Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: February 24, 2011

Y. Oiwa
H. Watanabe
H. Takagi
RCIS, AIST
Y. Ioku
Yahoo! Japan
T. Hayashi
Lepidum

August 23, 2010

Mutual Authentication Protocol for HTTP draft-oiwa-http-mutualauth-07

Abstract

This document specifies a mutual authentication method for Hyper-Text Transport Protocol (HTTP). This method provides true mutual authentication between an HTTP client and an HTTP server using password-based authentication. Unlike the Basic and Digest authentication methods, the Mutual authentication method specified in this document assures the user that the server truly knows the user's encrypted password. This prevents common phishing attacks: a phishing attacker controlling a fake website cannot convince a user that he authenticated to the genuine website. Furthermore, even when a user authenticates to an illegitimate server, the server cannot gain any information about the user's password. The Mutual authentication method is designed as an extension to the HTTP protocol, and is intended to replace existing authentication methods used in HTTP (the Basic method, Digest method, and authentication using HTML forms).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 24, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
   1.1. Terminology
   1.2. Document Structure Overview
2. Protocol Overview
   2.1. Messages
   2.2. Typical Flows of the protocol
   2.3. Alternative flows
3. Message Syntax
   3.1. Tokens and Extensive-tokens
   3.2. Numbers
   3.3. Strings
4. Messages
   4.1. 401-B0
   4.2. 401-B0-stale
   4.3. req-A1
   4.4. 401-B1
   4.5. req-A3
   4.6. 200-B4
   4.7. 200-Optional-B0
5. Authentication Realms
   5.1. Resolving ambiguities
6. Session Management
7. Validation Methods
8. Decision procedure for the client
9. Decision procedure for the server
10. Authentication-Control header
   10.1. Location-when-unauthenticated field
   10.2. Location-when-logout field
   10.3. Logout-timeout
11. Authentication Algorithms
   11.1. Support functions and notations
   11.2. Common functions for both settings
   11.3. Functions for discrete-logarithm settings
   11.4. Functions for elliptic-curve settings
12. Methods to extend this protocol
13. IANA Considerations
14. Security Considerations
   14.1. Security Properties
   14.2. Denial-of-service attacks to servers
   14.3. Implementation Considerations
   14.4. Usage Considerations

15. Notice on intellectual properties
16. References
  16.1. Normative References
  16.2. Informative References
Appendix A. (Informative) Generic syntax of headers
Appendix B. (Informative) Group parameters for discrete-logarithm based algorithms
Appendix C. (Informative) Derived numerical values
Appendix D. (Informative) Draft Remarks from the Authors
Appendix E. (Informative) Draft Change Log
  E.1. Changes in revision 07
  E.2. Changes in revision 06
  E.3. Changes in revision 05
  E.4. Changes in revision 04
  E.5. Changes in revision 03
  E.6. Changes in revision 02
§ Authors' Addresses

1. Introduction

This document specifies a mutual authentication method for Hyper-Text Transport Protocl (HTTP). The method, called as "Mutual Authentication Protocol" in this document, provides provides true mutual authentication between an HTTP client and an HTTP server, using just a simple password as a credential.

Currently available methods for authentication in HTTP and Web system have several deficiencies. Basic authentication method [RFC2617] sends a plaintext password to a server without any protections; Digest method uses a hash function which suffers from simple dictionary-based off-line attacks, and people begins to think it obsolete.

The authentication method proposed in this document solves these problems, substitutes these existing methods and serves as a long-term solution of Web authentications security. it has the following main characteristics:

- It provides "true" mutual authentication: as well as assuring the server that the user knows the password, it also assures the user that the server truly knows the user's encrypted password at the same time. It makes impossible for fake website owners to persuade users that he authenticated to the original websites.
- It uses only a password as a user's credential: unlike public-key-based security algorithms, the method does not rely on secret keys or other cryptographic data which have to be stored inside users' computers. The method can be used almost as a drop-in replacement to the current authentication methods like Basic or Digest, while ensuring much stronger security.
- It is secure: when the server failed to authentication user, the protocol will not reveal any bit of user's password.

By using the proposed method, users can discriminate between true and fake Web servers using their own passwords. Even when a user inputs his/her password to a fake website owned by illegitimate phishers, the user will certainly notices that the authentication has failed. Phishers cannot make such authentication attempt succeed, even if they forward received data from a user to the legitimate server or vice versa. Users can input sensitive data to the web forms after confirming that the mutual authentication has succeeded, without fear of phishing attacks.

The document also proposes several extensions to the current HTTP authentication framework, to replace current widely-used form-based Web authentication. Nowadays, majority of the Web sites on the Internet use custom application-layer authentication implementations using Web forms. Reasons of these may vary, but many people consider that the current HTTP Basic (and Digest, too) authentication method does not have functionality (including a good-feeling user interfaces) enough for supporting realistic Web-based applications. However, the method is very weak against phishing attacks, because the whole behavior of the authentication is controlled from the servers' side. To overcome this problem, we need to "modernize" the HTTP authentication framework so that better client-controlled secure methods can be used well with Web applications. The extensions proposed in this document include:

- multi-host single authentication within an Internet domain (Section 5),
- non-mandatory, optional authentication on HTTP (Section 4.7),
- log out from both server and client side (Section 10), and
- finer control for redirection depending on authentication status (Section 10).

1.1. 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 [RFC2119].

Terms "encouraged" and "advised" are used for suggestions which do not constitute "SHOULD"-level requirements. People MAY freely choose not to include the suggested items regarding [RFC2119], but following the suggestion will be a best practice; it will improve security, interoperability or operation performance.

This document distinguishes the terms "client" and "user" in the following way: A "client" is an entity understanding and talking HTTP and the specified authentication protocol, usually a computer software; on the contrary, a "user" is a (usually natural) person who want to access data resources using "a client".

The term "natural numbers" means non-negative integers (including zero) throughout this document.

1.2. Document Structure Overview

The whole document is organized as follows:

- Section 2 gives an overview presentation of the protocol design.
- The Sections from 3 to 9 define a general framework of the Mutual authentication protocol. This framework is independent from specific cryptographic primitives.
- Section 10 defines an optional extension to the the generic HTTP authentication framework, which is useful mostly to control the Web browser behavior of the authentication.
- Section 11 defines a few specific cryptographic algorithms to be used with this authentication framework.
- The sections after that contain general normative and informative information about the protocol.
- Appendices contain some information which may help developers to implement the protocol.

2. Protocol Overview

The protocol, as a whole, is designed as a natural extension to the HTTP protocol [RFC2616] using a framework defined in [RFC2617]. Internally, the server and the client will first perform a cryptographic key exchange, using the secret password as a "tweak" to the exchange. The key-exchange will only succeed when the secrets used by the both peer are correctly related (i.e. generated from the same password). Then the both peer will verify the authentication results by checking the sharing of the exchanged key. This section describes the brief image of the protocol and the exchanged messages.

2.1. Messages

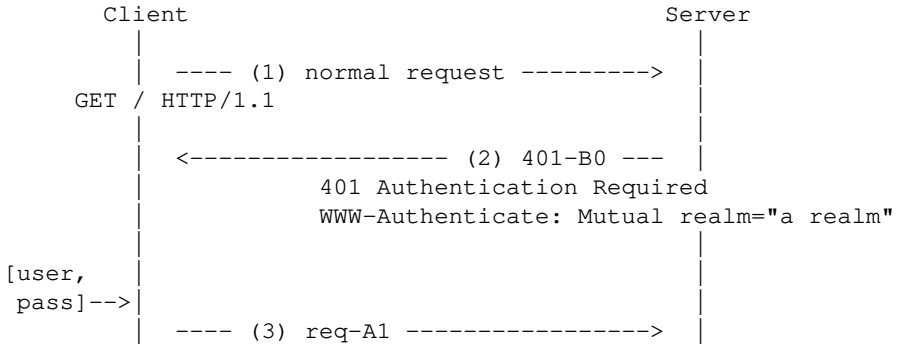
The authentication protocol uses seven kinds of messages to perform mutual authentication. These messages have a specific names within this specification.

- Authentication request messages: used by the servers to request clients to start mutual authentication.
  - 401-B0 message: a general message for start authentication protocol. It is also used as a message indicating an authentication failure.
  - 200-Optional-B0 message: a variant of 401-B0 message indicating that an authentication is not mandatory.
  - 401-B0-stale message: a message indicating that it has to start a new authentication trial.
- Authenticated key exchange messages: used by both peers to perform authentication and shares a cryptographic secret.
  - req-A1 message: a message sent from the client.
  - 401-B1 message: a message sent from the server as a response to req-A1 message.
- Authentication verification messages: used by both peers to verify authentication results.
  - req-A3 message: a message used by the client, requesting that the server authenticates and authorizes the client.
  - 200-B4 message: a successful response used by the server, also asserting that the server is authentic to the client at the same time.

In addition to above, either a request or a response without any HTTP headers related to this specification will be hereafter called a "normal request" or a "normal response", respectively.

2.2. Typical Flows of the protocol

In the typical case, a first client access to a resource protected by the Mutual authentication will follow the following protocol sequence.



