Network Working Group	E. Hammer-Lahav, Ed.
Internet-Draft	Yahoo!
Intended status: Standards Track	July 06, 2009
Expires: January 7, 2010	

The OAuth Protocol: Authentication
draft-ietf-oauth-authentication-01

Status of this Memo

Copyright Notice

Abstract

This document specifies the OAuth protocol authentication method. OAuth allows clients to access server resources on behalf of another party (such a different client or an end user). This document defines an HTTP authentication method which supports the ability to include two sets of credential with each request, one identifying the client and another identifying the resource owner on whose behalf the request is made.

Table of Contents

1. Introduction

The OAuth protocol provides a method for servers to allow third-party access to protected resources, without forcing their end users to share their credentials. This pattern is common among services that allow third-party developers to extend the service functionality, by building applications using an open API.
For example, a web user (resource owner) can grant a printing service (client) access to its private photos stored at a photo sharing service (server), without sharing its credentials with the printing service. Instead, the user authenticates directly with the photo sharing service and issue the printing service delegation-specific credentials.
OAuth introduces a third role to the traditional client-server authentication model: the resource owner. In the OAuth model, the client requests access to resources hosted by the server but not controlled by the client, but by the resource owner. In addition, OAuth allows the server to verify not only the resource owner's credentials, but also those of the client making the request.
In order for the client to access resources, it first has to obtain permission from the resource owner. This permission is expressed in the form of a token and matching shared-secret. The purpose of the token is to substitute the need for the resource owner to share its server credentials (usually a username and password pair) with the client. Unlike server credentials, tokens can be issued with a restricted scope and limited lifetime.
This specification consists of two parts. This document defines a method for making authenticated HTTP requests using two sets of credentials, one identifying the client making the request, and a second identifying the resource owner on whose behalf the request is being made.
[draft-ietf-oauth-web-delegation] (Hammer-Lahav, E., Ed., “The OAuth Protocol: Web Delegation,” .) defines a redirection-based user agent process for end users to authorize client access to their resources, by authenticating directly with the server and provisioning tokens to the client for use with the authentication method.

1.1. Terminology

TOC

client An HTTP client (per [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)) capable of making OAuth-authenticated requests.

server An HTTP server (per [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-

Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)) capable of accepting OAuth-authenticated requests.

protected resource An access-restricted resource (per [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)) which can be obtained from the server using an OAuth-authenticated request.

resource owner An entity capable of accessing and controlling protected resources by using credentials to authenticate with the server.

token An unique identifier issued by the server and used by the client to associate authenticated requests with the resource owner whose authorization is requested or has been obtained by the client. Tokens have a matching shared-secret that is used by the client to establish its ownership of the token, and its authority to represent the resource owner.

---

2. Notational Conventions

TOC

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] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).

---

3. Authenticated Requests

TOC

The HTTP authentication methods defined by [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.), enable clients to make authenticated HTTP requests. Clients using these methods gain access to protected resource by using their server credentials (typically a username and password pair), which allows the server to verify their authenticity. Using these methods for delegation requires the client to pretend it was the resource owner.

OAuth provides a method designed to include two sets of credentials with each request, one to identify the client, and another to identify the resource owner. Before a client can make authenticated requests on behalf of the resource owner, it must obtain a token authorized by the resource owner. [draft-ietf-oauth-web-delegation] (Hammer-Lahav, E., Ed., "The OAuth Protocol: Web Delegation," .) provides one such method in which the client can obtain a token authorized by the resource owner.

The client credentials take the form of a unique identifier, and an associated share-secret or RSA key pair. Prior to making authenticated requests, the client establishes a set of credentials with the server. The process and requirements for provisioning these are outside the scope of this specification. Implementers are urged to consider the security ramification of using client credentials, some of which are described in Section 11.7 (Secrecy of the Client Credentials).

Making authenticated requests requires prior knowledge of the server's configuration. OAuth provides multiple methods for including protocol parameters in requests (Section 7 (Parameter Transmission)), as well as multiple methods for the client to prove its rightful ownership of the credentials used (Section 6 (Signature)). The way in which clients discovery the required configuration is outside the scope of this specification.

