Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2012

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

July 5, 2011

Mutual Authentication Protocol for HTTP draft-oiwa-http-mutualauth-09

Abstract

This document specifies a mutual authentication method for the Hyper-text Transport Protocol (HTTP). This method provides a 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 the 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 January 6, 2012.

Copyright Notice

Table of Contents

1. Introduction

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

The currently available methods for authentication in HTTP and Web systems have several deficiencies. The Basic authentication method [RFC2617] sends a plaintext password to a server without any protection; the Digest method uses a hash function that suffers from simple dictionary-based off-line attacks, and people have begun to think it is obsolete.

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

- It provides "true" mutual authentication: in addition to 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 it impossible for fake website owners to persuade users that thee authenticated with the original websites.
- It uses only passwords as the user's credential: unlike public-key-based security algorithms, the method does not rely on secret keys or other cryptographic data that have to be stored inside the users' computers. The proposed method can be used as a drop-in replacement to the current authentication methods like Basic or Digest, while ensuring a much stronger level of security.
- It is secure: when the server fails to authenticate with a user, the protocol will not reveal any bit of the user's password.

Users can discriminate between true and fake Web servers using their own passwords by using the proposed method. Even when a user inputs his/her password to a fake website owned by illegitimate phishers, the user will certainly notice that the authentication has failed. Phishers will not be successful in their authentication attempts, even if they forward the received data from a user to a 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. A majority of the recent Web-sites on the Internet use custom application-layer authentication implementations using Web forms. The reasons for these may vary, but many people believe that the current HTTP Basic (and Digest, too) authentication method does not have enough functionality (including a good-feeling user interfaces) to support most of realistic Web-based applications. However, the method is very weak against phishing attacks, because the whole behavior of the authentication is controlled from the server side. To overcome this problem, we need to "modernize" the HTTP authentication framework so that better client-controlled secure methods can be used 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The terms "encouraged" and "advised" are used for suggestions that do not constitute "SHOULD"-level requirements. People MAY freely choose not to include the suggested items regarding [RFC2119], but complying with those suggestions would be a best practice; it will improve the security, interoperability, and/or operational 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 computer software; a "user" is a (usually natural) person who wants to access data resources using "a client".

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

1.2. Document Structure Overview

The entire document is organized as follows:

- Section 2 presents an overview of the protocol design.
- Sections 3 to 9 define a general framework of the Mutual authentication protocol. This framework is independent of specific cryptographic primitives.
- Section 10 defines an optional extension to the generic HTTP authentication framework, which is mostly useful for controlling the behavior of the Web browser for the authentication.
- Section 11 describes properties needed for cryptographic algorithms used with this protocol framework, and defines a few functions which will be shared among such cryptographic algorithms.
- The sections after that contain general normative and informative information about the protocol.
- The appendices contain some information that 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 peers are correctly related (i.e. generated from the same password). Then, both peers will verify the authentication results by confirming the sharing of the exchanged key. This section describes a brief image of the protocol and the exchanged messages.

2.1. Messages

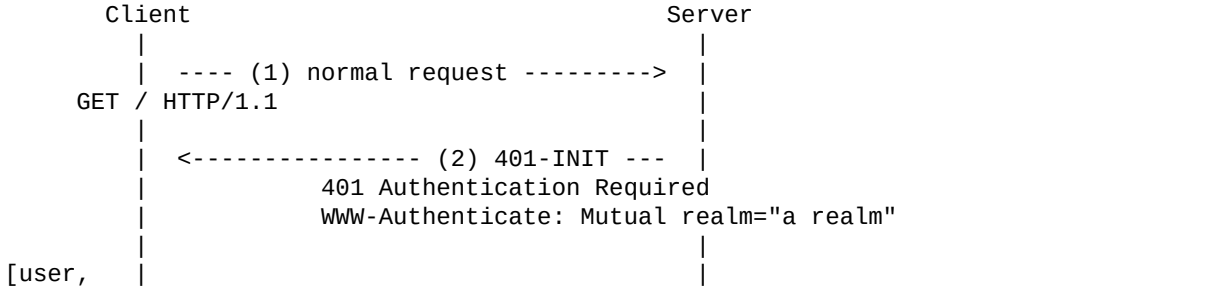
The authentication protocol uses seven kinds of messages to perform mutual authentication. These messages have specific names within this specification.