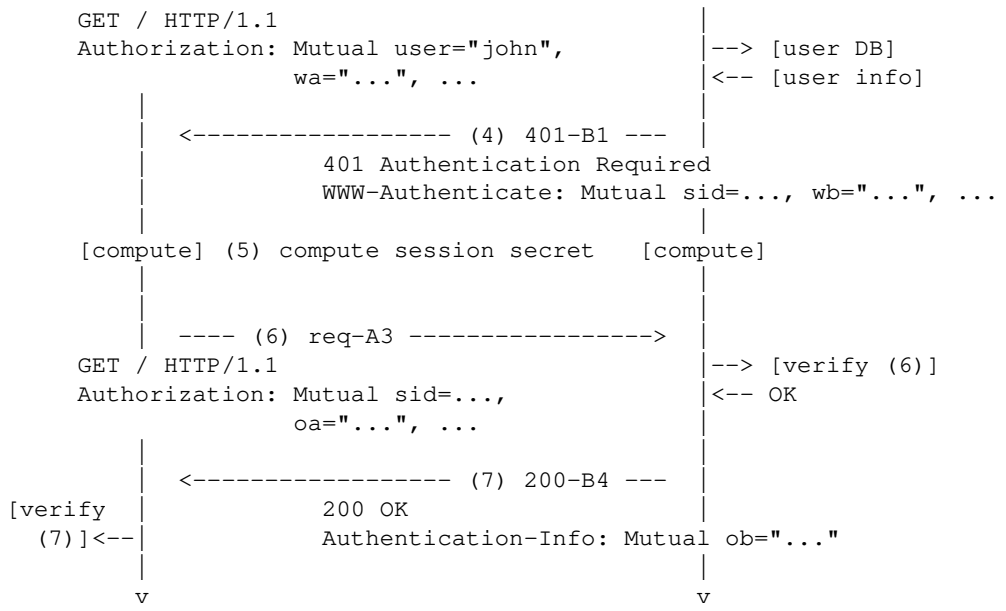


Figure 1: Typical communication flow on the first access to resource

---

- As usual in general HTTP protocol design, a client will first request a resource without any authentication attempt (1). If the requested resource is protected by the Mutual authentication, The server will respond with a message requesting authentication (401-B0) (2).
- The client processes the body of the message, and wait a user for inputing a user name and a password. If the user name and a password is available, The client will send a message with authenticated key exchange (req-A1) to start authentication (3).
- If the server has received a req-A1 message, The server looks up its user database for the user's authentication information. Then the server creates a new session identifier (sid) which will be used to identify sets of following messages, and responds back a message with a server-side authenticated key exchange value(401-B1) (4).
- At this point (5), both peers calculate a shared "session secret" using the exchanged values in key exchange messages. Only when both the server and the client have used secret credentials generated from the same password, the session secret values will match. This session secret will be used for the actual access authentication after this point.
- The client will send a request with a client-side authentication challenge (req-A3) (6), generated from the client-own session secret. The server will check the validity of the challenge using its own session secret.
- If the challenge from the client was correct, it means that the client certainly owns a credential based on the expected password (i.e. the client authentication succeeded.) The server will respond with a successful message (200-B4) (7). On the contrary to the usual one-way authentication (e.g. HTTP Basic authentication or POP APOP authentication), This message also contains a server-side authentication challenge.
  When the client's challenge was incorrect (e.g. because the user-supplied password was incorrect), the server will respond with the 401-B0 message (used in (2)) instead.
- The client MUST first check validity of the server-side authentication challenge contained in the message (7). If the challenge was equal to the expected value, the server authentication succeeded.
  If it is not the value expected, or if the message does not contain authentication challenge value, it means that the mutual authentication has been broken for some unexpected reasons. The client

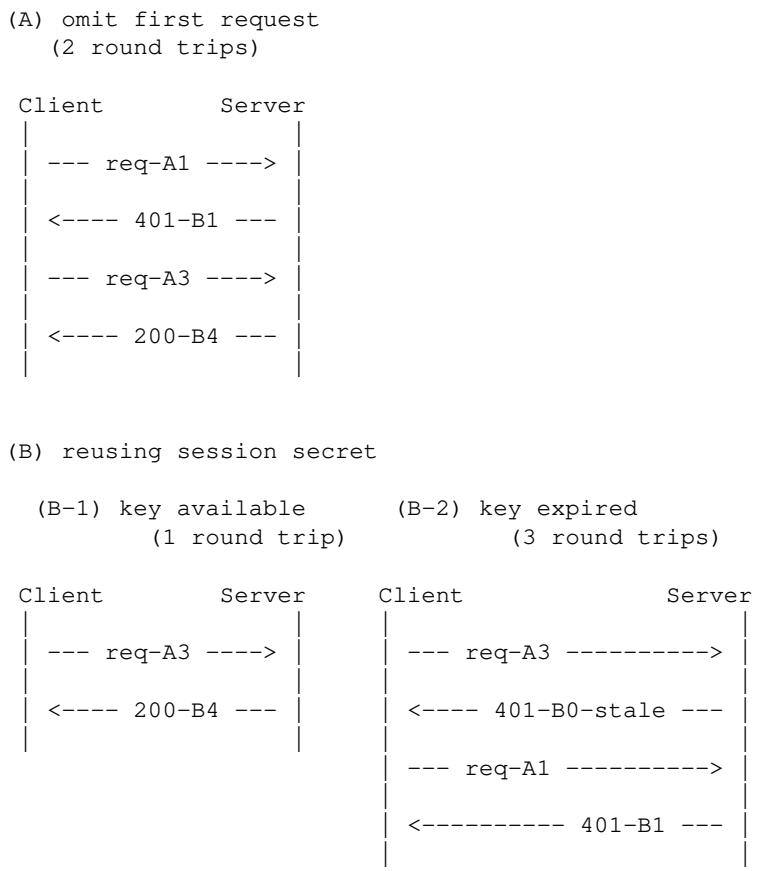
MUST NOT process any body and header values contained in this case. (Note: This case should not happen between a correctly-implemented server and a client.)

2.3. Alternative flows

As shown above, the typical flow for first authenticated request requires three request-response pairs. To reduce the protocol overhead, the protocol enables several short-cut flows which requires fewer messages.

- (case A) If the client knows that the resource is likely to require the authentication, the client MAY omit first unauthenticated request (1) and send req-A1 message immediately. This will reduce one round-trip of messages.
- (case B) If both the client and the server previously shared a session secret associated with a valid session identifier (sid), the client MAY directly send a req-A3 message using existing sid and corresponding session secret. This will further reduce one round-trip of the messages.
  In such cases, the server MAY have been thrown out the corresponding sessions from the session table. In this case, the server will send a 401-B0-stale message as a response to req-A3 message, indicating a new key exchange is required. The client SHOULD retry from constructing a req-A1 message in this case.

Figure 2 depicts the shortcut flows described above. Under appropriate setting and implementations, most of the requests to resources are expected to meet both criteria, and thus only one round-trip of request/response will be required for most cases.



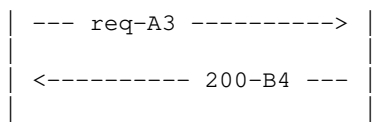


Figure 2: Several alternative flows on the protocol

For more details, see Section 8 and Section 9.

3. Message Syntax

The Mutual authentication protocol uses five headers: WWW-Authenticate (in responses with status code 401), Optional-WWW-Authenticate (in responses with non-401 status codes), Authentication-Control (in responses), Authorization (in requests), and Authentication-Info (in responses other than 401 status). These headers follow a common framework of the one described in [RFC2617] [Editorial Note: to be httpbis-p7]. The detailed syntax definitions for these headers are contained in Section 4.

These headers use some common syntax elements described in Figure 3. The syntax is denoted in the augmented BNF syntax defined in [RFC5234].

```
auth-scheme      = "Mutual"             ; see HTTP for other values
extension-field  = extension-token "=" value
token            = 1*(%x30-39 / %x41-5A / %x61-7A / "-" / "_")
extensive-token  = token / extension-token
extension-token  = "-" token 1*("." token)
value            = extensive-token / integer
                 / hex-fixed-number
                 / base64-fixed-number / string
integer          = "0" / (%x31-39 *%x30-39)      ; no leading zeros
hex-fixed-number = 1*(%x30-39 / %x41-46 / %x61-66)
base64-fixed-number = string
string           = %x22 *(%x20-21 / %x23-5B / %x5D-FF
                         / %x5C.22 / "\\") %x22
spaces           = 1*(" " / %x09)
```

Figure 3: the BNF syntax for the common elements used in the protocol

3.1. Tokens and Extensive-tokens

The tokens are case insensitive; Senders should SHOULD send these in lower-case, and receivers MUST accept both upper- and lower-cases. When tokens are used as (partial) inputs to any hash or other mathematical functions, it MUST always be used in lower-case. All hexadecimal numbers are also case-insensitive, and SHOULD be sent in lower-case.

Extensive-tokens are used in this protocol where the set of acceptable tokens may include non-standard extensions. Any non-standard extensions of this protocol MUST use the extension-tokens of format "-<token>.<domain-name>", where domain-name is a validly registered (sub-)domain name on the Internet owned by the party who defines extensions.

3.2. Numbers

The syntax definition of integers only allows representations which do not contain extra leading zeros.

The numbers represented as a hex-fixed-number MUST have even number of characters (i.e. multiple of eight bits). When these are generated from cryptographic values, those SHOULD have their "natural length": if these are generated from a hash function, these lengths SHOULD correspond to the hash size; if these are representing elements of a mathematical set (or group), its lengths SHOULD be the shortest which can represent all elements in the set. See Appendix C for information about the length of the fields used in this specification. Session-identifiers and other non-cryptographically generated values are represented in any (even) length determined by the side who generates it first, and the same length SHALL be used throughout the whole communications by both peers.

Numbers represented as base64-fixed-number SHALL be generated as follows: first, the number is converted to a big-endian octet-string representation. The length of the representation is determined in the same way as above. Then, the string is encoded by the Base 64 encoding [RFC4648] without any spaces and newlines, and then enclosed by two double-quotations.

3.3. Strings

All strings outside ASCII or equivalent character sets MUST be encoded using UTF-8 encoding [RFC3629] of the ISO 10646-1 character set [ISO.10646-1.1993]. Both peers are RECOMMENDED to reject any invalid UTF-8 sequences which cause decoding ambiguities (e.g. containing <"> in the second or later byte of the UTF-8 encoded characters).

To encode character strings to header values, these will first be encoded according to UTF-8 without leading BOM, then all occurrences of characters <"> and "\" will be escaped by prepending "\", and two <">s will be put around the string. These escaping backslashes and enclosing quotes SHALL be removed before any processing other than using them in header fields.

If strings are representing a domain name or URI which contains non-ASCII characters, host parts SHOULD be encoded as it is used in the HTTP protocol layer (e.g. in a Host: header); in current standard it will be the one defined in [RFC5890]. It SHOULD use lower-case ASCII characters.

For base64-fixed-numbers, which use the string syntax, see the previous section.

4. Messages

In this section we define seven kinds of messages used in the authentication protocol, Along with formats and requirements of the headers for each message.

To determine which message are expected to be sent, see Section 8 and Section 9.

In the descriptions below, the allowed type of values for each header field is shown in parenthesis after the key names. The type "algorithm-determined" means that the acceptable value type for the field is one of the types defined in Section 3, and is determined by the value of the "algorithm" field. The fields marked as "mandatory" SHALL be contained in the message. The fields marked as "non-mandatory" MAY either be contained or omitted in the message.

4.1. 401-B0

Every 401-B0 message SHALL be a valid HTTP 401 (Authentication Required) message containing one (and only one: hereafter not explicitly noticed) "WWW-Authenticate" header of the following format.

