Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: August 22, 2013

Y. Oiwa
H. Watanabe
H. Takagi
RISEC, AIST
B. Kihara
T. Hayashi
Lepidum
Y. Ioku
Yahoo! Japan

February 18, 2013

Common Template for HTTP Message-based Multi-hop Authentication draft-oiwa-httpauth-multihop-template-00

Abstract

This document specifies a common protocol design template for authentication on the Hyper-text Transport Protocol (HTTP) involving multi-hop message exchanges. To facilitate advanced authentication technologies such as hash-based exchanges, zero-knowledge password proof, or public-key authentications on HTTP, a kind of state management and key management facilities are required on the general HTTP authentication message framework. Also, to optimize performance of such authentication schemes, a well-designed mechanism for key caching and re-authentication are needed. The template defined in this document provides a generic foundation for implementing such advanced authentication technologies.

Status of this Memo

Table of Contents

1. Introduction

This document specifies a common protocol design template for authentication on the Hyper-text Transport Protocol (HTTP) involving multi-hop message exchanges.

To facilitate advanced authentication technologies such as hash-based exchanges, zero-knowledge password proof, or public-key authentications on HTTP, a kind of state management and key management facilities are required on the general HTTP authentication message framework. Also, to optimize performance of such authentication schemes, a well-designed mechanism for key caching and re-authentication are needed.

The template defined in this document provides a generic foundation for implementing such advanced authentication technologies. Such generic foundations can reduce cumbersomeness of both designers and implementors of such authentication protocols on HTTP. By using this template, protocol designers can easily apply any specific authenticated key exchange (or agreement) mechanisms onto HTTP protocol and enable authentication session management, shared-key based optimized re-authentication.

The design template provided on this document is mainly designed for multi-hop authentication mechanisms which do not use connection-based session managements. Some of existing authentication technologies applied on HTTP/1.0 or 1.1 are bound to underlying TCP connection, which violates strict definition of HTTP stateless semantics and not directly applicable to forthcoming HTTP/2.0. Retrofitting of such existing authentication schemes are out-of-scope of this specification (although, an additional specification for such retrofitting _may_ be defined on top of this template).

The template is defined using terminology and representation of existing HTTP/1.1, but it can be also directly applied on forthcoming HTTP/2.0.

1.1. How to Use This Document

This document is only providing a "template" for actual implementation of HTTP authentication: by itself only it will be useless. To use this document, there must be a specific definition document for each authentication schemes referring to this document. In other words, this document and such a specific definitions will compose "layers" of protocol definitions, the latter will exist upon the former.

However, for implementors' perspective, the definitions in this document can be implemented as a "base class" for multi-hop authentication: such class can be a common bases for "deriving" implementations of each authentication schemes, which will avoid duplicated implementation of same features and reduce burdens for testing such implementations one by one.

For terminology, this document uses the following three terms for referring each "layers" of protocols:

- "The authentication template" or "this template" will refer to the common protocol template defined in this document.
- "Authentication scheme(s)" will refer to a scheme which will realize a specific purpose/method of authentication. Examples of these schemes (which do not always depend on "this template") are Basic, Digest and others. Each of them will also correspond to a specific "auth-scheme" in the HTTP headers.

- "Sub-algorithms" or simply "algorithms" in an authentication scheme will refer to variations within a single authentication scheme which will provide a small differences of authentication properties such as cryptographic strength or others. Examples of them are "auth" and "auth-int" in Digest. Differences of used cryptographic primitives and/or parameters which provides the same functionalities except strengths (e.g. key lengths, hash choices etc.) will often fall into this category.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The terms "encouraged" and "advised" are used for suggestions that do not constitute "SHOULD"-level requirements. People MAY freely choose not to include the suggested items regarding [RFC2119], but complying with those suggestions would be a best practice; it will improve security, interoperability, and/or operational performance.

