Internet Engineering Task Force (IETF)
Request for Comments: 8120
Category: Experimental
ISSN: 2070-1721

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

April 2017

Mutual Authentication Protocol for HTTP

Abstract

This document specifies an authentication scheme for the Hypertext Transfer Protocol (HTTP) that is referred to as either the Mutual authentication scheme or the Mutual authentication protocol. 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 document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8120.

Copyright Notice

Copyright (c) 2017 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

This document specifies an authentication scheme for the Hypertext Transfer Protocol (HTTP) that is referred to as either the Mutual authentication scheme or the Mutual authentication protocol. This scheme 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 the most critical threats for Web systems. Plain-text password authentication techniques (Basic authentication and Web-form-based authentication) have been widely used for a long time. When these techniques are used with plain HTTP protocols, it is trivially easy for attackers to sniff the password credentials on the wire.

The Digest authentication scheme [RFC7616] uses SHA-256 and SHA-512/256 (formerly SHA-1 and MD5) hash algorithms to hide the raw user password from network sniffers. However, if the number of possible candidate users' passwords is not enough, newer and more 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 an offline password dictionary attack; the search capacity of these newer

computers reduces the effectiveness of users' memorable passwords, thereby threatening the effectiveness of such hash-based password protections.

Transport Layer Security (TLS) [RFC5246] provides 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 to any outside attackers. However, there is a pitfall related to TLS deployment on Web systems: if the users are fraudulently routed to a "wrong Website" via some kind of social engineering attack (e.g., phishing) and tricked into performing authentication on that site, the credentials will be sent to the attacker's server and trivially leaked. Attacks such as phishing have become a serious threat. In current Web system deployments, TLS certificates will be issued to almost any users of the Internet (including malicious attackers). Although those certificates include several levels of the "validation results" (such as corporate names) of the issued entities, the task of "checking" those validation results is left to the users of Web browsers, still leaving open the possibility of such social engineering attacks.

Another way to avoid such threats is to avoid password-based authentication and use some kinds of pre-deployed strong secret keys (on either the client side or the server side) for authentications. Several federated authentication frameworks, as well as HTTP Origin-Bound Authentication (HOBA) [RFC7486], are proposed and deployed on real Web systems to satisfy those needs. However, a type of authentication based on "human-memorable secrets" (i.e., passwords) is still required in several scenarios, such as initialization, key deployment to new clients, or recovery of secret accounts with lost cryptographic keys.

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

- No password information at all is exchanged in the communications. When the server and the user fail to authenticate with each other, the protocol will not reveal even the tiniest bit of information about the user's password. This prevents any kind of offline password dictionary attacks, even with the existence of phishing attacks.

- To successfully authenticate, the server, as well as client users, must own the valid registered credentials (authentication secret). This means that a phishing attacker cannot trick users into thinking that it is an "authentic" server. (It should be

pointed out that this is not true for Basic and Digest authentication; for example, servers using Basic authentication can answer "YES" to any clients without actually checking authentication at all.) Client users can ascertain whether or not the communicating peer is truly "the server" that registered their account beforehand. In other words, it provides "true" mutual authentication between servers and clients.

Given the information above, the proposed protocol can serve as a strong alternative to the Basic, Digest, and Web-form-based authentication schemes and also as a strong companion to the non-password-based authentication frameworks.

The proposed protocol will serve in the same way as does existing Basic or Digest authentication: it meets the requirements for new authentication schemes 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 that Web browsers have some "secure" way of displaying the authentication results. Having such a user interface in future browsers will greatly reduce the risk of impersonation by various kinds of social engineering attacks, in a manner similar to that of the "green padlock" for Extended Validation 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 correspond 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 that understands and implements HTTP and the specified authentication protocol -- usually computer software; a "user" is typically a human being 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 as 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:

- Section 2 presents an overview of the protocol design.
- Sections 3 through 11 define a general framework of the Mutual authentication protocol. This framework is independent of specific cryptographic primitives.
- Section 12 describes properties needed for cryptographic algorithms used with this protocol framework and defines a few functions that will be shared among such cryptographic algorithms.
- Sections 13 through 15 contain general normative and informative information about the protocol.
- Sections 16 and 17 describe IANA considerations and security considerations, respectively.

In addition, we will refer to the following two companion documents, as they are related to this specification:

- [RFC8121] defines cryptographic primitives that can be used with this protocol framework.
- [RFC8053] 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 HTTP [RFC7230] and uses the 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 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

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

- Authentication request messages: used by the servers to request that clients 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 the client has to start a new key exchange.
- 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.
- Authentication verification messages: used by both peers to verify the authentication results.
  - req-VFY-C message: a message used by the client to request that the server authenticate and authorize the client.
  - 200-VFY-S message: a response used by the server to indicate that client authentication succeeded. It also contains information necessary for the client to check the authenticity of the server.