---

4. Protocol Parameters

TOC

An OAuth-authenticated request includes several protocol parameters. Each parameter name begins with the oauth_ prefix, and the parameter names and values are case sensitive. Protocol parameters MUST NOT appear more than once per request. The parameters are:

oauth_consumer_key The identifier portion of the client credentials (equivalent to a username). The parameter name reflects a deprecated term (Consumer Key) used in previous revisions of the specification, and has been retained to maintain backward compatibility.

oauth_token The token value used to associate the request with the resource owner. If the request is not associated with a resource owner (no token), clients MAY omit the parameter.

oauth_signature_method The name of the signature method used by the client to sign the request, as defined in Section 6 (Signature).

oauth_signature The signature value as defined in Section 6 (Signature).

oauth_timestamp The timestamp value as defined in Section 5 (Nonce and Timestamp).

oauth_nonce The nonce value as defined in Section 5 (Nonce and Timestamp).

oauth_version

> The protocol version. If omitted, the protocol version defaults to 1.0.

Server-specific request parameters MUST NOT begin with the oauth_ prefix.

---

5. Nonce and Timestamp

TOC

Unless otherwise specified by the server, the timestamp is expressed in the number of seconds since January 1, 1970 00:00:00 GMT. The timestamp value MUST be a positive integer and MUST be equal or greater than the timestamp used in previous requests with the same client credentials and token credentials combination.
A nonce is a random string, uniquely generated to allows the server to verify that a request has never been made before and helps prevent replay attacks when requests are made over a non-secure channel. The nonce value MUST be unique across all requests with the same timestamp, client credentials, and token combinations.
To avoid the need to retain an infinite number of nonce values for future checks, servers MAY choose to restrict the time period after which a request with an old timestamp is rejected. Server applying such restriction SHOULD provide a way for the client to sync its clock with the server's clock.

---

6. Signature

TOC

OAuth-authenticated requests can have two sets of credentials included via the oauth_consumer_key parameter and the oauth_token parameter. In order for the server to verify the authenticity of the request and prevent unauthorized access, the client needs to prove it is the rightful owner of the credentials. This is accomplished using the shared-secret (or RSA key) part of each set of credentials.
OAuth provides three methods for the client to prove its rightful ownership of the credentials: HMAC–SHA1, RSA–SHA1, and PLAINTEXT. These methods are generally referred to as signature methods, even though PLAINTEXT does not involve a signature. In addition, RSA–SHA1 utilizes an RSA key instead of the shared-secrets associated with the client credentials.
OAuth does not mandate a particular signature method, as each implementation can have its own unique requirements. Servers are free to implement and document their own custom methods. Recommending any particular method is beyond the scope of this specification.

The client declares which signature method is used via the oauth_signature_method parameter. It then generates a signature (or a sting of an equivalent value), and includes it in the oauth_signature parameter. The server verifies the signature as specified for each method.
The signature process does not change the request or its parameter, with the exception of the oauth_signature parameter.

6.1. Signature Base String

TOC

The signature base string is a consistent, reproducible concatenation of several request elements into a single string. The string is used as an input to the HMAC-SHA1 and RSA-SHA1 signature methods, or potential future extension.
The signature base string does not cover the entire HTTP request. Most notably, it does not include the entity-body in most requests, nor does it include most HTTP entity-headers. The importance of the signature base string scope is that the authenticity of the excluded components cannot be verified using the signature.

6.1.1. Collect Request Parameters

TOC

The signature base string includes a specific set of request parameters. In order for the parameter to be included in the signature base string, they MUST be used in their unencoded form.
For example, the URI:

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

contains the following raw-form parameters:

Name	Value
b5	=%3D
a3	a
c@	
a2	r b
c2	
a3	2q

Note that the value of b5 is =%3D and not ==. Both c@ and c2 have empty values.

The request parameters, which include both protocol parameters and request-specific parameters, are extracted and restored to their original unencoded form, from the following sources:

* The OAuth HTTP Authorization header (Authorization Header). The realm parameter MUST be excluded if present.
* The HTTP request entity-body, but only if:
  - The entity-body is single-part.
  - The entity-body follows the encoding requirements of the application/x-www-form-urlencoded content-type as defined by [W3C.REC-html40-19980424] (Hors, A., Jacobs, I., and D. Raggett, “HTML 4.0 Specification,” April 1998.).
  - The HTTP request entity-header includes the Content-Type header set to application/x-www-form-urlencoded.
* The query component of the HTTP request URI as defined by [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) section 3.

The oauth_signature parameter MUST be excluded if present.
In many cases, clients have direct access to the parameters in their original, unencoded form. In such cases, clients SHOULD use the unencoded values instead of extracting them. This option is not available for servers when validating incoming requests. Even though the parameters are encoded again in the process, they are decoded because each of the three sources uses a different encoding algorithm.
The output of this step is a list of unencoded parameter name / value pairs.