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

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

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

1.3. Document Structure and Related Documents

The entire document is organized as follows:

- Section 2 presents an overview of the protocol design.
- Sections 3 to 8 define a general template for the multi-hop authentication protocol. This template is independent of specific cryptographic primitives and authentication schemes.
- Section 9 describes requirements for each authentication schemes used with this protocol template, and defines a few functions which will be shared among such cryptographic algorithms.
- The sections after that contain general normative and informative information about the protocol.
- The appendices contain some information that may help developers to implement the protocol.

2. Protocol Overview

The protocol template, as a whole, is designed as a natural extension to the HTTP protocol [I-D.ietf-httpbis-p1-messaging] using a framework defined in [I-D.ietf-httpbis-p7-auth]. Internally, the server and the client will first perform a cryptographic key exchange, defined for each authentication schemes. The key-exchange will derive the same session keys only when the clients and servers are agreed with the authentication credentials used. Then, both peers will verify the authentication results by confirming the sharing of the exchanged key. This section describes a brief image of the protocol and the exchanged messages.

2.1. Messages Overview

The authentication protocol template uses six kinds of messages to perform multi-hop authentication. These messages have specific names within this specification.

- Authentication request messages: used by the servers to request clients to start authentication.
  - 401-INIT message: a general message to start the authentication exchange. It is also used as a message indicating an authentication failure.
  - 401-STALE message: a message indicating that it has to start a new authentication trial.
- Authenticated key exchange messages: used by both peers to perform authentication and the sharing of a session key (shared secret).
  - req-KEX-C1 message: a message sent from the client.
  - 401-KEX-S1 message: a message sent from the server as a response to a req-KEX-C1 message.
- Authentication verification messages: used by both peers to verify the authentication results.
  - req-VFY-C message: a message used by the client, requesting that the server authenticates and authorizes the client.
  - 200-VFY-S message: a successful response used by the server, and also asserting that the server is authentic to the client simultaneously.

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

2.2. Typical Flows of the Protocol

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

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

Figure 1: Typical communication flow for first access to resource

- As usual in general HTTP protocol designs, a client will at first request a resource without any authentication attempt (1). If the requested resource is protected by the authentication, the server will respond with a message requesting authentication (401-INIT) (2).
- The client processes the body of the message, and waits for the user to input the authentication credentials (such as a user name and a password). When the credentials to be used become available, the client will send a message with the authenticated key exchange (req-KEX-C1) to start the authentication (3).
- If the server has received a req-KEX-C1 message, the server looks up the user’s authentication information within its user database. Then the server creates a new session identifier (sid) that will be used to identify sets of the messages that follow it, and responds back with a message containing a server-side authenticated key exchange value (401-KEX-S1) (4).
- At this point (5), both peers calculate a shared "session secret" using the exchanged values in the key exchange messages. It is assumed that underlying authentication protocol will generate the same "session secret" on both sides only when the user authentication succeeds. This session secret will be used for the actual access authentication after this point.
- The client will send a request with a client-side authentication verification value (req-VFY-C) (6), generated from the client-owned session secret. The server will check the validity of the verification value using its own session secret.
- If the authentication verification value from the client was correct, it means that the client definitely owns the credentials required for authentication. (i.e. the client authentication succeeded.) The server will respond with a successful message (200-VFY-S) (7).
  When the client’s verification value is incorrect (e.g. because the user-supplied password was incorrect), the server will respond with the 401-INIT message (the same one as used in (2)) instead.
- The response (200-VFY-S) may contain the server-side authentication verification value (7). When the underlying authentication mechanism supports bidirectional authentication, clients can check server’s identity using this information.

2.3. Alternative Flows

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

- (case A) If the client knows that the resource is likely to require the authentication, the client MAY omit the first unauthenticated request (1) and immediately send a key exchange (req-KEX-C1 message). This will reduce one round-trip of messages.
- (case B) If both the client and the server previously shared a session secret associated with a valid session identifier (sid), the client MAY directly send a req-VFY-C message using the existing session identifier and corresponding session secret. This will further reduce one round-trip of messages.
  In such cases, the server MAY have thrown out the corresponding sessions from the session table.

In this case, the server will respond with a 401-STALE message, indicating a new key exchange is required. The client SHOULD retry constructing a req-KEX-C1 message in this case.