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

2.2. Typical Flows of the Protocol

In typical cases, client access to a resource protected by the Mutual authentication scheme will use the following protocol sequence:

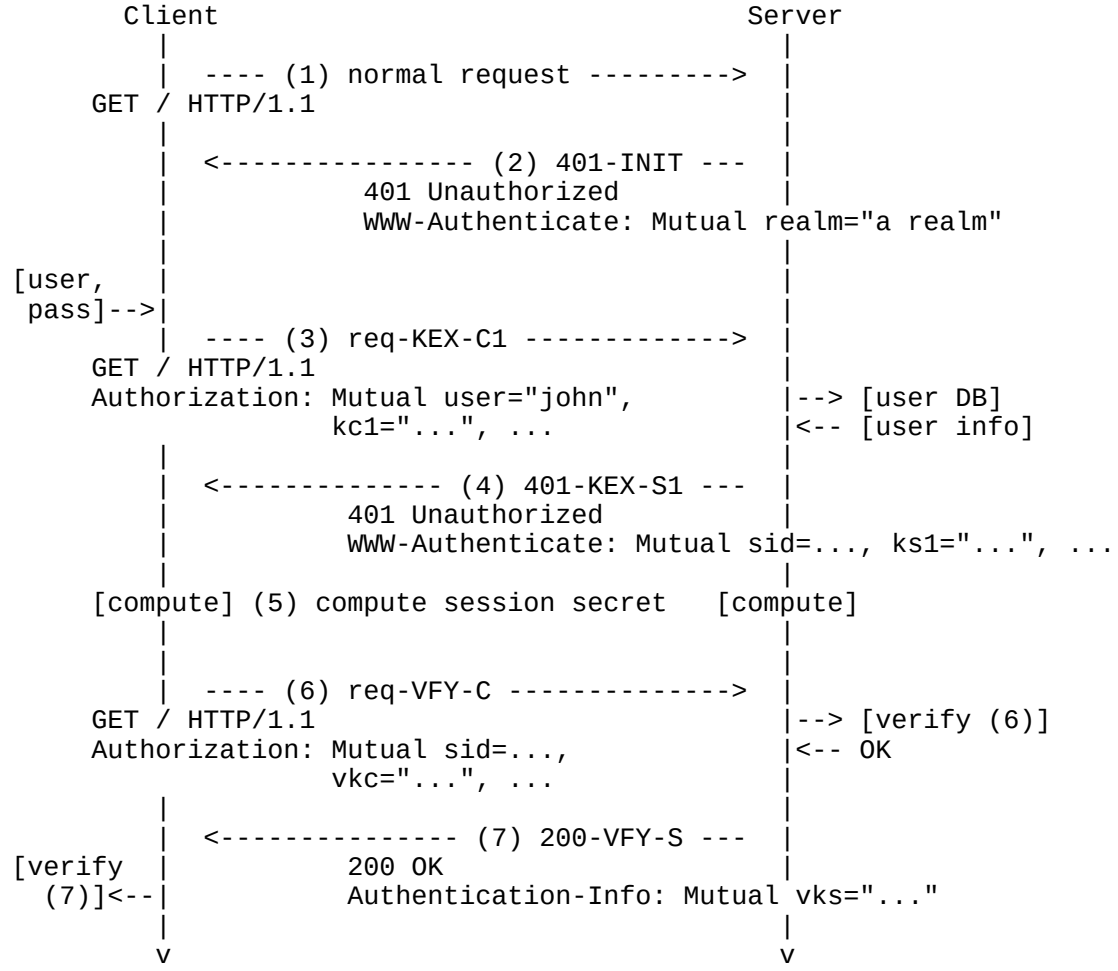

Figure 1: Typical Communication Flow for First Access to Resource

- As is typical 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 protocol, 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 username and password. If the username and password are available, the client will send a message with the authenticated key exchange (req-KEX-C1) to start the authentication (3).

- If the server has received a req-KEX-C1 message, the server looks up the user's authentication information within its user database. Then, the server creates a new session identifier (sid) that will be used to identify sets of the messages that follow it and responds with a message containing a server-side authenticated key exchange value (401-KEX-S1) (4).

- At this point (5), both peers calculate a shared "session secret" using the exchanged values in the key exchange messages. Only when both the server and the client have used secret credentials generated from the same password will the session secret values match. This session secret will be used for access authentication of every individual request/response pair after this point.

- 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.

- If the authentication verification value from the client was correct, then 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). Unlike 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 a 401-INIT message (the same message as the message used in (2)) instead.

- The client MUST first check the validity of the server-side authentication verification value contained in the message (7). If the value was equal to the expected value, server authentication succeeded.

If it is not the expected value or the message does not contain the authentication verification value, then 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; such a scenario could be caused by either a man-in-the-middle attack or incorrect implementation.)

2.3. Alternative Flows

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

- 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 the number of round trips by one.

- Case B: If both the client and the server previously shared a session secret associated with a valid sid, the client MAY directly send a req-VFY-C message using the existing sid and corresponding session secret. This will further reduce the number of round trips by one.

  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 that a new key exchange is required. The client SHOULD try again to construct a req-KEX-C1 message in this case.

Figure 2 depicts the shortcut flows described above. When using appropriate settings and implementations, most of the requests to resources are expected to meet both criteria; thus, only one round trip of request/response will be required.