- Authentication request messages: used by the servers to request clients to start mutual authentication.
  - 401-INIT message: a general message to start the authentication protocol. It is also used as a message indicating an authentication failure.
  - 200-Optional-INIT message: a variant of the 401-INIT message indicating that an authentication is not mandatory.
  - 401-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 the sharing of a cryptographic secret.
  - req-KEX-C1 message: a message sent from the client.
  - 401-KEX-S1 message: a message sent from the server as a response to a req-KEX-C1 message.
- Authentication verification messages: used by both peers to verify the authentication results.
  - req-VFY-C message: a message used by the client, requesting that the server authenticates and authorizes the client.
  - 200-VFY-S message: a successful response used by the server, and also asserting that the server is authentic to the client simultaneously.

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



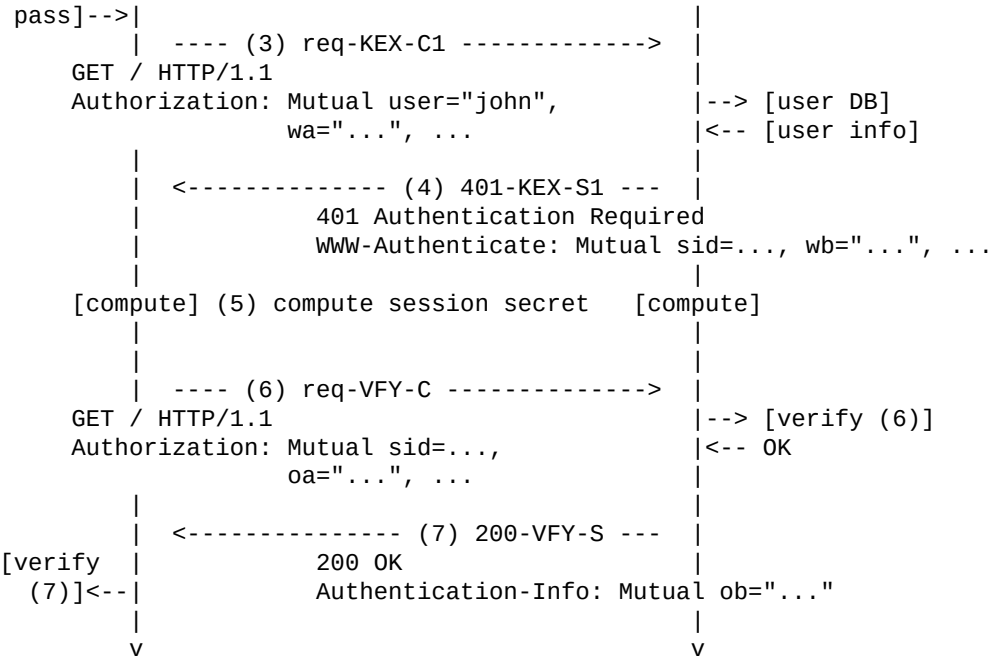


Figure 1: Typical communication flow for first access to resource

---

- As usual in general HTTP protocol designs, a client will at 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-INIT) (2).
- The client processes the body of the message, and waits for the user to input the user name and a password. If the user name and the password are available, the client will send a message with the authenticated key exchange (req-KEX-C1) to start the authentication (3).
- If the server has received a req-KEX-C1 message, the server looks up the user's authentication information within its user database. Then the server creates a new session identifier (sid) that will be used to identify sets of the messages that follow it, and responds back with a message containing a server-side authenticated key exchange value (401-KEX-S1) (4).
- At this point (5), both peers calculate a shared "session secret" using the exchanged values in the key exchange messages. Only when both the server and the client have used secret credentials generated from the same password will the session secret values 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 verification value (req-VFY-C) (6), generated from the client-owned session secret. The server will check the validity of the verification value using its own session secret.
- If the authentication verification value from the client was correct, it means that the client definitely owns the credential based on the expected password (i.e. the client authentication succeeded.) The server will respond with a successful message (200-VFY-S) (7). Contrary to the usual one-way authentication (e.g. HTTP Basic authentication or POP APOP authentication), this message also contains a server-side authentication verification value.
  When the client's verification value is incorrect (e.g. because the user-supplied password was incorrect), the server will respond with the 401-INIT message (the same one as used in (2)) instead.
- The client MUST first check the validity of the server-side authentication verification value contained in the message (7). If the value was equal to the expected one, the server authentication succeeded.