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

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

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


    (B) reusing session secret (re-authentication)

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

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

Figure 2: Several alternative flows on protocol

For more details, see Sections 7 and 8.

3. Message Syntax

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

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

Each authentication scheme using this template SHALL specify a single token specific to the underlying scheme (like Basic or Digest). All of the "auth-scheme" contained in all of those headers MUST be that token.

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

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

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

3.1. Values

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

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

3.1.1. Tokens

Tokens will have syntax of the "token" defined in HTTP. The canonical format for tokens are unquoted tokens.

3.1.2. Strings

All character strings outside ASCII character sets MUST be encoded using the UTF-8 encoding [RFC3629] for the ISO 10646-1 character set [ISO.10646-1.1993], without any leading BOM characters. Both peers are RECOMMENDED to reject any invalid UTF-8 sequences that might cause decoding ambiguities (e.g., containing <"> in the second or later byte of the UTF-8 encoded characters).

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

The canonical format for strings are quoted-string.

3.1.3. Numbers

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

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

Figure 3: BNF syntax for number types

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

The numbers represented as a hex-fixed-number MUST include an even number of characters (i.e. multiples of eight bits). Those values are case-insensitive, and SHOULD be sent in lower-case. When these values are generated from any cryptographic values, they SHOULD have their "natural length": if these are generated from a hash function, these lengths SHOULD correspond to the hash size; if these are representing elements of a mathematical set (or group), its lengths SHOULD be the shortest for representing all the elements in the set. For example, any results of SHA-256 hash function will be represented by 64 characters, and any elements in 2048-bit prime field (modulo a 2048-bit integer) will be represented by 512 characters, regardless of how much 0's will be appear in front of such representations. Session-identifiers and other non-cryptographically generated values are represented in any (even) length determined by the side who generates it first, and the same length SHALL be used throughout the all communications by both peers.

The numbers represented as base64-fixed-number SHALL be generated as follows: first, the number is converted to a big-endian radix-256 binary representation as an octet string. The length of the representation is determined in the same way as mentioned above. Then, the string is encoded using the Base 64 encoding [RFC4648] without any spaces and newlines. Implementations decoding base64-fixed-number SHOULD reject any input data with invalid characters, excess/insufficient paddings, or non-canonical pad bits (See Sections 3.1 to 3.5 of [RFC4648]).

The canonical format for integer and hex-fixed-number are unquoted tokens, and that for base64-fixed-number is quoted-string (as it will contain equal, plus signs and slashs).

4. Messages

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

To determine which message are expected to be sent, see Sections 7 and 8.

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 and the auth-scheme to be used. The parameters marked "mandatory" SHALL be contained in the message. The parameters marked "non-mandatory" MAY either be contained or omitted in the message. Each parameter SHALL appear in each headers exactly once at most.

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

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

4.1. 401-INIT and 401-STALE

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

algorithm:
: (mandatory token) specifies the authentication sub-algorithm to be used. The set of allowed value for this field MUST be specified within each specification for a specific authentication protocol.

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

validation:
: (mandatory token) specifies the method of host validation. The value MUST be one of the tokens described in Section 6, or the tokens specified in other supplemental specification documentation.

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

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

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

  - reauth-needed: server-side application requires a new authentication trial, regardless of the current status.
  - invalid-parameters: authentication was not even tried in the server-side because some parameters are not acceptable.
  - internal-error: authentication was not even tried in the server-side because there is some troubles on the server-side.

- user-unknown: a special case of auth-failed, suggesting that the provided user-name is invalid. The use of this parameter is NOT RECOMMENDED for security implications, except for special-purpose applications which makes this value sense.
- invalid-credential: ditto, suggesting that the provided user-name was valid but authentication was failed. The use of this parameter is NOT RECOMMENDED as the same as the above.
- authz-failed: authentication was successful, but access to the specified resource is not authorized to the specific authenticated user. (It is different from 403 responses which suggest that the reason of inaccessibility is other that authentication.)