6.1.2. Normalize Request Parameters

TOC

The parameter collected in Section 6.1.1 (Collect Request Parameters) are normalized into a single string as follows:

1. First, the name and value of each parameter are encoded (Percent Encoding).
2. The parameters are sorted by name, using lexicographical byte value ordering. If two or more parameters share the same name, they are sorted by their value.

3. The name of each parameter is concatenated to its corresponding value using an = character (ASCII code 61) as separator, even if the value is empty.

4. The sorted name / value pairs are concatenated together into a single string by using an & character (ASCII code 38) as separator.

For example, the list of parameters from the previous section would be normalized as follows:

Encoded:

Name	Value
b5	%3D%253D
a3	a
c%40	
a2	r%20b
c2	
a3	2q

Sorted:

Name	Value
a2	r%20b
a3	2q
a3	a
b5	%3D%253D
c%40	
c2	

Concatenated Pairs:

Name=Value
a2=r%20b
a3=2q
a3=a
b5=%3D%253D
c%40=
c2=

And concatenated together into a single string:

```
a2=r%20b&a3=2q&a3=a&b5=%3D%253D&c%40=&c2=
```

6.1.3. Construct Base String URI

TOC

The signature base string incorporates the scheme, authority, and path of the request URI as defined by [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) section 3. The request URI query component is included through the normalized parameters string (Normalize Request Parameters), and the fragment component is excluded.
This is done by constructing a base string URI representing the request without the query or fragment components. The base string URI is constructed as follows:

1. The scheme and host MUST be in lowercase.

2. The host and port values MUST match the content of the HTTP request Host header, if present. If the Host header is not present, the client MUST use the hostname and port used to make the request. Servers SHOULD remove potential ambiguity in such cases by specifying the expected host value.

3. The port MUST be included if it is not the default port for the scheme, and MUST be excluded if it is the default. Specifically, the port MUST be excluded when an http request uses port 80 or when an https request uses port 443. All other non-default port numbers MUST be included.

4. If the URI includes an empty path, it MUST be included as /.

For example:

The request URI	Is included in base string as
HTTP://EXAMPLE.com:80/r/x?id=123	http://example.com/r/x
https://example.net:8080?q=1#top	https://example.net:8080/

TOC

6.1.4. Concatenate Base String Elements

Finally, the signature base string is put together by concatenating its elements together. The elements MUST be concatenated in the following order:

1. The HTTP request method in uppercase. For example: HEAD, GET, POST, etc. If the request uses a custom HTTP method, it MUST be encoded (Percent Encoding).
2. An & character (ASCII code 38).
3. The base string URI from Section 6.1.3 (Construct Base String URI), after being encoded (Percent Encoding).
4. An & character (ASCII code 38).
5. The normalized request parameters string from Section 6.1.2 (Normalize Request Parameters), after being encoded (Percent Encoding).

6.2. HMAC-SHA1

TOC

The HMAC-SHA1 signature method uses the HMAC-SHA1 signature algorithm as defined in [RFC2104] (Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” February 1997.):

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

The HMAC-SHA1 function variables are used in following way:

text is set to the value of the signature base string from Section 6.1.4 (Concatenate Base String Elements).

key is set to the concatenated values of:

1. The client shared-secret, after being encoded (Percent Encoding).
2. An & character (ASCII code 38), which MUST be included even when either secret is empty.
3. The token shared-secret, after being encoded (Percent Encoding).

digest is used to set the value of the oauth_signature protocol parameter, after the result octet string is base64-encoded per

> [RFC2045] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” November 1996.) section 6.8.

6.3. RSA-SHA1

TOC

The RSA-SHA1 signature method uses the RSASSA-PKCS1-v1_5 signature algorithm as defined in [RFC3447] (Jonsson, J. and B. Kaliski, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” February 2003.) section 8.2 (also known as PKCS#1), using SHA-1 as the hash function for EMSA-PKCS1-v1_5. To use this method, the client MUST have established client credentials with the server which included its RSA public key (in a manner which is beyond the scope of this specification).
The signature base string is signed using the client's RSA private key per [RFC3447] (Jonsson, J. and B. Kaliski, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” February 2003.) section 8.2.1:

```
S = RSASSA-PKCS1-V1_5-SIGN (K, M)
```

Where:

> **K** is set to the client's RSA private key,
>
> **M** is set to the value of the signature base string from Section 6.1.4 (Concatenate Base String Elements), and
>
> **S** is the result signature used to set the value of the oauth_signature protocol parameter, after the result octet string is base64-encoded per [RFC2045] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” November 1996.) section 6.8.

