Mutual Authentication Protocol for HTTP

draft-oiwa-http-mutualauth-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 22, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

This document specifies the "Mutual authentication protocol for Hyper-Text Transport Protocol". This protocol provides true mutual authentication between HTTP clients and servers using simple password-based authentication. Unlike Basic and Digest HTTP access authentication protocol, the protocol ensures that server knows the
user's entity (encrypted password) upon successful authentication. This prevents common phishing attacks: phishing attackers cannot convince users that the user has been authenticated to the genuine website. Furthermore, even when a user has been authenticated against an illegitimate server, the server cannot gain any bit of information about user's passwords. The protocol is designed as an extension to the HTTP protocol, and the protocol design intends to replace existing authentication mechanism such as Basic/Digest access authentications and form-based authentications.

Table of Contents

1. Introduction ................................................. 3
   1.1. Requirements Language ................................. 4
2. Protocol Overview ............................................ 4
3. Message Syntax .............................................. 5
   3.1. Tokens and Extensive-tokens .......................... 6
   3.2. Numbers ............................................. 6
   3.3. Strings ............................................ 7
4. Messages ................................................... 7
   4.1. 401-B0 ............................................. 8
   4.2. 401-B0-stale ....................................... 9
   4.3. req-A1 ............................................ 9
   4.4. 401-B1 ............................................ 9
   4.5. req-A3 ............................................ 10
   4.6. 200-B4 ........................................... 11
5. Decision procedure for the client ............................ 12
6. Authentication Algorithms .................................. 14
7. Validation Methods ......................................... 16
8. Session Management ......................................... 17
9. Extension 1: Optional Mutual Authentication ................ 17
10. Methods to extend this protocol ........................... 18
11. IANA Considerations ....................................... 19
12. Security Considerations ................................... 19
   12.1. General Assumptions ................................. 19
   12.2. Implementation Considerations ..................... 19
13. Notice on intellectual properties .......................... 20
14. References .............................................. 20
   14.1. Normative References ............................... 20
   14.2. Informative References ............................ 21
Authors' Addresses .......................................... 22
Intellectual Property and Copyright Statements ................ 23
Introduction

This document specifies the "Mutual authentication protocol for Hyper-Text Transport Protocol". This protocol provides true mutual authentication between HTTP clients and servers using simple password-based authentication. Unlike Basic and Digest HTTP access authentication protocol [RFC2617], the protocol ensures that server knows the user's entity (encrypted password) upon successful authentication. This prevents common phishing attacks: phishing attackers cannot convince users that the user has been authenticated to the genuine website. Furthermore, even when a user has been authenticated against an illegitimate server, the server cannot gain any bit of information about user's passwords.

Recently, phishing attacks are getting more and more sophisticated. Phishers not only steal user’s password directly, but imitate successful authentication to steal user’s sensitive information, check the password validity by forwarding the password to the legitimate server, or employ a man-in-the-middle attack to hijack user’s login session. Existing countermeasures such as one-time passwords cannot completely solve these problems.

The protocol prevents such attacks by providing users a way to discriminate between true and fake web servers using their own passwords. Even when a user inputs his/her password to a fake website, using this authentication method, any information about the password does not leak to the phisher, and the user certainly notices that the mutual authentication has failed. Phishers cannot make such authentication attempt succeed, even if they forward received data from a user to the legitimate server or vice versa. Users can safely input sensitive data to the web forms after confirming that the mutual authentication has succeeded.

To achieve this goal, this protocol uses a mechanism in ISO/IEC 11770-4 [ISO.11770-4.2006], a kind of PAKE (Password-Authenticated Key Exchange) authentication algorithms as a basis. The use of PAKE mechanism allows users to use familiar ID/password based accesses, without fear of leaking any password information to the communication peer. The protocol, as a whole, is designed as a natural extension to the HTTP protocol [RFC2616].

The design also considers to replace current form-based Web authentication, which is very vulnerable against phishing attacks. To this purpose, several extensions to current HTTP authentication mechanism [RFC2617] are introduced.
1.1. Requirements Language

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

2. Protocol Overview

The following sequence is a typical sequence for the first access to the resource.

- If the server (S) has received a request for mutual-authentication protected resources from the Client (C) (which is not a req-A1 nor a req-A3 message), it sends a 401-B0 message to C.

  When C has received a 401-B0 message, C SHOULD check validity of the message. If succeed, C processes the body of the message, and enables the password entry field.

- If the user has input the username and password as a response to the 401-B0 message, C creates a value s_A, calculates the value w_A, and construct and send a req-A1 message.

  If S has received an req-A1 message, S should record the received w_A value, validate w_A using T(w_A), and then look up the username from the user table. If the user is found, S prepares a new session id (sid), record it into a session table, and then construct s_B, calculate w_B, and then send an 401-B1 message.

  If there is no matching user found, the server SHOULD construct a fake w_B value, and let the protocol going on by sending an 401-B1 message.

- When C has received an 401-B1 message as a response for a req-A1 message, C should compute z and K_i, compute o_A, and send an req-A3 message.

  If C receives any messages other than 401-B1, C MUST NOT process the message body and treat it as a fatal communication error condition. This case includes the reception of HTTP OK (200-status) message.

- If S has received an req-A3 message, S should look up the received sid from the session table. If no matching sid message is received, or if S has not received the corresponding req-A1 message beforehand, S SHOULD send an 401-B0-stale message.
Otherwise, S should compute o_A' and check its value. If the validation has failed, the server SHOULD send an 401-B0 message.