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:

algorithm, realm:
: MUST be the same value as it is when received from the server.

user:
: (non-mandatory, string) is the UTF-8 encoded name of the user. This field MUST be present unless the authentication scheme defines other means of identifying the authenticating users other than the textual user name. If this name comes from a user input, client software SHOULD prepare the string using the preparation mechanism defined with each scheme (see Section 11 for more information) before encoding it to UTF-8.

kc1:
: (mandatory, algorithm-determined) is the client-side key exchange value K_{c1}, which is specified by the algorithm that is used.

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:

algorithm, realm:
: MUST be the same value as it is when received from the client.

sid:
: (mandatory, hex-fixed-number) MUST be a session identifier, which is a random integer. The sid SHOULD have uniqueness of at least 80 bits or the square of the maximal estimated transactions concurrently available in the session table, whichever is larger. See Section 5 for more details.

ks1:
: (mandatory, algorithm-determined) is the server-side key exchange value K_{s1}, which is specified by the algorithm.

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

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

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

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

4.4. req-VFY-C

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

The parameters contained in the header are as follows:

algorithm, realm:
: MUST be the same value as it is when received from the server for the session.

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

nc:
: (mandatory, integer) is a nonce value that is unique among the requests sharing the same sid. The values of the nonces SHOULD satisfy the properties outlined in Section 5.

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

4.5. 200-VFY-S

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

The parameters contained in the header are as follows:

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

algorithm, realm:
: MUST be the same value as it is when received from the client.

vks:
: (mandatory, algorithm-determined) is the server-side authentication verification value VK_S, which is specified by the algorithm. If the algorithm specification do not specify any specific value for this field, the value SHALL the token "0".

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. Session Management

In this authentication protocol template, a session represented by an sid is set up using first four messages (first request, 401-INIT, req-KEX-C1 and 401-KEX-S1). After sharing a session secret, this session, along with the secret, can be used for one or more requests for resources protected by the same realm in the same server. Note that session management is only an inside detail of the protocol and usually not visible to normal users. If a session expires, the client and server SHOULD automatically re-establish another session without informing the users.

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

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

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

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

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

Servers MUST check for duplication of the received nonces, and if any duplication is detected, the server MUST discard the session and respond with a 401-STALE message, as outlined in Section 8. The server MAY also reject other invalid nonce values (such as ones above the nc-max limit) by sending a 401-STALE message.

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

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

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

6. Host Validation Methods

The "validation method" specifies a method to "relate" (or "bind") authentication processed by this template with other authentications already performed in the underlying layers and to prevent man-in-the-middle attacks. It decides the value vh that is an input to the authentication protocols.

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

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

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

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

If the HTTP protocol is used on a non-encrypted channel (TCP and SCTP, for example), the validation type MUST be "host". If HTTP/TLS [RFC2818] (HTTPS) protocol is used with the server certificates, the validation type MUST be "tls-cert". If HTTP/TLS protocol is used without any kind of server certificates, the validation type MUST be "tls-key".

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

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

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

7. Decision Procedure for Clients

To securely implement the protocol, the user client must be careful about accepting the authenticated responses from the server. This also holds true for the reception of "normal responses" from HTTP servers.

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

Step 1 (step_new_request):
  If the client software needs to access a new Web resource, check whether the resource is expected to be inside some authentication realm for which the user has already been authenticated by the authentication scheme. If yes, go to Step 2. Otherwise, go to Step 5.

Step 2:
  Check whether there is an available sid for the authentication realm you expect. If there is one, go to Step 3. Otherwise, go to Step 4.

Step 3 (step_send_vfy_1):
  Send a req-VFY-C request.
  - If you receive a 401-INIT message with a different authentication realm than expected, go to Step 6.
  - If you receive a 401-STALE message, go to Step 9.
  - If you receive a 401-INIT message, go to Step 13.
  - If you receive a 200-VFY-S message, go to Step 14.
  - If you receive a normal response, go to Step 11.

Step 4 (step_send_kex1_1):
  Send a req-KEX-C1 request.
  - If you receive a 401-INIT message with a different authentication realm than expected, go to Step 6.
  - If you receive a 401-KEX-S1 message, go to Step 10.
  - If you receive a 401-INIT message with the same authentication realm, go to Step 13 (see Note 1).
  - If you receive a normal response, go to Step 11.

