HTTPAUTH Working Group | Y. Oiwa
Internet-Draft | H. Watanabe
Intended status: Experimental | H. Takagi
Expires: January 2, 2014 | RISEC, AIST
B. Kihara
T. Hayashi
Lepidum
Y. Ioku
Yahoo! Japan
July 1, 2013

Mutual Authentication Protocol for HTTP
draft-ietf-httpauth-mutual-00

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

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uses a hash function that suffers from simple dictionary-based off-line attacks, and people have begun to think it is obsolete.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. This makes it impossible for fake website owners to persuade users that they have 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, along with [I-D.ietf-httpauth-extension], also proposes several extensions to the current HTTP authentication framework, to replace current widely-used form-based Web authentication. The extensions provided include:

- Multi-host single authentication within an Internet domain (Section 5),
- non-mandatory, optional authentication on HTTP (Section 8),
- log out from both server and client side (Section 8), and
- finer control for redirection depending on authentication status (Section 8).

1.1. Relations to other technologies

1.1.1. Technologies updated or superceded by this proposal

1.1.1.1. HTTP Basic and Digest authentication

The main purpose of this proposal is obviously providing an upgrade for the two existing HTTP authentication methods, Basic and Digest [RFC2617].

HTTP Basic authentication, as its name suggests, provides very simple authentication mechanism using plain-text password directly upon the HTTP transport. HTTP Digest authentication focuses on mitigating the fundamental weakness of Basic authentication by using MD5-based hashing to the authentication, but that has almost failed to deploy due to improper implementations, interoperability problems, and missing feature implementations before MD5 has deprecated by its cryptographic weakness. Digest also has a fundamental problem that the server-side must posses a password-equivalent to perform authentication, which increases risks of server-side data leakage.

1.1.1.2. HTML Form authentication

Another aim of this protocol is (at least) partially replacing the HTML form authentication. Because of inflexibility of the HTTP Basic authentication, recent Web applications tend to use application-level implementations for user authentication using HTML Forms and Web browser rendering engines. However, that method has many potential security weaknesses as same as the HTTP Basic authentication as it uses plaintext. Considering server-impersonations and existence of human-forging rogue servers (i.e. phishing), script-based implementations of hash-based authentication does not help, because its behavior is completely controlled by the web-page content itself, which is possibly provided by such a rogue server. This also closes any possibilities for extending HTML forms to implement cryptography, as its user-interface could not be prevented from being imitated using plain-text forms. Using HTTP-level authentication is better in this field, because it is under the control of the client software (Web browsers), which can enforce security checks regardless of server-provided contents.

Of course, we could not ignore the strong reasons of favoring Form authentication over Basic authentication: its flexibility. HTTP authentication framework lacks many features for recent Web applications, mainly for interactions between HTTP-level authentications and application-level management of "authentication sessions". As long as current HTTP-layer (and lower-layer) authentication are used, the new method would share the same problem. To solve this problem, this protocol has a companion mechanism for application-level control of authentication behaviors as a separate draft [I-D.ietf-httpauth-extension]. By using this additional mechanism, Web applications can implement most of these required features as easy as just calling an already-provided API for them.

1.1.2. Technologies not updated by this proposal

1.1.2.1. Federated identity/authorization management

There are several technologies (protocols, frameworks, or systems) for managing authentications/authorizations involving multiple-parties: some of those examples are OAuth [RFC6749], OpenID Connect [OIDF.Connect.Standard], SAML [OASIS.saml-core-2.0-os] etc. These technologies can be further divided to two categories: federated authentication and authorization delegation, although some of these technologies cover both.

Federated authentication provides so-called "three-legged authentication": provided the result of user authentication to a single entity (identity provider) and the user’s consent, the mechanism can provide other entities assertion of the user’s identity without performing a separate identity management by every entity. Authorization delegation gives a mechanism for transferring a part of the user’s privilege on an entity (resource owners) to another entity without requiring users give away the full credential for the authentication.

Essentially, both of those technologies are transforming a result of conventional, one-by-one (two-legged) authentication into a multi-party privilege management. The purpose of this protocol is to secure the very part of the two-legged authentication, and so it can be naturally combined with existing federated management frameworks for increasing security of the entire system.

Additionally, this protocol can provide a secure peer-to-peer shared key generated during authentication to the higher-layer applications Section 12. These keys can be possibly used by such federating mechanisms in future for simplifying/securing the framework.