WWW-Authenticate: Mutual algorithm=xxxx, validation=xxxx, realm="xxxx", stale=0, version=-draft07

```
header-401-B0 = "WWW-Authenticate" ":" [spaces]
                auth-scheme spaces fields-401-B0
fields-401-B0 = field-401-B0 *([spaces] "," spaces field-401-B0)
field-401-B0  = version / algorithm / validation
              / auth-domain / realm / pwd-hash / stale
              / extension-field
version       = "version"     "=" extensive-token
algorithm     = "algorithm"   "=" extensive-token
validation    = "validation"  "=" extensive-token
auth-domain   = "auth-domain" "=" string
realm         = "realm" "=" string
pwd-hash      = "pwd-hash" "=" extensive-token
stale         = token
```

Figure 4: the BNF syntax for the header in 401-B0 header

The header SHALL contain all of the fields marked as "mandatory" below, and MAY contain those marked as "non-mandatory".

version:
: (mandatory extensive-token) should be the token "-draft07" in this specification. The behavior when other values are specified is undefined.

algorithm:
: (mandatory extensive-token) specifies the authentication algorithm to be used. The value MUST be one of the tokens described in Section 11, or the tokens specified in other supplemental specification documentations.

validation:
: (mandatory extensive-token) specifies the method of host validation. The value MUST be one of the tokens described in Section 7, or the tokens specified in other supplemental specification documentations.

auth-domain:
: (non-mandatory string) specifies authentication domain, the set of hosts on which authentication credentials are valid. It MUST be one of the strings described in Section 5. If the value is omitted, it is assumed to be the host part of the requested URI.

realm:
: (mandatory string) is a UTF-8 encoded string representing the name of the authentication realm inside the authentication domain.

pwd-hash:
: (non-mandatory extensive-token) specifies the hash algorithm (hereafter referred to by ph) used for additionally hashing the password. The valid tokens are
  - none: ph(p) = p
  - md5: ph(p) = MD5(p)
  - digest-md5: ph(p) = MD5(username | ":" | realm | ":" | p), the same value as MD5(A1)

for "MD5" algorithm in [RFC2617].

- sha1: ph(p) = SHA1(p)

If omitted, the value "none" is assumed. The use of "none" is recommended.

stale:

(mandatory token) MUST be "0".

The algorithm specified in this header will determine the types and the values for w_A, w_B, o_A and o_B.

4.2. 401-B0-stale

A 401-B0-stale message is a variant of 401-B0 message, which means that the client has sent a request message which is not for any active session.

WWW-Authenticate: Mutual algorithm=xxxx, validation=xxxx, realm="xxxx", stale=1, version=-draft07

The header MUST contain the same fields as in 401-B0, except that stale field holds the token 1.

4.3. req-A1

Every req-A1 message SHALL be a valid HTTP request message containing a "Authorization" header of the following format.

Authorization: Mutual algorithm=xxxx, validation=xxxx, realm="xxxx", user="xxxx", wa=xxxx, version=-draft07

```
header-req-A1 = "Authorization" ":" [spaces]
                auth-scheme spaces fields-req-A1
fields-req-A1 = field-req-A1 *([spaces] "," spaces field-req-A1)
field-req-A1  = version / algorithm / validation
              / auth-domain / realm / user / wa
              / extension-field
user          = "user" "=" string
wa            = "wa"   "=" value
```

Figure 5: the BNF syntax for the header in req-A1 message

The header SHALL contain the fields with the following keys:

version:

(mandatory, extensive-token) should be the token "-draft07" in this specification. The behavior when other values are specified is undefined.

algorithm, validation, auth-domain, realm:

MUST be the same value as it is received from the server.

user:

(mandatory, string) is the UTF-8 encoded name of the user.

wa:

(mandatory, algorithm-determined) is the client-side key exchange value w_A, which is specified by the used algorithm (see Section 11).

4.4. 401-B1

Every 401-B1 message SHALL be a valid HTTP 401 (Authentication Required) message containing a "WWW-Authenticate" header of the following format.

WWW-Authenticate: Mutual algorithm=xxxx, validation=xxxx, realm="xxxx", sid=xxxx, wb=xxxx, nc-max=x, nc-window=x, time=x, path="xxxx", version=-draft07

```
header-401-B1 = "WWW-Authenticate" ":" [spaces]
                auth-scheme spaces fields-401-B1
fields-401-B1 = field-401-B1 *([spaces] "," spaces field-401-B1)
field-401-B1  = version / algorithm / validation
              / auth-domain / realm / sid / wb
              / nc-max / nc-window / time / path
              / extension-field
sid           = "sid"       "=" string
wb            = "wb"        "=" value
nc-max        = "nc-max"    "=" integer
nc-window     = "nc-window" "=" integer
time          = "time"      "=" integer
path          = "path"      "=" string
```

Figure 6: the BNF syntax for the header in 401-B1 message

The header SHALL contain the fields with the following keys:

version:
: (mandatory, extensive-token) should be the token "-draft07" in this specification. The behavior when other values are specified is undefined.

algorithm, validation, auth-domain, realm:
: MUST be the same value as it is received from the client.

sid:
: (mandatory, hex-fixed-number) MUST be a session identifier, which is a random integer. The sid SHOULD have uniqueness of at least 80 bits or the square of the maximal estimated transactions concurrently available in the session table, whichever is larger. Sids are local to each authentication realm concerned: the same sids for different authentication realms SHOULD be treated as independent ones.

wb:
: (mandatory, algorithm-determined) is the server-side key exchange value w_B, which is specified by the algorithm (see Section 11).

nc-max:
: (mandatory, integer) is the maximal value of nonce counts which S accepts.

nc-window:
: (mandatory, integer) the number of available nonce slots which the server will accept. The value of nc-window is RECOMMENDED to be 32 or more.

time:
: (mandatory, integer) represents the suggested time (in seconds) which the client can reuse the session represented by sid. It is RECOMMENDED to be at least 60. The value of this field is not directly linked to the duration that the server keeps track of the session represented by sid.

path:

(non-mandatory, string) specifies for which path in the URI space the same authentication is expected to apply. The value is a space-separated list of URIs, in the same format as it is specified in domain parameter [RFC2617] for the Digest authentications, and clients are RECOMMENDED to recognize it. The all path elements contained in the field MUST be inside the specified auth-domain: if not, clients SHOULD ignore such elements.

4.5. req-A3

Every req-A3 message SHALL be a valid HTTP request message containing a "Authorization" header of the following format.

Authorization: Mutual algorithm=xxxx, validation=xxxx, realm="xxxx", sid=xxxx, nc=x, oa=xxxx, version=-draft07

```
header-req-A3 = "Authorization" ":" [spaces]
                auth-scheme spaces fields-req-A3
fields-req-A3 = field-req-A3 *([spaces] "," spaces field-req-A3)
field-req-A3  = version / algorithm / validation
              / auth-domain / realm / sid / nc / oa
              / extension-field
nc            = "nc" "=" integer
oa            = "oa" "=" value
```

Figure 7: the BNF syntax for the header in req-A3 message

The fields contained in the header are as follows:

version:

(mandatory, extensive-token) should be the token "-draft07" in this specification. The behavior when other values are specified is undefined.

algorithm, validation, auth-domain, realm:

MUST be the same value as it is received from the server for the session.

sid:

(mandatory, hex-fixed-number) MUST be one of the sid values which has been received from the server for the same authentication realm.

nc:

(mandatory, integer) is a nonce value which is unique among the requests sharing the same sid. Values of nonces SHOULD satisfy the properties outlined in Section 6.

oa:

(mandatory, algorithm-determined) is the client-side authentication challenge value o_A, which is specified by the algorithm (see Section 11).

4.6. 200-B4

Every 200-B4 message SHALL be a valid HTTP message which is not 401 (Authentication Required) type, containing an "Authentication-Info" header of the following format.

Authentication-Info: Mutual sid=xxxx, ob=xxxx, version=-draft07

```
header-200-B4  = "Authentication-Info" ":" [spaces]
                 auth-scheme spaces fields-200-B4
fields-200-B4  = field-200-B4 *([spaces] "," spaces field-200-B4)
field-200-B4   = version / sid / ob / logout-timeout
ob             = "ob"             "=" value
logout-timeout = "logout-timeout" "=" integer
```

Figure 8: the BNF syntax for the header in 200-B4 message

The fields contained in the header are as follows:

version:
: (mandatory, extensive-token) should be the token "-draft07" in this specification. The behavior when other values are specified is undefined.

sid:
: (mandatory, hex-fixed-number) MUST be the value received from the client.

ob:
: (mandatory, algorithm-determined) is the server-side authentication challenge value o_B, which is specified by the algorithm (see Section 11).

logout-timeout:
: (non-mandatory, integer) is a number of seconds after which the client should re-validate the user's password for the current authentication realm. As a special case, the value 0 means that the client SHOULD automatically forget the user-inputted password to the current authentication realm and revert to the unauthenticated state (i.e. server-initiated logout). This does not, however, mean that the long-term memories for the passwords (such as password reminders and auto fill-ins) should be removed. If a new value of timeout is received for the same authentication realm, it overrides the previous timeout.

The header MUST be sent before the content body: it MUST NOT be sent in a trailer of a chunked-encoded response. If a "100 Continue" response is sent from the server, The Authentication-Info header SHOULD be included in that response, instead of the final response.

4.7. 200-Optional-B0

The 200-Optional-B0 messages enables a non-mandatory authentication, which is not possible under current HTTP authentication mechanism. In several Web applications, users can access the same contents both as a guest user and as a authenticated users. In usual Web applications, it is implemented using HTTP cookies [RFC2965] and custom form-based authentications. The new method of authentication using this message will provide a replacement for those authentication systems. The support for this message is RECOMMENDED, unless the protocol is used for some specific applications in which authentication is always mandatory.

Servers MAY send HTTP successful responses (response code 200, 206 and others) containing the Optional-WWW-Authenticate header, when it is allowed to send 401-B0 responses (with one exception described below). Such responses are hereafter called 200-Optional-B0 responses.

