

HTTPAUTH Working Group
Internet-Draft
Intended status: Experimental
Expires: May 18, 2017

Y. Oiwa
H. Watanabe
H. Takagi
ITRI, AIST
K. Maeda
T. Hayashi
Lepidum
Y. Ioku
Individual
November 14, 2016

Mutual Authentication Protocol for HTTP
draft-ietf-httpauth-mutual-11

Abstract

This document specifies a mutual authentication scheme for the Hypertext Transfer Protocol (HTTP). This scheme provides true mutual authentication between an HTTP client and an HTTP server using password-based authentication. Unlike the Basic and Digest authentication schemes, the Mutual authentication scheme specified in this document assures the user that the server truly knows the user's encrypted password.

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 May 18, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal

Provisions Relating to IETF Documents

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

Table of Contents

| | | |
|------------------------|--|--------------------|
| 1. | Introduction | 4 |
| 1.1. | Terminology | 6 |
| 1.2. | Document Structure and Related Documents | 6 |
| 2. | Protocol Overview | 7 |
| 2.1. | Messages Overview | 7 |
| 2.2. | Typical Flows of the Protocol | 8 |
| 2.3. | Alternative Flows | 10 |
| 3. | Message Syntax | 11 |
| 3.1. | Non-ASCII extended header parameters | 12 |
| 3.2. | Values | 13 |
| 3.2.1. | Tokens | 13 |
| 3.2.2. | Strings | 14 |
| 3.2.3. | Numbers | 14 |
| 4. | Messages | 15 |
| 4.1. | 401-INIT and 401-STALE | 16 |
| 4.2. | req-KEX-C1 | 18 |
| 4.3. | 401-KEX-S1 | 19 |
| 4.4. | req-VFY-C | 20 |
| 4.5. | 200-VFY-S | 20 |
| 5. | Authentication Realms | 21 |
| 5.1. | Resolving Ambiguities | 22 |
| 6. | Session Management | 23 |
| 7. | Host Validation Methods | 25 |
| 7.1. | Applicability notes | 26 |
| 7.2. | Notes on tls-unique | 27 |
| 8. | Authentication Extensions | 27 |
| 9. | String Preparation | 28 |
| 10. | Decision Procedure for Clients | 28 |
| 10.1. | General Principles and Requirements | 28 |
| 10.2. | State machine for the client (informative) | 30 |
| 11. | Decision Procedure for Servers | 35 |
| 12. | Authentication Algorithms | 37 |
| 12.1. | Support Functions and Notations | 38 |
| 12.2. | Default Functions for Algorithms | 39 |
| 13. | Application Channel Binding | 40 |
| 14. | Application for Proxy Authentication | 41 |

| | | |
|------------------------------------|---|---------------------------|
| <u>15.</u> | <u>Methods to Extend This Protocol</u> | <u>42</u> |
| <u>16.</u> | <u>IANA Considerations</u> | <u>42</u> |
| <u>16.1.</u> | <u>Registry for Authentication Algorithms</u> | <u>42</u> |
| <u>16.2.</u> | <u>Registry for Validation Methods</u> | <u>43</u> |
| <u>17.</u> | <u>Security Considerations</u> | <u>44</u> |
| <u>17.1.</u> | <u>Security Properties</u> | <u>44</u> |
| <u>17.2.</u> | <u>Secrecy of Credentials</u> | <u>44</u> |
| <u>17.3.</u> | <u>Denial-of-service Attacks to Servers</u> | <u>45</u> |
| <u>17.3.1.</u> | <u>On-line Active Password Attacks</u> | <u>45</u> |
| 17.4. | Communicating the status of mutual authentication with users | <u>45</u> |
| <u>17.5.</u> | <u>Implementation Considerations</u> | <u>46</u> |
| <u>17.6.</u> | <u>Usage Considerations</u> | <u>47</u> |
| <u>18.</u> | <u>References</u> | <u>47</u> |
| <u>18.1.</u> | <u>Normative References</u> | <u>47</u> |
| <u>18.2.</u> | <u>Informative References</u> | <u>48</u> |
| <u>Appendix A.</u> | <u>(Informative) Draft Change Log</u> | <u>50</u> |
| <u>A.1.</u> | <u>Changes in Httpauth WG Revision 11</u> | <u>50</u> |
| <u>A.2.</u> | <u>Changes in Httpauth WG Revision 10</u> | <u>50</u> |
| <u>A.3.</u> | <u>Changes in Httpauth WG Revision 09</u> | <u>50</u> |
| <u>A.4.</u> | <u>Changes in Httpauth WG Revision 08</u> | <u>50</u> |
| <u>A.5.</u> | <u>Changes in Httpauth WG Revision 07</u> | <u>51</u> |
| <u>A.6.</u> | <u>Changes in Httpauth WG Revision 06</u> | <u>51</u> |
| <u>A.7.</u> | <u>Changes in Httpauth WG Revision 05</u> | <u>51</u> |
| <u>A.8.</u> | <u>Changes in Httpauth WG Revision 04</u> | <u>51</u> |
| <u>A.9.</u> | <u>Changes in Httpauth WG Revision 03</u> | <u>51</u> |
| <u>A.10.</u> | <u>Changes in Httpauth WG Revision 02</u> | <u>51</u> |
| <u>A.11.</u> | <u>Changes in Httpauth WG Revision 01</u> | <u>52</u> |
| <u>A.12.</u> | <u>Changes in Httpauth Revision 00</u> | <u>52</u> |
| <u>A.13.</u> | <u>Changes in HttpBis Revision 00</u> | <u>52</u> |
| <u>A.14.</u> | <u>Changes in Revision 12</u> | <u>52</u> |
| <u>A.15.</u> | <u>Changes in Revision 11</u> | <u>52</u> |
| <u>A.16.</u> | <u>Changes in Revision 10</u> | <u>53</u> |
| <u>A.17.</u> | <u>Changes in Revision 09</u> | <u>54</u> |
| <u>A.18.</u> | <u>Changes in Revision 08</u> | <u>54</u> |
| <u>A.19.</u> | <u>Changes in Revision 07</u> | <u>54</u> |
| <u>A.20.</u> | <u>Changes in Revision 06</u> | <u>55</u> |
| <u>A.21.</u> | <u>Changes in Revision 05</u> | <u>55</u> |
| <u>A.22.</u> | <u>Changes in Revision 04</u> | <u>55</u> |
| <u>A.23.</u> | <u>Changes in Revision 03</u> | <u>55</u> |
| <u>A.24.</u> | <u>Changes in Revision 02</u> | <u>56</u> |
| <u>A.25.</u> | <u>Changes in Revision 01</u> | <u>56</u> |
| Authors' | Addresses | <u>56</u> |

1. Introduction

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

Password-stealing attacks are one of most critical threats for Web systems. For a long time, plain-text password authentications (Basic and Web form-based) are widely used (and are in use now). When these are used with plain HTTP protocols, it is trivially easy for attackers to sniff the password credentials on the wire.

Digest authentication scheme [[RFC7616](#)] uses a SHA-2 (formerly SHA-1 and MD5) hash algorithms to hide the raw user password from the sniffing. However, if the number of possible candidates of users' password is not enough, recent powerful computers can compute possible hash values for billions of password candidates, and compare these with the sniffed values to find out the correct password. This kind of attack is called "offline password dictionary attacks"; recently, the size of possible search space by computers is quite competing with possibility of user's memorable passwords, threatening the effectiveness of such hash-based password protections.

TLS [[RFC5246](#)] provides a strong cryptographic protection against the network-based sniffing of passwords and other communication contents. If TLS is correctly used by both server operators and client users, passwords and other credentials will not be available for any outside attackers. However, there is a pit-hole in the TLS deployment on the Web systems; if the users are forged into a "wrong website" by some kind of social attacks and tricked to perform authentication on that site, the credentials will be sent to the attacker's server and trivially leaked. Such attacks are called "Phishing", and becoming a real threats in these days. In the current Web system deployment, TLS certificates will be issued to almost any users of Internet (including malicious attackers). Although those certificate includes several levels of the "validation results" (such as corporate names) of the issued entities, task of "checking" those validation results are left to the users of Web browsers, still leaving the possibility of such social attacks.