1.1.2.2. HTTPS and HTTPS client-certificate authentication

This protocol will not replace the wide-spread and widely-accepted technology of SSL/TLS and HTTPS [RFC2818]. This protocol will be still relying on the HTTPS for the integrity and secrecy of the HTTP payload. This protocol ensures users the integrity and secrecy of the authentication credentials, and authenticity of the talking peer server.

Client certificate (and other public-key-based) authentications have a fair-amount of applications (mainly for high-assurance applications), and there are possible needs for redesigning/updating the whole framework. However, currently public-key-based user-authentication and connection-based user identification is out-of-scope of this proposal.

1.1.2.3. Protocols for local identity-management frameworks

There are several existing frameworks for managing user identity of tightly-managed, closed group of users, such as Kerberos [RFC3961] / GSS-API [RFC2743] etc. Some of these have defined a bridging protocol for HTTP authentication. This protocol does not currently aim to replace such existing frameworks.

More precisely, requirements for those framework and usual Web user authentication differ fundamentally. In such framework, user authentication in performed first, and the result of the authentication tends to be shared in all applications, sometimes even shared regardless of the underlying protocols. In those systems, it is almost never likely to use a multiple identity to be used inside a single server and inside a single client machine at the same time. In such applications, connection-based or even a machine-based authentication can be used without a trouble. This is not a case for the general Web authentication applications.

1.1.2.4. HTTP and HTTP authentication architecture

Although HTTP and generic HTTP authentication architecture lacks some required features (see above), the whole structure of per-request, per-resource authentication is well-suited for general Web applications compared with connection-based or machine-based authentication/authorization framework (those which tie user identity to either connections or machines). The whole protocol in this specification is designed on top of the framework of [I-D.ietf-httpbis-p7-auth]. Small extensions to the framework in this specification and [I-D.ietf-httpauth-extension], which are designed for filling the missing features, are carefully designed so that it can be implemented easily only by the client-side without changing the whole framework.

1.2. 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 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.

This document treats target (codomain) of hash functions to be natural numbers. The notation OCTETS(H(s)) gives a usual octet-string output of hash function H applied to string s.

1.3. Document Structure and Related Documents

The entire document is organized as follows:

- Section 2 presents an overview of the protocol design.
- Sections 3 to 10 define a general framework of the Mutual authentication protocol. This framework is independent of specific cryptographic primitives.
- 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.

In addition, there are two companion documents which are referred from/related to this specification:

- [I-D.oiwa-httpauth-mutual-algo]: defines a cryptographic primitives which can be used with this protocol framework. [draft note: it is separated from this document so that it may be replaced with another crypto in future.]
- [I-D.ietf-httpauth-extension]: defines a small but useful extensions to the current HTTP authentication framework so that it can support application-level semantics of existing Web systems.

2. Protocol Overview

The protocol, as a whole, is designed as a natural extension to the HTTP protocol [I-D.ietf-httpbis-p1-messaging] using a framework defined in [I-D.ietf-httpbis-p7-auth]. 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 Overview

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.

```
       Client                                 Server
         |                                       |
         |  ---- (1) normal request --------->  |
     GET / HTTP/1.1                              |
         |                                       |
         |  <---------------- (2) 401-INIT ---  |
         |            401 Authentication Required
         |            WWW-Authenticate: Mutual realm="a realm"
         |                                       |
[user,   |                                       |
 pass]-->|                                       |
         |  ---- (3) req-KEX-C1 ------------->  |
     GET / HTTP/1.1                              |
     Authorization: Mutual user="john",          |--> [user DB]
                    kc1="...", ...               |<-- [user info]
         |                                       |
         |  <-------------- (4) 401-KEX-S1 ---  |
         |           401 Authentication Required
         |           WWW-Authenticate: Mutual sid=..., ks1="...", ...
         |                                       |
     [compute] (5) compute session secret   [compute]
         |                                       |
         |                                       |
         |  ---- (6) req-VFY-C -------------->  |
     GET / HTTP/1.1                              |--> [verify (6)]
     Authorization: Mutual sid=...,              |<-- OK
                    vkc="...", ...               |
         |                                       |
         |  <--------------- (7) 200-VFY-S ---  |
[verify  |           200 OK                      |
  (7)]<--|           Authentication-Info: Mutual vks="..."
         |                                       |
         v                                       v
```