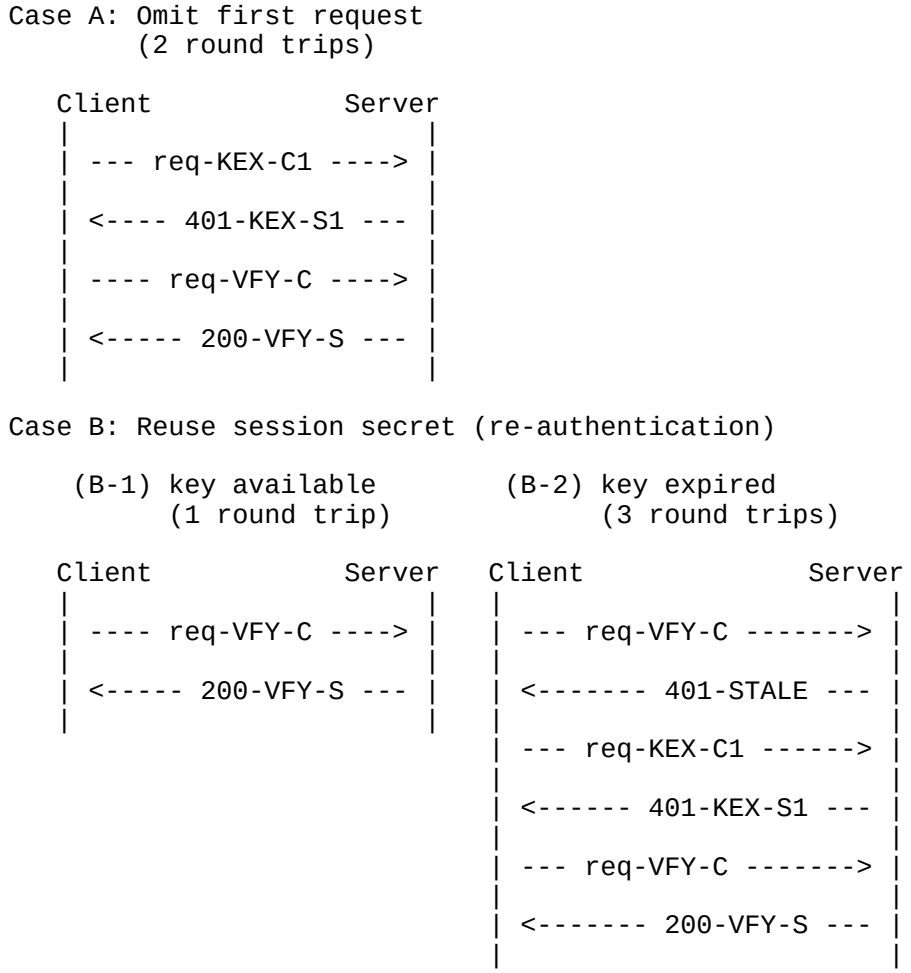

Figure 2: Several Alternative Protocol Flows

For more details, see Sections 10 and 11.

3. Message Syntax

Throughout this specification, the syntax is denoted in the extended augmented BNF syntax as defined in [RFC7230] and [RFC5234]. The following elements are used in this document per [RFC5234], [RFC7230], and [RFC7235]: DIGIT, ALPHA, SP, auth-scheme, quoted-string, auth-param, header-field, token, challenge, and credentials.

The Mutual authentication protocol uses three headers: WWW-Authenticate (usually in responses with a 401 status code), Authorization (in requests), and Authentication-Info (in responses other than a 401 status code). These headers follow the frameworks described in [RFC7235] and [RFC7615]. See Section 4 for more details regarding these headers.

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 element contained in those headers MUST be set to "Mutual" when using the protocol specified in this document. The syntax for "challenge" and "credentials" to be used with the "Mutual" auth-scheme SHALL be name-value pairs (#auth-param), not the "token68" parameter 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 correctly handling, those cases.

3.1. Non-ASCII Extended Header Parameters

All of the parameters contained in the above three headers, except for 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 per the syntax specified in [RFC5987].

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

If a value (except for 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 syntax used.

For example, a parameter "user" with the 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 that the "realm" parameter be in its plain form (not as an extended "realm*" parameter), so the syntax specified in RFC 5987 MUST NOT be used for this parameter.