Another direction to avoid such threats is to avoid password-based authentication and use some kind of pre-deployed strong secret keys (either on client side or on server-side) for authentications. Several federated authentication framework as well as HOBA [[RFC7486](#)] are proposed and deployed on the real Web systems to satisfy those needs. However, a kind of authentication based on "human-memorable

secret" (i.e. passwords) is still required on several situations within those systems, such is initialization, key deployment to new clients, or recovery of secret accounts with lost cryptographic keys.

The Mutual authentication protocol proposed in this document is a strong cryptographic solution for password authentications. It mainly provides the two key features:

- o No password information, at all, is exchanged in the communications. When the server and the user fails to authenticate with each other, the protocol will not reveal the tiniest bit of information about the user's password. This prevents any kind of off-line password dictionary attacks, even with the existence of Phishing attacks.
- o To successfully authenticate, the server must own the valid registered credentials (authentication secret), as well as client users. (Non-intuitively, this is not true for Basic and Digest authentication. For example, servers for Basic authentications can answer "YES" to any clients, without actually checking authentication at all.) This means that phishing attackers cannot forge users that they are the "authentic" servers. Client users can assert whether the communicating peer is "the server" who have registered their account beforehand. In other words, it provides "true" mutual authentication between servers and clients.

Given these, the proposed protocol can serve as a strong alternative to the Basic, Digest, and web-form-based authentications, and also as a strong companion to the non-password-based authentication frameworks.

The proposed protocol will serve in the same way as existing Basic/Digest authentication: it meets the requirement for new authentication scheme for HTTP as described in [Section 5.1.2 of \[RFC7235\]](#). Additionally, to communicate authentication results more reliably between the server and the client user, it suggests for Web browsers to have some "secure" way of displaying the authentication results. Having such an user interface in future browser will greatly reduce the risk of impersonation by kinds of social attacks, similarly in the manner of the "green padlock" for extended verification TLS certificates.

Technically, the authentication scheme proposed in this document is a general framework for using password-based authenticated key exchange (PAKE) and similar stronger cryptographic primitives with HTTP. The two key features shown above are corresponding to the nature of PAKE.

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 [\[RFC2119\]](#).

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 both the input (domain) and the output (codomain) of hash functions to be octet strings. When a natural number output for a hash function is required, it will be written as INT(H(s)).

1.2. Document Structure and Related Documents

The entire document is organized as follows:

- o [Section 2](#) presents an overview of the protocol design.
- o Sections [3](#) to [11](#) define a general framework of the Mutual authentication protocol. This framework is independent of specific cryptographic primitives.
- o [Section 12](#) describes properties needed for cryptographic algorithms used with this protocol framework, and defines a few functions which will be shared among such cryptographic algorithms.
- o The sections after that contain general normative and informative information about the protocol.

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

- o [\[I-D.ietf-httpauth-mutual-algo\]](#): defines cryptographic primitives which can be used with this protocol framework.
- o [\[I-D.ietf-httpauth-extension\]](#): defines 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 [[RFC7230](#)] using a framework defined in [[RFC7235](#)]. 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 provides a brief outline 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.

- o 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.
 - * 401-STALE message: a message indicating that client has to start a new key exchange.
- o 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: an intermediate response to a req-KEX-C1 message from the server.
- o Authentication verification messages: used by both peers to verify the authentication results.
 - * req-VFY-C message: a message used by the client, requesting the server authenticate and authorize the client.
 - * 200-VFY-S message: a response used by the server to indicate the successful client-authentication. It also contains information necessary for the client to check the authenticity of the server.

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 scheme will use the following protocol sequence.

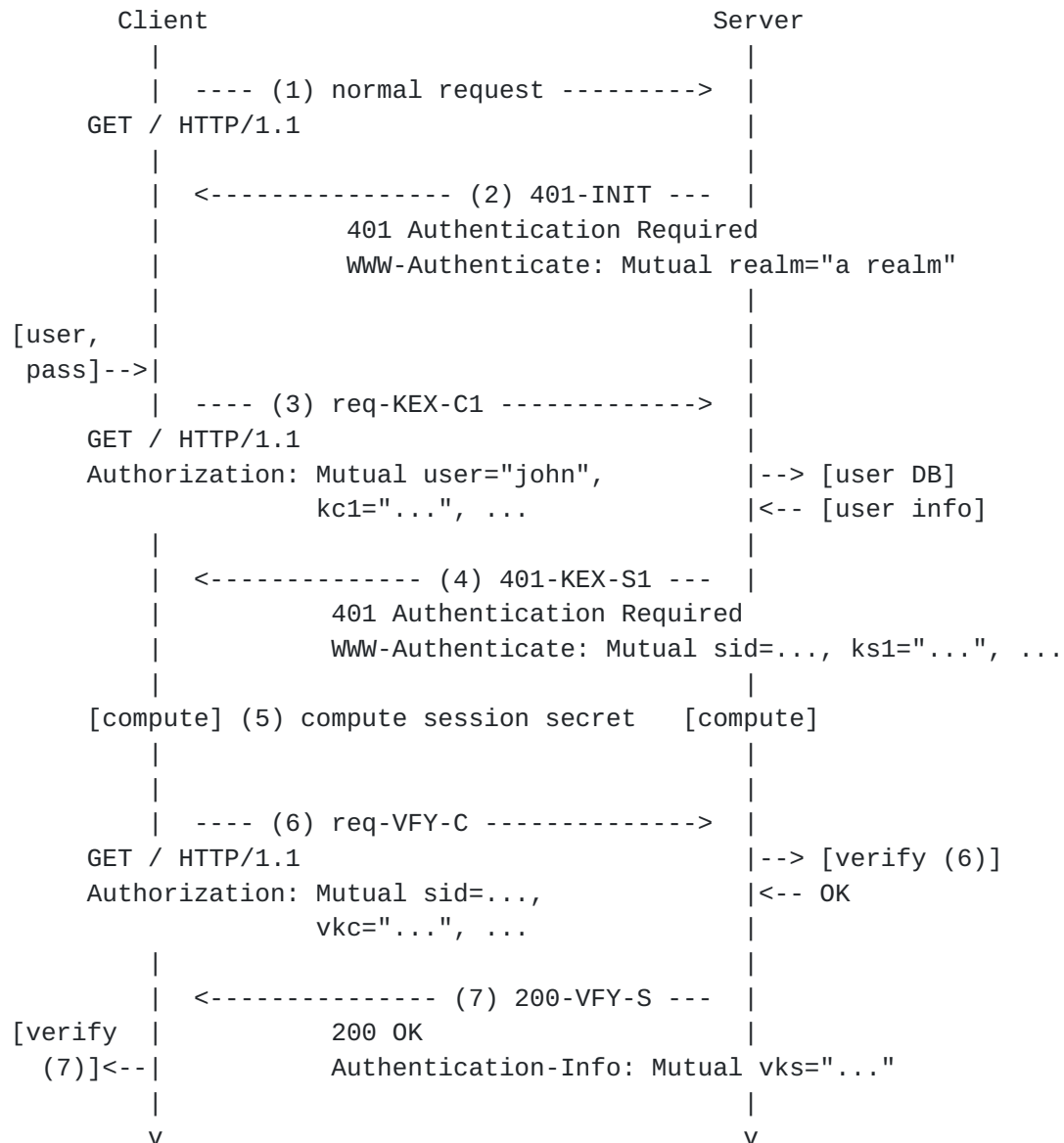


Figure 1: Typical communication flow for first access to resource

- o 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).

- o 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).
- o 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).
- o 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 access authentication of every individual request/response pair after this point.
- o The client will send a request with a client-side authentication verification value (req-VFY-C) (6), calculated from the client-generated session secret. The server will check the validity of the verification value using its own version of the session secret.
- o 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 [[RFC1939](#)]), 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.

- o 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, 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 the HTTP response in this case. (Note: This case should not happen between a correctly implemented server and client without any active attacks. The possible cause of such a case might be either a man-in-the-middle attack or an incorrect implementation.)

2.3. Alternative Flows

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

- o (case A) If the client knows that the resource is likely to require 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.
- o (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.

The server **MAY** have thrown out the corresponding session from the session table. If so, 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 criteria, and thus only one round-trip of request/response will be required.