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,the session secret values will match. This session secret will be used for access authentication of every individual request 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.

```
(A) omit first request
   (2 round trips)

 Client             Server
```

```
 |                       |
 | --- req-KEX-C1 ----> |
 |                       |
 | <---- 401-KEX-S1 --- |
 |                       |
 | ---- req-VFY-C ----> |
 |                       |
 | <----- 200-VFY-S --- |
 |                       |


(B) reusing session secret (re-authentication)

  (B-1) key available        (B-2) key expired
         (1 round trip)             (3 round trips)

 Client            Server   Client              Server
 |                       |   |                         |
 | ---- req-VFY-C ----> |   | --- req-VFY-C -------> |
 |                       |   |                         |
 | <----- 200-VFY-S --- |   | <------- 401-STALE --- |
 |                       |   |                         |
                             | --- req-KEX-C1 ------> |
                             |                         |
                             | <------ 401-KEX-S1 --- |
                             |                         |
                             | --- req-VFY-C -------> |
                             |                         |
                             | <------- 200-VFY-S --- |
                             |                         |
```

Figure 2: Several alternative flows on protocol

For more details, see Sections 9 and 10.

3. Message Syntax

Throughout this specification, The syntax is denoted in the extended augmented BNF syntax defined in [I-D.ietf-httpbis-p1-messaging] and [RFC5234]. The following elements are quoted from [RFC5234], [I-D.ietf-httpbis-p1-messaging] and [I-D.ietf-httpbis-p7-auth]: DIGIT, ALPHA, SP, auth-scheme, quoted-string, auth-param, header-field, token, challenge, and credential.

The Mutual authentication protocol uses three headers: WWW-Authenticate (in responses with status code 401), Authorization (in requests), and Authentication-Info (in responses other than 401 status). These headers follow a common framework described in [I-D.ietf-httpbis-p7-auth]. The detailed meanings for these headers are contained in Section 4.

The framework in [I-D.ietf-httpbis-p7-auth] defines the syntax for the headers WWW-Authenticate and Authorization as the syntax elements "challenge" and "credentials", respectively. The "auth-scheme" contained in those headers MUST be "Mutual" throughout this protocol specification. The syntax for "challenge" and "credentials" to be used with the "Mutual" auth-scheme SHALL be name-value pairs (#auth-param), not the "b64token" defined in [I-D.ietf-httpbis-p7-auth].

The Authentication-Info: header used in this protocol SHALL contain the value in same syntax as those the "WWW-Authenticate" header, i.e. the "challenge" syntax element.

In HTTP, the WWW-Authenticate header may contain more than one challenges. Client implementations SHOULD be aware of and be capable of handle those cases correctly.

3.1. Values

The parameter values contained in challenge/credentials MUST be parsed strictly conforming to the HTTP semantics (especially un-quoting of the string parameter values). In this protocol, those values are further categorized into the following value types: tokens (bare-token and extensive-token), string, integer, hex-fixed-number, and base64-fixed-number.

For clarity, implementations are encouraged to use the canonical representations specified in the following subsections for sending values. Recipients SHOULD accept both quoted and unquoted representations interchangeably as specified in HTTP.

3.1.1. Tokens

For sustaining both security and extensibility at the same time, this protocol defines a stricter sub-syntax for the "token" to be used. The extensive-token values SHOULD follow the following syntax (after HTTP value parsing):

```
bare-token       = 1*(DIGIT / ALPHA / "-" / "_")
extension-token  = "-" bare-token 1*("." bare-token)
extensive-token  = bare-token / extension-token
```

Figure 3: BNF syntax for token values

The tokens (bare-token and extension-token) are case insensitive; Senders 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.

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 SHOULD use the extension-tokens with format "-<bare-token>.<domain-name>", where <domain-name> is a validly registered (sub-)domain name on the Internet owned by the party who defines the extensions.

Bare-tokens and extensive-tokens are also used for parameter names (of course in the unquoted form). Requirements for using the extension-token for the parameter names are the same as the above.

The canonical format for bare-tokens and tokens are unquoted tokens.

3.1.2. Strings

All character 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], without any leading BOM characters. 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).

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.

The canonical format for strings are quoted-string (as it may contain equal sign, plus sign and slashs).

3.1.3. Numbers

The following syntax definitions gives a syntax for number-type values:

```
integer           = "0" / (%x31-39 *DIGIT)       ; no leading zeros
hex-fixed-number = 1*(2(DIGIT / %x41-46 / %x61-66))
base64-fixed-number = 1*( ALPHA / DIGIT /
                         "-" / "." / "_" / "~" / "+" / "/" ) *"="
```

Figure 4: BNF syntax for number types

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). Those values are case-insensitive, and SHOULD be sent in lower-case. When these values 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 radix-256 binary representation as an octet string. 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. Implementations decoding base64-fixed-number SHOULD reject any input data with invalid characters, excess/insufficient paddings, or non-canonical pad bits (See Sections 3.1 to 3.5 of [RFC4648]).