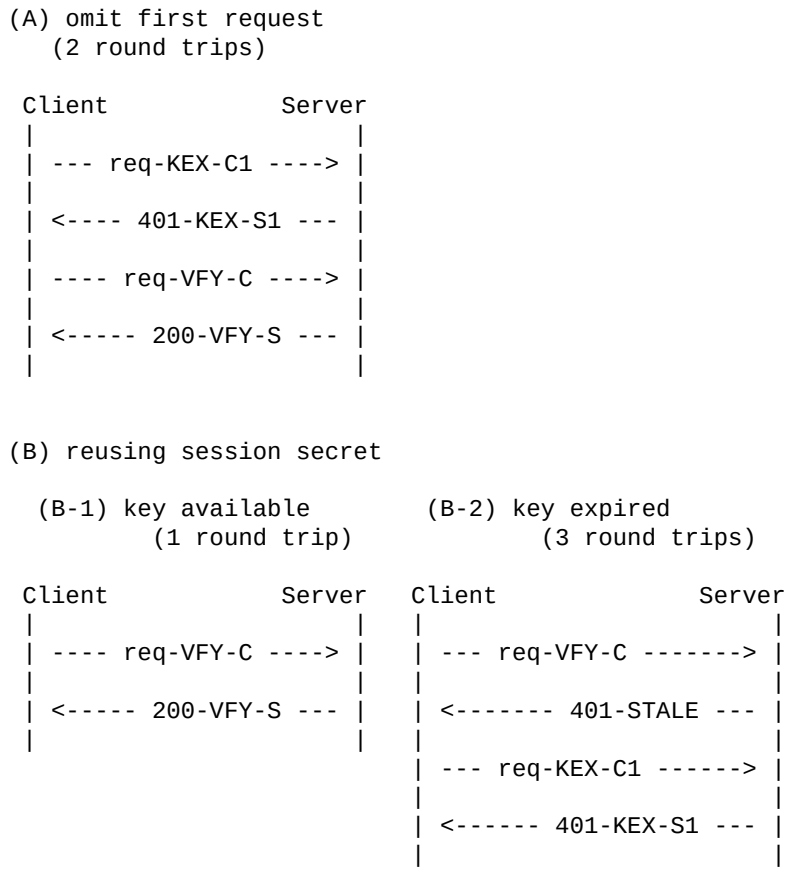
If it is not the value expected, or if the message does not contain the authentication verification value, it means that the mutual authentication has been broken for some unexpected reason. The client MUST NOT process any body or header values contained in this case. (Note: This case should not happen between a correctly-implemented server and a client.)

2.3. Alternative flows

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

- (case A) If the client knows that the resource is likely to require the authentication, the client MAY omit the first unauthenticated request (1) and immediately send a key exchange (req-KEX-C1 message). 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-VFY-C message using the existing session identifier and corresponding session secret. This will further reduce one round-trip of messages.
  In such cases, the server MAY have thrown out the corresponding sessions from the session table. In this case, the server will respond with a 401-STALE message, indicating a new key exchange is required. The client SHOULD retry constructing a req-KEX-C1 message in this case.

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



```
| --- req-VFY-C -------> |
|                        |
| <------- 200-VFY-S --- |
|                        |
```

Figure 2: Several alternative flows on protocol

For more details, see Sections 8 and 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 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: BNF syntax for common elements used in protocol

3.1. Tokens and Extensive-tokens

The tokens are case insensitive; Senders SHOULD send these in lower-case, and receivers MUST accept both upper- and lower-cases. When tokens are used as the (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 with format "-<token>.<domain-name>", where <domain-name> is a validly registered (sub-)domain name on the Internet owned by the party who defines the extensions.

3.2. Numbers

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

The numbers represented as a hex-fixed-number MUST include an even number of characters (i.e. multiples of eight bits). When these are generated from any cryptographic values, they 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 for representing all the elements in the set. For example, any results of SHA-256 hash function will be represented by 64 characters, and any elements in 2048-bit prime field (modulo a 2048-bit integer) will be represented by 512 characters, regardless of how much 0's will be appear in front of such representations. 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 all communications by both peers.

The 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 mentioned above. Then, the string is encoded using the Base 64 encoding [RFC4648] without any spaces and newlines, and then enclosed by two double-quotations.

3.3. Strings

All the strings outside ASCII character sets MUST be encoded using the UTF-8 encoding [RFC3629] for the ISO 10646-1 character set [ISO.10646-1.1993]. Both peers are RECOMMENDED to reject any invalid UTF-8 sequences that might 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, they will first be encoded according to UTF-8 without a leading BOM, then all occurrences of the 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 when using them in a header field.

If strings are representing a domain name or URI that contains non-ASCII characters, the host parts SHOULD be encoded as it is used in the HTTP protocol layer (e.g. in a Host: header); under current standards 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 the seven kinds of messages used in the authentication protocol along with the formats and requirements of the headers for each message.

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

In the descriptions below, the type of allowable values for each header field is shown in parenthesis after the key names. The "algorithm-determined" type means that the acceptable value 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 "mandatory" SHALL be contained in the message. The fields marked "non-mandatory" MAY either be contained or omitted in the message. Each field SHALL appear in each headers exactly once at most.