(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
(1 round trip)

| Client | Server |
|----------------------|--------|
| | |
| ---- req-VFY-C ----> | |
| | |
| <----- 200-VFY-S --- | |
| | |

(B-2) key expired
(3 round trips)

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

Figure 2: Several alternative protocol flows

For more details, see Sections [10](#) and [11](#).

3. Message Syntax

Throughout this specification, the syntax is denoted in the extended augmented BNF syntax defined in [[RFC7230](#)], and [[RFC5234](#)]. The following elements are quoted from [[RFC5234](#)], [[RFC7230](#)] and [[RFC7235](#)]: DIGIT, ALPHA, SP, auth-scheme, quoted-string, auth-param, header-field, token, challenge, and credential.

The Mutual authentication protocol uses three headers:

WWW-Authenticate (usually 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 [\[RFC7235\]](#) and [\[RFC7615\]](#). The detailed meanings for these headers are contained in [Section 4](#).

The framework in [\[RFC7235\]](#) 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 [\[RFC7235\]](#).

The Authentication-Info: header used in this protocol SHALL follow the syntax defined in [\[RFC7615\]](#).

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

[3.1](#). Non-ASCII extended header parameters

All of parameters contained in the above three headers, except the "realm" field, MAY be extended to ISO 10646-1 values using the framework described in [\[RFC5987\]](#). All servers and clients MUST be capable of receiving and sending values encoded in [\[RFC5987\]](#) syntax.

If a value to be sent contains only ASCII characters, the field MUST be sent using plain [RFC 7235](#) syntax. The syntax as extended by [RFC 5987](#) MUST NOT be used in this case.

If a value (except the "realm" header) contains one or more non-ASCII characters, the parameter SHOULD be sent using the syntax defined in [Section 3.2 of \[RFC5987\]](#) as "ext-parameter". Such a parameter MUST have a charset value of "UTF-8", and the language value MUST always be omitted (have an empty value). The same parameter MUST NOT be sent more than once, regardless of the used syntax.

For example, a parameter "user" with value "Renee of France" SHOULD be sent as < user="Renee of France" >. If the value is "Ren<e acute>e of France", it SHOULD be sent as < user*=UTF-8'Ren%C3%89e%20of%20France > instead.

[\[RFC7235\]](#) requires the realm parameter to be in its plain form (not as an extended "realm*" parameter), so [RFC 5987](#) syntax MUST NOT be used for this parameter.

3.2. Values

The parameter values contained in challenge/credentials **MUST** be parsed strictly conforming to the HTTP semantics (especially unquoting 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 **RECOMMENDED** to use the canonical representations specified in the following subsections for sending values. However, recipients **MUST** accept both quoted and unquoted representations interchangeably as specified in HTTP.

3.2.1. Tokens

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

```

bare-token           = bare-token-lead-char *bare-token-char
bare-token-lead-char = %x30-39 / %x41-5A / %x61-7A
bare-token-char      = %x30-39 / %x41-5A / %x61-7A / "-" / "_"
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, they **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 extension of this protocol **MAY** use either the bare-tokens allocated by IANA (under the procedure described in [Section 16](#)), or extension-tokens with the format "-<bare-token>.<domain-name>", where <domain-name> is a valid (sub-)domain name on the Internet owned by the party who defines the extension.

Bare-tokens and extensive-tokens are also used for parameter names, in the unquoted form. Requirements for using the extension-token for the parameter names are the same as the previous paragraph.

The canonical format for bare-tokens and extensive-tokens is the

unquoted representation.

3.2.2. Strings

All character strings MUST be encoded to octet strings using the UTF-8 encoding [RFC3629] for the Unicode character set [Unicode]. Such strings MUST NOT contain any leading BOM markers (also known as ZERO WIDTH NO-BREAK SPACE, U+FEFF or EF BB BF). Both peers are RECOMMENDED to reject any invalid UTF-8 sequences that might cause decoding ambiguities (e.g., containing "<" in the second or later bytes 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 is quoted-string (as it may contain equal signs, plus signs and slashes), unless the parameter containing the string value will use extended syntax defined in [RFC5987]. (An [RFC5987] extended parameter will have an unquoted encoded value, as defined therein.)

3.2.3. Numbers

The following syntax definitions give a syntax for numeric 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 / "+" / "/" ) 0*2"="
```

Figure 4: BNF syntax for numbers

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

A number represented as a hex-fixed-number MUST include an even number of hexadecimal digits (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 MUST have their "natural length"; if these values are generated from a hash function, these lengths correspond to the hash size; if these are representing elements of a mathematical set (or group), these lengths SHALL be the shortest for representing all the elements in the set. For example, the results of the SHA-256 hash function will be represented by 64 digits, and any elements in a 2048-bit prime field (modulo a 2048-bit integer) will be represented by 512 digits,

regardless of how much zeros appear in front of such representations. Session-identifiers and other non-cryptographically generated values are represented in any (even) length determined by the side that generates it first, and the same length MUST be used throughout 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 (described in [Section 4 of \[RFC4648\]](#)) without any spaces and newlines. Implementations decoding base64-fixed-number SHOULD reject any input data with invalid characters, excess/insufficient padding, 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 in what circumstances each message is expected to be sent, see Sections [10](#) and [11](#).

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 header exactly once at most.

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

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

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

Every message in this section contains a "version" field, to detect future, incompatible revisions of the protocol. Implementations of the protocol described in this specification MUST always send a token "1", and recipients MUST reject messages that contain any other value as a version, unless another specification defines a behavior for that version.

4.1. 401-INIT and 401-STALE

Every 401-INIT or 401-STALE message SHALL be a valid HTTP 401-status (Authentication Required) message (or other 4XX status if sensible) containing one and only one (hereafter not explicitly noted) "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 "1".

algorithm: (mandatory extensive-token) specifies the authentication algorithm to be used. The value MUST be one of the tokens specified in [\[I-D.ietf-httpauth-mutual-algo\]](#) or another supplemental specification.

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 another supplemental specification.

auth-scope: (non-mandatory string) specifies the authentication scope, 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-server" type domain in [Section 5](#).

realm: (mandatory string) is a string representing the name of the authentication realm inside the authentication scope. As specified in [\[RFC7235\]](#), this value MUST always be sent in the quoted-string form, and an [\[RFC5987\]](#) encoding MUST NOT be used. The realm value sent from the server SHOULD be an ASCII string. Clients MAY treat any non-ASCII value

received in this field as a binary blob, an NFC-normalized UTF-8 string, or an error.

reason: (mandatory extensive-token) SHALL be an extensive-token that 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 for some reason, possibly with a bad authentication credential.

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

- * reauth-needed: the server-side application requires a new authentication trial, regardless of the current status.
- * invalid-parameters: the server did not attempt authentication because some parameters were not acceptable.
- * internal-error: the server did not attempt authentication because there are some troubles on the server-side.
- * user-unknown: this is a special case of auth-failed, suggesting that the provided user name is invalid. The use of this parameter is NOT RECOMMENDED due to security implications, except for special-purpose applications where it makes sense.
- * invalid-credential: ditto, suggesting that the provided user name was valid but authentication still failed. The use of this parameter is

NOT RECOMMENDED for security reasons.

- * authz-failed: authentication was successful, but access to the specified resource is not authorized to the specific authenticated user. (It might be used along with either a 401 or 403 status to indicate that the authentication result is one of the existing reasons for the failed authorization.)

It is RECOMMENDED to record the reasons to a kind of diagnostic log, for an example, or shown to the client user immediately. It will be helpful to find out later that the reason of the failed authentication is either technical reasons of user errors.

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 "1".

algorithm, validation, auth-scope, realm: MUST be the same values as received from the server.

user: (mandatory, string) is the UTF-8 encoded name of the user. The string SHOULD be prepared according to the method presented in [Section 9](#).

kc1: (mandatory, algorithm-determined) is the client-side key exchange value K_c1, which is specified by the algorithm that is used.

[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 "1".

algorithm, validation, auth-scope, realm: MUST be the same values as 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 maximum 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 maximum value of nonce numbers that the server accepts.

nc-window: (mandatory, integer) the number of available nonce number slots that the server will accept. The value of the nc-window parameter is RECOMMENDED to be 128 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 for 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 [[RFC7616](#)] for Digest authentications. All path elements contained in the parameter MUST be inside the specified auth-scope; if not, clients SHOULD ignore such elements. For better performance,

recognition of this parameter by clients is important.