3.2. Values

The parameter values contained in challenges or credentials MUST be parsed in strict conformance with HTTP semantics (especially the unquoting of 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, it is RECOMMENDED that implementations 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 the parsing of HTTP values):

```
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 functions 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 (see 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 those described in 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 UTF-8 encoding [RFC3629] for the Unicode character set [Unicode]. Such strings MUST NOT contain any leading Byte Order Marks (BOMs) (also known as ZERO WIDTH NO-BREAK SPACE, U+FEFF, or EF BB BF). It is RECOMMENDED that both peers reject any invalid UTF-8 sequences that might cause decoding ambiguities (e.g., containing <"> in the second or subsequent bytes of the UTF-8 encoded characters).

If strings represent a domain name or URI that contains non-ASCII characters, the host parts SHOULD be encoded as they (the parts) are used in the HTTP protocol layer (e.g., in a Host: header); per current standards, the A-label as defined in [RFC5890] will be used. Lowercase ASCII characters SHOULD be used.

The canonical format for strings is quoted-string (as it may contain equals signs ("="), plus signs ("+"), and slashes ("/")), unless the parameter containing the string value will use extended syntax as defined in [RFC5987]. (Per [RFC5987], an extended parameter will have an unquoted encoded value.)

3.2.3. Numbers

The following syntax definitions provide 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 they are generated from a hash function, their lengths correspond to the hash size; if they represent elements of a mathematical set (or group), their lengths SHALL be the shortest lengths that represent 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 many 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 in 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 the technique mentioned above. Then, the string is encoded using base64 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 or insufficient padding, or non-canonical pad bits (see Sections 3.1 through 3.5 of [RFC4648]).

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

4. Messages

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

To determine under what circumstances each message is expected to be sent, see Sections 10 and 11.

In the descriptions below, the types of allowable values for each header parameter are shown in parentheses 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 be either contained in the message or omitted from it. 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 them. However, all credentials and challenges MUST meet the following criteria:

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

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

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

4.1. 401-INIT and 401-STALE

Every 401-INIT or 401-STALE message SHALL be a valid HTTP 401 (Unauthorized) status message (or some other 4xx status message, if appropriate) 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 [RFC8121] 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, i.e., 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 as described 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 encoding as specified in [RFC5987] 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 ("NFC" stands for "Normalization Form C"), or an error.

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

* initial: Authentication was not attempted because there was no Authorization header in the corresponding request.

* stale-session: The provided sid in the request was either unknown to the server or expired in the server.

* auth-failed: The authentication trial failed for some reason, possibly because of a bad authentication credential.

Implementations MAY support the following tokens or any extensive-tokens defined outside of this specification. If clients receive any unknown tokens, they SHOULD treat them 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 problems on the server side.

* user-unknown: This is a special case of auth-failed; it suggests that the provided username is invalid. Due to security implications, the use of this parameter is NOT RECOMMENDED, except for special-purpose applications where it would make sense to do so.

* invalid-credential: This is another special case of auth-failed; it suggests that the provided username was valid but authentication still failed. For security reasons, the use of this parameter is NOT RECOMMENDED.

* 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 (Unauthorized) or 403 (Forbidden) status code to indicate that the authentication result is one of the existing reasons for the failed authorization.)

It is RECOMMENDED that the reason for failure be recorded to some type of diagnostic log, shown to the client user immediately, or both. It will be helpful to find out later whether the reason for the failure is technical or caused by user error.

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, any messages with the "reason" parameter value "stale-session" will be called "401-STALE" messages hereafter, because these messages 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 those 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 (Unauthorized) status 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 those 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) is the number of available nonce number slots that the server will accept. It is RECOMMENDED that the value of the "nc-window" parameter 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 that the time be set to at least 60 (seconds). However, the server is not required to guarantee that the session represented by the sid will be available (e.g., alive, usable) for the time specified in this parameter.

path:
(non-mandatory, string) specifies to 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 that specified in the "domain" parameter [RFC7616] for Digest authentications. All path elements contained in the "path" parameter MUST be inside the specified auth-scope; if not, clients SHOULD ignore such elements. For better performance, it is important that clients recognize and use this parameter.

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 those 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 (Unauthorized) 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)" [RFC7231] 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 usernames and passwords is valid. If the server requests authentication for an authentication realm for which the client is already authenticated, the client will automatically perform the authentication using the already-known credentials. However, for different authentication realms, clients MUST NOT automatically reuse usernames and passwords for another realm.