The canonical format for integer and hex-fixed-number are unquoted tokens, and that for base64-fixed-number is quoted-string.

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 9 and 10.

In the descriptions below, the type of allowable values for each header parameter is shown in parenthesis after each parameter name. The "algorithm-determined" type means that the acceptable value for the parameter is one of the types defined in Section 3, and is determined by the value of the "algorithm" parameter. The parameters marked "mandatory" SHALL be contained in the message. The parameters marked "non-mandatory" MAY either be contained or omitted in the message. Each parameter SHALL appear in each headers exactly once at most.

All credentials and challenges MAY contain any parameters not explicitly specified in the following sections. Recipients who do not understand such parameters MUST silently ignore those. However, all credentials and challenges MUST meet the following criteria:

- For responses, the parameters "reason", any "ks*" (where * stands for any decimal integers), and "vks" are mutually exclusive: any challenge MUST NOT contain two or more parameters among them. They MUST NOT contain any "kc*" and "vkc" parameters.
- For requests, the parameters "kc*" (where * stands for any decimal integers), and "vks" are mutually exclusive and any challenge MUST NOT contain two or more parameters among them. They MUST NOT contain any "ks*" and "vks" parameters.

4.1. 401-INIT and 401-STALE

Every 401-INIT or 401-STALE message SHALL be a valid HTTP 401-status (Authentication Required) message containing one (and only one: hereafter not explicitly noticed) "WWW-Authenticate" header containing a "reason" parameter in the challenge. The challenge SHALL contain all of the parameters marked "mandatory" below, and MAY contain those marked "non-mandatory".

version:
: (mandatory extensive-token) should be the token "-wg-draft00" 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 [I-D.oiwa-httpauth-mutual-algo] 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 "single-port" type domain in Section 5.

realm:
: (mandatory string) is a UTF-8 encoded string representing the name of the authentication realm inside the authentication domain. As specified in [I-D.ietf-httpbis-p7-auth], this value MUST always be sent in the quoted-string form.

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.

reason:

(mandatory extensive-token) SHALL be an extensive-token which describes the possible reason of the failed authentication/authorization. Both servers and clients SHALL understand and support the following three tokens:

- initial: authentication was not tried because there was no Authorization header in the corresponding request.
- stale-session: the provided sid; in the request was either unknown to or expired in the server.
- auth-failed: authentication trial was failed by some reasons, possibly with a bad authentication credentials.

Implementations MAY support the following tokens or any extensive-tokens defined outside this specification. If clients has received any unknown tokens, these SHOULD treat these as if it were "auth-failed" or "initial".

- reauth-needed: server-side application requires a new authentication trial, regardless of the current status.
- invalid-parameters: authentication was not even tried in the server-side because some parameters are not acceptable.
- internal-error: authentication was not even tried in the server-side because there is some troubles on the server-side.
- user-unknown: a special case of auth-failed, suggesting that the provided user-name is invalid. The use of this parameter is NOT RECOMMENDED for security implications, except for special-purpose applications which makes this value sense.
- invalid-credential: ditto, suggesting that the provided user-name was valid but authentication was failed. The use of this parameter is NOT RECOMMENDED as the same as the above.
- authz-failed: authentication was successful, but access to the specified resource is not authorized to the specific authenticated user. (It is different from 403 responses which suggest that the reason of inaccessibility is other that authentication.)

The algorithm specified in this header will determine the types (among those defined in Section 3) and the values for K_{c1}, K_{s1}, VK_c and VK_s.

Among these messages, those with the reason parameter of value "stale-session" will be called "401-STALE" messages hereafter, because these have a special meaning in the protocol flow. Messages with any other reason parameters will be called "401-INIT" messages.

4.2. req-KEX-C1

Every req-KEX-C1 message SHALL be a valid HTTP request message containing an "Authorization" header with a credential containing a "kc1" parameter.

The credential SHALL contain the parameters with the following names:

version:

(mandatory, extensive-token) should be the token "-wg-draft00" 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.

kc1:

(mandatory, algorithm-determined) is the client-side key exchange value K_{c1}, which is specified by the algorithm that is used.

rekey-sid:

(non-mandatory, hex-fixed-number): reserved for future extensions (see rekey-method in "200-VFY-S" message).