HTTP/1.1 200 OK
Optional-WWW-Authenticate: Mutual version=-draft07, algorithm=xxxx, validation=xxxx, realm="xxxx", stale=0

```
header-200-Optional-B0 = "Optional-WWW-Authenticate" ":" [spaces]
                         auth-scheme spaces fields-401-B0
```

Figure 9: the BNF syntax for the header in 200-Optional-B0 header

The fields contained in the Optional-WWW-Authenticate header is the same as the 401-B0 message described in Section 4.1. For authentication-related matters, a 200-Optional-B0 message will have the same meaning as a 401-B0 message with a corresponding WWW-Authenticate header. (The behavior for other matters, such as caching, MAY be different between 200-Optional-B0 and 401-B0 messages.)

The 200-Optional-B0 message is a only place where an Optional-WWW-Authenticate header is allowed. If a server is to send a 401-B1 or a 401-B0-stale responses, it SHALL NOT replace it with 200-Optional-B0 or similar responses. Furthermore, if a server is going to send a 401-B0 message as a responses to req-A3 message with a correct realm, the server MUST send a 401-B0 message, not a 200-Optional-B0 message.

Servers requesting non-mandatory authentication SHOULD send the path field in 401-B1 messages with an appropriate value. Clients supporting non-mandatory authentication MUST recognize the field, and MUST send either req-A1 or req-A3 request for the URI space inside the specified paths, instead of a normal request without an Authorization header.

5. Authentication Realms

In this protocol, an "authentication realm" is defined as a set of resources (URIs) for which the same set of user names and passwords is valid for. If the server requests authentication for the authentication realm which the client is already authenticated, the client will automatically perform authentication using the already-known secrets. On the contrary, for the different authentication realms, clients SHOULD NOT automatically reuse the usernames and passwords for another realm.

Just like Basic and Digest access authentication protocol, Mutual authentication protocol supports multiple, separate authentication realms to be set up inside each host. Furthermore, the protocol supports that a single authentication realm spans over several hosts in the same Internet domain.

Each authentication realm is defined and distinguished by the triple of an "authentication algorithm", an "authentication domain", and a "realm" parameter. Server operators are NOT RECOMMENDED to use the same pair of an authentication domain and a realm for different authentication algorithms, however.

Authentication algorithms are defined in Section 4 and Section 11. The realm parameter is a string as defined in Section 4. Authentication domains are described in the rest of this section.

An authentication domain specifies the range of hosts which the authentication realm spans over. In the protocol, it MUST be one of the following strings.

- The string in format "<scheme>://<host>:<port>", where scheme, host and port are the URI parts of the requested URI. Even if the request-URI does not have a port part, the string will include the one (i.e. 80 for http and 443 for https). Use this when authentication is only valid for specific protocol (such as https).
- The "host" part of the requested URI. This is the default value. Authentication realms in this kind

of authentication domain will span over several protocols (i.e. http and https) and ports, but not over different hosts.

- String in format "*.<domain-postfix>", where "domain-postfix" is either the host part of the requested URI, or any domain in which the requested host is included (this means that the specification "*.example.com" is valid for all of hosts "www.example.com", "web.example.com", "www.sales.example.com" and "example.com"). The domain-postfix sent from servers MUST be equal to or included in a valid Internet domain assigned to specific organization: if clients knows, by some means such as blacklists for HTTP cookies, that the specified domain is not to be assigned to any specific organization (e.g. "*.com" or "*.jp"), clients are RECOMMENDED to reject the authentication request.

In the above specifications, every "scheme", "host" and "domain" MUST be in lower-case, and any internationalized domain names beyond the ASCII character set SHALL be represented in the way these are sent in the underlying HTTP protocol, represented in lower-case characters; i.e. these SHALL be in the form of the LDH labels in IDNA [RFC5890]. All "port"s MUST be in the shortest, unsigned, decimal number notation. Not obeying these requirements will cause failure of valid authentication attempts.

5.1. Resolving ambiguities

In the above definition of authentication domains, several domains will overwrap with each other. Depending on the "path" parameters given in the "401-B1" message (see Section 4), There may be several candidates when the client is to send a request with authentication credentials included (at the Steps 3 and 4 of the decision procedure shown in Section 8).

If such choices are required, the following procedure SHOULD be followed.

- If the client has previously sent a request to the same URI, and it remembers the authentication realm requested by 401-B0 messages at that time, use that realm.
- In other cases, use one of authentication realms which represents most-specific authentication domains. In the list of possible domain specifications shown above, one described earlier has priority over ones described after that.
  If there are several choices with different domain-postfix specifications, the one which has the longest domain-postfix has priority over ones with shorter domain-postfix.
- If there are realms with the same specifications of authentication domain, there is no defined priority: client MAY choose any one of possible choices.

If possible, server operators are encouraged to avoid such ambiguities by setting "path" parameters properly.

6. Session Management

In the Mutual authentication protocol, a session represented by an sid is set up by the first 4 messages (first request, 401-B0, req-A1 and 401-B1), and a "session secret" (z) associated to the session is established. After having a session secret, this session, along with the secret, can be used for one or more requests for resources protected by the same realm in the same server. Note that the session management is only an inside detail of the protocol and usually not visible to normal users. If a session expires, the client and server SHOULD automatically reestablish another session without telling it to the users.

The sessions are local to each port of a host inside an authentication domain; clients MUST establish separate sessions for each port of a host to be accessed.

The server SHOULD accept at least one req-A3 request for each session, given that the request reaches the server in a time window specified by the timeout field in the 401-B1 message, and that there are no emergent reasons (such as flooding attacks) to forget the sessions. After that, the server MAY discard any session at any time and MAY send 401-B0-stale messages for any req-A3 requests.

The client MAY send two or more requests using a single session specified by the sid. However, for all such requests, each value of the nonce (in the nc field) MUST satisfy the following conditions:

- It is a natural number.
- The same nonce has not yet sent previously in the same session.
- It is not larger than the nc-max value which has been sent from the server in the session represented by the sid.
- It is larger than (largest-nc - nc-window), where largest-nc is the maximal value of nc which has previously been sent in the session, and nc-window is the value of the nc-window field which has been sent from the server in the session.

The last condition allows servers to reject any nonce values which is "significantly" smaller than the "current" value (defined by the value of nc-window) of the nonce used in the session involved. In other words, servers MAY treat such nonces as "already received". This restriction enables servers to implement duplicated nonce detection in a constant amount of memory (for each session).

Servers MUST check for duplication of the received nonces, and if any duplication is detected, the server MUST discard the session and respond by a 401-B0-stale message, as outlined in Section 9. The server MAY also reject other invalid nonce values (such as ones above the nc-max limit) by sending a 401-B0-stale message.

For example, consider the nc-window value of the current session to be 32, the nc-max to be 100, and that the client has already used the following nonce values beforehand: {1-20, 22, 24, 30-38, 45-60, 63-72}. Then the nonce values which can be used for next request is one of the following set: {41-44, 61-62, 73-100}. The values {0, 21, 23, 25-29, 39-40} MAY be rejected by the server because these are not above the current "window limit" (40 = 72 - 32).

Typically, clients can ensure the above property by using a monotonically-increasing integer counter counting from zero upto the value of nc-max.

Values of nonces and nonce-related values MUST always be treated as natural numbers within infinite range. Implementations using fixed-width integers or fixed-precision floating numbers MUST handle integer overflow correctly and carefully. Such implementations are RECOMMENDED to accept any larger values which cannot be represented in the fixed-width integer representations, as long as other limits such as internal header-length restrictions are not involved. The protocol is designed carefully so that both clients and servers can implement the protocol only with fixed-width integers, by rounding any overflowed values to the maximum possible value.

7. Validation Methods

The "validation method" specifies a method to "relate" the mutual authentication processed by this protocol with other authentications already performed in the underlying layers and to prevent man-in-the-middle attacks. It decides the value of v which is an input to authentication protocols.

The valid tokens for the validation field and corresponding values of v are as follows:

host:
: hostname validation: The value v will be the ASCII string in the following format: "<scheme>://<host>:<port>", where scheme, host and port are the URI components correspond to the currently accessing resource. The scheme and host are lower-case, and the port is in a shortest decimal representation. Even if the request-URI does not have a port part, v will include the one.

tls-cert:
: TLS certificate validation: The value v will be the octet string of the hash value of the public key certificate used in underlying TLS [RFC5246] (or SSL) connection. The hash value is defined as the value of the whole signed certificate (specified as "Certificate" in [RFC5280]), hashed by the hash algorithm specified by the authentication algorithm used.

tls-key:
: TLS shared-key validation: The value v will be the octet string of the shared master secret negotiated in underlying TLS (or SSL) connection.

If the HTTP protocol is used on non-encrypted channel (TCP and SCTP, for example), the validation type MUST be "host". If HTTP/TLS [RFC2818] (https) protocol is used with server certificates, the validation type MUST be either "tls-cert" or "tls-key". If HTTP/TLS protocol is used with anonymous Diffie-Hellman key exchange, the validation type MUST be "tls-key" (but see the note below).

Clients MUST validate this field upon reception of 401-B0 messages.

However, when the client is a Web browser with any scripting capabilities, underlying TLS channel used with HTTP/TLS MUST provide server identity verification. This means (1) anonymous Diffie-Hellman key exchange ciphersuite MUST NOT be used, and (2) the verification of server certificate provided from the server MUST be employed.

For other systems, when underlying TLS channel used with HTTP/TLS does not provide server identity verification, the client SHOULD ensure that all responses are validated using the Mutual authentication protocol, regardless of the existence of the 401-B0 responses.

Note: The protocol defines two variants for validation on TLS connections. The method "tls-key" method is more secure. However, there are some situations where tls-cert is more preferable.

- When TLS accelerating proxies are used. In this case, it is difficult for the authenticating server to acquire the TLS key information which is used between the client and the proxy. It is not the case for client-side "tunneling" proxies using CONNECT method extension of HTTP.
- When a black-box implementation of the TLS protocol is used on either peer.

Implementations supporting Mutual authentication over HTTPS protocol SHOULD support "tls-cert" validation. Support for "tls-key" validation is OPTIONAL for both servers and clients.

8. Decision procedure for the client

To securely implement the protocol, the user client must be careful for accepting authenticated responses from the server. This also holds upon reception of "normal responses" (responses which do not contain Mutual-related headers) from HTTP servers.

Clients SHOULD implement the decision procedure equivalent to the one shown below. (Unless implementers understand what is required for the security, they should not alter this.) Especially, clients SHOULD NOT accept "normal responses" unless explicitly allowed below. The labels on the steps are for informational purpose only. Entries within each step are checked in top-to-bottom order, and the first clause satisfied SHOULD be taken.

