+q HTTP/1.1\n
1361471629\n
example.com\n
```

5.3. Keyed Message Digest Algorithms

The client uses a cryptographic algorithm together with a session key to calculate a keyed message digest. This specification defines two algorithms: "hmac-sha-1" and "hmac-sha-256", and provides an extension registry for additional algorithms.

5.3.1. hmac-sha-1

"hmac-sha-1" uses the HMAC-SHA1 algorithm, as defined in [RFC2104]:

```
mac = HMAC-SHA1 (key, text)
```

Where:

text

is set to the value of the input string as described in Section 5.2,

key

is set to the session key provided by the authorization server, and

mac

is used to set the value of the "mac" attribute, after the result string is base64-encoded per Section 6.8 of [RFC2045].

5.3.2. hmac-sha-256

"hmac-sha-256" uses the HMAC algorithm, as defined in [RFC2104], with the SHA-256 hash function, defined in [NIST-FIPS-180-3]:

```
mac = HMAC-SHA256 (key, text)
```

Where:

text

is set to the value of the input string as described in Section 5.2,

key

is set to the session key provided by the authorization server, and

mac

is used to set the value of the "mac" attribute, after the result string is base64-encoded per Section 6.8 of [RFC2045].

6. Verifying the Authenticator

When receiving a message with an authenticator the following steps are performed:

1. When the authorization server receives a message with a new access token (and consequently a new session key) then it obtains the session key by retrieving the content of the access token (which requires decryption of the session key contained inside the token). The content of the access token, in particular the audience field and the scope, MUST be verified as described in Alternatively, the kid parameter is used to look-up a cached session key from a previous exchange.
2. Recalculate the keyed message digest, as described in Section 5.3, and compare the request MAC to the value received from the client via the "mac" attribute.
3. Verify that no replay took place by comparing the value of the ts (timestamp) header with the local time. The processing of authenticators with stale timestamps is described in Section 6.1.

Error handling is described in Section 6.2.

6.1. Timestamp Verification

The timestamp field enables the server to detect replay attacks. Without replay protection, an attacker can use an eavesdropped request to gain access to a protected resource. The following procedure is used to detect replays:

o At the time the first request is received from the client for each key identifier, calculate the difference (in seconds) between the request timestamp and the local clock. The difference is stored

locally for later use.

- For each subsequent request, apply the request time delta to the timestamp included in the message to calculate the adjusted request time.
- Verify that the adjusted request time is within the allowed time period defined by the authorization server. If the local time and the calculated time based in the request differ by more than the allowable clock skew (e.g., 5 minutes) a replay has to be assumed.

6.2. Error Handling

If the protected resource request does not include an access token, lacks the keyed message digest, contains an invalid key identifier, or is malformed, the server SHOULD return a 401 (Unauthorized) HTTP status code.

For example:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: MAC
```

The "WWW-Authenticate" request header field uses the framework defined by [RFC2617] as follows:

```
challenge   = "MAC" [ 1*SP #param ]
param       = error / auth-param
error       = "error" "=" ( token / quoted-string)
```

Each attribute MUST NOT appear more than once.

If the protected resource request included a MAC "Authorization" request header field and failed authentication, the server MAY include the "error" attribute to provide the client with a human-readable explanation why the access request was declined to assist the client developer in identifying the problem.