As is the case for 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 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, it is NOT RECOMMENDED that server operators 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 spanned by the authentication realm. In this protocol, it MUST be one of the following kinds of strings:

- 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, as presented in Section 6.2 of [RFC6454].

- 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 several protocols (e.g., HTTP and HTTPS) and ports but will not span different hosts.

- 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, via 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"), it is RECOMMENDED that clients 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 lowercase characters, i.e., these domain names SHALL be in the form of LDH ("letters, digits, hyphen") labels as defined in the Internationalized Domain Names for Applications (IDNA) specification [RFC5890]. A "port" MUST be given in shortest unsigned decimal number notation. Not obeying these requirements will cause valid authentication attempts to fail.

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 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 realms to which the client can be authenticated. In such cases, clients face an ambiguous choice regarding which credentials to send for a new request (see Steps 3 and 4 of the decision procedure presented in Section 10).

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

The following procedure is one possible tactic for resolving ambiguities in such cases:

- If the client has previously sent a request to the same URI and it remembers the authentication realm requested by the 401-INIT message at that time, use that realm.

- 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 those with shorter domain-postfixes.

- 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 they are provided by the same server. The same session identifiers provided either from different servers or for different realms MUST be treated as being independent of 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 if 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:

- It is a natural number.
- The same nonce number 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 largest value of nc that 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 those above the nc-max limit) by sending a 401-STALE message.

For example, assume that the nc-window value of the current session is 128 and 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}. The nonce number that can then be used for the next request is a number from 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 that use fixed-width integer representations, fixed-precision floating-point numbers, or similar representations SHOULD NOT reject any larger values that 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 another possible secure transport is used, this corresponds to the idea of "channel binding" as 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:
: 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 server-side resource currently being accessed. The scheme and host are in lower case, and the port is listed in shortest decimal notation. Even if the request URI does not have a port part, vh 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-related notes regarding this validation method.)

If HTTP is used on a non-encrypted channel (TCP and the Stream Control Transmission Protocol (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 is NOT automatic in Diffie-Hellman-based key exchanges with separate exchanges 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 preferable:

- When TLS accelerating proxies are used. In this case, 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.

- 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 (support of dynamic contents), the underlying TLS channel used with HTTP/TLS MUST provide server identity verification. This means that (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 Section 3.1 of [RFC5929], the "tls-unique" verification value will be changed by possible TLS renegotiation, causing an interoperability problem. TLS renegotiations 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 precautions SHOULD be taken:

- Both peers must be aware that the vh values used for vkc (in req-VFY-C messages) and vks (in 200-VFY-S messages) may be different. These values MUST be retrieved from underlying TLS libraries each time they are used.

- 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 renegotiation 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, the TLS extension defined in [RFC7627] SHOULD be used when "tls-unique" host verification is to be used.

8. Authentication Extensions

It is RECOMMENDED that interactive clients (e.g., Web browsers) supporting this protocol support non-mandatory authentication and the Authentication-Control header defined in [RFC8053], except for the "auth-style" parameter. This specification also proposes (but does not mandate) that the default "auth-style" be "non-modal". Web applications SHOULD, however, consider the security impacts of the behavior 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

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

- Usernames received from users SHOULD be prepared using the "UsernameCasePreserved" profile defined in Section 3.3 of [RFC7613].

- 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 usernames 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 that does not contain mutual-authentication-related headers) from HTTP servers.

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

- Any kind of "normal response" MUST only be accepted for the very first request in the sequence. Any "normal response" returned for the second or subsequent requests in the sequence SHALL be considered invalid.

- By the same principle, if any response is related to an authentication realm that 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 subsequent request in the sequence SHALL be considered invalid.

- A req-KEX-C1 message MAY be sent as either an initial request or a response to a 401-INIT or 401-STALE message. However, to avoid infinite loops of messages, the req-KEX-C1 message SHOULD NOT be sent more than once in the sequence for a single authentication realm. A 401-KEX-S1 response MUST be accepted only when the corresponding request is req-KEX-C1.

- A req-VFY-C message MAY be sent if there is a valid session secret shared between the client and the server, as established by req-KEX-C1 and 401-KEX-S1 messages. If any response with a 401 status code is returned for such a message, the corresponding session secret SHOULD be discarded as unusable.

  In particular, upon the reception of a 401-STALE response, the client SHOULD try to establish a new session by sending a req-KEX-C1 message, but only once within the request/response sequence.

- A 200-VFY-S message MUST be accepted only as a response to a req-VFY-C message 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:

- AUTH-SUCCEED: A 200-VFY-S message with the correct VK_s value was returned in response to the req-VFY-C request in the sequence.

- AUTH-REQUIRED: Two cases exist:

  * 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 a req-VFY-C (or req-KEX-C1) message, which means that the user-supplied authentication credentials were not accepted.

- 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 then have a status of "UNAUTHENTICATED".

If a request is a sub-request for a resource included in another resource (e.g., embedded images, style sheets, frames), clients MAY treat an AUTH-REQUESTED status the same way they would treat an UNAUTHENTICATED status. In other words, the client MAY ignore the 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 of the security consequences of the above rules, we strongly advise that the decision procedure below be followed. In particular, clients SHOULD NOT accept "normal responses" unless explicitly allowed in the rules. The labels in the steps below 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 to see whether the resource is expected to be inside some authentication realm for which the user has already been authenticated via the Mutual authentication scheme. If yes, go to Step 2. Otherwise, go to Step 5.

Step 2:
Check to see 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 a 401-INIT message is received 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 to see 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 to see 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 authentication other than Mutual authentication, 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; it is RECOMMENDED that the client process the content sent from the server and ask the user for a username and password. When those are supplied by the user, go to Step 9.

Step 13 (step_rcvd_init_failed):
The authentication failed for some reason, possibly because the password or 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, then the mutual authentication succeeded. The client will be in the "AUTH-SUCCEED" status.

An unexpected value is interpreted as a fatal communication error.

If a user explicitly asks 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 it is NOT RECOMMENDED that servers initiate them.

Figure 5 shows an informative diagram of the client state.

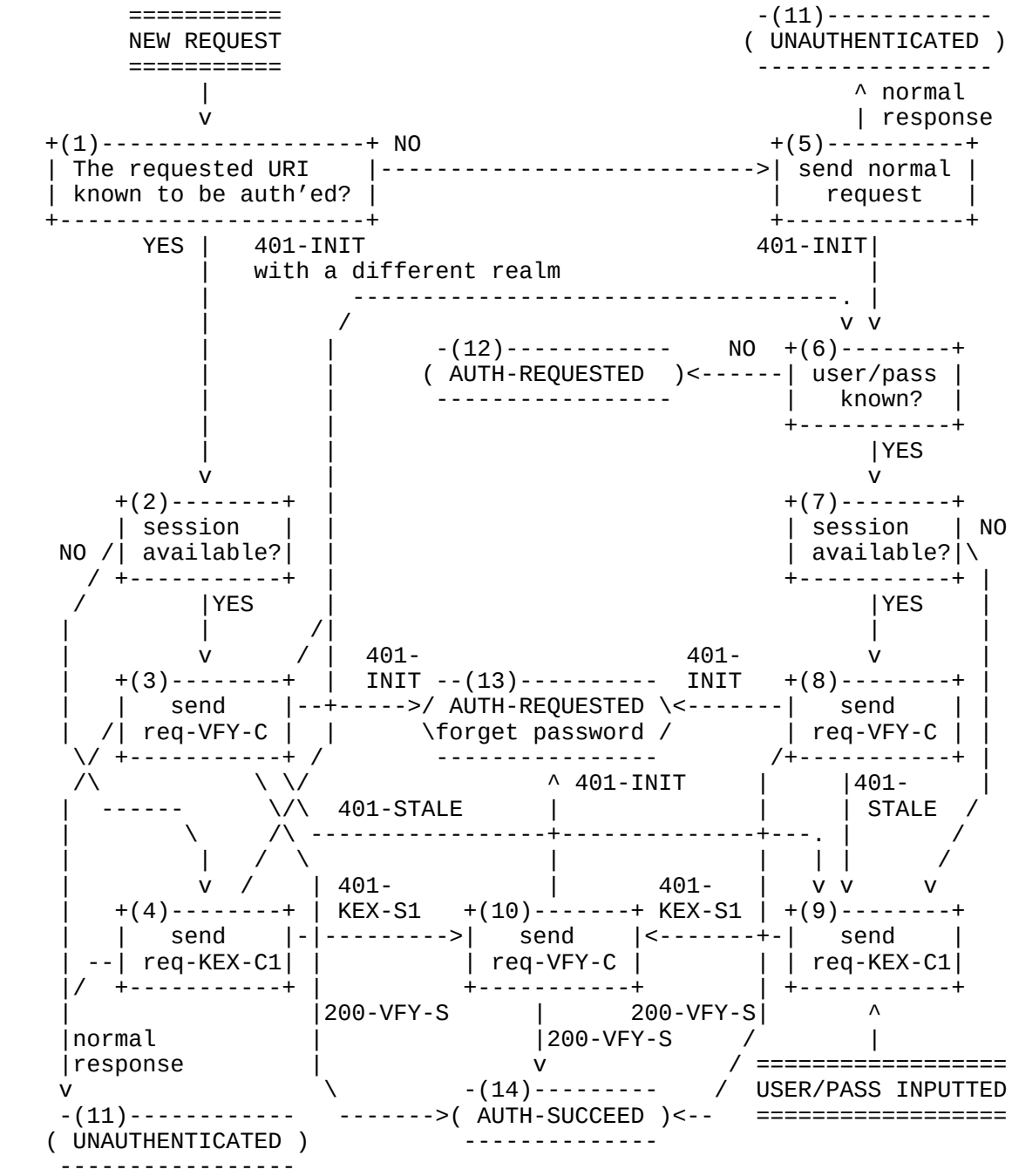

Figure 5: State Diagram for Clients

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:

- The session identifier, which is 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 username received from the client.
- A boolean flag indicating 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 value between (largest-nc - nc-window + 1) and max_nc, a boolean flag indicating 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 regarding 401-INIT messages with a plus sign):

- 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.

- 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 realm, send a 401-INIT response.
  - If the server cannot validate the parameter "kc1", send a 401-INIT (+) response.
  - If the received username is either invalid, unknown, or unacceptable, create a new session, mark it as 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 value is initialized to zero.

  The created session is in the "key exchanging" state.

- When the server receives a req-VFY-C request:
  - If the requested resource is not protected by the Mutual authentication, send a "normal response".
  - If the authentication realm specified in the req-VFY-C request is not the expected realm, send a 401-INIT response.

  If none of the 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 (+) response or a 401-STALE message.
  - If the nc value in the request is larger than the "nc-max" parameter sent from the server or it is not larger than (largest-nc - nc-window) (when in the "authenticated" state), 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 the 401-STALE message was sent.

* 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 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 be either 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 the "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" state 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 to, and following the specification of, the optional authentication defined in [RFC8053], the 401-INIT messages marked with plus signs cannot be replaced with a successful response with an Optional-WWW-Authenticate header. Every other 401-INIT can be a response with an Optional-WWW-Authenticate header.

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 username will not be accepted. Instead, postpone it until the response to the next req-VFY-C request.

Note 3: If the request is not rejected in this clause, the server will be required, in the next step, to determine whether the same nc value was previously received from the client. If that is impossible, the server MUST send a 401-STALE response in this step. If the server does not remember the whole history of the nc values received from the client, the server MUST send a 401-STALE message in this clause.

12. Authentication Algorithms

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

- The server-side authentication credential J, derived from the client-side authentication credential pi.
- Key exchange values K_c1, K_s1 (exchanged on the wire) and S_c1, S_s1 (kept secret in each peer).
- Shared session secret (z), to be computed by both server and client.
- A hash function H to be used with the protocol, along with its output size hSize.
- The value nIterPi, the number of iterations for the key derivation operation.

Specifications for cryptographic algorithms used with this framework MUST specify whether those algorithms will (1) use the default functions defined below for values pi, VK_c, and VK_s or (2) define their own comparable functions.

All algorithms used with this protocol SHOULD provide secure mutual authentication between clients and servers and generate a cryptographically strong shared secret value (z) that is equally strong or stronger than the hash function H. If any passwords (or passphrases 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 passwords. Furthermore, it is RECOMMENDED that the function J for deriving the server-side authentication credential J(pi) be one-way, if possible, so that pi cannot 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 for the last digit, which is represented by an octet within the range 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 encoding used for the subcomponents of object identifiers in ASN.1 encoding [ITU.X690.2015] and is 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 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)

(Note: Unlike the colon-separated format used in the Basic and Digest HTTP authentication schemes, 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 INT(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 = INT(PBKDF2(HMAC_H, password, VS(algorithm) | VS(auth-scope) |
VS(realm) | VS(username), nIterPi, hSize / 8))
```