Step 1 (step_new_request):

If the client software needs to get a new Web resource, check whether the resource is expected to be inside some authentication realm for which the user has already authenticated by the Mutual authentication scheme. If yes, go to Step 2. Otherwise, go to Step 5.

Step 2:

Check whether there is an available sid for the authentication realm you expect. If there is one, go to Step 3. Otherwise, go to Step 4.

Step 3 (step_send_a3_1):

Send a req-A3 request.

- If you receive a 401-B0 message with a different authentication realm than expected, go to Step 6.
- If you receive a 200-Optional-B0 message with a different authentication realm than expected, go to Step 6.
- If you receive a 401-B0-stale message, go to Step 9.
- If you receive a 401-B0 message, go to Step 13.
- If you receive a 200-B4 message, go to Step 14.
- If you receive a normal response, go to Step 11.

Step 4 (step_send_a1_1):

Send a req-A1 request.

- If you receive a 401-B0 message with a different authentication realm than expected, go to Step 6.
- If you receive a 200-Optional-B0 message with a different authentication realm than expected, go to Step 6.
- If you receive a 401-B1 message, go to Step 10.
- If you receive a 401-B0 message with the same authentication realm, go to Step 13 (see Note 1).
- If you receive a normal response, go to Step 11.

Step 5 (step_send_normal_1):

Send a request without any Mutual authentication headers.

- If you receive a 401-B0 message, go to Step 6.
- If you receive a 200-Optional-B0 message, go to Step 6.
- If you receive a normal response, go to Step 11.

Step 6 (step_rcvd_b0):

Check whether you know the user’s password for the requested authentication realm. If yes, go to Step 7. Otherwise, go to Step 12.

Step 7:

Check whether there is an available sid for the authentication realm you expect. If there is one, go to Step 8. Otherwise, go to Step 9.

Step 8 (step_send_a3):

Send a req-A3 request.

- If you receive a 401-B0-stale message, go to Step 9.
- If you receive a 401-B0 message, go to Step 13.
- If you receive a 200-B4 message, go to Step 14.

Step 9 (step_send_a1):

Send a req-A1 request.

- If you receive a 401-B1 message, go to Step 10.
- If you receive a 401-B0 message, go to Step 13 (See Note 1).

Step 10 (step_rcvd_b1):

Send a req-A3 request.

- If you receive a 401-B0 message, go to Step 13.
- If you receive a 200-B4 message, go to Step 14.

Step 11 (step_rcvd_normal):

This case means that the resource requested is out of the authenticated area. The client will be in the "UNAUTHENTICATED" status. If the response contains a request for authentications other than Mutual, it MAY be handled normally.

Step 12 (step_rcvd_b0_unknown):

This case means that the resource requested requires Mutual authentication, and the user is not authenticated yet. The client will be in the "AUTH_REQUESTED" status, is RECOMMENDED to process the content sent from the server and ask user a username and a password. If the user has inputted those, go to Step 9.

Step 13 (step_rcvd_b0_failed):

This case means that in some reason the authentication failed: possibly the password or the username is invalid for the authenticated resource. Forget the password for the authentication realm and go to Step 12.

Step 14 (step_rcvd_b4):

Check the validity of the received o_b value. If it is equal to the expected value, it means that the mutual authentication has been succeeded. The client will be in the "AUTH_SUCCEEDED" status.

If the value is unexpected, it is a fatal communication error.

If a user requests to log out explicitly (via user interfaces), the client MUST forget user's password, go to step 5 and reload the current resource without authentication credential.

Note 1:

These transitions are valid for clients, but not recommended for servers to initiate.

Any kind of response (including a normal response) other than those shown in the above procedure SHOULD be interpreted as fatal communication error, and in such cases user clients MUST NOT process any data (response body and other content-related headers) sent from the server. However, as a handling for exceptional error cases, clients MAY accept a message without an Authentication-Info header, if it is a Server-Error (5xx) status. The client will be in the "UNAUTHENTICATED" status in these cases.

The client software SHOULD display the three client status to the end-user. For an interactive client, however, if a request is a sub-request for a resource included to another page (e.g. embedded images, style sheets, frames etc.), its status MAY be omitted from being shown, and any "AUTH_REQUESTED" statuses MAY be treated in the same way as an "UNAUTHENTICATED" status.

Figure 10 shows the full client-side state diagram.

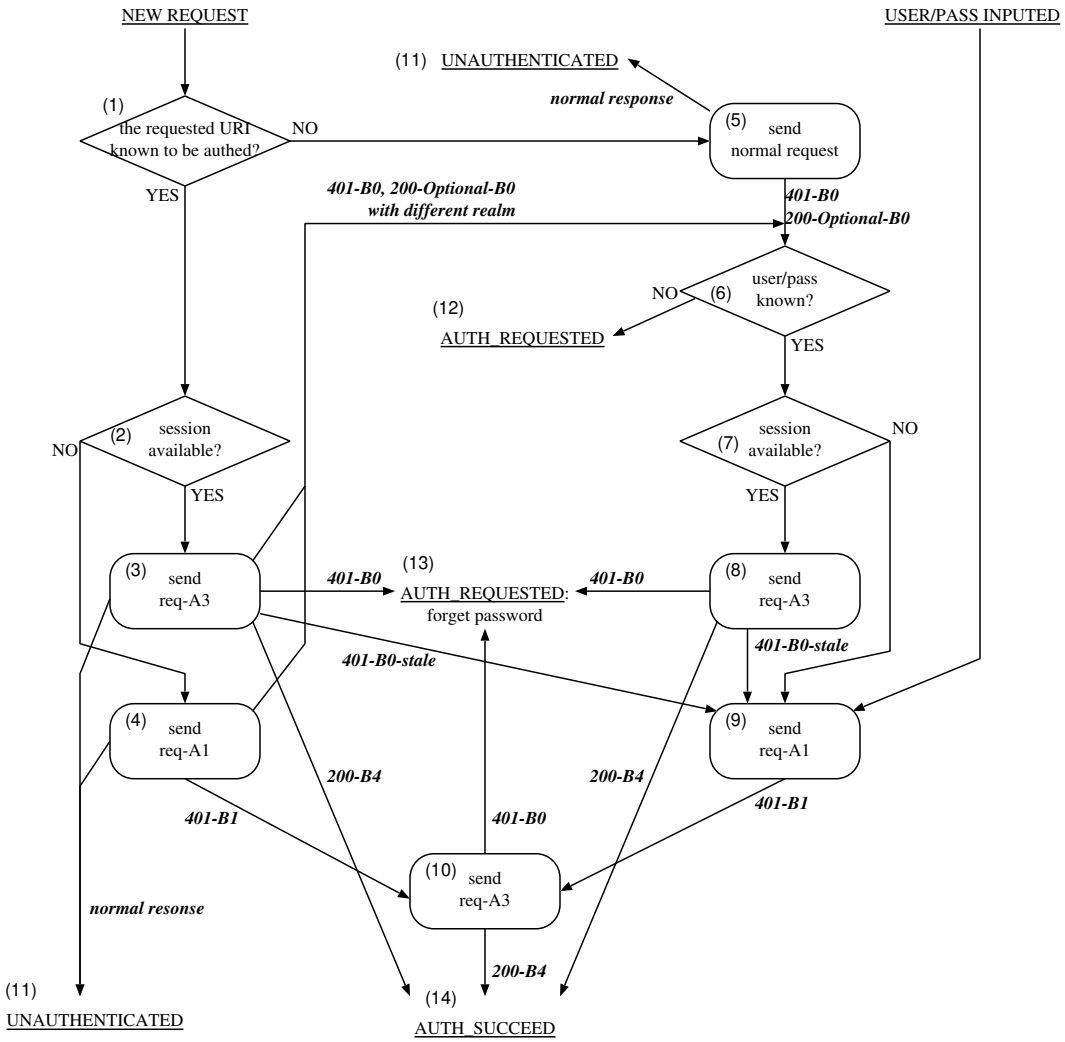


Figure 10: State diagram for clients

---

9. Decision procedure for the server

Each server SHOULD have a table of session states. This table need not be persistent in a long term; it MAY be cleared upon server restart, reboot and others. Each entry of the table SHOULD contain at least the following information:

- The session identifier, the value of the sid field
- The algorithm used
- The authentication realm
- The state of the protocol: one of "wa received", "authenticated", "rejected", and "inactive"
- The user name received from the client
- The boolean flag whether the session is fake
- When the state is "wa received", the values of wa and sb
- When the state is "authenticated", the following information:
  - The value of the session secret z
  - The largest nc received from the client (largest-nc)

- For each possible nc values between (largest-nc - nc-window + 1) and max_nc, a flag whether a request with corresponding nc has been received.

The table MAY contain other information.

Servers SHOULD respond to the client requests according to the following procedure:

- When the server receives a normal request:
  - If the requested resource is not protected by the Mutual Authentication, send a normal response.
  - If the resource is protected by the Mutual Authentication, send a 401-B0 response.
  - If the resource is protected by the optional Mutual Authentication, send a 200-Optional-B0 response.
- When the server receives a req-A1 request:
  - If the requested resource is not protected by the Mutual Authentication, send a normal response.
  - If the authentication realm specified in the req-A1 request is not the expected one, send either a 401-B0 or a 200-Optional-B0 response.
  - If the server cannot validate the field wa, send a 401-B0 response.
  - If the received user name is either invalid, unknown or unacceptable, create a new session, mark it as a "fake" session, compute a random value as wb, and send a fake 401-B1 response. (Note: the server SHOULD NOT send 401-B0 response in this case, because it will leak the information to the client that the specified user will not be accepted. Instead, postpone it to the response for the next req-A3 request.)
  - Otherwise, create a new session, compute wb and send a 401-B1 response.

  The created session has "wa received" state.
- When the server receives a req-A3 request:
  - If the requested resource is not protected by the Mutual Authentication, send a normal response.
  - If the authentication realm specified in the req-A3 request is not the expected one, send either a 401-B0 or a 200-Optional-B0 response.