Step 5 (step_send_normal_1):
  Send a request without any authentication headers related to this specification.
  - If you receive a 401-INIT message, go to Step 6.
  - If you receive a normal response, go to Step 11.

Step 6 (step_rcvd_init):
  Check whether you know the user’s authentication credential for the requested authentication realm. If yes, go to Step 7. Otherwise, go to Step 12.

Step 7:
  Check whether there is an available sid for the authentication realm you expect. If there is one, go to Step 8. Otherwise, go to Step 9.

Step 8 (step_send_vfy):
  Send a req-VFY-C request.
  - If you receive a 401-STALE message, go to Step 9.
  - If you receive a 401-INIT message, go to Step 13.

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

Step 9 (step_send_kex1):

Send a req-KEX-C1 request.

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

Step 10 (step_rcvd_kex1):

Send a req-VFY-C request.

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

Step 11 (step_rcvd_normal):

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

Step 12 (step_rcvd_init_unknown):

The requested resource requires a authentication, and the user is not yet authenticated. The client will be in the "AUTH-REQUESTED" status, and is RECOMMENDED to process the content sent from the server, and to ask user for any user’s authentication credentials. 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 used authentication credentials are invalid for the authenticated resource. Forget such authentication credentials (or disable, whichever appropriate for the specific kind of credentials) for the authentication realm and go to Step 12.

Step 14 (step_rcvd_vfy):

Check the validity of the received VK_S value. If it is equal to the expected value, it means that the server authentication has succeeded. The client will be in the "AUTH-SUCCEEDED" status.

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

Note 1:

These transitions MAY be accepted by clients, but NOT RECOMMENDED for servers to initiate.

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

Figure 4 shows a diagram of the client-side state.