4.1. 401-INIT

Every 401-INIT 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-INIT = "WWW-Authenticate" ":" [spaces]
                  auth-scheme spaces fields-401-INIT
fields-401-INIT = field-401-INIT *([spaces] "," spaces field-401-INIT)
field-401-INIT  = 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: BNF syntax for header in 401-INIT header

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

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

algorithm:
: (mandatory extensive-token) specifies the authentication algorithm to be used. The value MUST be one of the tokens specified in &algo_draft; or other supplemental specification documentation.

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 documentation.

auth-domain:
: (non-mandatory string) specifies the authentication domain, the set of hosts for which the 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-STALE

A 401-STALE message is a variant of the 401-INIT message, which means that the client has sent a request message that 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-INIT, except that the stale field contains token 1.

4.3. req-KEX-C1

Every req-KEX-C1 message SHALL be a valid HTTP request message containing an "Authorization" header of the following format.

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

```
header-req-KEX-C1 = "Authorization" ":" [spaces]
                    auth-scheme spaces fields-req-KEX-C1
fields-req-KEX-C1 = field-req-KEX-C1
                    *([spaces] "," spaces field-req-KEX-C1)
field-req-KEX-C1  = 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-KEX-C1 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 is undefined when other values are specified.

algorithm, validation, auth-domain, realm:

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

user:

(mandatory, string) is the UTF-8 encoded name of the user. If this name comes from a user input, client software SHOULD prepare the string using SASLprep [RFC4013] before encoding it to UTF-8.

wa:

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

4.4. 401-KEX-S1

Every 401-KEX-S1 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-KEX-S1 = "WWW-Authenticate" ":" [spaces]
                    auth-scheme spaces fields-401-KEX-S1
fields-401-KEX-S1 = field-401-KEX-S1
                    *([spaces] "," spaces field-401-KEX-S1)
field-401-KEX-S1  = 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-KEX-S1 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 is undefined when other values are specified.

algorithm, validation, auth-domain, realm:

MUST be the same value as it is when 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. Session identifiers are local to each concerned authentication realm: 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.

nc-max:

(mandatory, integer) is the maximal value of nonce counts that the server accepts.

nc-window:

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

time:

(mandatory, integer) represents the suggested time (in seconds) that the client can reuse the session represented by the 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 the sid.

path:

(non-mandatory, string) specifies which path in the URI space the same authentication is expected to be applied. The value is a space-separated list of URIs, in the same format as it was 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-VFY-C

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

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

```
header-req-VFY-C = "Authorization" ":" [spaces]
                   auth-scheme spaces fields-req-VFY-C
fields-req-VFY-C = field-req-VFY-C
                   *([spaces] "," spaces field-req-VFY-C)
field-req-VFY-C  = 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-VFY-C message

The fields contained in the header are as follows:

version:

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

algorithm, validation, auth-domain, realm:

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

sid:

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

nc:

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

oa:

(mandatory, algorithm-determined) is the client-side authentication verification value o_A, which is specified by the algorithm.

4.6. 200-VFY-S

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

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

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

Figure 8: BNF syntax for header in 200-VFY-S message

The fields contained in the header are as follows:

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

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

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

logout-timeout:
: (non-mandatory, integer) is the number of seconds after which the client should re-validate the user's password for the current authentication realm. The value 0 means that the client SHOULD automatically forget the user-inputted password for 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 the password reminders and auto fill-ins) should be removed. If a new timeout value 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 the 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-INIT

The 200-Optional-INIT messages enable a non-mandatory authentication, which is not possible under the current HTTP authentication mechanism. In several Web applications, users can access the same contents as both a guest user and an authenticated user. In most Web applications, it is implemented using HTTP cookies [RFC6265] and custom form-based authentications. The new authentication method using this message will provide a replacement for these authentication systems. Support for this message is RECOMMENDED, unless the protocol is used for some specific applications in which the 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-INIT responses (with one exception described below). Such responses are hereafter called 200-Optional-INIT responses.

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

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

Figure 9: BNF syntax for header in 200-Optional-INIT header

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

The 200-Optional-INIT message is the only place where an Optional-WWW-Authenticate header is allowed. If a server is supposed to send a 401-KEX-S1 or a 401-STALE response, it SHALL NOT replace it with 200-Optional-INIT or similar responses. Furthermore, if a server is going to send a 401-INIT message as a response to a req-VFY-C message with a correct realm, the server MUST send a 401-INIT message, not a 200-Optional-INIT message.