For example:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: MAC error="The MAC credentials expired"
```

7. Example

[Editor's Note: Full example goes in here.]

8. Security Considerations

As stated in [RFC2617], the greatest sources of risks are usually found not in the core protocol itself but in policies and procedures surrounding its use. Implementers are strongly encouraged to assess how this protocol addresses their security requirements and the security threats they want to mitigate.

8.1. Key Distribution

This specification describes a key distribution mechanism for providing the session key (and parameters) from the authorization server to the client. The interaction between the client and the authorization server requires Transport Layer Security (TLS) with a ciphersuite offering confidentiality protection. The session key MUST NOT be transmitted in clear since this would completely destroy the security benefits of the proposed scheme. Furthermore, the obtained session key MUST be stored so that only the client instance has access to it. Storing the session key, for example, in a cookie allows other parties to gain access to this confidential information and compromises the security of the protocol.

8.2. Offering Confidentiality Protection for Access to Protected Resources

This specification can be used with and without Transport Layer Security (TLS).

Without TLS this protocol provides a mechanism for verifying the integrity of requests and responses, it provides no confidentiality protection. Consequently, eavesdroppers will have full access to request content and further messages exchanged between the client and the resource server. This could be problematic when data is exchanged that requires care, such as personal data.

When TLS is used then confidentiality can be ensured and with the use of the TLS channel binding feature it ensures that the TLS channel is cryptographically bound to the used MAC token. TLS in combination with channel bindings bound to the MAC token provide security superiour to the OAuth Bearer Token.

The use of TLS in combination with the MAC token is highly recommended to ensure the confidentiality of the user's data.

8.3. Authentication of Resource Servers

This protocol allows clients to verify the authenticity of resource servers in two ways:

1. The resource server demonstrates possession of the session key by computing a keyed message digest function over a number of HTTP fields in the response to the request from the client.
2. When TLS is used the resource server is authenticated as part of the TLS handshake.

8.4. Plaintext Storage of Credentials

The MAC key works in the same way passwords do in traditional authentication systems. In order to compute the keyed message digest, the client and the resource server must have access to the MAC key in plaintext form.

If an attacker were to gain access to these MAC keys - or worse, to the resource server's or the authorization server's database of all such MAC keys - he or she would be able to perform any action on behalf of any client.

It is therefore paramount to the security of the protocol that these session keys are protected from unauthorized access.

8.5. Entropy of Session Keys

Unless TLS is used between the client and the resource server, eavesdroppers will have full access to requests sent by the client. They will thus be able to mount offline brute-force attacks to recover the session key used to compute the keyed message digest. Authorization servers should be careful to generate fresh and unique session keys with sufficient entrophy to resist such attacks for at least the length of time that the session keys are valid.

For example, if a session key is valid for one day, authorization servers must ensure that it is not possible to mount a brute force attack that recovers the session key in less than one day. Of course, servers are urged to err on the side of caution, and use the longest session key reasonable.

It is equally important that the pseudo-random number generator (PRNG) used to generate these session keys be of sufficiently high quality. Many PRNG implementations generate number sequences that may appear to be random, but which nevertheless exhibit patterns, which make cryptanalysis easier. Implementers are advised to follow the guidance on random number generation in [RFC4086].

8.6. Denial of Service / Resource Exhaustion Attacks

This specification includes a number of features which may make resource exhaustion attacks against resource servers possible. For example, a resource server may need to need to consult backend databases and the authorization server to verify an incoming request including an access token before granting access to the protected resource.

An attacker may exploit this to perform a denial of service attack by sending a large number of invalid requests to the server. The computational overhead of verifying the keyed message digest alone is, however, not sufficient to mount a denial of service attack since keyed message digest functions belong to the computationally fastest cryptographic algorithms. The usage of TLS does, however, require additional computational capability to perform the asymmetric cryptographic operations. For a brief discussion about denial of service vulnerabilities of TLS please consult Appendix F.5 of RFC 5246 [RFC5246].

8.7. Timing Attacks

This specification makes use of HMACs, for which a signature verification involves comparing the received MAC string to the expected one. If the string comparison operator operates in observably different times depending on inputs, e.g. because it compares the strings character by character and returns a negative result as soon as two characters fail to match, then it may be possible to use this timing information to determine the expected MAC, character by character.

Implementers are encouraged to use fixed-time string comparators for MAC verification. This means that the comparison operation is not terminated once a mismatch is found.

8.8. CSRF Attacks

A Cross-Site Request Forgery attack occurs when a site, evil.com, initiates within the victim's browser the loading of a URL from or the posting of a form to a web site where a side-effect will occur, e.g. transfer of money, change of status message, etc. To prevent this kind of attack, web sites may use various techniques to determine that the originator of the request is indeed the site itself, rather than a third party. The classic approach is to include, in the set of URL parameters or form content, a nonce generated by the server and tied to the user's session, which indicates that only the server could have triggered the action.

Recently, the Origin HTTP header has been proposed and deployed in some browsers. This header indicates the scheme, host, and port of the originator of a request. Some web applications may use this Origin header as a defense against CSRF.