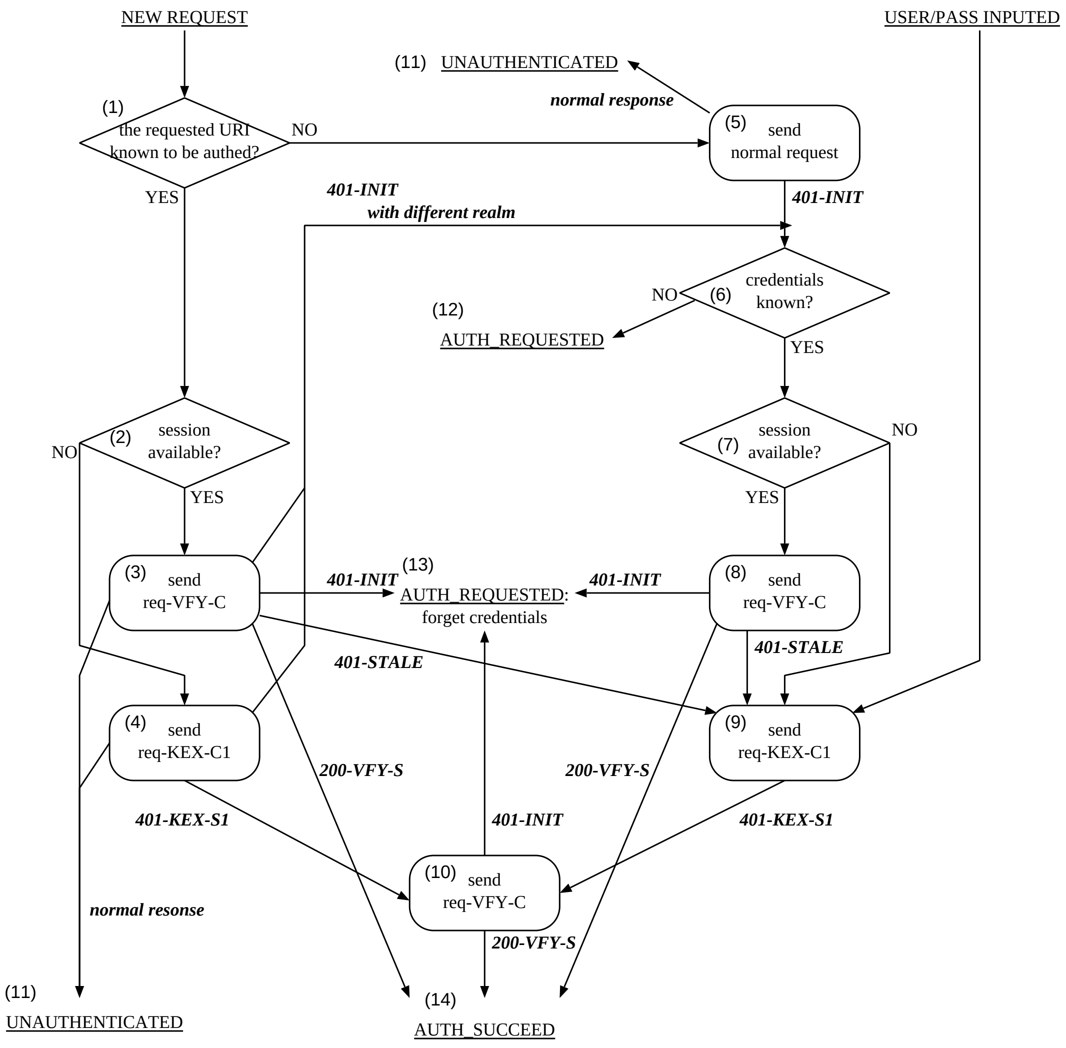

Figure 4: State diagram for clients

8. Decision Procedure for Servers

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

- The session identifier, the value of the sid parameter.
- The algorithm used.
- The authentication realm.
- The state of the protocol: one of "key exchanging", "authenticated", "rejected", or "inactive".
- The user name received from the client
- The boolean flag noting whether or not the session is fake.
- When the state is "key exchanging", the values of K_{c1} and S_{s1}.
- When the state is "authenticated", the following information:
  - The value of the session secret z
  - The largest nc received from the client (largest-nc)

    - For each possible nc values between (largest-nc - nc-window + 1) and max_nc, a flag whether or not a request with the corresponding nc has been received.

The table MAY contain other information.

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

- When the server receives a normal request:
    - If the requested resource is not protected by the authentication, send a normal response.
    - If the resource is protected by the authentication, send a 401-INIT response.
- When the server receives a req-KEX-C1 request:
    - If the requested resource is not protected by the 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. (Note: the server SHOULD NOT send a 401-INIT response in this case, because it will leak the information to the client that the specified user will not be accepted. Instead, postpone it to the response for the next req-VFY-C request.)
    - Otherwise, create a new session, compute K_{s1} and send a 401-KEX-S1 response.

  The created session has the "key exchanging" state.
- When the server receives a req-VFY-C request:
    - If the requested resource is not protected by the 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 lookup the session corresponding to the received sid and the authentication realm.
    - If the session corresponding to the received sid could not be found, or it is in the "inactive" state, send a 401-STALE response.
    - If the session is in the "rejected" state, send either a 401-INIT or a 401-STALE message.
    - If the session is in the "authenticated" state, and the request has an nc value that was previously received from the client, send a 401-STALE message. The session SHOULD be changed to the "inactive" status.
    - If the nc value in the request is larger than the nc-max parameter sent from the server, or if it is not larger then (largest-nc - nc-window) (when in "authenticated" status), the server MAY (but not REQUIRED to) send a 401-STALE message. The session SHOULD be changed to the "inactive" status if so.
    - If the session is a "fake" session, or if the received vkc is incorrect, then send a 401-INIT response. If the session is in the "key exchanging" state, it SHOULD be changed to the "rejected" state; otherwise, it MAY either be changed to the "rejected" status or kept in the previous state.
    - Otherwise, send a 200-VFY-S response. If the session was in the "key exchanging" state, the session SHOULD be changed to an "authenticated" state. The maximum nc and nc flags of the state SHOULD be updated properly.