where

- PBKDF2 is the password-based key derivation function defined in [RFC8018],
- HMAC_H is the Hashed Message Authentication Code (HMAC) function, defined in [RFC2104], composed from the hash function H, and
- 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 via the following equations:

```
VK_c = INT(H(octet(4) | OCTETS(K_c1) | OCTETS(K_s1) | OCTETS(z) |
VI(nc) | VS(vh)))

VK_s = INT(H(octet(3) | OCTETS(K_c1) | OCTETS(K_s1) | OCTETS(z) |
VI(nc) | 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 authenticated), the following value b_1 MAY be used:

```
b_1 = H(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 = 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 in the previous sections can be applied (with modifications) to proxy authentication. In such cases, the following alterations MUST be applied:

- The 407 (Proxy Authentication Required) status code is to be sent and recognized in places where the 401 status code is used,
- The Proxy-Authenticate header is to be used in places where the WWW-Authenticate header is used,
- The Proxy-Authorization header is to be used in places where the Authorization header is used,
- The Proxy-Authentication-Info header is to be used in places where the Authentication-Info header is used,

- The "auth-scope" parameter is fixed to the hostname of the proxy, which means that it covers all requests processed by the specific proxy,
- The limitation for the paths contained in the "path" parameter of 401-KEX-S1 messages is disregarded,
- The omission of the "path" parameter of 401-KEX-S1 messages means that the authentication realm will potentially cover all requests processed by the proxy,
- The scheme, hostname, and port of the proxy are used for host validation tokens, and
- Authentication extensions defined in [RFC8053] 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 extensions, as well as extensions that are in the process of being standardized, 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 of this document, [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) are reserved for this purpose. If those specifications use names other than those mentioned above, it is RECOMMENDED that extension-tokens be used to avoid any parameter-name conflicts 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

16.1. Addition to HTTP Authentication Schemes Registry

IANA has added the following entry to the "HTTP Authentication Schemes" registry:

- Authentication Scheme Name: Mutual
- Reference: RFC 8120

16.2. Registry for Authentication Algorithms

This document establishes the "HTTP Mutual Authentication Algorithms" registry. The registry manages case-insensitive ASCII strings. The strings MUST follow the extensive-token syntax defined in Section 3.

When bare-tokens are used for the authentication-algorithm parameter, they 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].

Registrations for an authentication algorithm are required to include descriptions of the authentication algorithms. Reviewers assigned by the IESG are advised to examine minimum security requirements and consistency of the key exchange algorithm descriptions.

It is advised that new registrations provide the following information:

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

[RFC8121] defines the initial contents of this registry.

16.3. Registry for Validation Methods

This document establishes the "HTTP Mutual Authentication Host Validation Methods" registry. The registry manages case-insensitive ASCII strings. The strings MUST follow the extensive-token syntax defined in Section 3.

When bare-tokens are used for the validation parameter, they 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].

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

It is advised that new registrations provide the following information:

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

The initial contents of this registry are as follows:

Token	Description	Reference
host	Hostname verification only	RFC 8120, Section 7
tls-server-end-point	TLS certificate-based	RFC 8120, Section 7
tls-unique	TLS unique key-based	RFC 8120, Section 7

17. Security Considerations

17.1. Security Properties

- The protocol is secure against passive eavesdropping and replay attacks. However, the protocol relies on transport security (including DNS integrity) for data secrecy and integrity. HTTP/TLS SHOULD be used where transport security is not assured and/or data confidentiality is important.