Servers requesting non-mandatory authentication SHOULD send the path field in the 401-KEX-S1 messages with an appropriate value. Clients supporting non-mandatory authentication MUST recognize the field, and MUST send either a req-KEX-C1 or a req-VFY-C 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 an authentication realm that the client is already authenticated for, the client will automatically perform the authentication using the already-known secrets. However, for the different authentication realms, the clients SHOULD NOT automatically reuse the usernames and passwords for another realm.

Just like in Basic and Digest access authentication protocols, 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 within 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. However, server operators are NOT RECOMMENDED to use the same pair of an authentication domain and a realm for different authentication algorithms.

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

An authentication domain specifies the range of hosts that the authentication realm spans over. In this 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 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 within this kind of authentication domain will span over several protocols (i.e. http and https) and ports, but not over different hosts.
- The 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 the servers MUST be equal to or included in a valid Internet domain assigned to a specific organization: if clients know, by some means such as a blacklist for HTTP cookies, that the specified domain is not to be assigned to any specific organization (e.g. "*.com" or "*.jp"), the 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 they 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 definitions of authentication domains, several domains will overlap each other. Depending on the "path" parameters given in the "401-KEX-S1" message (see Section 4), there may be several candidates when the client is going to send a request including an authentication credential (Steps 3 and 4 of the decision procedure presented 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-INIT messages at that time, use that realm.
- In other cases, use one of authentication realms representing the most-specific authentication domains. From the list of possible domain specifications shown above, each one has priority over ones described after that.
  If there are several choices with different domain-postfix specifications, the one that has the longest domain-postfix has priority over ones with a shorter domain-postfix.
- If there are realms with the same authentication domain, there is no defined priority: the client MAY choose any one of the possible choices.

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

6. Session Management

In the Mutual authentication protocol, a session represented by an sid is set up using first four messages (first request, 401-INIT, req-KEX-C1 and 401-KEX-S1), and a "session secret" (z) associated with the session is established. After sharing 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 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 informing the users.

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

The server SHOULD accept at least one req-VFY-C request for each session, given that the request reaches the server in a time window specified by the timeout field in the 401-KEX-S1 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-STALE messages for any req-VFY-C 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 was not sent within the same session.
- It is not larger than the nc-max value that was 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 was previously sent in the session, and nc-window is the value of the nc-window field which was received from the server in the session.

The last condition allows servers to reject any nonce values that are "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 with a 401-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-STALE message.

For example, assume the nc-window value of the current session is 32, nc-max is 100, and that the client has already used the following nonce values: {1-20, 22, 24, 30-38, 45-60, 63-72}. Then the nonce values that 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 they are not above the current "window limit" (40 = 72 - 32).

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

The values of the nonces and any nonce-related values MUST always be treated as natural numbers within an infinite range. Implementations using fixed-width integers or fixed-precision floating numbers MUST correctly and carefully handle integer overflows. Such implementations are RECOMMENDED to accept any larger values that 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 the clients and servers can implement the protocol using only 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 v that is an input to the 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 corresponding to the currently accessing resource. The scheme and host are in 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 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 the underlying TLS [RFC5246] (or SSL) connection. The hash value is defined as the value of the entire 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 the underlying TLS (or SSL) connection.

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

If the validation type "tls-cert" is used, the server certificate provided on TLS connection MUST be verified to make sure that the server actually owns the corresponding secret key.

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

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

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

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

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

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

8. Decision procedure for client

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

Clients SHOULD implement a decision procedure equivalent to the one shown below. (Unless implementers understand what is required for the security, they should not alter this.) In particular, clients SHOULD NOT accept "normal responses" unless explicitly allowed below. The labels on the steps are for informational purposes only. Action 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 access a new Web resource, check whether the resource is expected to be inside some authentication realm for which the user has already been 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_vfy_1):
: Send a req-VFY-C request.
  - If you receive a 401-INIT message with a different authentication realm than expected, go to Step 6.
  - If you receive a 200-Optional-INIT message with a different authentication realm than expected, go to Step 6.
  - If you receive a 401-STALE message, go to Step 9.
  - If you receive a 401-INIT message, go to Step 13.
  - If you receive a 200-VFY-S message, go to Step 14.
  - If you receive a normal response, go to Step 11.

Step 4 (step_send_kex1_1):
: Send a req-KEX-C1 request.
  - If you receive a 401-INIT message with a different authentication realm than expected, go to Step 6.
  - If you receive a 200-Optional-INIT message with a different authentication realm than expected, go to Step 6.
  - If you receive a 401-KEX-S1 message, go to Step 10.
  - If you receive a 401-INIT 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-INIT message, go to Step 6.
- If you receive a 200-Optional-INIT message, go to Step 6.
- If you receive a normal response, go to Step 11.