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 "1".

algorithm, validation, auth-scope, realm: MUST be the same values as 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 request number that is unique among the requests sharing the same sid. The values of the nonce numbers 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 does not have a 401 (Authentication Required) status code and SHALL contain 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 "1".

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.

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. 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 credentials. However, for different authentication realms, clients MUST NOT automatically reuse user names and passwords for another realm.

Just like in the Basic and Digest access authentication protocols, the Mutual authentication protocol supports multiple, separate protection spaces to be set up inside each host. Furthermore, the protocol allows a single authentication realm to span over several hosts within the same Internet domain.

Each authentication realm is defined and distinguished by the triple of an "authentication algorithm", an "authentication scope", and a "realm" parameter. However, server operators are NOT RECOMMENDED to use the same pair of an authentication scope and a realm with different authentication algorithms.

The realm parameter is a string as defined in [Section 4](#). Authentication scopes are described in the remainder of this section.

An authentication scope specifies the range of hosts that the authentication realm spans over. In this protocol, it MUST be one of the following kinds of strings.

- o Single-server type: A string in the format "<scheme>://<host>" or "<scheme>://<host>:<port>", where <scheme>, <host>, and <port> are the corresponding URI parts of the request URI. If the default port (i.e., 80 for http and 443 for https) is used for the underlying HTTP communications, the port part MUST be omitted, regardless of whether it was present in the request-URI. In all other cases, the port part MUST be present, and it MUST NOT contain leading zeros. Use this format when authentication is only valid for a specific protocol (such as https). This format is equivalent to the ASCII serialization of a Web Origin, presented in [Section 6.2 of \[RFC6454\]](#).
- o Single-host type: The "host" part of the requested URI. This is the default value. Authentication realms within this kind of authentication scope will span over several protocols (e.g., http

and https) and ports, but not over different hosts.

- o Wildcard-domain type: A string in the 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 by 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"), 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 domain names SHALL be in the form of LDH labels in IDNA [[RFC5890](#)]. A "port" MUST be given 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 scopes, several scopes may overlap each other. If a client has already been authenticated to several realms applicable to the same server, the client may have a multiple lists of the "path" parameters received with the "401-KEX-S1" message (see [Section 4](#)). If these path lists have any overlap, a single URI may belong to multiple possible candidate of realms to be authenticated to. In such cases, clients faces an ambiguity in deciding which credentials to send for a new request (in steps 3 and 4 of the decision procedure presented in [Section 10](#)).

In such cases, a client MAY send request which belong to any of these candidate realms freely, or it MAY simply send an unauthenticated request and see for which realm the server requests an authentication. Server operators are RECOMMENDED to provide properly-configured "path" parameters (more precisely, disjoint path sets for each realms) for clients so that such ambiguities will not occur.

The following procedure is one possible tactic for resolving ambiguity in such cases.

- o If the client has previously sent a request to the same URI, and if it remembers the authentication realm requested by the 401-INIT message at that time, use that realm.
- o In other cases, use one of the authentication realms representing the most-specific authentication scopes. The list of possible domain specifications shown above is given from most specific to least specific.

If there are several choices with different wildcard-domain specifications, the one that has the longest domain-postfix has priority over ones with shorter domain-postfixes.

- o If there are realms with the same authentication scope, there is no defined priority; the client MAY choose any one of the possible choices.

6. Session Management

In the Mutual authentication protocol, a session represented by an sid is set up using four messages (first request, 401-INIT, req-KEX-C1 and 401-KEX-S1), after which a "session secret" (z) associated with the session is established. After mutually establishing a session secret, this session, along with the secret, can be used for one or more requests for resources protected by the same realm on 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 user.

Sessions and session identifiers are local to each server (defined by scheme, host, and port), even if an authentication scope covers multiple servers; 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 by the same server. The same session identifiers provided either from different servers or for different realms MUST be treated as independent or each other.

The server SHOULD accept at least one req-VFY-C request for each session, if the request reaches the server in a time window specified by the timeout parameter in the 401-KEX-S1 message, and there are no emergent reasons (such as flooding attacks) to forget the session. After that, the server MAY discard any session at any time and MAY send 401-STALE messages for any further req-VFY-C requests received for that session.

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 number (in the nc parameter) MUST satisfy the following conditions:

- o It is a natural number.
- o The same nonce number was not sent within the same session.
- o It is not larger than the nc-max value that was sent from the server in the session represented by the sid.
- o It is larger than (largest-nc - nc-window), where largest-nc is the largest value of nc which was previously sent in the session, and nc-window is the value of the nc-window parameter that was received from the server for the session.

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

Servers MUST check for duplication of the received nonce numbers, and if any duplication is detected, the server MUST discard the session and respond with a 401-STALE message, as outlined in [Section 11](#). The server MAY also reject other invalid nonce numbers (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 128, nc-max is 400, and that the client has already used the following nonce numbers: {1-120, 122, 124, 130-238, 255-360, 363-372}. Then the nonce number that can be used for the next request is one of the following set: {245-254, 361, 362, 373-400}. The values {0, 121, 123, 125-129, 239-244} MAY be rejected by the server because they are not above the current "window limit" ($244 = 372 - 128$).

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

The values of the nonce numbers and any nonce-related values MUST always be treated as natural numbers within an infinite range. Implementations which uses fixed-width integer representations, fixed-precision floating-point numbers, or similar representations SHOULD NOT reject any larger values which overflow such representative limits, and MUST NOT silently truncate them using any

modulus-like rounding operation (e.g., by mod 2^{32}). Instead, the whole protocol is carefully designed so that recipients MAY replace any such overflowing values (e.g. 2^{80}) with some reasonably-large maximum representative integer (e.g., $2^{31} - 1$ or others).

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 determines the value `vh` that is an input to the authentication protocols.

When HTTPS or other possible secure transport is used, this corresponds to the idea of "channel binding" described in [\[RFC5929\]](#). Even when HTTP is used, similar, but somewhat limited, "binding" is performed to prevent a malicious server from trying to authenticate itself to another server as a valid user by forwarding the received credentials.

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

`host:` host-name 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
 server-side resource currently being accessed. 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-server-end-point:` TLS endpoint (certificate) validation: The
 value `vh` will be the octet string of the hash value of
 the server's public key certificate used in the
 underlying TLS [\[RFC5246\]](#) connection, processed as
 specified in [Section 4.1 of \[RFC5929\]](#).

`tls-unique:` TLS shared-key validation: The value `vh` will be the
 channel binding material derived from the Finished
 messages, as defined in [Section 3.1 of \[RFC5929\]](#).
 (Note: see [Section 7.2](#) for some security notices when
 using this validation method.)

If HTTP is used on a non-encrypted channel (TCP and SCTP, for

example), the validation type MUST be "host". If HTTP/TLS [[RFC2818](#)] (HTTPS) is used with a server certificate, the validation type MUST be "tls-server-end-point". If HTTP/TLS is used with an anonymous Diffie-Hellman key exchange, the validation type MUST be "tls-unique" (see the note below).

If the validation type "tls-server-end-point" is used, the server certificate provided in the TLS connection MUST be verified at least to make sure that the server actually owns the corresponding private key. (Note: this verification is automatic in some RSA-based key exchanges but NOT automatic in Diffie-Hellman-based key exchanges with separate exchange for server verification.)

Clients MUST validate this parameter upon receipt of 401-INIT messages.

Note: The protocol defines two variants of validation on the TLS connections. The "tls-unique" method is technically more secure. However, there are some situations where tls-server-end-point is more preferable.

- o 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 the HTTP CONNECT method.
- o When a black-box implementation of the TLS protocol is used on either peer.

[7.1.](#) Applicability notes

When the client is a Web browser with any scripting capabilities (dynamic contents support), the underlying TLS channel used with HTTP/TLS MUST provide server identity verification. This means (1) anonymous Diffie-Hellman key exchange cipher suites MUST NOT be used, and (2) verification of the server certificate provided by the server MUST be performed. This is to prevent loading identity-unauthenticated scripts or dynamic contents, which are referenced from the authenticated page.

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

7.2. Notes on tls-unique

As described in the interoperability note in the above channel binding specification, the tls-unique verification value will be changed by possible TLS renegotiation, causing an interoperability problem. TLS re-negotiations are used in several HTTPS server implementations for enforcing some security properties (such as cryptographic strength) for some specific responses.