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

9. Applying for Specific Authentication Schemes

Each authentication scheme to use this template MUST at least provide a definitions for the following functions:

- A token for distinguishing the protocol from any others (like Basic or Digest), to be used as "auth-scheme"s.
- A set of tokens which will be allowed in the "algorithm" field of 401-INIT message.
- * A string preparation algorithm based on [I-D.ietf-precis-framework]. (see Section 11)

Furthermore, for each sub-algorithm defined by the "algorithm" field, the following MUST be defined:

- A format for representing fields "kc1", "ks1", "vkc" and "vks".
- An algorithm for computing key exchange values K_{c1}, K_{s1}.
- A hash function H to be used with the algorithm.
- * An algorithm for computing authentication confirmation values VK_c, VK_s. Values derived by these algorithms SHOULD depend on the value of "nc" value used for each re-authenticating requests using the same "sid". It SHOULD also depend on the value of the host verification value "vh".
- * If possible, an algorithm for computing "application channel binding keys" (see Section 10).

For items marked with asterisks (*), default template functions are provided in the following sections.

9.1. Default Functions for Algorithms

If there are no specific (such as compatibility) requirements for values VK_c, VK_s, schemes MAY use the default functions for computing VK_c and VK_s, defined in this section. Designers of specific authentication schemes MAY choose either to use this default function or not, depending on the nature and the background settings for each authentication schemes to be defined.

To use this default function, the algorithm specification SHALL specify the following values.

- Shared secret z, to be computed in both server-side and client side using exchanged values.

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

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

The definitions of any support functions in the above definitions are provided in Appendix A.

10. Application Channel Binding

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

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

The following definitions are assuming that the authentication scheme uses the default function shown above for computing VK_c and VK_s. If not, the specification for the authentication scheme is encouraged to provide an alternative means for this purpose (e.g., either to specify the function for z, or to specify functions b_1 and b_2).

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

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

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

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

The definitions of any support functions in the above definitions are provided in Appendix A.

11. String Preparation

For proper internationalization of the protocol to be designed, each authentication scheme SHOULD specify algorithms for preparing string inputs, unless the underlying protocol does not use any kind of human-readable (i.e., possibly-non-ASCII-capable) identifier or passwords.

If some algorithm is suitable for each specific authentication scheme in relation to other existing protocols, that one should be used (e.g. [RFC4013] or [I-D.melnikov-precis-saslprepbis] for any SASL-related authentication algorithms).

If there is no specific one to be chosen, schemes may choose the following default choice: use [I-D.melnikov-precis-saslprepbis] for user-identifiers and passwords-like strings, except that case mapping of upper-case and title-case letters will NOT be applied (i.e., the string will be left case-sensitive, for keeping compatibility with existing HTTP-based authentication mechanisms).

12. Application for Proxy Authentication

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

- The 407 status is to be sent and recognized for places where the 401 status is used,
- Proxy-Authenticate: header is to be used for places where WWW-Authenticate: is used,
- Proxy-Authorization: header is to be used for places where Authorization: is used,
- Proxy-Authentication-Info: header is to be used for places where Authentication-Info: is used,
- The omission of the path parameter of 401-KEX-S1 messages means that the authentication realm will potentially cover all requests processed by the proxy,
- The scheme, host name and the port of the proxy is used for host validation tokens.

13. Methods to extend this protocol template

The template is designed to have fair amount of flexibility for implementing several authentication schemes. However, if needed, specifications defining authentication schemes or authentication algorithms MAY define its own representations for the parameters "kc1", "ks1", "vkc", and "vks", and/or add parameters to the messages containing those parameters in supplemental specifications, provided that syntactic and semantic requirements in Section 3, [I-D.ietf-httpbis-p1-messaging] and [I-D.ietf-httpbis-p7-auth] are satisfied.

If there is more than two round-trips of messages needed for performing authentication, messaged named "req-KEX-C2", "401-KEX-S2", "req-KEX-C3" and so on MAY be used between 401-KEX-S1 and req-VFY-C messages. These messages MUST have algorithm, realm, and sid fields as the same as req-KEX-C1 and 401-KEX-S1. and they SHOULD have fields named "kc2", "ks2", "kc3" and so on, respectively.