4.3. 401-KEX-S1

Every 401-KEX-S1 message SHALL be a valid HTTP 401-status (Authentication Required) response message containing a "WWW-Authenticate" header with a challenge containing a "ks1" parameter.

The challenge SHALL contain the parameters with the following names:

version:

(mandatory, extensive-token) should be the token "-wg-draft00" 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. See Section 6 for more details.

ks1:

(mandatory, algorithm-determined) is the server-side key exchange value K_{s1}, 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 parameter 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 parameter 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 parameter MUST be inside the specified auth-domain: if not, clients SHOULD ignore such elements.

4.4. req-VFY-C

Every req-VFY-C message SHALL be a valid HTTP request message containing an "Authorization" header with a credential containing a "vkc" parameter.

The parameters contained in the header are as follows:

version:
: (mandatory, extensive-token) should be the token "-wg-draft00" 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.

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

4.5. 200-VFY-S

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

The parameters contained in the header are as follows:

version:
: (mandatory, extensive-token) should be the token "-wg-draft00" 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.

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

rekey-method:
: (non-mandatory, extensive-token): defining a credential used for reestablishing a new session with a new sid. It must be either omitted or the token "passwords" at the current specification. The bare-tokens "refresh-key" and "refresh-key-global" are reserved for future extensions.

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.

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 protection spaces 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.

- Single-server type: The string in format "<scheme>://<host>:<port>", where <scheme>, <host>, and <port> are the corresponding URI parts of the request 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). The port part MUST NOT contain leading zeros. Use this when authentication is only valid for specific protocol (such as https).
- Single-host type: 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.
- Wildcard-domain type: 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 [RFC6265], 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 9).

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

- If the client has previously sent a request to the same URI, and if 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 earlier 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 re-establish another session without informing the users.

Sessions and session identifiers are local to each server (defined by scheme, host and port) inside an authentication domain; the clients MUST establish separate sessions for each port of a host to be accessed. Furthermore, sessions and identifiers are also local to each authentication realm, even if these are provided from the same server. The same session identifiers provided either from different servers or for different realms SHOULD be treated as independent ones.

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 parameter 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 parameter) 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 parameter 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 10. 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. Host Validation Methods

The "validation method" specifies a method to "relate" (or "bind") 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 vh that is an input to the authentication protocols.

The valid tokens for the validation parameter and corresponding values of vh are as follows:

host:
: hostname validation: The value vh 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 the default port number.

tls-cert:
: TLS certificate validation: The value vh 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 "tls-cert". 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 parameter 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. Authentication Extensions

Interactive clients (e.g. Web browsers) supporting this protocol are RECOMMENDED to support non-mandatory authentication and the Authentication-Control header defined there, except the "auth-style" parameter. This specification also proposes (however, not mandates) default "auth-style" to be "non-modal". Web applications SHOULD however consider the security impacts of the behaviors of clients that do not support these headers.

Authentication-initializing messages with the Optional-WWW-Authenticate header are used where 401-INIT response is valid. Such a message is called a 200-Optional-INIT message in this document. (It will not replace other 401-type messages such as 401-STALE and 401-KEX-S1.)

9. Decision Procedure for Clients

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 VK_S 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 MAY be accepted by 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 5 shows a diagram of the client-side state.