The server verifies the signature per [RFC3447] (Jonsson, J. and B. Kaliski, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” February 2003.) section 8.2.2:

```
RSASSA-PKCS1-V1_5-VERIFY ((n, e), M, S)
```

Where:

> **(n, e)** is set to the client's RSA public key,
>
> **M** is set to the value of the signature base string from Section 6.1.4 (Concatenate Base String Elements), and

S

> is set to the octet string value of the oauth_signature protocol parameter received from the client.

6.4. PLAINTEXT

TOC

The PLAINTEXT method does not employ a signature algorithm and does not provide any security as it transmits secrets in the clear. It SHOULD only be used with a transport-layer mechanisms such as TLS or SSL. It does not use the signature base string.
The oauth_signature protocol parameter is set to the concatenated value of:

1. The client shared-secret, after being encoded (Percent Encoding).
2. An & character (ASCII code 38), which MUST be included even when either secret is empty.
3. The token shared-secret, after being encoded (Percent Encoding).

---

7. Parameter Transmission

TOC

When making an OAuth-authenticated request, protocol parameters SHALL be included in the request using one and only one of the following locations, listed in order of decreasing preference:

1. The HTTP Authorization header as described in Section 7.1 (Authorization Header).
2. The HTTP request entity-body as described in Section 7.2 (Form-Encoded Body).
3. The HTTP request URI query as described in Section 7.3 (Request URI Query).

In addition to these three methods, future extensions may provide other methods for including protocol parameters in the request.

TOC

7.1. Authorization Header

Protocol parameters can be transmitted using the HTTP Authorization header as defined by [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.) with the auth-scheme name set to OAuth (case-insensitive). For example:

```
Authorization: OAuth realm="http://server.example.com/",
   oauth_consumer_key="0685bd9184jfhq22",
   oauth_token="ad180jjd733klru7",
   oauth_signature_method="HMAC-SHA1",
   oauth_signature="wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D",
   oauth_timestamp="137131200",
   oauth_nonce="4572616e48616d65724c61686176",
   oauth_version="1.0"
```

Protocol parameters SHALL be included in the Authorization header as follows:

1. Parameter names and values are encoded per Parameter Encoding (Percent Encoding).

2. Each parameter's name is immediately followed by an = character (ASCII code 61), a " character (ASCII code 34), the parameter value (MAY be empty), and another " character (ASCII code 34).

3. Parameters are separated by a , character (ASCII code 44) and OPTIONAL linear whitespace per [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.).

4. The OPTIONAL realm parameter MAY be added and interpreted per [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.), section 1.2.

Servers MAY indicate their support for the OAuth auth-scheme by returning the HTTP WWW-Authenticate response header upon client requests for protected resources. As per [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.) such a response MAY include additional HTTP WWW-Authenticate headers:
For example:

```
WWW-Authenticate: OAuth realm="http://server.example.com/"
```

The realm parameter defines a protection realm per [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.), section 1.2.

7.2. Form-Encoded Body

TOC

Protocol parameters can be transmitted in the HTTP request entity-body, but only if the following REQUIRED conditions are met:

- The entity-body is single-part.
- The entity-body follows the encoding requirements of the application/x-www-form-urlencoded content-type as defined by [W3C.REC-html40-19980424] (Hors, A., Jacobs, I., and D. Raggett, “HTML 4.0 Specification,” April 1998.).
- The HTTP request entity-header includes the Content-Type header set to application/x-www-form-urlencoded.

For example (line breaks are for display purposes only):

```
oauth_consumer_key=0685bd9184jfhq22&oauth_token=ad180jjd733klr
u7&oauth_signature_method=HMAC-SHA1&oauth_signature=wOJIO9A2W5
mFwDgiDvZbTSMK%2FPY%3D&oauth_timestamp=137131200&oauth_nonce=4
572616e48616d6d65724c61686176&oauth_version=1.0
```

The entity-body MAY include other request-specific parameters, in which case, the protocol parameters SHOULD be appended following the request-specific parameters, properly separated by an & character (ASCII code 38).