Step 6 (step_rcvd_init):

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_vfy):

Send a req-VFY-C request.

- If you receive a 401-STALE message, go to Step 9.
- If you receive a 401-INIT message, go to Step 13.
- If you receive a 200-VFY-S message, go to Step 14.

Step 9 (step_send_kex1):

Send a req-KEX-C1 request.

- If you receive a 401-KEX-S1 message, go to Step 10.
- If you receive a 401-INIT message, go to Step 13 (See Note 1).

Step 10 (step_rcvd_kex1):

Send a req-VFY-C request.

- If you receive a 401-INIT message, go to Step 13.
- If you receive a 200-VFY-S message, go to Step 14.

Step 11 (step_rcvd_normal):

The requested resource 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_init_unknown):

The requested resource requires a Mutual authentication, and the user is not yet authenticated. The client will be in the "AUTH_REQUESTED" status, and is RECOMMENDED to process the content sent from the server, and to ask user for a user name and a password. When those are supplied from the user, proceed to Step 9.

Step 13 (step_rcvd_init_failed):

For 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_vfy):

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

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

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

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 a fatal communication error, and in such cases the clients MUST NOT process any data (response body and other content-related headers) sent from the server. However, to handle 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 in 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 a diagram of the client-side state.

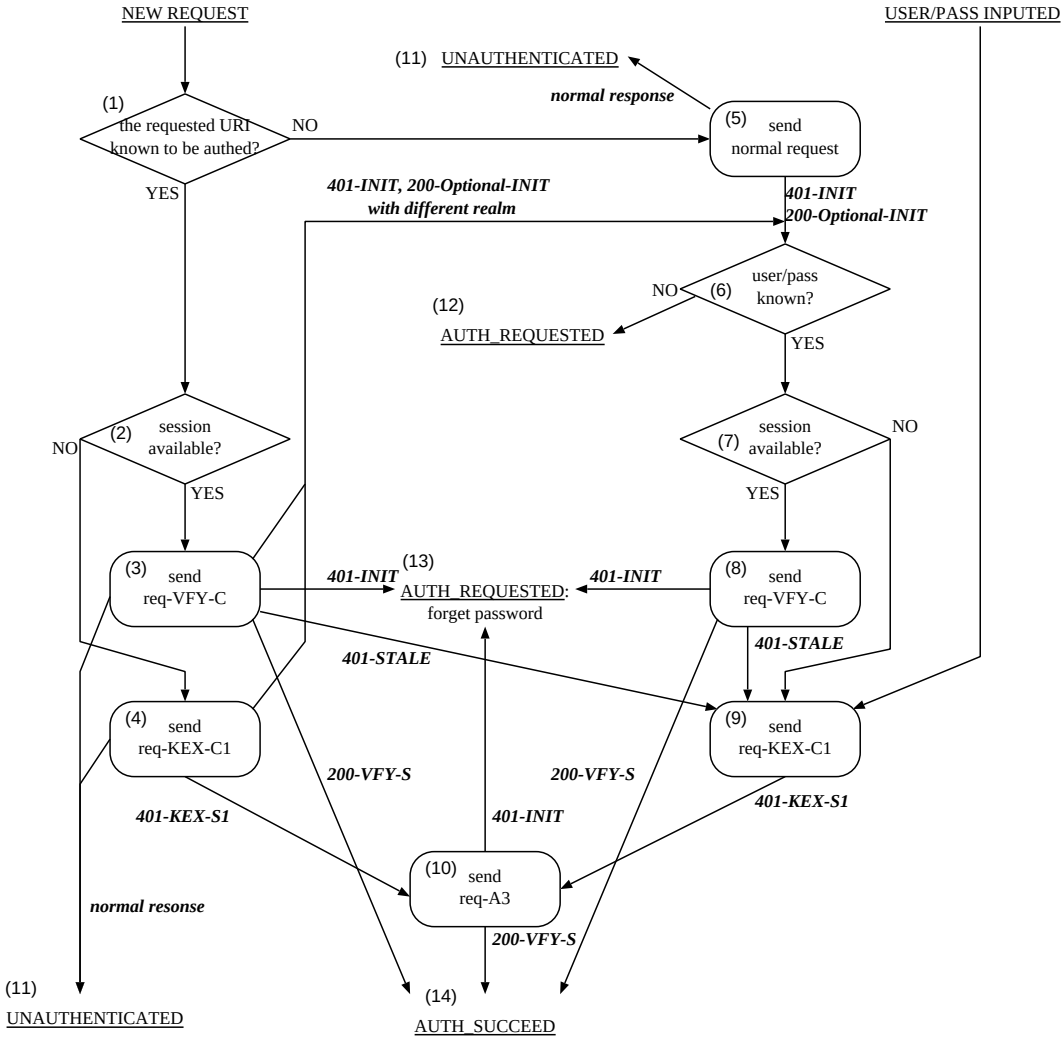


Figure 10: State diagram for clients

9. Decision procedure for the server

Each server SHOULD have a table of session states. This table need not be persistent over a long term; it MAY be cleared upon server restart, reboot, or others. Each entry in 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", or "inactive".
- The user name received from the client
- The boolean flag noting whether or not 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 or not a request with the 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-INIT response.
  - If the resource is protected by the optional Mutual Authentication, send a 200-Optional-INIT response.
- When the server receives a req-KEX-C1 request:
  - If the requested resource is not protected by the Mutual Authentication, send a normal response.
  - If the authentication realm specified in the req-KEX-C1 request is not the expected one, send either a 401-INIT or a 200-Optional-INIT response.
  - If the server cannot validate the field wa, send a 401-INIT response.
  - If the received user name is either invalid, unknown or unacceptable, create a new session, mark it a "fake" session, compute a random value as wb, and send a fake 401-KEX-S1 response. (Note: the server SHOULD NOT send a 401-INIT 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-VFY-C request.)
  - Otherwise, create a new session, compute wb and send a 401-KEX-S1 response.

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

  If none of above holds true, 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-STALE response.
- If the session is in the "rejected" state, send either a 401-INIT or a 401-STALE message.
- If the session is a "fake" session, or if the received oa is incorrect, then send a 401-INIT response. If the session is in the "wa received" state, it SHOULD be changed to the "rejected" state; otherwise, it MAY either be changed to the "rejected" status or kept in the previous state.