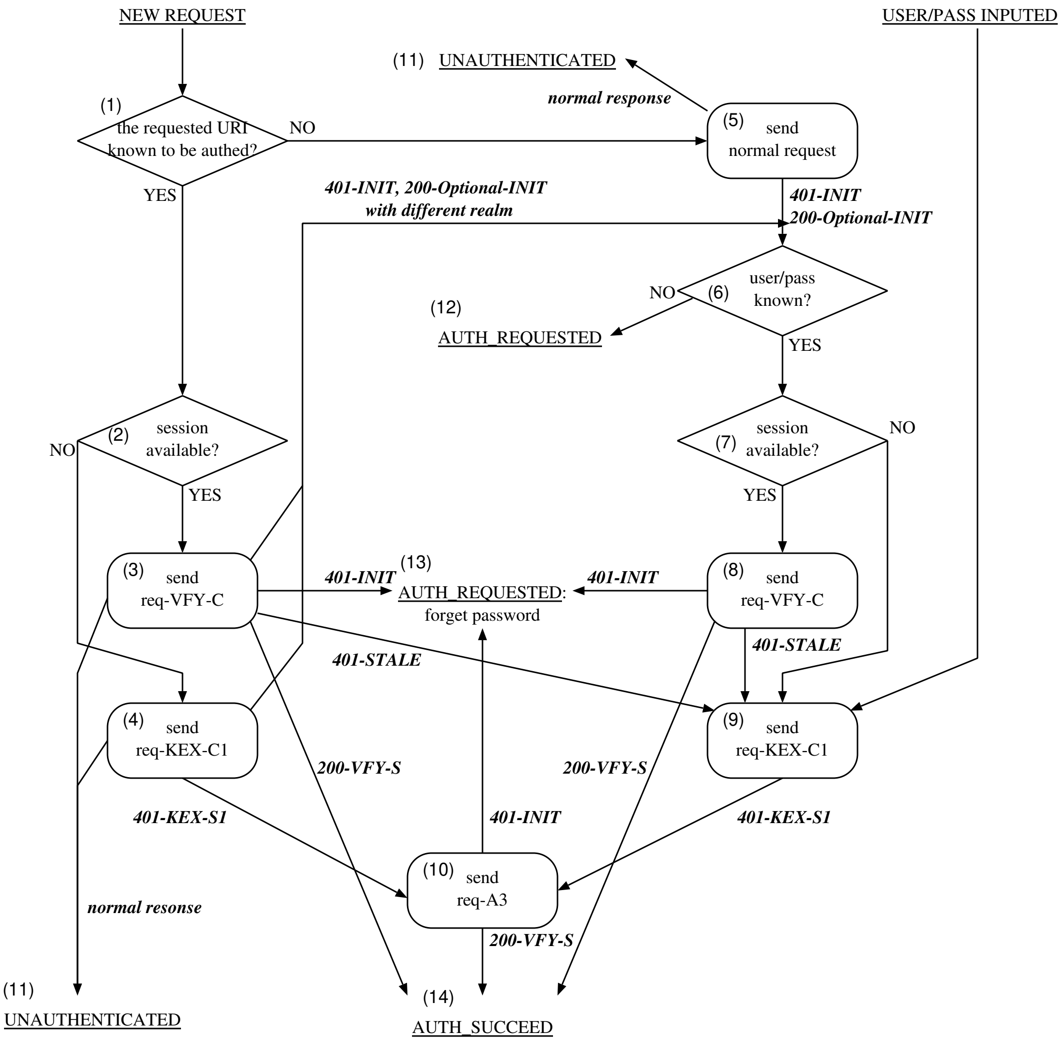

Figure 5: State diagram for clients

---

10. Decision Procedure for Servers

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 parameter.
- The algorithm used.
- The authentication realm.
- The state of the protocol: one of "key exchanging", "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 "key exchanging", the values of K_{c1} and S_{s1}.
- 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 parameter kc1, 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 K_{s1}, 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 K_{s1} and send a 401-KEX-S1 response.

  The created session has the "key exchanging" 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 in the "inactive" state, 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 in the "authenticated" 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 parameter sent from the server, or if it is not larger then (largest-nc - nc-window) (when in "authenticated" status), the server MAY (but not REQUIRED to) send a 401-STALE message. The session SHOULD be changed to the "inactive" status if so.
  - If the session is a "fake" session, or if the received vkc is incorrect, then send a 401-INIT response. If the session is in the "key exchanging" 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.
  - Otherwise, send a 200-VFY-S response. If the session was in the "key exchanging" 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 "key exchanging" status SHOULD be kept unless there is an emergency situation such as a server reboot or a table capacity overflow.

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 K_{c1}, K_{s1} (exchanged on wire) and S_{c1}, S_{s1} (kept secret in each peer).
- 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.

Some examples:

VI(0) = "\000" (in C string notation)

VI(100) = "d"

VI(10000) = "\316\020"

VI(1000000) = "\275\204@"

VS("") = "\000"

VS("Tea") = "\003Tea"

VS("Caf<e acute>" [in UTF-8]) = "\005Caf\303\251"

VS([10000 "a"s]) = "\316\020aaaaa..." (10002 octets)

[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 3.1.3 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 parameter 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 VK_c and VK_s are derived by the following equation.

VK_c = H(octet(4) | OCTETS(K_{c1}) | OCTETS(K_{s1}) | OCTETS(z) | VI(nc) | VS(vh))
VK_s = H(octet(3) | OCTETS(K_{c1}) | OCTETS(K_{s1}) | OCTETS(z) | VI(nc) | VS(vh))

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

12. Application Channel Binding

Applications and upper-layer communication protocols may need authentication binding to the HTTP-layer authenticated user. Such applications MAY use the following values as a standard shared secret.