If an implementation supports the "tls-unique" verification method, the following caution SHOULD be taken:

- o Both peers must be aware that the vh values used for vkc (in req-VFY-C) and for vks (in 200-VFY-S) may be different. These values MUST be retrieved from underlying TLS libraries each time they are used.
- o After calculating the values vh and vkc to send a req-VFY-C request, Clients SHOULD NOT initiate TLS renegotiation until the end of the corresponding response header is received. An exception is that clients can and SHOULD perform TLS re-negotiation as a response to the server's request for TLS renegotiation, before receipt of the beginning of the response header.

Also, implementers MUST take care of session resumption attacks regarding tls-unique channel binding mechanisms and master secrets. As a mitigation, a TLS extension defined in [\[RFC7627\]](#) SHOULD be used when tls-unique host verification is to be used.

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 in [\[I-D.ietf-httpauth-extension\]](#), except for the "auth-style" parameter. This specification also proposes (however, does not mandate) the default "auth-style" 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 only where the 401-INIT response is valid. It will not replace other 401-type messages such as 401-STALE and 401-KEX-S1. That is, the reason field of such a message MUST be "initial" (or any extensive-tokens NOT defined in [Section 4.1](#)).

9. String Preparation

It is important for interoperability that user names and passwords used in this protocol are binary-comparable regardless of the user's input methods and/or environments. To ensure this, the following preparation SHOULD be performed:

- o User names received from users SHOULD be prepared using the "UsernameCasePreserved" profile defined in [Section 3.3 of \[RFC7613\]](#).
- o Passwords received from users SHOULD be prepared using the "OpaqueString" profile defined in [Section 4.2 of \[RFC7613\]](#).

In both cases, it is the sender's duty to correctly prepare the character strings. If any non-prepared character string is received from the other peer of the communication, the behavior of its recipient is not defined; the recipient MAY either accept or reject such input.

Server applications SHOULD also prepare user names and passwords accordingly upon registration of user credentials.

In addition, binary-based "interfaces" of implementations MAY require and assume that the string is already prepared accordingly; when a string is already stored as a binary Unicode string form, implementations MAY omit preparation and Unicode normalization (performing UTF-8 encoding only) before using it. When a string is already stored as an octet blob, implementations MAY send it as is.

10. Decision Procedure for Clients

10.1. General Principles and Requirements

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

As usual in the HTTP authentication, a single user-level request may result in exchange of two-or-more HTTP requests and responses in sequence. The following normative rules MUST be followed by the clients implementing this protocol:

- o Any kind of a "normal response" MUST only be accepted for the very first request in the sequence. Any "normal response" returned for

the second or later requests in the sequence SHALL be considered invalid.

- o In the same principle, if any response is related to an authentication realm which is different from that of the client's request (for example, a 401-INIT message requesting authentication on another realm), it MUST only be accepted for the very first request in the sequence. Such a response returned for a second or later request in the sequence SHALL be considered invalid.
- o A req-KEX-C1 message MAY be sent either as a initial request or as a response to 401-INIT or 401-STALED. However, it SHOULD NOT be sent more than once in the sequence for a single authentication realm, to avoid infinite loops of messages. A 401-KEX-S1 response MUST be accepted only when the corresponding request is req-KEX-C1.
- o A req-VFY-C message MAY be sent if there is a valid session secret shared between the client and the server, established by req-KEX-C1 and 401-KEX-S1. If any response with 401 status is returned for such a message, the corresponding session secret SHOULD be discarded as unusable. Especially, upon the reception of a 401-STALED response, the client SHOULD try establishing a new session by sending req-KEX-C1, but only once within the request/response sequence.
- o A 200-VFY-S message MUST be accepted only as a response to req-VFY-C and nothing else. The VK_s values of such response messages MUST always be checked against the correct value, and if it is incorrect, the whole response SHOULD be considered invalid.

The final status of the client request following the message exchange sequence shall be determined as follows:

- o AUTH-SUCCESS: A 200-VFY-S message with the correct VK_s value was returned in response to the req-VFY-C request in the sequence.
- o AUTH-REQUIRED: Two cases exists.
 - * A 401-INIT message was returned from the server, and the client does not know how to authenticate to the given authentication realm.
 - * A 401-INIT response was returned for req-VFY-C (or req-KEX-C1), which means the user-supplied authentication credentials were not accepted.

- o UNAUTHENTICATED: a normal response is returned for an initial request of any kind in the sequence.

Any kind of response (including a normal response) other than those explicitly allowed in the above rules SHOULD be interpreted as a fatal communication error. In such cases, the clients MUST NOT process any data (the 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 has a Server-Error (5xx) status code. In such cases, they SHOULD be careful about processing the body of the content (ignoring it is still RECOMMENDED, as it may possibly be forged by intermediate attackers), and the client will be in the "UNAUTHENTICATED" status then.

If a request is a sub-request for a resource included in another resource (e.g., embedded images, style sheets, frames etc.), clients MAY treat an AUTH-REQUESTED status as the same as an UNAUTHENTICATED status. In other words, the client MAY ignore server's request to start authentication with new credentials via sub-requests.

10.2. State machine for the client (informative)

The following state machine describes the possible request-response sequences derived from the above normative rules. If implementers are not quite sure on the security consequences of the above rules, it is strongly advised to follow the decision procedure below. In particular, clients SHOULD NOT accept "normal responses" unless explicitly allowed in the rules. 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 is to be followed.

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 expected authentication realm. 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 a 401-STALE message is received, go to Step 9.
- * If a 401-INIT message is received, go to Step 13.
- * If a 200-VFY-S message is received, go to Step 14.
- * If a normal response is received, go to Step 11.

Step 4 (step_send_kex1_1):

Send a req-KEX-C1 request.

- * If a 401-INIT message is received with a different authentication realm than expected, go to Step 6.
- * If a 401-KEX-S1 message is received, go to Step 10.
- * If a 401-INIT message is received with the same authentication realm, go to Step 13 (see Note 1).
- * If a normal response is received, go to Step 11.

Step 5 (step_send_normal_1):

Send a request without any Mutual authentication headers.

- * If a 401-INIT message is received, go to Step 6.
- * If a normal response is received, go to Step 11.

Step 6 (step_rcvd_init):

Check whether the user's password for the requested authentication realm is known. If yes, go to Step 7. Otherwise, go to Step 12.

Step 7:

Check whether there is an available sid for the expected authentication realm. 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 a 401-STALE message is received, go to Step 9.
- * If a 401-INIT message is received, go to Step 13.
- * If a 200-VFY-S message is received, go to Step 14.

Step 9 (step_send_kex1):

Send a req-KEX-C1 request.

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

Step 10 (step_rcvd_kex1):

Send a req-VFY-C request.

- * If a 401-INIT message is received, go to Step 13.
- * If a 200-VFY-S message is received, 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 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 the 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 user-provided credentials for the authentication realm and go to Step 12.

Step 14 (step_rcvd_vfy):

The received message is the 200-VFY-S message, which always contains a vks field. 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 the user interface), 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 are **NOT RECOMMENDED** for servers to initiate.

Figure 5 shows an informative diagram of the client state.