- If the session is in the "active" state, and the request has an nc value that was previously received from the client, send a 401-STALE message. The session SHOULD be changed to the "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 to) send a 401-STALE message. The session SHOULD be changed to the "inactive" status if so.
- Otherwise, send a 200-VFY-S response. If the session was in the "wa received" state, the session SHOULD be changed to an "authenticated" state. The maximum nc and nc flags of the state SHOULD be updated properly.

At any time, the server MAY change any state entries with both the "rejected" and "authenticated" statuses to the "inactive" status, and MAY discard any "inactive" states from the table. The entries with the "wa received" status SHOULD be kept unless there is an emergency situation such as a server reboot or a 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 provides a more precise control of the client behavior for Web applications using the Mutual authentication protocol. This header will usually be generated in the application layer, as opposed to WWW-Authenticate headers which will be generated by the Web servers.

Support of this header is RECOMMENDED for interactive clients and not required for non-interactive clients. Web applications SHOULD consider the security impacts of the behaviors of clients that 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 where any unauthenticated clients should be redirected to. This header may be used, for example, when there is a central login page for the entire Web application. The value of this field MUST be a string that contains an absolute URL location. If a given URL is not absolute, the clients MAY consider it a relative URL from the current location.

This field MAY be used with a 401-INIT and 200-Optional-INIT message; however, use of this field with 200-Optional-INIT messages is not recommended. If there is a 200-VFY-S, 401-STALE or 401-KEX-S1 message with this field, the clients MUST ignore this field.

When a client receives a message with this field, if and only if the client's state after processing the response is either Step 12 or 13 (i.e., a state in which the client will process the response body and ask for the user's password), the client will treat the entire response as if it were a 303 "See Other" response with a Location header that contains 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 process authentication without the 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-INIT message. If this is not satisfied, the 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 the user explicitly request a logout. The value of this field MUST be a string that contains an absolute URL location. If a given URL is not absolute, the clients MAY consider it a relative URL from the current location.

This field MAY be used with 200-VFY-S messages. If there is a 401-INIT, 401-KEX-S1, 401-STALE, 200-Optional-INIT or normal 200 message with this field, the clients MUST ignore this field.

When the user 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 (as if it received a 303 response), instead of reloading the page without the authentication credential. Web applications are encouraged 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 the "Authentication-Info" header. This field will be used with 200-VFY-S messages. If both are specified, clients are RECOMMENDED to use the one with the smaller value.

11. Authentication Algorithms

Cryptographic authentication algorithms which are used with this protocol will be defined separately. The algorithm definition MUST at least provide a definitions for the following functions:

- The server-side authentication credential J, derived from user-side authentication credential pi.
- Key exchange values w_A and w_B.
- Shared secret z, to be computed in both server-side and client side.
- A hash function H to be used with the protocol.