  If none of above is hold, the server will lookup the session corresponding to the received sid and the authentication realm.
  - If the session corresponding to the received sid could not be found, or it is inactive, send a 401-B0-stale response.
  - If the session is in "rejected" state, send either a 401-B0 or a 401-B0-stale message.
  - If the session is a "fake" session, or if the received oa is incorrect, then send a 401-B0 response. If the session is "wa received" state, it SHOULD be changed to a "rejected" state; otherwise, it MAY either be changed to a "rejected" status or keep the previous state.
  - If the session is in "active" state, and request has a nc value which was previously received from the client, send either a 401-B0-stale message. The session SHOULD be changed to "inactive" status.
  - If the nc value in the request is larger than the nc-max field sent from the server, or if it is not larger then (largest-nc - nc-window) (when in "authenticated" status), the server MAY (not REQUIRED) send either a 401-B0-stale message. The session SHOULD be changed to a "inactive" status if did so.
  - Otherwise, send a 200-B4 response. If the session was "wa received" state, the session SHOULD be changed to an "authenticated" state. The maximum nc and the nc flags of the state SHOULD be updated properly.

At any time, the server MAY change any state entries with both "rejected" and "authenticated" status to "inactive" status, and MAY discard any "inactive" states from the table. The entries with "wa received" status SHOULD be kept unless there is an emergency situation such as server reboot and table capacity overflow.

10. Authentication-Control header

```
Authentication-Control-header
                    = "Authentication-Control" ":" [spaces]
                      auth-scheme spaces Auth-Ctrl-fields
Auth-Ctrl-fields    = Auth-Ctrl-field
                      *([spaces] "," spaces Auth-Ctrl-field)
Auth-Ctrl-field     = loc-when-unauthed / loc-when-logout
                    / logout-timeout
                    / extension-field
loc-when-unauthed = "location-when-unauthenticated" "=" string
loc-when-logout   = "location-when-logout" "=" string
```

Figure 11: the BNF syntax for the Authentication-Control header

The Authentication-Control header gives more precise control for the client behavior for Web applications using Mutual Access Control Protocol. This headers may usually be generated in an application layer, as opposed to WWW-Authenticate headers which will be generated by Web servers.

Support of this header is RECOMMENDED for interactive clients and not required for non-interactive clients. Web applications SHOULD consider security impacts of behavior of clients which do not support this header.

The "auth-scheme" of this header and other authentication-related headers within the same message MUST be equal. This document does not define any behavior associated with this header, when the "auth-scheme" of this header is not "Mutual".

10.1. Location-when-unauthenticated field

Authentication-Control: Mutual
location-when-unauthenticated="http://www.example.com/login.html"

The field "location-when-unauthenticated" specifies a location which any unauthenticated clients should be redirected to. This header may be used, for example, when there is a central login page for the whole Web application. The value of this field MUST be a string that contains an absolute URL location. If a given URL is not absolute, clients MAY consider it as a relative URL from the current location.

This field MAY be used with 401-B0 and 200-Optional-B0 messages; however, use of this with 200-Optional-B0 messages is not recommended. If there is a 200-B4, 401-B0-stale or 401-B1 message with this field, clients MUST ignore this field.

When a client receives a message with this field, if and only if the client's state after the processing the response is either Step 12 or Step 13 (i.e., a state in which the client will process response body and ask user's password), the client will treat the whole response as if it were a 303 "See Other" response with a Location header with the value of this field (i.e., client will be redirected to the specified location with a GET request). Unlike a normal 303 response, if the client can proceed authentication

without user's interaction (like steps 3, 4, 8, 9 and 10), this field is ignored.

The specified location SHOULD be included in a set of locations specified in the "auth-domain" field of the corresponding 401-B0 message. If this is not satisfied, clients MAY ignore this field.

10.2. Location-when-logout field

Authentication-Control: Mutual location-when-logout="http://www.example.com/byebye.html"

The field "location-when-logout" specifies a location where the client is to be redirected when users request logout explicitly. The value of this field MUST be a string that contains an absolute URL location. If a given URL is not absolute, clients MAY consider it as a relative URL from the current location.

This field MAY be used with 200-B4 messages. If there is a 401-B0, 401-B1, 401-B0-stale, 200-Optional-B0 or normal 200 message with this field, clients MUST ignore this field.

When users of a client request to terminate an authentication session, and if the client currently displays a page supplied by a response with this field, the client will be redirected to the specified location by a new GET request (like received a 303 response), instead of reloading the page without authentication credentials. It is recommendable for Web applications to send this field with an appropriate value for any responses (except those with redirection (3XX) statuses) for non-GET requests.

10.3. Logout-timeout

Authentication-Control: Mutual logout-timeout=300

The field "logout-timeout" has the same meaning as the field of the same name in "Authentication-Info" headers. This field will be used with 200-B4 messages. If both are specified, clients are recommended to use the one with the smaller value.

11. Authentication Algorithms

This document specifies only one family of the authentication algorithm. The family consists of four authentication algorithms, which only differ in underlying mathematical groups and security parameters. The algorithms do not add any additional fields. The tokens for algorithms are

- iso-kam3-ec-p256-sha256: for the 256-bit prime-field elliptic-curve setting with SHA-256 hash function.
- iso-kam3-ec-p521-sha512: for the 521-bit prime-field elliptic-curve setting with SHA-512 hash function.
- iso-kam3-dl-2048-sha256: for the 2048-bit discrete-logarithm setting with SHA-256 hash function.
- iso-kam3-dl-4096-sha512: for the 4096-bit discrete-logarithm setting with SHA-512 hash function.

For the elliptic-curve settings, the underlying groups are the elliptic curves over prime fields P-256 and P-521, respectively, specified in the appendix D.1.2 of FIPS PUB 186-3 [FIPS.186-3.2009] specification. The hash functions H are SHA-256 for P-256 curve and SHA-512 for P-521 curve, respectively, defined in FIPS PUB 180-2 [FIPS.180-2.2002]. The representation of fields wa, wb, oa, and ob is hex-fixed-number.

For discrete-logarithm settings, the underlying groups are 2048-bit and 4096-bit MODP groups defined in [RFC3526], respectively. See Appendix B for the exact specification of the group and associated parameters. The hash functions H are SHA-256 for the 2048-bit group and SHA-512 for the 4096-bit group, respectively. The representation of fields wa, wb, oa, and ob is base64-fixed-number.

The clients SHOULD support at least "iso-kam3-dl-2048-sha256" algorithm, and are advised to support all of the above four algorithms whenever possible. The server software implementations SHOULD support at least "iso-kam3-dl-2048-sha256" algorithm, unless it is known that users will not use it.

Note: This algorithm is based on the Key Agreement Mechanism 3 (KAM3) defined in Section 6.3 of ISO/IEC 11770-4 [ISO.11770-4.2006] with a few modifications/improvements. However, implementers should use this document as the normative reference, because the algorithm has been changed in several minor details as well as major improvements.

11.1. Support functions and notations

The algorithm definitions use several support functions and notations defined below:

The integers in the specification is decimal, or hexadecimal when prefixed with "0x".

The function octet(c) generates a single octet string whose code value is equal to c. The operator |, when applied to octet strings, denotes the concatenation of two operands.

The function VI encodes natural numbers into octet strings in the following manner: numbers are represented in big-endian radix-128 string, where each digit is represented by a octet within 0x80–0xff except the last digit represented by a octet within 0x00–0x7f. The first octet MUST NOT be 0x80. For example, VI(i) = octet(i) for i < 128, and VI(i) = octet(0x80 + (i >> 7)) | octet(i & 127) for 128 <= i < 16384. This encoding is the same as the one used for subcomponents of object identifiers in the ASN.1 encoding [ITU.X690.1994], and available as a "w" conversion in the pack function of several scripting languages.

The function VS encodes variable-length octet string into uniquely-decoded, self-delimited octet string, as in the following manner:

VS(s) = VI(length(s)) | s

where length(s) is a number of octets (not characters) in s.

[Editorial note: Unlike the colon-separated notion used in the Basic/Digest HTTP authentication scheme, the string generated by a concatenation of the VS-encoded strings will be unique, regardless of the characters included in the strings encoded.]

The function OCTETS converts an integer to corresponding radix-256 big-endian octet string having its natural length: See Section 3.2 for the definition of the "natural length".

Note: The definition of OCTETS() is different from the function GE2OS_x in the original ISO specification, which takes the shortest representation.

11.2. Common functions for both settings

The password-based string pi used by this authentication is derived in the following manner:

pi = H(VS(algorithm) | VS(auth-domain) | VS(realm) | VS(username) | VS(ph(password)).

The values of algorithm, realm and auth-domain are taken from the values contained in the 401-B0 (or 200-Optional-B0, hereafter implied) message. When pi is used in the context of an octet string, it SHALL have the natural length derived from the size of the output of function H (e.g. 32 octets for SHA-256). The function ph is defined by the value of the pwd-hash field given in a 401-B0 message. The password SHALL be encoded as a UTF-8 string before passed to ph.

The values o_A and o_B are derived by the following equation.

o_A = H(octet(4) | OCTETS(w_A) | OCTETS(w_B) | OCTETS(z) | VI(nc) | VS(v))
o_B = H(octet(3) | OCTETS(w_A) | OCTETS(w_B) | OCTETS(z) | VI(nc) | VS(v))

The equations for J, w_A, T, z, and w_B are specified differently for the discrete-logarithm setting and the elliptic-curve setting. These equations are defined later in this section.

11.3. Functions for discrete-logarithm settings

In this section, the equation (x / y mod z) denotes a natural number w less than z which satisfies (w * y) mod z = x mod z.

For the discrete-logarithm, we refer some of the domain parameters by the following symbols:

- q: for "the prime" of the group.
- g: for "the generator" associated with the group.
- r: for the order of the subgroup generated by g.

The function J is defined as

J(pi) = g^(pi) mod q.

The value of w_A is derived as

w_A = g^(s_A) mod q,

where s_A is a random integer within range [1, r-1] and r is the size of the subgroup generated by g. In addition, s_A MUST be larger than log(q)/log(g) (so that g^(s_A) > q).

The value of w_A SHALL satisfy 1 < w_A < q-1. The server MUST check this condition upon reception.

The value of w_B is derived from J(pi) and w_A as:

w_B = (J(pi) * w_A^(H(octet(1) | OCTETS(w_A))))^s_B mod q,

where s_B is a random number within range [1, r-1]. The value of w_B MUST satisfy 1 < w_B < q-1. If this condition is not hold, the server MUST retry with another value of s_B. The client MUST check this condition upon reception.

The value z in the client side is derived by the following equation:

z = w_B^((s_A + H(octet(2) | OCTETS(w_A) | OCTETS(w_B))) / (s_A * H(octet(1) | w_A) + pi) mod r) mod q.