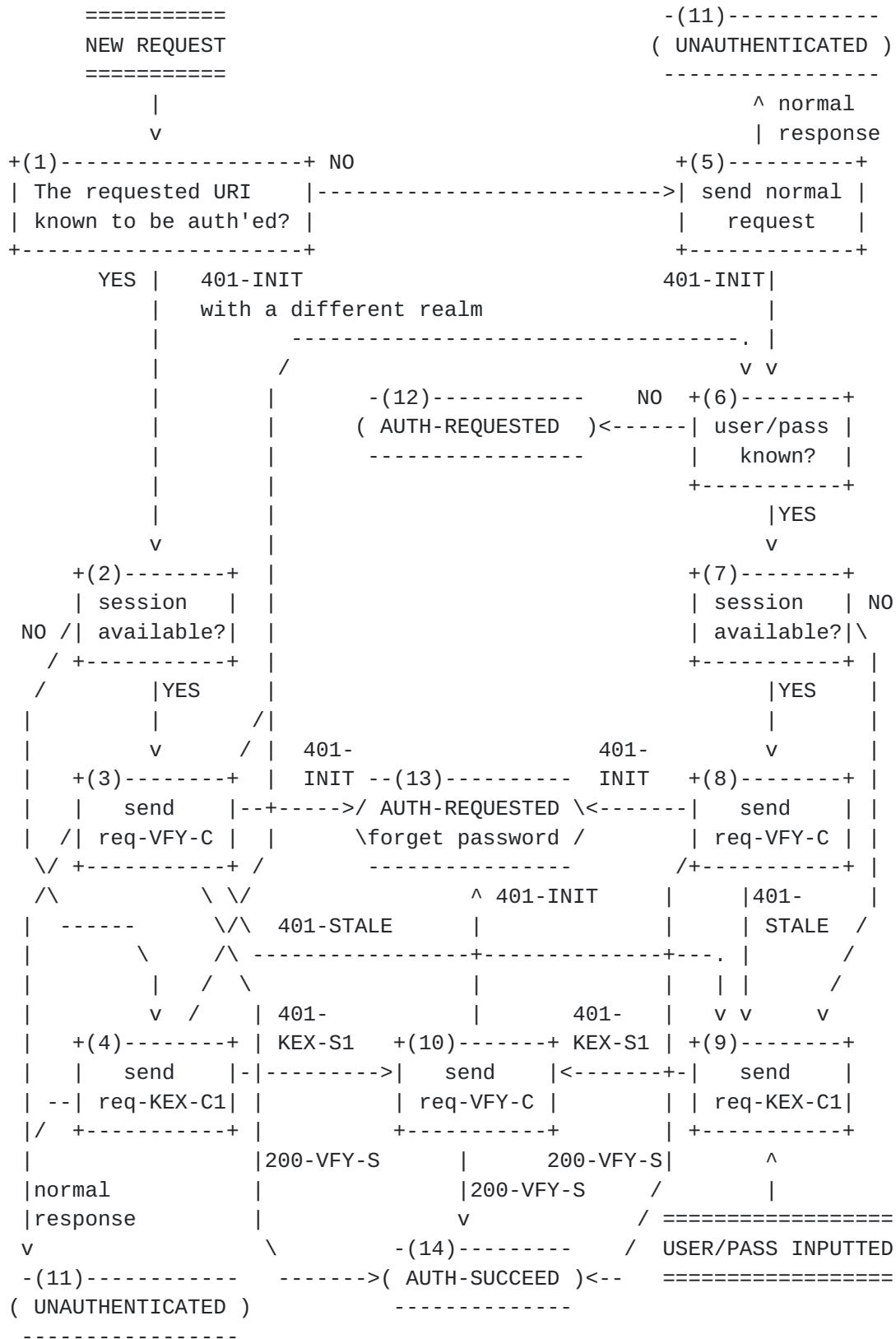


Figure 5: State diagram for clients

11. Decision Procedure for Servers

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

- o The session identifier, which is the value of the sid parameter.
- o The algorithm used.
- o The authentication realm.
- o The state of the protocol: one of "key exchanging", "authenticated", "rejected", or "inactive".
- o The user name received from the client.
- o A boolean flag representing whether or not the session is fake.
- o When the state is "key exchanging", the values of K_c1 and S_s1.
- o 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 boolean 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: (See Note 1 below for 401-INIT message with a plus sign)

- o 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.

- o 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 a 401-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. (See Note 2.)
 - * Otherwise, create a new session, compute K_s1 and send a 401-KEX-S1 response. The created session is marked as not fake, and its largest-nc is initialized to zero.

The created session has the "key exchanging" state.

- o 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 a 401-INIT response.

If none of above holds true, the server will look up 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 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 is not REQUIRED to; See Note 3) send a 401-STALE message. The session is changed to the "inactive" state if so did.

- * 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 is changed to the "inactive" state.
- * 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 MUST be changed to the "rejected" state; otherwise, it MAY either be changed to the "rejected" state 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 MUST be updated appropriately.

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

Note 1: In relation with and following the specification of the optional authentication defined in [[I-D.ietf-httpauth-extension](#)], the 401-INIT messages marked with the pluses cannot be replaced with a successful responses with an Optional-WWW-Authenticate header. Every other 401-INIT can be a response with an Optional-WWW-Authenticate.

Note 2: 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 name will not be accepted. Instead, postpone it to the response for the next req-VFY-C request.

Note 3: The next case implies that, when the request is not rejected in this clause, the server must be able to determine whether the same nc value was previously received from the client. If the server does not remember a whole history of the nc values received from the client, the server MUST send a 401-STALE message on this clause.

12. Authentication Algorithms

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

- o The server-side authentication credential J, derived from client-side authentication credential pi.

- o Key exchange values K_{c1} , K_{s1} (exchanged on wire) and S_{c1} , S_{s1} (kept secret in each peer).
- o Shared session secret z , to be computed by both server and client.
- o A hash function H to be used with the protocol, along with its output size $hSize$.
- o The number of iterations for password hashing $nIterPi$, if it uses the default password hashing function defined below.

Specifications for cryptographic algorithms used with this framework MUST specify whether these will use the default functions defined below for values pi , VK_c , and VK_s ; or, these will define their own versions for these.

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 J for deriving server-side authentication credential $J(pi)$ is RECOMMENDED to be one-way so that pi should not be easily computed from $J(pi)$.

12.1. 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(i)` generates an octet string containing a single octet of value i . 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 as big-endian radix-128 strings, where each digit is represented by an octet within the range `0x80-0xff` except the last digit, which is represented by a octet within the range `0x00-0x7f`. The first octet MUST NOT be `0x80`. For example, $VI(i) = \text{octet}(i)$ for $i < 128$, and $VI(i) = \text{octet}(0x80 + (i >> 7)) | \text{octet}(i \& 127)$ for $128 \leq i < 16384$. This encoding is the same as the one used for the sub-components of object identifiers in the

ASN.1 encoding [[ITU.X690.1994](https://www.itu-t.org/ITU-T/rec/itu-t-rec-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:

$$VS(s) = VI(\text{length}(s)) \mid s$$

where $\text{length}(s)$ is a number of octets (not characters) in s .

Some examples:

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

$VI(100) = "d"$

$VI(10000) = "\316\020"$

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

$VS("") = "\000"$

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

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

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

(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 3.2.3](#) for the definition of "natural length".

The function INT converts an octet string into a natural number, where the input string is treated as being in radix-256 big-endian notation. The identity $\text{INT}(\text{OCTETS}(n)) = n$ always holds for any natural number n .

[12.2.](#) Default Functions for Algorithms

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

The client-side password-based (credential) `pi` used by this authentication is a natural number derived in the following manner:

$$pi = \text{INT}(\text{PBKDF2}(\text{HMAC_H}, \text{password}, \text{VS}(\text{algorithm}) \mid \text{VS}(\text{auth-scope}) \mid \text{VS}(\text{realm}) \mid \text{VS}(\text{username}), nIterPi, hSize / 8)),$$

where

- o PBKDF2 is the password-based key derivation function defined in [\[RFC2898\]](#),
- o HMAC_H is the HMAC function, defined in [\[RFC2104\]](#), composed from the hash function H, and
- o hSize is the output size of hash H in bits.

The values of `algorithm`, `realm`, and `auth-scope` are taken from the values contained in the 401-INIT message. If the password comes from user input, it SHOULD first be prepared according to the method presented in [Section 9](#). Then, the password SHALL be encoded as a UTF-8 string.

The values `VK_c` and `VK_s` are derived by the following equation.

$$VK_c = \text{INT}(\text{H}(\text{octet}(4) \mid \text{OCTETS}(K_c1) \mid \text{OCTETS}(K_s1) \mid \text{OCTETS}(z) \mid \text{VI}(nc) \mid \text{VS}(vh)))$$

$$VK_s = \text{INT}(\text{H}(\text{octet}(3) \mid \text{OCTETS}(K_c1) \mid \text{OCTETS}(K_s1) \mid \text{OCTETS}(z) \mid \text{VI}(nc) \mid \text{VS}(vh)))$$

[13.](#) 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 application or protocol. If there is no appropriate value to be specified, use an empty 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.

$$b_1 = \text{H}(\text{H}(\text{octet}(6) \mid \text{OCTETS}(K_c1) \mid \text{OCTETS}(K_s1) \mid \text{OCTETS}(z) \mid \text{VI}(0)$$

| 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 b₂ MAY be used.

b₂ = H(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](#).