These values are parameterized with an optional octet string (t) which may be arbitrarily chosen by each applications or protocols. If there is no appropriate value to be specified, use a null string for t.

For applications requiring binding to either an authenticated user or a shared-key session (to ensure that the requesting client is certainly authenticated), the following value b_1 MAY be used.

b_1 = OCTETS(H(OCTETS(H(octet(6) | OCTETS(K_{c1}) | OCTETS(K_{s1}) | OCTETS(z) | VI(0) | VS(vh))) | VS(t))).

For applications requiring binding to a specific request (to ensure that the payload data is generated for the exact HTTP request), the following value b_2 MAY be used.

b_2 = OCTETS(H(OCTETS(H(octet(7) | OCTETS(K_{c1}) | OCTETS(K_{s1}) | OCTETS(z) | VI(nc) | VS(vh))) | VS(t))).

Note: Channel bindings to lower-layer transports (TCP and TLS) are defined in Section 7.

13. Application for Proxy Authentication

The authentication scheme defined by the previous sections can be applied m.m. for proxy authentications. In such cases, the following alterations MUST be applied:

- The 407 status is to be sent and recognized for places where the 401 status is used,
- Proxy-Authenticate: header is to be used for places where WWW-Authenticate: is used,
- Proxy-Authorization: header is to be used for places where Authorization: is used,
- Proxy-Authentication-Info: header is to be used for places where Authentication-Info: is used,
- The auth-domain parameter is fixed to the host-name of the proxy, which means to cover all requests processed through the specific proxy,
- The limitation for the paths contained in the path parameter of 401-KEX-S1 messages is disregarded,
- The omission of the path parameter of 401-KEX-S1 messages means that the authentication realm will potentially cover all requests processed by the proxy,
- The scheme, host name and the port of the proxy is used for host validation tokens, and
- Authentication extension in [I-D.ietf-httpauth-extension] is not applicable.

The requirements for client software to display the authentication status to the end-user is also not applicable for proxy authentication. If the client software supports both end-to-end and proxy authentication using this protocol, it SHOULD be careful that the authentication status of the proxy communication will never be confused by users with authentication statuses of the end-to-end resource authentications.

14. Methods to Extend This Protocol

If a private 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. (standardized or being-standardizing extensions MAY use either bare-tokens or extension-tokens.)

Specifications defining authentication algorithms MAY use other representations for the parameters "kc1", "ks1", "vkc", and "vks", replace those parameter names, and/or add parameters to the messages containing those parameters in supplemental specifications, provided that syntactic and semantic requirements in Section 3, [I-D.ietf-httpbis-p1-messaging] and [I-D.ietf-httpbis-p7-auth] are satisfied. Any parameters starting with "kc", "ks", "vkc" or "vks" and followed by decimal natural numbers (e.g. kc2, ks0, vkc1, vks3 etc.) are reserved for this purpose. If those specifications use names other than those mentioned above, it is RECOMMENDED to use extension-tokens to avoid any parameter 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 parameters provided that the domain part in the token is appropriately used.

15. IANA Considerations

When bare-tokens are used for the authentication-algorithm, pwd-hash, and validation parameters 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.

16. Security Considerations

16.1. 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).

16.2. 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.

16.3. 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 parameter), 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.

16.4. 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" parameter 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.

17. 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.

18. References

18.1. Normative References

[I-D.ietf-httpauth-extension] Oiwa, Y., Watanabe, H., Takagi, H., Kihara, B., Hayashi, T., and Y. Ioku, "HTTP Authentication Extensions for Interactive Clients," draft-ietf-httpauth-extension-00 (work in progress), July 2013.

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

[I-D.ietf-httpbis-p7-auth] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication," draft-ietf-httpbis-p7-auth-22 (work in progress), February 2013 (TXT).

[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).

18.2. Informative References

[I-D.ietf-precis-framework] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation and Comparison of Internationalized Strings in Application Protocols," draft-ietf-precis-framework-08 (work in progress), April 2013 (TXT).

[I-D.oiwa-httpauth-mutual-algo] Oiwa, Y., Watanabe, H., Takagi, H., Kihara, B., Hayashi, T., and Y. Ioku, "Mutual Authentication Protocol for HTTP: KAM3-based Cryptographic Algorithms," draft-oiwa-httpauth-mutual-algo-00 (work in progress), Jul 2013.

[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.

[OASIS.saml-core-2.0-os] Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0," OASIS Standard saml-core-2.0-os, March 2005.