The value z in the server side is derived by the following equation:

z = (w_A * g^(H(octet(2) | OCTETS(w_A) | OCTETS(w_B))))^s_B mod q.

11.4. Functions for elliptic-curve settings

For the elliptic-curve setting, we refer some of the domain parameters by the following symbols:

- q: for the prime used to define the group,
- G: for the defined point called the generator,
- r: for the order of the subgroup generated by G.

The function P(p) converts a curve point p to an integer representing the point p, by computing x * 2 + (y mod 2), where (x, y) are the coordinates of the point p. P'(z) is the inverse of function P, that is, it converts an integer z to a point p which satisfies P(p) = z. If such p is exist, it is uniquely defined. Otherwise, z does not represent a valid curve point. The operation [x] * p denotes an integer-multiplication of point p: it calculates p + p + ... (x times) ... + p. See literatures on elliptic-curve cryptography for the exact algorithms for those. 0_E represents the infinity point. The equation (x / y mod z) denotes an natural number w less than z which satisfies (w * y) mod z = x mod z.

the function J is defined as

J(pi) = [pi] * G.

The value of w_A is derived as

w_A = P(W_A), where W_A = [s_A] * G.

where s_A is a random number within range [1, r-1]. The value of w_A MUST represent a valid curve point, and W_A SHALL NOT be 0_E. The server MUST check this condition upon reception.

The value of w_B is derived from J(pi) and W_A = P'(w_A) as:

w_B = P(W_B), where W_B = [s_B] * (J(pi) + [H(octet(1) | OCTETS(w_A))] * W_A).

where s_B is a random number within range [1, r-1]. The value of w_B MUST represent a valid curve point and satisfy [4] * P'(w_B) <> 0_E. If this condition does not hold, the server MUST retry with another value of s_B. The client MUST check this condition upon reception.

The value z in the client side is derived by the following equation:

z = P([(s_A + H(octet(2) | OCTETS(w_A) | OCTETS(w_B))) / (s_A * H(octet(1) | OCTETS(w_A)) + pi) mod r] * W_B), where W_B = P'(w_B).

The value z in the server side is derived by the following equation:

z = P([s_B] * (W_A + [H(octet(2) | OCTETS(w_A) | OCTETS(w_B))] * G)), where W_A = P'(w_A).

12. Methods to extend this protocol

If a non-standard extension to the this protocol is implemented, it MUST use the extension-tokens defined in Section 3 to avoid conflicts with this protocol and other extensions.

Authentication algorithms other than those defined in this document MAY use other representations for keys "wa", "wb", "oa" and "ob", replace those keys, and/or add fields to the messages containing those fields by supplemental specifications. Two-octet keys from "wc" to "wz" and from "oc" to "oz" are reserved for this purpose. If those specifications use keys other than shown above, it is RECOMMENDED to use extension-tokens to avoid any key-name conflict with the future extension of this protocol.

Extension-tokens MAY be freely used for any non-standard, private and/or experimental uses for those fields provided that the domain part in the token is appropriately used.

13. IANA Considerations

The tokens used for authentication-algorithm, pwd-hash, and validation fields MUST be allocated by IANA. To acquire registered tokens, a specification for the use of such tokens MUST be available as an RFC, as outlined in [RFC5226].

Note: More formal declarations will be added in future drafts to meet RFC 5226 requirements.

14. Security Considerations

14.1. Security Properties

- The protocol is secure against passive eavesdropping and replay attacks. However, the protocol relies on transport security including DNS integrity for data secrecy and integrity. HTTP/TLS SHOULD be used where transport security is not assured and/or data secrecy is important.
- When used with HTTP/TLS, if TLS server certificates are reliably verified, the protocol gives true protection against active man-in-the-middle attacks.
- Even if the server certificate is not used or is unreliable, the protocol gives protection against active man-in-the-middle attacks for each HTTP request/response pair. However, in such cases, JavaScript or similar scripting facilities can be used to affect Mutually-authenticated contents from other contents not protected by this authentication mechanism. This is the reason why this protocol requires that valid TLS server certificates MUST be presented (Section 7).

14.2. Denial-of-service attacks to servers

The protocol requires a server-side table of active sessions, which may become a critical point of the server resource consumptions. For proper operation, the protocol requires that at least one key verification request is processed for each session identifier. After that, servers MAY discard sessions internally at any time, without causing any operational problems to clients. Clients will silently reestablishes a new session then.

However, if a malicious client sends too many requests of key exchanges (req-A1 messages) only, resource starvation might occur. In such critical situations, servers MAY discard any kind of existing sessions regardless of these statuses. One way to mitigate such attacks are that servers MAY have a number and a time limits for unverified pending key exchange requests (in the "wa received" status).

This is a common weakness of authentication protocols with almost any kind of negotiations or states, including Digest authentication method and most Cookie-based authentication implementations. However, regarding the resource consumption, a situation of the mutual authentication method is a slightly better than the Digest, because HTTP requests without any kind of authentication requests will not generate any kind of sessions. Session identifiers are only generated after a client starts a key negotiation. It means that simple clients such as web crawlers will not accidentally consume server-side resources for session managements.

14.3. Implementation Considerations

- To securely implement the protocol, the Authentication-Info headers in the 200-B4 messages MUST always be validated by the client. If the validation fails, the client MUST NOT process any content sent with the message, including the body part. Non-compliance to this requirement will allow phishing attacks.
- The authentication status on the client-side SHOULD be visible to the users of the client. In addition, the method for asking user's name and passwords SHOULD be carefully designed so that (1) the user can easily distinguish request of this authentication methods from other existing authentication methods such as Basic and Digest methods, and (2) the Web contents cannot imitate the user-interfaces for this protocol.
  An informational memo regarding user-interface considerations and recommendations for implementing this protocol will be separately published.
- For HTTP/TLS communications, when a web form is submitted from Mutually-authenticated pages with the validation methods of "tls-cert" to a URI which is protected by the same realm (so indicated by the path field), if server certificate has been changed since the pages has been received, the peer is RECOMMENDED to be revalidated using a req-A1 message with an "Expect: 100-continue" header. The same applies when the page is received with the validation methods of "tls-key", and when the TLS session has been expired.
- Server-side storage of user passwords are advised to have the values encrypted by one-way function J(pi), instead of the real passwords, those hashed by ph, or pi.

14.4. Usage Considerations

- The user-names inputted by user may be sent automatically to any servers sharing the same auth-domain. This means that when host-type auth-domain is used for authentication in HTTPS site, and when an HTTP server on the same host requests Mutual authentication with the same realm, the client will send the user-name in a clear text. If user-names have to be kept secret against eavesdropping, the server must use full-scheme-type auth-domain parameter. On the contrary, passwords are not exposed to eavesdroppers even on HTTP requests.
- "Pwd_hash" field is only provided for backward compatibility for password databases, and using "none" function is the mostly secure choice and RECOMMENDED. If values other than "none" is used, you must ensure that the hash values of the passwords were not exposed to the public. Note that hashed password databases for plain-text authentications are usually not considered secret.
- If the server provides several ways of storing server-side password database, it is advised to store the values encrypted by one-way function J(pi), instead of the real passwords, those hashed by ph,

or pi.

15. Notice on intellectual properties

The National Institute of Advanced Industrial Science and Technology (AIST) and Yahoo! Japan, Inc. has jointly submitted a patent application about the protocol proposed in this documentation to the Patent Office of Japan. The patent is intended to be open to any implementors of this protocol and its variants under non-exclusive royalty-free manner. For the detail of the patent application and its status, please contact the author of this document.

The elliptic-curve based authentication algorithms might involve several existing patents of third-parties. The authors of the document take no position regarding the validity or scope of such patents, and other patents as well.

16. References

16.1. Normative References

[FIPS.180-2.2002] National Institute of Standards and Technology, "Secure Hash Standard," FIPS PUB 180-2, August 2002.

[FIPS.186-3.2009] National Institute of Standards and Technology, "Digital Signature Standard (DSS)," FIPS PUB 186-3, June 2009.

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

[RFC2818] Rescorla, E., "HTTP Over TLS," RFC 2818, May 2000 (TXT).

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)," RFC 3526, May 2003 (TXT).

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646," STD 63, RFC 3629, November 2003 (TXT).

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," RFC 4648, October 2006 (TXT).

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," STD 68, RFC 5234, January 2008 (TXT).

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

16.2. Informative References

[ISO.10646-1.1993] International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane," ISO Standard 10646-1, May 1993.

[ISO.11770-4.2006] International Organization for Standardization, “Information technology – Security techniques – Key management – Part 4: Mechanisms based on weak secrets,” ISO Standard 11770-4, May 2006.

[ITU.X690.1994] International Telecommunications Union, “Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),” ITU-T Recommendation X.690, 1994.

[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).

[RFC2965] Kristol, D. and L. Montulli, “HTTP State Management Mechanism,” RFC 2965, October 2000 (TXT, HTML, XML).

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” RFC 5280, May 2008 (TXT).

[RFC5890] Klensin, J., “Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework,” RFC 5890, August 2010 (TXT).

[RFC5929] Altman, J., Williams, N., and L. Zhu, “Channel Bindings for TLS,” RFC 5929, July 2010 (TXT).

Appendix A. (Informative) Generic syntax of headers

Several headers (e.g. WWW-Authenticate: headers in 401-B0, 401-B0-stale, and 401-B1 messages) shares common header names. To parse these headers, one MAY use the following general syntax definition of the message syntax:

```
header           = header-name ":" [spaces] auth-scheme
                   spaces fields
header-name      = "WWW-Authenticate" / "Optional-WWW-Authenticate"
                 / "Authorization" / "Authentication-info"
                 / "Authentication-Control"
auth-scheme      = "Mutual"              ; see HTTP for other values
fields           = field *([spaces] "," spaces field)
field            = key "=" value         ; either a specific or
                                         ;        an extension field
key              = extensive-token
token            = 1*(%x30-39 / %x41-5A / %x61-7A / "-" / "_")
extensive-token  = token / extension-token
extension-token  = "-" token 1*("." token)
value            = extensive-token / integer
                 / hex-fixed-number
                 / base64-fixed-number / string
```

```
integer          = "0" / (%x31-39 *%x30-39)      ; no leading zeros
hex-fixed-number = 1*(%x30-39 / %x41-46 / %x61-66)
base64-fixed-number = string
string           = %x22 *(%x20-21 / %x23-5B / %x5D-FF
                          / %x5C.22 / "\\") %x22
spaces           = 1*(" " / %x09)
```

Figure 12: the common BNF syntax for the headers in the protocol

---

In this way of parsing, messages will be distinguished by the fields contained in a header corresponding to the authentication. The procedure below determines the kind of a message which each HTTP request/response belongs to.