7.3. Request URI Query

TOC

Protocol parameters can be transmitted by being added to the HTTP request URI as a query parameter as defined by [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) section 3.
For example (line breaks are for display purposes only):

```
GET /example/path?oauth_consumer_key=0685bd9184jfhq22&
oauth_token=ad180jjd733klru7&oauth_signature_method=HM
AC-SHA1&oauth_signature=wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%
3D&oauth_timestamp=137131200&oauth_nonce=4572616e48616
d6d65724c61686176&oauth_version=1.0 HTTP/1.1
```

The request URI MAY include other request-specific query parameters, in which case, the protocol parameters SHOULD be appended following the request-specific parameters, properly separated by an & character (ASCII code 38).

---

8. Server Response

TOC

Servers receiving an authenticated request MUST:

* Recalculate the request signature independently and compare it to the value received from the client.
* Ensure that the nonce / timestamp / token combination has not been used before, and MAY reject requests with stale timestamps.
* If a token is present, verify the scope and status of the client authorization by using the token, and MAY choose to restrict token usage to the client to which it was issued.
* Ensure that the protocol version used is 1.0.

If the request fails verification, the server SHOULD respond with the appropriate HTTP response status code. The server MAY include further details about why the request was rejected in the response body. The following status codes SHOULD be used:

* 400 (Bad Request)
  - Unsupported parameters
  - Unsupported signature method
  - Missing parameters
  - Duplicated protocol parameters
* 401 (Unauthorized)
  - Invalid client credentials
  - Invalid or expired token

-Invalid signature

-Invalid or used nonce

9. Percent Encoding

TOC

OAuth uses the following percent-encoding rules:

1. Text values are first encoded as UTF-8 octets per [RFC3629] (Yergeau, F., “UTF-8, a transformation format of ISO 10646,” November 2003.) if they are not already. This does not include binary values which are not intended for human consumption.

2. The values are then escaped using the [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) percent-encoding (%XX) mechanism as follows:

   *Characters in the unreserved character set as defined by [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” January 2005.) section 2.3 (ALPHA, DIGIT, "-", ".", "_", "~") MUST NOT be encoded.

   *All other characters MUST be encoded.

   *The two hexadecimal characters use to represent encoded characters MUST be upper case.

10. IANA Considerations

TOC

This memo includes no request to IANA.

11. Security Considerations

TOC

As stated in [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.), 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.

11.1. RSA-SHA1 Signature Method

TOC

When used with RSA−SHA1 signatures, the OAuth protocol does not use the token shared-secret, or any provisioned client shared-secret. This means the protocol relies completely on the secrecy of the private key used by the client to sign requests.

11.2. PLAINTEXT Signature Method

TOC

When used with the PLAINTEXT method, the protocol makes no attempts to protect credentials from eavesdroppers or man-in-the-middle attacks. The PLAINTEXT method is only intended to be used in conjunction with a transport-layer security mechanism such as TLS or SSL which does provide such protection.

11.3. Confidentiality of Requests

TOC

While OAuth provides a mechanism for verifying the integrity of requests, it provides no guarantee of request confidentiality. Unless further precautions are taken, eavesdroppers will have full access to request content. Servers should carefully consider the kinds of data likely to be sent as part of such requests, and should employ transport-layer security mechanisms to protect sensitive resources.

11.4. Spoofing by Counterfeit Servers

TOC

OAuth makes no attempt to verify the authenticity of the server. A hostile party could take advantage of this by intercepting the client's requests and returning misleading or otherwise incorrect responses. Service providers should consider such attacks when developing services based on OAuth, and should require transport-layer security for any requests where the authenticity of the server or of request responses is an issue.

11.5. Proxying and Caching of Authenticated Content

TOC

The HTTP Authorization scheme (Authorization Header) is optional. However, [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.) relies on the Authorization and WWW-Authenticate headers to distinguish authenticated content so that it can be protected. Proxies and caches, in particular, may fail to adequately protect requests not using these headers.

For example, private authenticated content may be stored in (and thus retrievable from) publicly-accessible caches. Servers not using the HTTP Authorization header (Authorization Header) should take care to use other mechanisms, such as the Cache-Control header, to ensure that authenticated content is protected.

11.6. Plaintext Storage of Credentials

TOC

The client shared-secret and token shared-secret function the same way passwords do in traditional authentication systems. In order to compute the signatures used in methods other than RSA-SHA1, the server must have access to these secrets in plaintext form. This is in contrast, for example, to modern operating systems, which store only a one-way hash of user credentials.

If an attacker were to gain access to these secrets - or worse, to the server's database of all such secrets - he or she would be able to perform any action on behalf of any resource owner. Accordingly, it is critical that servers protect these secrets from unauthorized access.