[OIDF.Connect.Standard] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, "OpenID Connect Standard 1.0 - draft 10," May 2012.

[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).

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1," RFC 2743, January 2000 (TXT).

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

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5," RFC 3961, February 2005 (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).

[RFC6749] Hardt, D., “The OAuth 2.0 Authorization Framework,” RFC 6749, October 2012 (TXT).

Appendix A. (Informative) Draft Remarks from Authors

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

- 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.
- Adopt [I-D.ietf-precis-framework] for replacing SASLprep reference. Especially, use NFC canonicalization instead of NFKC.
- Adding test vectors for ensuring implementation correctness.
- Possibly adding a method for servers to detect availability of Mutual authentication on client-side.
- Possible support for optional key renewal and cross-site federated authentication.

Appendix B. (Informative) Draft Change Log

B.1. Changes in Httpauth Revision 00

- Changed the version token.
- Renamed "verification tokens" to "Host verification tokens" and variables "v" to "vh" for clarification. (Back-ported from draft-oiwa-httpauth-multihop-template-00)

B.2. Changes in HttpBis Revision 00

None.

B.3. Changes in Revision 12

- Added a reason "authz-failed".

B.4. Changes in Revision 11

- Message syntax definition reverted to pre-07 style as httpbis-p1 and p7 now defines a precise rule for parameter value parsing.
- Replaced "stale" parameter with more infomative/extensive "reason" parameter in 401-INIT and 401-STALE.
- Reserved "rekey-sid" and "rekey-method" parameters for future extensions.
- Added descriptions for replacing/non-replacing existing technologies.

B.5. Changes in Revision 10

- The authentication extension parts (non-mandatory authentication and authentication controls) are separated to yet another draft.
- The default auth-domain parameter is changed to the full scheme-host-port syntax, which is consistent with usual HTTP authentication framework behavior.
- Provision for application channel binding is added.
- Provision for proxy access authentication is added.
- Bug fix: syntax specification of sid parameter was wrong: it was inconsistent with the type specified in the main text (the bug introduced in -07 draft).
- Terminologies for headers are changed to be in harmony with httpbis drafts (e.g. field to parameter).
- Syntax definitions are changed to use HTTP-extended ABNF syntax, and only the header values are shown for header syntax, in harmony with httpbis drafts.
- Names of parameters and corresponding mathematical values are now renamed to more informative ones. The following list shows correspondence between the new and the old names.

new name	old name	description
S_{c1}, S_{s1}	s_a, s_b	client/server-side secret randoms
K_{c1}, K_{s1}	w_a, w_b	client/server-side exchanged key components
kc1, ks1	wa, wb	parameter names for those
VK_c, VK_s	o_a, o_b	client/server-side key verifiers
vkc, vks	oa, ob	parameter names for those
z	z	session secrets

B.6. Changes in Revision 09

- The (default) cryptographic algorithms are separated to another draft.
- Names of the messages are changed to more informative ones than before. The following is the correspondence table of those names:

new name	old name	description
401-INIT	401-B0	initial response
401-STALE	401-B0-stale	session key expired

req-KEX-C1	req-A1	client->server key exchange
401-KEX-S1	401-B1	server->client key exchange
req-VFY-C	req-A3	client->server auth. verification
200-VFY-S	200-B4	server->client auth. verification
200-Optional-INIT	200-Optional-B0	initial with non-mandatory authentication

B.7. Changes in Revision 08

- The English text has been revised.

B.8. 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 parameters 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.

B.9. 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.

B.10. Changes in Revision 05

- A new parameter 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.

B.11. 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.

B.12. 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.

B.13. 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 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.

B.14. Changes in Revision 01

- Several texts are rewritten for clarification.
- Added several security consideration clauses.

Authors' Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
3-11-46 Nakouji
Amagasaki, Hyogo
JP
Email: mutual-auth-contact-ml@aist.go.jp

Hajime Watanabe
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
Tsukuba Central 2
1-1-1 Umezono
Tsukuba-shi, Ibaraki
JP

Hiromitsu Takagi
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
Tsukuba Central 2
1-1-1 Umezono
Tsukuba-shi, Ibaraki

JP

Boku Kihara
Lepidum Co. Ltd.
#602, Village Sasazuka 3
1-30-3 Sasazuka
Shibuya-ku, Tokyo
JP

Tatsuya Hayashi
Lepidum Co. Ltd.
#602, Village Sasazuka 3
1-30-3 Sasazuka
Shibuya-ku, Tokyo
JP

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