[14.](#) Application for Proxy Authentication

The authentication scheme defined by the previous sections can be applied (with modifications) for proxy authentication. In such cases, the following alterations MUST be applied:

- o The 407 status is to be sent and recognized in places where the 401 status is used,
- o Proxy-Authenticate header is to be used in places where WWW-Authenticate is used,
- o Proxy-Authorization header is to be used in places where Authorization is used,
- o Proxy-Authentication-Info header is to be used in places where Authentication-Info is used,
- o The auth-scope parameter is fixed to the host-name of the proxy, which means it covers all requests processed through the specific proxy,
- o The limitation for the paths contained in the path parameter of 401-KEX-S1 messages is disregarded,
- o 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,
- o The scheme, host name, and the port of the proxy is used for host validation tokens, and
- o Authentication extensions in [[I-D.ietf-httpauth-extension](#)] are not applicable.

15. 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-standardized 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](#), [[RFC7230](#)] and [[RFC7235](#)] 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 future extensions to 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 used in the manner defined in [Section 3](#).

16. IANA Considerations

This document requires an additional entry to the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" as follows:

- o Authentication Scheme Name: "Mutual"
- o Pointer to specification text: (this document)

When bare-tokens are used for the authentication-algorithm and validation parameters, these MUST be allocated by IANA. To acquire registered tokens, the usage of such tokens MUST be reviewed by a designated expert, as outlined in [[RFC5226](#)].

16.1. Registry for Authentication Algorithms

This document establishes a registry for HTTP Mutual authentication algorithms. The registry manages case-insensitive ASCII strings. The strings MUST follow the extensive-token syntax defined in [Section 3](#).

Registrations for an authentication algorithm are required to include a description of the authentication algorithms. Reviewers assigned

by IESG are advised to examine minimum security requirements and consistency of the key exchange algorithm descriptions.

New registrations are advised to provide the following information:

- o Token: a token used in HTTP headers for identifying the algorithm.
- o Description: A brief description of the algorithm.
- o Specification: A reference for a specification defining the algorithm.

The initial content of this registry is empty. [[Editorial Note: A separate document [[I-D.ietf-httpauth-mutual-algo](#)] will effectively define the initial content of the registry.]]

[16.2.](#) Registry for Validation Methods

This document establishes a registry for HTTP Mutual authentication host validation methods. The registry manages case-insensitive ASCII strings. The strings MUST follow the extensive-token syntax defined in [Section 3](#).

Registrations for a validation method are required to include a description of the validation method. Reviewers assigned by IESG are advised to examine its use-case requirements and security consequence of its introduction.

New registrations are advised to provide the following information:

- o Token: a token used in HTTP headers for identifying the method.
- o Description: A brief description of the method.
- o Specification: A reference for a specification defining the method.

The initial content of this registry is as follows:

| +-----+-----+-----+ | | |
|----------------------|------------------------|---------------------------|
| Token | Description | Specification |
| +-----+-----+-----+ | | |
| host | Host name verification | Section 7 |
| | only | |
| tls-server-end-point | TLS certificate-based | Section 7 |
| tls-unique | TLS unique key-based | Section 7 |
| +-----+-----+-----+ | | |

17. Security Considerations

17.1. Security Properties

- o 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 confidentiality is important.
- o When used with HTTP/TLS, if TLS server certificates are reliably verified, the protocol provides true protection against active man-in-the-middle attacks.
- o 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](#)).

17.2. Secrecy of Credentials

The client-side password credential MUST be kept secret all the time, and SHOULD NOT be used with any other (possibly insecure) authentication purpose. Loss of control of the credential will directly affect the control of corresponding server-side account.

Use of client-side credential with THIS authentication scheme is always safe, even if the connected server peer is not trustful (condition of Phishing). However, if it is used with other authentication schemes (such as Web forms), and if the recipient is rogue, the result will be obvious.

The server-side password credential (J) is also important to be kept secret. If it is stolen, and if the client's choice of password is not strong, the person aware of server-side password credential can employ a off-line dictionary attack to search for the client password. However, if the client has chosen a strong password, so that the attacker cannot guess the client's password from dictionary candidate, the client is still well protected from any attacks.

The shared session secret (z) MUST be kept secret inside the server/client software; if it is lost, and if the session is still active, it will lead to session hijacking. After the session is expired, the key is valueless for attackers.

17.3. Denial-of-service Attacks to Servers

The protocol requires a server-side table of active sessions, which may become a critical point for server resource consumption. 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 reestablish a new session then.

However, if a malicious client sends too many requests for 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 their statuses. One way to mitigate such attacks is that servers MAY have a number and a time limit for unverified, pending key exchange requests (in the "key exchanging" state).

This is a common weakness of authentication protocols with almost any kind of negotiations or states, including Digest authentication scheme and most Cookie-based authentication implementations. However, regarding the resource consumption, the situation for the mutual authentication scheme is a slightly better than for 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.

17.3.1. On-line Active Password Attacks

Although the protocol provides very strong protection against off-line dictionary attacks from eavesdropped traffic, the protocol, by its nature, cannot prevent active password attacks in which the attackers sends so many authentication trial requests for every possible password.

Possible countermeasures for preventing such attacks may be rate-limiting of password authentication trials, statistics-based intrusion detection measures, or similar protection schemes. If the server operators assume that the passwords of users are not strong enough, it may be desirable to introduce such ad-hoc countermeasures.

17.4. Communicating the status of mutual authentication with users

This protocol is designed for two goals. The first goal is just providing a secure alternative for existing Basic and Digest authentication. The second goal is to provide users a way to detect

forged rogue servers imitating a user's registered account on a server, commonly known as (a part or kind of) Phishing attacks.

For this protocol to effectively work as some countermeasure to such attacks, it is very important that end users of clients be notified of the result of the mutual authentication performed by this protocol, especially the three states "AUTH-SUCCEED", "UNAUTHENTICATED", and "AUTH-REQUIRED" defined in [Section 10](#). The design of secure user interfaces of the HTTP interactive clients is out of the scope of this document, but if possible, having some kind of UI indication for the three states above will be desirable for the user's security benefit.

Of course, in such cases, the user interfaces for asking passwords for this authentication shall be clearly identifiable against imitation by other insecure password input fields (such as forms). If the passwords are known to malicious attackers outside of the protocol, the protocol cannot work as an effective security measures.

[17.5](#). Implementation Considerations

- o 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.
- o For HTTP/TLS communications, when a web form is submitted from Mutually-authenticated pages with the "tls-server-end-point" 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 re-validated using a req-KEX-C1 message with an "Expect: 100-continue" header. The same applies when the page is received with the "tls-unique" validation method, and when the TLS session has expired.
- o For better protection against possible password database stealing, server-side storage of user passwords should contain the values encrypted by the one-way function $J(\pi)$, instead of the real passwords or those hashed by π .
- o If the TLS 1.2 is used for underlying HTTP/TLS communications, follow best practices in [[RFC7525](#)].

17.6. Usage Considerations

- o The user names inputted by a user may be sent automatically to any servers sharing the same auth-scope. This means that when a host-type auth-scope 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 clear text. If user names have to be kept secret against eavesdropping, the server must use the full-scheme-type auth-scope parameter and HTTPS. Contrarily, passwords are not exposed to eavesdroppers even on HTTP requests.
- o If the server provides several ways for storing server-side password secrets in the password database, it is desirable for better security to store the values encrypted by using the one-way function $J(\pi)$, instead of the real passwords or those hashed by π .

18. References

18.1. Normative References

- [I-D.ietf-httpauth-extension]
Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "HTTP Authentication Extensions for Interactive Clients", [draft-ietf-httpauth-extension-09](#) (work in progress), August 2016.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), DOI 10.17487/RFC2104, February 1997, <<http://www.rfc-editor.org/info/rfc2104>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", [RFC 2898](#), DOI 10.17487/RFC2898, September 2000, <<http://www.rfc-editor.org/info/rfc2898>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, [RFC 3629](#), DOI 10.17487/RFC3629, November 2003, <<http://www.rfc-editor.org/info/rfc3629>>.

- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", [RFC 4648](#), DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, [RFC 5234](#), DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", [RFC 5987](#), DOI 10.17487/RFC5987, August 2010, <<http://www.rfc-editor.org/info/rfc5987>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", [RFC 7235](#), DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.
- [RFC7613] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", [RFC 7613](#), DOI 10.17487/RFC7613, August 2015, <<http://www.rfc-editor.org/info/rfc7613>>.
- [RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", [RFC 7615](#), DOI 10.17487/RFC7615, September 2015, <<http://www.rfc-editor.org/info/rfc7615>>.
- [Unicode] The Unicode Consortium, "The Unicode Standard", <<http://www.unicode.org/versions/latest/>>.

18.2. Informative References

- [I-D.ietf-httpauth-mutual-algo]
Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "Mutual Authentication Protocol for HTTP:

KAM3-based Cryptographic Algorithms",
[draft-ietf-httpauth-mutual-algo-07](#) (work in progress),
November 2016.

[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.

[RFC1939] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, [RFC 1939](#), DOI 10.17487/RFC1939, May 1996,
<<http://www.rfc-editor.org/info/rfc1939>>.

[RFC2818] Rescorla, E., "HTTP Over TLS", [RFC 2818](#), DOI 10.17487/
[RFC2818](#), May 2000,
<<http://www.rfc-editor.org/info/rfc2818>>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), DOI 10.17487/RFC5226, May 2008,
<<http://www.rfc-editor.org/info/rfc5226>>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", [RFC 5890](#), DOI 10.17487/RFC5890, August 2010,
<<http://www.rfc-editor.org/info/rfc5890>>.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", [RFC 5929](#), DOI 10.17487/RFC5929, July 2010,
<<http://www.rfc-editor.org/info/rfc5929>>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", [RFC 6265](#), DOI 10.17487/RFC6265, April 2011,
<<http://www.rfc-editor.org/info/rfc6265>>.

[RFC6454] Barth, A., "The Web Origin Concept", [RFC 6454](#), DOI 10.17487/RFC6454, December 2011,
<<http://www.rfc-editor.org/info/rfc6454>>.

[RFC7486] Farrell, S., Hoffman, P., and M. Thomas, "HTTP Origin-Bound Authentication (HOBAs)", [RFC 7486](#), DOI 10.17487/
[RFC7486](#), March 2015,
<<http://www.rfc-editor.org/info/rfc7486>>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer

Security (TLS) and Datagram Transport Layer Security (DTLS)", [BCP 195](#), [RFC 7525](#), DOI 10.17487/RFC7525, May 2015, <<http://www.rfc-editor.org/info/rfc7525>>.

[RFC7564] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", [RFC 7564](#), DOI 10.17487/RFC7564, May 2015, <<http://www.rfc-editor.org/info/rfc7564>>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", [RFC 7616](#), DOI 10.17487/RFC7616, September 2015, <<http://www.rfc-editor.org/info/rfc7616>>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", [RFC 7627](#), DOI 10.17487/RFC7627, September 2015, <<http://www.rfc-editor.org/info/rfc7627>>.

[Appendix A](#). (Informative) Draft Change Log

[To be removed on final publication]

[A.1](#). Changes in Httpauth WG Revision 11

- o Reflecting IESG comments.

[A.2](#). Changes in Httpauth WG Revision 10

- o Small rephrasing and a typo fix.

[A.3](#). Changes in Httpauth WG Revision 09

- o Reflected AD review comments.
- o Authors' addresses updated.

[A.4](#). Changes in Httpauth WG Revision 08

- o Minor text update, in sync with httpauth-extension.
- o The version token is raised to "1".

[A.5.](#) Changes in Httpauth WG Revision 07

- o Several comments from reviewers are reflected to the text.
- o The password-hash has been completely dropped.
- o The version token is raised to "1".

[A.6.](#) Changes in Httpauth WG Revision 06

- o The auth-domain parameter has been renamed to auth-scope, following suggestions on the mailing list.
- o The digest-md5 password-hash has been dropped, as Digest with MD5 hash is now obsoleted.

[A.7.](#) Changes in Httpauth WG Revision 05

- o Minimum nonce number window has increased to 128. (HTTP 2.0 recommends at least 100 concurrent sessions to exist)
- o Reference to TLS session hash extension added for tls-unique security issues.
- o Comments in the previous F2F meeting has been reflected to the text.

[A.8.](#) Changes in Httpauth WG Revision 04

- o Merged httpauthprep proposal into general PRECIS Username/Password profile.
- o Adopting [RFC 5987](#) extended syntax for non-ASCII parameter values.
- o Refer [draft-ietf-httpbis-auth-info](#) for Authentication-Info header. This results in a different syntax for that header.

[A.9.](#) Changes in Httpauth WG Revision 03

- o Incompatible change: Single-port type authentication realm label has been changed to harmonize with Web Origin. (That is, the default ports (80 and 443) are to be omitted.)

[A.10.](#) Changes in Httpauth WG Revision 02

- o Major change: introduction of password-strengthening function PBKDF2.

- o Changed [Section 10](#) to adopt "list of requirements" style. Strict definition of state machine is now a derived, informational definition.

[A.11.](#) Changes in Httpauth WG Revision 01

- o Changed "tls-key" verification to "tls-unique" verification, and "tls-cert" to "tls-server-end-point", adopting [RFC 5929](#).
- o Adopted PRECIS framework [[RFC7564](#)].
- o Reverted reservation of "rekey-sid" and "rekey-method" parameters.
- o Degraded secure UI requirement to application note level, non-normative.
- o Adjusted levels of several requirements.
- o Added warning text for handling of exceptional 5XX responses.
- o Dropped several references for optional authentications, except one "Note".
- o Several textual fixes, improvements and revisions.

[A.12.](#) Changes in Httpauth Revision 00

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

[A.13.](#) Changes in Httpbis Revision 00

None.

[A.14.](#) Changes in Revision 12

- o Added a reason "authz-failed".

[A.15.](#) Changes in Revision 11

- o Message syntax definition reverted to pre-07 style as httpbis-p1 and p7 now defines a precise rule for parameter value parsing.
- o Replaced "stale" parameter with more informative/extensive "reason" parameter in 401-INIT and 401-STALE.

- o Reserved "rekey-sid" and "rekey-method" parameters for future extensions.
- o Added descriptions for replacing/non-replacing existing technologies.

A.16. Changes in Revision 10

- o The authentication extension parts (non-mandatory authentication and authentication controls) are separated to yet another draft.
- o The default auth-domain parameter is changed to the full scheme-host-port syntax, which is consistent with usual HTTP authentication framework behavior.
- o Provision for application channel binding is added.
- o Provision for proxy access authentication is added.
- o 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).
- o Terminologies for headers are changed to be in harmony with httpbis drafts (e.g. field to parameter).
- o 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.
- o 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 | | | |
| +-----+-----+-----+-----+-----+-----+ | | | | | |

A.17. Changes in Revision 09

- o The (default) cryptographic algorithms are separated to another draft.
- o 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 shared secret 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 |

A.18. Changes in Revision 08

- o The English text has been revised.

A.19. Changes in Revision 07

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

[A.20.](#) Changes in Revision 06

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

[A.21.](#) Changes in Revision 05

- o 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.
- o A new header "Authentication-Control" is added for precise control of application-level authentication behavior.

[A.22.](#) Changes in Revision 04

- o 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.
- o The "tls-key" verification is now OPTIONAL.
- o Several description fixes and clarifications.

[A.23.](#) Changes in Revision 03

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

A.24. Changes in Revision 02

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

A.25. Changes in Revision 01

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

Authors' Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Information Technology Research Institute
Tsukuba Central 1
1-1-1 Umezono
Tsukuba-shi, Ibaraki
JP

Email: y.oiwa@aist.go.jp

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

Email: h-watanabe@aist.go.jp

Hiromitsu Takagi
National Institute of Advanced Industrial Science and Technology
Information Technology Research Institute
Tsukuba Central 1
1-1-1 Umezono
Tsukuba-shi, Ibaraki
JP

Email: takagi.hiromitsu@aist.go.jp

Kaoru Maeda
Lepidum Co. Ltd.
Village Sasazuka 3, Suite #602
1-30-3 Sasazuka
Shibuya-ku, Tokyo
JP

Email: maeda@lepidum.co.jp

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

Email: hayashi@lepidum.co.jp

Yuichi Ioku
Individual

Email: mutual-work@ioku.org