It is RECOMMENDED that 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. It is strongly encouraged that specifications for authentication schemes do not rename or remove there fields, as they are important for distinguishing message types.

14. IANA Considerations

[TBD]

15. Security Considerations

15.1. Security Properties

- The protocol template relies on transport security including DNS integrity for data secrecy and integrity, regardless of any underlying authentication algorithm to be used. HTTP/TLS SHOULD be used where transport security is not assured and/or data secrecy is important.
- When used with HTTP/TLS, if TLS server certificates are reliably verified, the protocol provides true protection against active man-in-the-middle attacks.

15.2. Denial-of-service Attacks to Servers

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

However, if a malicious client sends too many requests of key exchanges (req-KEX-C1 messages) only, resource starvation might occur. In such critical situations, servers MAY discard any kind of existing sessions regardless of these statuses. One way to mitigate such attacks are that servers MAY have a number and a time limits for unverified pending key exchange requests (in the "wa received" status).

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

16. References

16.1. Normative References

[I-D.ietf-httpbis-p1-messaging]	Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," draft-ietf-httpbis-p1-messaging-21 (work in progress), October 2012 (TXT).
[I-D.ietf-httpbis-p7-auth]	Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication," draft-ietf-httpbis-p7-auth-21 (work in progress), October 2012 (TXT).
[I-D.ietf-precis-framework]	Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation and Comparison of Internationalized Strings in Application Protocols," draft-ietf-precis-framework-06 (work in progress), September 2012 (TXT).
[I-D.melnikov-precis-saslprepbis]	Saint-Andre, P. and A. Melnikov, "Preparation and Comparison of Internationalized Strings Representing Simple User Names and Passwords," draft-melnikov-precis-saslprepbis-04 (work in progress), September 2012 (TXT).
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC3629]	Yergeau, F., "UTF-8, a transformation format of ISO 10646," STD 63, RFC 3629, November 2003 (TXT).
[RFC4013]	Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords," RFC 4013, February 2005 (TXT).
[RFC4648]	Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," RFC 4648, October 2006 (TXT).
[RFC5234]	Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," STD 68, RFC 5234, January 2008 (TXT).
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008 (TXT).

16.2. Informative References

[ISO.10646-1.1993]	International Organization for Standardization, “Information Technology - Universal Multiple-octet coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane,” ISO Standard 10646-1, May 1993.
[ITU.X690.1994]	International Telecommunications Union, “Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),” ITU-T Recommendation X.690, 1994.
[RFC2617]	Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” RFC 2617, June 1999 (TXT, HTML, XML).
[RFC2818]	Rescorla, E., “HTTP Over TLS,” RFC 2818, May 2000 (TXT).
[RFC5280]	Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” RFC 5280, May 2008 (TXT).
[RFC5890]	Klensin, J., “Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework,” RFC 5890, August 2010 (TXT).
[RFC5929]	Altman, J., Williams, N., and L. Zhu, “Channel Bindings for TLS,” RFC 5929, July 2010 (TXT).

Appendix A. (Normative) Support Functions and Notations

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

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

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

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

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

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

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

Some examples:

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

VI(100) = "d"

VI(10000) = "\316\020"

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

VS("") = "\000"

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

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

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

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

The function OCTETS converts an integer into the corresponding radix-256 big-endian octet string having its natural length: See Section 3.1.3 for the definition of "natural length".

Appendix B. (Informative) Draft Remarks from Authors

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

- Whether to keep TLS-key validation or not.
- When keeping tls-key validation, whether to use "TLS channel binding" [RFC5929] for "tls-key" verification (Section 6). Note that existing TLS implementations should be considered to determine this.

Authors' Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
Tsukuba Central 2
1-1-1 Umezono
Tsukuba-shi, Ibaraki
JP
Email: mutual-auth-contact-ml@aist.go.jp

Hajime Watanabe
National Institute of Advanced Industrial Science and Technology

Hiromitsu Takagi

National Institute of Advanced Industrial Science and Technology

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

Tatsuya Hayashi
Lepidum Co. Ltd.

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