To keep this specification simple, HTTP headers are not part of the string to be MAC'ed. As a result, MAC authentication cannot defend against header spoofing, and a web site that uses the Host header to defend against CSRF attacks cannot use MAC authentication to defend against active network attackers. Sites that want the full protection of MAC Authentication should use traditional, cookie-tied CSRF defenses.

8.9. Protecting HTTP Header Fields

This specification provides flexibility for selectively protecting header fields and even the body of the message. At a minimum the following fields are included in the keyed message digest.

9. IANA Considerations

9.1. JSON Web Token Claims

This document adds the following claims to the JSON Web Token Claims registry established with [I-D.ietf-oauth-json-web-token]:

o Claim Name: "kid"
o Change Controller: IETF
o Specification Document(s): [[this document]]
o Claim Name: "mac_key"
o Change Controller: IETF
o Specification Document(s): [[this document]]

9.2. MAC Token Algorithm Registry

This specification establishes the MAC Token Algorithm registry.

Additional keyed message digest algorithms are registered on the advice of one or more Designated Experts (appointed by the IESG or their delegate), with a Specification Required (using terminology from [RFC5226]). However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests should be sent to the [TBD]@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for MAC Algorithm: example"). [[Note to RFC-EDITOR: The

name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: http-mac-ext-review.]]

Within at most 14 days of the request, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

Decisions (or lack thereof) made by the Designated Expert can be first appealed to Application Area Directors (contactable using app-ads@tools.ietf.org email address or directly by looking up their email addresses on http://www.iesg.org/ website) and, if the appellant is not satisfied with the response, to the full IESG (using the iesg@iesg.org mailing list).

IANA should only accept registry updates from the Designated Expert(s), and should direct all requests for registration to the review mailing list.

9.2.1. Registration Template

Algorithm name:

The name requested (e.g., "example").

Change controller:

For standards-track RFCs, state "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, e-mail address, home page URI) may also be included.

Specification document(s):

Reference to document that specifies the algorithm, preferably including a URI that can be used to retrieve a copy of the document. An indication of the relevant sections may also be included, but is not required.

9.2.2. Initial Registry Contents

The HTTP MAC authentication scheme algorithm registry's initial contents are:

- Algorithm name: hmac-sha-1
- Change controller: IETF
- Specification document(s): [[this document]]

- o Algorithm name: hmac-sha-256
- o Change controller: IETF
- o Specification document(s): [[this document]]

9.3. OAuth Access Token Type Registration

This specification registers the following access token type in the OAuth Access Token Type Registry.

9.3.1. The "mac" OAuth Access Token Type

Type name:

mac

Additional Token Endpoint Response Parameters:

secret, algorithm

HTTP Authentication Scheme(s):

MAC

Change controller:

IETF

Specification document(s):

[[this document]]

9.4. OAuth Parameters Registration

This specification registers the following parameters in the OAuth Parameters Registry established by [RFC6749].

9.4.1. The "mac_key" OAuth Parameter

Parameter name: mac_key
Parameter usage location: authorization response, token response
Change controller: IETF
Specification document(s): [[this document]]
Related information: None

9.4.2. The "mac_algorithm" OAuth Parameter

Parameter name: mac_algorithm
Parameter usage location: authorization response, token response

Change controller: IETF
Specification document(s): [[this document]]
Related information: None

9.4.3. The "kid" OAuth Parameter

Parameter name: kid
Parameter usage location: authorization response, token response
Change controller: IETF
Specification document(s): [[this document]]
Related information: None

10. Acknowledgments

This document is based on OAuth 1.0 and we would like to thank Eran Hammer-Lahav for his work on incorporating the ideas into OAuth 2.0. As part of this initial work the following persons provided feedback: Ben Adida, Adam Barth, Phil Hunt, Rasmus Lerdorf, James Manger, William Mills, Scott Renfro, Justin Richer, Toby White, Peter Wolanin, and Skylar Woodward

Further work in this document was done as part of OAuth working group conference calls late 2012/early 2013 and in design team conference calls February 2013. The following persons (in addition to the OAuth WG chairs, Hannes Tschofenig, and Derek Atkins) provided their input during these calls: Bill Mills, Justin Richer, Phil Hunt, Prateek Mishra, Mike Jones, George Fletcher, John Bradley, Tony Nadalin, Thomas Hardjono, Brian Campbell

11. References

11.1. Normative References

[I-D.ietf-httpbis-p1-messaging]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-22 (work in progress), February 2013.

[I-D.ietf-jose-json-web-encryption]
Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-08 (work in progress), December 2012.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token

(JWT)", draft-ietf-oauth-json-web-token-06 (work in progress), December 2012.

[I-D.richer-oauth-introspection]
Richer, J., "OAuth Token Introspection", draft-richer-oauth-introspection-03 (work in progress), February 2013.

[I-D.tschofenig-oauth-audience]
Tschofenig, H., "OAuth 2.0: Audience Information", draft-tschofenig-oauth-audience-00 (work in progress), February 2013.

[NIST-FIPS-180-3]
National Institute of Standards and Technology, "Secure Hash Standard (SHS). FIPS PUB 180-3, October 2008".

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

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

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

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

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[W3C.REC-html401-19991224] Hors, A., Raggett, D., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, <http://www.w3.org/TR/1999/REC-html401-19991224>.

11.2. Informative References

[I-D.tschofenig-oauth-security] Tschofenig, H. and P. Hunt, "OAuth 2.0 Security: Going Beyond Bearer Tokens", draft-tschofenig-oauth-security-01 (work in progress), December 2012.

Authors' Addresses

Justin Richer
The MITRE Corporation

Email: jricher@mitre.org

William Mills
Yahoo! Inc.

Phone:
Email: wmills@yahoo-inc.com

Hannes Tschofenig (editor)
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at