All algorithm used with this protocol SHOULD provide secure mutual authentication between client and servers, and generate a cryptographically strong shared secret value z, equivalently strong to or stronger than the hash function H. If any passwords (or pass-phrases or any equivalents, i.e. weak secrets) are involved, these SHOULD NOT be guessable from any data transmitted in the protocol, even if an attacker (either an eavesdropper or an active server) knows the possible thoroughly-searchable candidate list of the passwords. Furthermore, if possible, the function for deriving server-side authentication credential J is RECOMMENDED to be one-way so that pi should not be easily computed from J(pi).

11.1. Support functions and notations

In this section we define several support functions and notations to be shared by several algorithm definitions:

The integers in the specification are in decimal, or in 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 the 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 a variable-length octet string into a 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 to be encoded.]

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

11.2. Default functions for algorithms

The functions defined in this section are common default functions among authentication algorithms.

The client-side 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-INIT (or 200-Optional-INIT, 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 determined by the value of the pwd-hash field given in a 401-INIT message. If the password comes from a user input, it SHOULD first be prepared using SASLprep [RFC4013]. Then, 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))

Specifications for cryptographic algorithms used with this framework MAY override the functions pi, o_A, and o_B defined above. In such cases implementations MUST use the ones defined with such algorithm specifications.

12. Methods to extend this protocol

If a non-standard extension to 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 the fields "wa", "wb", "oa", and "ob", replace those keys, and/or add fields to the messages containing those fields in 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 those mentioned 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 the 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 the future drafts to meet the 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 provides true protection against active man-in-the-middle attacks.
- Even if the server certificate is not used or is unreliable, the protocol provides 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 the 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-KEX-C1 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-VFY-S messages MUST always be validated by the client. If the validation fails, the client MUST NOT process any content sent with the message, including other headers and 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 for the user's name and passwords SHOULD be carefully designed so that (1) the user can easily distinguish the request from this authentication method from any other 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 "tls-cert" validation method to a URI that is protected by the same realm (so indicated by the path field), if the server certificate has been changed since the pages were received, the peer is RECOMMENDED to be revalidated using a req-KEX-C1 message with an "Expect: 100-continue" header. The same applies when the page is received with the "tls-key" validation method, and when the TLS session has expired.

- Server-side storages of user passwords are advised to contain 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 a 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 on an HTTPS site, and when an HTTP server on the same host requests Mutual authentication within 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. Contrarily, passwords are not exposed to eavesdroppers even on HTTP requests.
- The "Pwd_hash" field is only provided for backward compatibility of password databases. The use of "none" function is the most secure choice and is RECOMMENDED. If values other than "none" are 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 for storing server-side password secrets into the password database, it is advised to store the values encrypted by using the 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 on 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 details of the patent application and its status, please contact the author of this document.

The elliptic-curve based authentication algorithms might involve several existing third-party patents. 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

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

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

[RFC4013] Zeilenga, K., “SASLprep: Stringprep Profile for User Names and Passwords,” RFC 4013, February 2005 (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.

[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).

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

[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).

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

Appendix A. (Informative) Generic syntax of headers

Several headers (e.g. WWW-Authenticate: headers in 401-INIT, 401-STALE, and 401-KEX-S1 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: Common BNF syntax for 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 each HTTP request/response.

- 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 within 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-INIT message.
  - If the header contains a stale field with a value of 1, it is a 401-STALE message.
  - If the header contains an sid field, it is a 401-KEX-S1 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 the "Mutual" scheme, it is a 200-VFY-S message.
  - If it contains a Optional-WWW-Authenticate header with the "Mutual" scheme, it is a 200-Optional-INIT 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-VFY-C message.

- If the header do not contain an sid field, it is a req-KEX-C1 message.

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

Appendix B. (Informative) Draft Remarks from Authors

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

- Restructuring of the draft, further separating the HTTP extensions part from this document.
- 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 TLS implementations 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.
- Possible support for optional key renewal and cross-site federated authentication.

Appendix C. (Informative) Draft Change Log

C.1. Changes in revision 09

Note: the token for the header field "version" is NOT changed from draft-07, because the protocol semantics has not been changed in this revision.

- The (default) cryptographic algorithms are separated to another draft.
- Names of the messages are changed to more informative one (like KEX-C for client key-exchange) than before.

C.2. Changes in revision 08

- The English text has been revised.

C.3. 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.

C.4. 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.

C.5. Changes in revision 05

- A new field called "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" is added for precise control of application-level authentication behavior.

C.6. 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.

C.7. Changes in revision 03

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

C.8. 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-KEX-S1 and req-VFY-C messages are changed to contain authentication realm information.
- Bugs on equations for o_A and o_B are fixed.
- Detailed equations for the entire algorithm are 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