- When used with HTTP/TLS, if TLS server certificates are reliably verified, the protocol provides true protection against active man-in-the-middle attacks.

- Even if the server certificate is not used or is unreliable, the protocol provides protection against active man-in-the-middle attacks for each HTTP request/response pair. However, in such cases, JavaScript or similar scripts that are not authenticated by this authentication mechanism can affect mutually authenticated contents to circumvent the protection. This is why this protocol stipulates that valid TLS server certificates MUST be shown from the server to the client (Section 7).

17.2. Secrecy of Credentials

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

The use of a client-side credential with THIS authentication scheme is always safe, even if the connected server peer is not trustworthy (e.g., a phishing scenario). However, if it is used with other authentication schemes (such as Web forms) and the recipient is rogue, the result will be obvious.

It is also important that the server-side password credential (J) be kept secret. If it is stolen and the client’s choice of password is not strong, anyone who is aware of the server-side password credential can employ an offline dictionary attack to search for the client’s password. However, if the client has chosen a strong password so that an attacker cannot guess the client’s password from dictionary candidates, 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 the session is still active, session hijacking will result. After the session expires, the key is of no value to attackers.

17.3. Denial-of-Service Attacks on 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 be processed for each session identifier. After that, servers MAY discard sessions internally at any time without causing any operational problems for clients. Clients will then silently re-establish a new session.

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 set number and time limits 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 the Digest authentication scheme and most cookie-based authentication implementations. However, regarding resource consumption, the situation for the Mutual authentication scheme is slightly better than that 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, so that simple clients such as Web crawlers will not accidentally consume server-side resources for session management.

17.3.1. Online Active Password Attacks

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

Possible countermeasures for preventing such attacks may be the 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 with two goals in mind. The first goal is simply to provide a secure alternative to existing Basic and Digest authentication schemes. The second goal is to provide users with a way to detect forged rogue servers imitating (e.g., via a phishing attack) a user's registered account on a server.

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

Of course, in such cases, the user interfaces for requesting passwords for this authentication shall be protected against imitation (for example, 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 measure.

17.5. Implementation Considerations

- To securely implement the protocol, the Authentication-Info headers in the 200-VFY-S messages MUST always be validated by the client. If the validation fails, the client MUST NOT process any content sent with the message, including other headers and the body part. Non-compliance with this requirement will allow phishing attacks.

- For HTTP/TLS communications, when a Web form is submitted from mutually authenticated pages via 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, it is RECOMMENDED that the peer be revalidated using a req-KEX-C1 message with an "Expect: 100-continue" header. The same applies when the page is received via the "tls-unique" validation method and when the TLS session has expired.

- 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 pi.

- If TLS 1.2 [RFC5246] is used for underlying HTTP/TLS communications, follow the best practices specified in [RFC7525].

17.6. Usage Considerations

- The usernames 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 an HTTP server on the same host requests the Mutual authentication scheme within the same realm, the client will send the username in clear text. If usernames have to be kept secret (protected from eavesdroppers), the server must use the full-scheme-type "auth-scope" parameter and HTTPS. Passwords, on the other hand, are not exposed to eavesdroppers, even in HTTP requests.

- If the server provides several ways to store server-side password secrets in the password database, it is desirable, for purposes of better security, to store the values encrypted by using the one-way function J(pi) instead of the real passwords or those hashed by pi.

18. References

18.1. Normative References

[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>.

[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>.

[RFC8018] Moriarty, K., Ed., Kaliski, B., and A. Rusch, "PKCS #5: Password-Based Cryptography Specification Version 2.1", RFC 8018, DOI 10.17487/RFC8018, January 2017, <http://www.rfc-editor.org/info/rfc8018>.

[RFC8053] Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "HTTP Authentication Extensions for Interactive Clients", RFC 8053, DOI 10.17487/RFC8053, January 2017, <http://www.rfc-editor.org/info/rfc8053>.

[Unicode] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>.

18.2. Informative References

[ITU.X690.2015]
International Telecommunication 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, ISO/IEC 8825-1, August 2015, <https://www.itu.int/rec/T-REC-X.690/>.

[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>.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7486] Farrell, S., Hoffman, P., and M. Thomas, "HTTP Origin-Bound Authentication (HOBA)", 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>.

[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>.

[RFC8121] Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "Mutual Authentication Protocol for HTTP: Cryptographic Algorithms Based on the Key Agreement Mechanism 3 (KAM3)", RFC 8121, DOI 10.17487/RFC8121, April 2017, <http://www.rfc-editor.org/info/rfc8121>.

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
Japan
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
Japan
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
Japan
Email: takagi.hiromitsu@aist.go.jp

Kaoru Maeda
Individual Contributor
Email: kaorumaeda.ml@gmail.com

Tatsuya Hayashi
Lepidum Co. Ltd.
Village Sasazuka 3, Suite #602
1-30-3 Sasazuka
Shibuya-ku, Tokyo
Japan
Email: hayashi@lepidum.co.jp

Yuichi Ioku
Individual Contributor
Email: mutual-work@ioku.org