11.7. Secrecy of the Client Credentials

TOC

In many cases, the client application will be under the control of potentially untrusted parties. For example, if the client is a freely available desktop application, an attacker may be able to download a copy for analysis. In such cases, attackers will be able to recover the client credentials.

Accordingly, servers should not use the client credentials alone to verify the identity of the client. Where possible, other factors such as IP address should be used as well.

TOC

11.8. Cryptographic Attacks

SHA-1, the hash algorithm used in HMAC−SHA1 signatures, has been shown (De Canniere, C. and C. Rechberger, “Finding SHA-1 Characteristics: General Results and Applications,” .) [SHA1-CHARACTERISTICS] to have a number of cryptographic weaknesses that significantly reduce its resistance to collision attacks. Practically speaking, these weaknesses are difficult to exploit, and by themselves do not pose a significant risk to users of OAuth. They may, however, make more efficient attacks possible, and NIST has announced (National Institute of Standards and Technology, NIST., “NIST Brief Comments on Recent Cryptanalytic Attacks on Secure Hashing Functions and the Continued Security Provided by SHA-1, August, 2004.,” .) [SHA-COMMENTS] that it will phase out use of SHA-1 by 2010. Servers should take this into account when considering whether SHA-1 provides an adequate level of security for their applications.

11.9. Signature Base String Limitations

TOC

The signature base string has been designed to support the signature methods defined in this specification. When designing additional signature methods, the signature base string should be evaluated to ensure compatibility with the algorithms used.
Since the signature base string does not cover the entire HTTP request, such as most request entity-body, most entity-headers, and the order in which parameters are sent, servers should employ additional mechanisms to protect such elements.

Appendix A. Examples

TOC

[[TODO]]

Appendix B. Acknowledgments

TOC

This specification is directly based on the [OAuth Core 1.0 Revision A] (OAuth, OCW., “OAuth Core 1.0,” .) community specification which was the product of the OAuth community. OAuth was modeled after existing proprietary protocols and best practices that have been independently implemented by various web sites. This specification was orignially authored by: Mark Atwood, Dirk Balfanz, Darren Bounds, Richard M. Conlan, Blaine Cook, Leah Culver, Breno de Medeiros, Brian Eaton,

Kellan Elliott-McCrea, Larry Halff, Eran Hammer-Lahav, Ben Laurie, Chris Messina, John Panzer, Sam Quigley, David Recordon, Eran Sandler, Jonathan Sergent, Todd Sieling, Brian Slesinsky, and Andy Smith.

Appendix C. Document History

TOC

[[To be removed by the RFC editor before publication as an RFC.]]
-01

- Moved all subsection from section 3 to the document root.
- Synced acknoledgments section with web-delegation draft.

-00

- Transitioned from the individual submission draft-hammer-oauth-02 to working group draft.
- Split draft-hammer-oauth-02 into two drafts, one dealing with authentication (this draft) and another dealing with web delegation draft-ietf-oauth-web-delegation.

---

12. References

TOC

12.1. Normative References

TOC

[RFC2045]	Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” RFC 2045, November 1996 (TXT).
[RFC2104]	Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104, February 1997 (TXT).
[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[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 (TXT, PS, PDF, HTML, XML).

[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 (TXT, HTML, XML).
[RFC3447]	Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," RFC 3447, February 2003 (TXT).
[RFC3629]	Yergeau, F., "UTF-8, a transformation format of ISO 10646," STD 63, RFC 3629, November 2003 (TXT).
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," STD 66, RFC 3986, January 2005 (TXT, HTML, XML).
[W3C.REC-html40-19980424]	Hors, A., Jacobs, I., and D. Raggett, "HTML 4.0 Specification," World Wide Web Consortium Recommendation REC-html40-19980424, April 1998 (HTML).
[draft-ietf-oauth-web-delegation]	Hammer-Lahav, E., Ed., "The OAuth Protocol: Web Delegation."

12.2. Informative References

TOC

[OAuth Core 1.0 Revision A]	OAuth, OCW., "OAuth Core 1.0."
[SHA-COMMENTS]	National Institute of Standards and Technology, NIST., "NIST Brief Comments on Recent Cryptanalytic Attacks on Secure Hashing Functions and the Continued Security Provided by SHA-1, August, 2004.."
[SHA1-CHARACTERISTICS]	De Canniere, C. and C. Rechberger, "Finding SHA-1 Characteristics: General Results and Applications."

Author's Address

TOC

	Eran Hammer-Lahav (editor)
	Yahoo!
Email:	eran@hueniverse.com
URI:	http://hueniverse.com