- If the message is a response with a "401" status:
  - If it does not contain any WWW-Authenticate header, it is an error.
  - If the WWW-Authenticate header specifies a scheme other than "Mutual", it is a normal response in this draft's scope.
  - Otherwise, the response contains a "WWW-Authenticate: Mutual" header. If the header contains both sid and stale fields, it is an error.
  - If the header contains a stale field with a value of 0, it is a 401-B0 message.
  - If the header contains a stale field with a value of 1, it is a 401-B0-stale message.
  - If the header contains an sid field, it is a 401-B1 message.
- If the message is a response other than a "401" status:
  - If it contains both Authentication-Info and Optional-WWW-Authenticate headers, it is an error.
  - If it contains a Authentication-Info header with a scheme "Mutual", it is a 200-B4 message.
  - If it contains a Optional-WWW-Authenticate header with "Mutual" scheme, it is a 200-Optional-B0 message.
  - If it contains a Optional-WWW-Authenticate header with a scheme other than "Mutual", it is either an error or a normal response, and the behavior is not defined in this specification.
  - Otherwise, it is a normal response.
- If the message is a request:
  - If it does not contain an Authorization header, or it contains an Authorization header with a scheme other than Mutual, it is a normal request.
  - Otherwise, the request contains a "Authorization: Mutual" header. If the header contains an sid field, it is a req-A3 message.
  - If the header do not contain an sid field, it is a req-A1 message.

Implementations MAY perform checks stricter than the procedure above, according to the definitions in Section 3.

Appendix B. (Informative) Group parameters for discrete-logarithm based algorithms

The MODP group used for the iso-kam3-dl-2048-sha256 algorithm is defined by the following parameters.

The prime is:

```
q = 0xFFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
      29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
      EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
      E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
      EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
      C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
      83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
      670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
      E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
      DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
      15728E5A 8AACAA68 FFFFFFFF FFFFFFFF.
```

The generator is:

```
g = 2.
```

The size of the subgroup generated by g is:

```
r = (q - 1) / 2 =
    0x7FFFFFFF FFFFFFFF E487ED51 10B4611A 62633145 C06E0E68
      94812704 4533E63A 0105DF53 1D89CD91 28A5043C C71A026E
      F7CA8CD9 E69D218D 98158536 F92F8A1B A7F09AB6 B6A8E122
      F242DABB 312F3F63 7A262174 D31BF6B5 85FFAE5B 7A035BF6
      F71C35FD AD44CFD2 D74F9208 BE258FF3 24943328 F6722D9E
      E1003E5C 50B1DF82 CC6D241B 0E2AE9CD 348B1FD4 7E9267AF
      C1B2AE91 EE51D6CB 0E3179AB 1042A95D CF6A9483 B84B4B36
      B3861AA7 255E4C02 78BA3604 650C10BE 19482F23 171B671D
      F1CF3B96 0C074301 CD93C1D1 7603D147 DAE2AEF8 37A62964
      EF15E5FB 4AAC0B8C 1CCAA4BE 754AB572 8AE9130C 4C7D0288
      0AB9472D 45565534 7FFFFFFF FFFFFFFF.
```

The MODP group used for the iso-kam3-dl-4096-sha512 algorithm is defined by the following parameters.

The prime is:

```
q = 0xFFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
      29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
      EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
      E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
      EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
      C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
      83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
      670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
      E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
      DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
      15728E5A 8AAAC42D AD33170D 04507A33 A85521AB DF1CBA64
      ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
      ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B
      F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
      BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31
      43DB5BFC E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7
      88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA
      2583E9CA 2AD44CE8 DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6
      287C5947 4E6BC05D 99B2964F A090C3A2 233BA186 515BE7ED
      1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
      93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34063199
      FFFFFFFF FFFFFFFF.
```

The generator is:

```
g = 2.
```

The size of the subgroup generated by g is:

```
r = (q - 1) / 2 =
    0x7FFFFFFF FFFFFFFF E487ED51 10B4611A 62633145 C06E0E68
      94812704 4533E63A 0105DF53 1D89CD91 28A5043C C71A026E
      F7CA8CD9 E69D218D 98158536 F92F8A1B A7F09AB6 B6A8E122
      F242DABB 312F3F63 7A262174 D31BF6B5 85FFAE5B 7A035BF6
      F71C35FD AD44CFD2 D74F9208 BE258FF3 24943328 F6722D9E
      E1003E5C 50B1DF82 CC6D241B 0E2AE9CD 348B1FD4 7E9267AF
      C1B2AE91 EE51D6CB 0E3179AB 1042A95D CF6A9483 B84B4B36
      B3861AA7 255E4C02 78BA3604 650C10BE 19482F23 171B671D
      F1CF3B96 0C074301 CD93C1D1 7603D147 DAE2AEF8 37A62964
      EF15E5FB 4AAC0B8C 1CCAA4BE 754AB572 8AE9130C 4C7D0288
      0AB9472D 45556216 D6998B86 82283D19 D42A90D5 EF8E5D32
      767DC282 2C6DF785 457538AB AE83063E D9CB87C2 D370F263
      D5FAD746 6D8499EB 8F464A70 2512B0CE E771E913 0D697735
      F897FD03 6CC50432 6C3B0139 9F643532 290F958C 0BBD9006
      5DF08BAB BD30AEB6 3B84C460 5D6CA371 047127D0 3A72D598
      A1EDADFE 707E8847 25C16890 54908400 8D391E09 53C3F36B
      C438CD08 5EDD2D93 4CE1938C 357A711E 0D4A341A 5B0A85ED
      12C1F4E5 156A2674 6DDDE16D 826F477C 97477E0A 0FDF6553
      143E2CA3 A735E02E CCD94B27 D04861D1 119DD0C3 28ADF3F6
      8FB094B8 67716BD7 DC0DEEBB 10B8240E 68034893 EAD82D54
      C9DA754C 46C7EEE0 C37FDBEE 48536047 A6FA1AE4 9A0318CC
      FFFFFFFF FFFFFFFF.
```

Appendix C. (Informative) Derived numerical values

This section gives several numerical values for implementing this protocol, derived from the above specifications. The values shown in this section are for informative purpose only.

	dl-2048	**dl-4096**	**ec-p256**	**ec-p521**	
Size of w_A etc.	2048	4096	257	522	(bits)
Size of H(...)	256	512	256	512	(bits)
length of OCTETS(w_A) etc.	256	512	33	66	(octets)
length of wa, wb field values.	346 *	686 *	66	132	(octets)
length of oa, ob field values.	46 *	90 *	64	128	(octets)
minimum allowed s_A	2048	4096	1	1	

(The numbers marked with * include enclosing quotation marks.)

Appendix D. (Informative) Draft Remarks from the Authors

The following items are currently under consideration for future revisions by the authors.

- Restructuring of the draft, possibly separating it to several parts, e.g. introduction, general HTTP extensions and Mutual authentication.
- Format of the "Authentication-Control" header and other header fields extending the general

HTTP authentication scheme, and harmonization of those with other draft proposals.

- Whether to keep TLS-key validation or not.
- When keeping tls-key validation, whether to use "TLS channel binding" [RFC5929] for "tls-key" verification (Section 7). Note that existing implementations of TLS should be considered to determine this.
- Adding test vectors for ensuring implementation correctness.
- Possibly adding a method for servers to detect availability of Mutual authentication on client-side.
- Applying the protocol for proxy authentication/authorization.

Appendix E. (Informative) Draft Change Log

E.1. Changes in revision 07

- Adapt to httpbis HTTP/1.1 drafts:
  - Changed definition of extensive-token.
  - LWSP continuation-line (%0D.0A.20) deprecated.
- To simplify the whole spec, the type of nonce-counter related fields are change from hex-integer to integer.
- Algorithm tokens are renamed to include names of hash algorithms.
- Clarified the session management, added details of server-side protocol decisions.
- The whole draft was reorganized; introduction and overview has been rewritten.

E.2. Changes in revision 06

- Integrated Optional Mutual Authentication to the main part.
- Clarified the decision procedure for message recognitions.
- Clarified that a new authentication request for any sub-requests in interactive clients may be silently discarded.
- Typos and confusing phrases are fixed.
- Several "future considerations" are added.

The field "version" is NOT changed from the previous draft, as the semantics has not been changed.

E.3. Changes in revision 05

- A new field "version" is added for supporting future incompatible changes with a single implementation. In the (first) final specification its value will be changed to 1.
- A new header "Authentication-Control" added for precise control of application-level authentication behavior.

E.4. Changes in revision 04

- Changed text of patent licenses: the phrase "once the protocol is accepted as an Internet standard" is removed so that the sentence also covers the draft versions of this protocol.
- The "tls-key" verification is now OPTIONAL.
- Several description fixes and clarifications.

E.5. Changes in revision 03

- Wildcard domain specifications (e.g. "*.example.com") is allowed for auth-domain parameters (Section 4.1).
- Specification of the "tls-cert" verification is updated (incompatible change).
- State transitions fixed.
- Requirements for servers about w_a values clarified.
- RFC references are updated.

E.6. Changes in revision 02

- Auth-realm is extended to allow full-scheme type.
- A decision diagram for clients and decision procedures for servers are added.
- 401-B1 and req-A3 messages are changed to have authentication realm information.
- Bugs on equations for o_A and o_B is fixed.
- Detailed equations for the whole algorithm is included.
- Elliptic-curve algorithms are updated.
- Several clarifications and other minor updates.

Authors' Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Research Center for Information Security
Room #1003, Akihabara Daibiru
1-18-13 Sotokanda
Chiyoda-ku, Tokyo
JP
Phone: +81 3-5298-4722
Email: mutual-auth-contact@m.aist.go.jp

Hajime Watanabe
National Institute of Advanced Industrial Science and Technology

Hiromitsu Takagi
National Institute of Advanced Industrial Science and Technology

Yuichi Ioku
Yahoo! Japan, Inc.
Midtown Tower
9-7-1 Akasaka
Minato-ku, Tokyo
JP

Tatsuya Hayashi
Lepidum Co. Ltd.
#602, Village Sasazuka 3
1-30-3 Sasazuka

Shibuya-ku, Tokyo
JP