If the validation has succeeded, the server SHOULD calculate o_B, and send a 200-B4 message.

When C has received a 401-B0-stale message as a response to req-A3 message, and it is the first time that it has received a 401-B0-stale message, C SHOULD redo constructing req-A1 message.

Instead, when C has received an 401-B0 message, it means the authentication has been failed, possibly due to the wrong password has been given. C MAY ignore the body of the 401-B0 message in this case.

When C has received an 200-B4 message, C MUST first compute the value of o_B' and validate the value o_B sent from the server. If it has not verified successfully, C MUST ignore the body of the message, and treat it as a fatal communication error condition. If it has succeed, C will process the body of the message.

If C receives any messages other than 401-B0-stale or 401-B1, C MUST NOT process the message body and other headers and treat it as a fatal communication error condition. This case includes the reception of usual HTTP OK (200-status) messages.

For the second or later request to the server, if the client knows that the resource is likely to require the authentication, the client MAY omit first unauthenticated request and send req-A1 message immediately. Furthermore, if client owns a valid session ID (sid), the client MAY send a req-A3 message using existing sid. In either case, the first (and only the first) response from the server MAY be a normal, unauthenticated message, and client MAY accept such messages. For more detail, see Section 5.

3. Message Syntax

The Mutual authentication protocol uses four headers:
WWW-Authenticate (in responses with status code 401), Optional-WWW-Authenticate (in responses with positive status codes), Authorization (in requests), and Authentication-info (in positive responses). These three headers share the common syntax described in Figure 1. The syntax is denoted in the augmented BNF syntax defined in [RFC2616], plus that "a"..."b" means any ASCII characters between "a" and "b" inclusive. The syntax is a subset of the one described in [RFC2617].
header = header-name ":": spaces "Mutual" spaces fields
header-name = "WWW-Authenticate" | "Optional-WWW-Authenticate"
               | "Authorization" | "Authentication-info"
spaces = 1*(SP | HT | CR LF 1*(SP | HT))
fields = field 0*("," spaces fields)
field = key "=" value
       key = extensive-token
extension-token = token | extension-token
extension-token = token "@" token
       token = 1*("0"..."9" | "A"..."Z" | "a"..."z"
               | "-" | ";" | ".")
value = extensive-token | integer | hex-integer
       | hex-fixed-number
integer = "0" | ("1"..."9") 0*("0"..."9")
hex-integer = "0" | ("1"..."9" | "A"..."F" | "a"..."f")
               | 0*("0"..."9" | "A"..."F" | "a"..."f")
hex-fixed-number = 1*("0"..."9" | "A"..."F" | "a"..."f")
base64-fixed-number = string
       string = "<" *(<TEXT except "<" and ">" and ")" and ";" and "," and "))" *("<")
Figure 1: the BNF syntax for the headers used in the protocol

3.1. Tokens and Extensive-tokens

The tokens MUST be interpreted case-insensitive, and SHOULD be sent
in the same case as shown in the specification. When these are used as (partial) inputs to any hash or other mathematical functions, it
MUST be used in lower-case. All hex-fixed-number or hex-integer
numbers are also case-insensitive, and SHOULD be sent in lower-case.

Extensive-tokens are used where the set of acceptable tokens are
testensible. Any non-standard extensions of this protocol MUST use
the extension-tokens of format "<token>@<domain-name>", where domain-
name is the valid registered (sub-)domain name on the Internet owned
by the party who defines extensions.

3.2. Numbers

The syntax definitions of integer and hex-integer only allow
representations which do not contain extra leading 0s.

The numbers represented as a hex-fixed-number SHALL have even
characters (i.e. multiple of eight bits). When these are generated
from the cryptographic values, those SHOULD have the natural length:
if these are generated from a hash function, these lengths SHOULD
correspond to the hash size; if these are representing elements of a mathematical group, its lengths SHOULD be the shortest which can represent all elements in the group. Other values such as session-id are represented in any (even) length determined by the side who generates it first, and the same length SHALL be used throughout the whole communications by both peers.

The numbers represented as a base64-fixed-number SHALL be generated as follows: first, the number is converted to a big-endian octet-string representation. The length of the representation is determined in the same way as above. Then, the string is encoded by the Base 64 encoding [RFC3548], and then enclosed by two double-quotations.

3.3. Strings

All strings outside ASCII or equivalent character sets SHOULD be encoded using UTF-8 encoding [RFC3629] of the ISO 10646-1 character set [ISO.10646-1.1993]. Both peers SHOULD reject any invalid UTF-8 sequences which causes decoding ambiguities (e.g. containing "\" in the second or later byte of the UTF-8 encoded characters). To encode character strings, these will first be encoded according to UTF-8 without leading BOM, then all occurrences of characters "<" and "\" will be escaped by prepending "\"", and two "\"s will be put around the string. If the contents of the strings are comma-separated values, the commas in the values are also quoted by "\".

If strings are representing a domain name or URI which contains non-ASCII characters, the host parts SHOULD be encoded using puny-code defined in [RFC3492] instead of UTF-8, and SHOULD use lower-case ASCII characters.

For Base64-fixed-numbers, which use the string syntax, see the previous section.

4. Messages

In this section, formats and requirements of the headers for each message are presented. The allowed type for values for each header field is shown in parenthesis after the key names.

Note: The term "optional" here means that omitting the field is allowed and has specific meanings in communications (i.e. it is not generally "OPTIONAL" defined in [RFC2119]).
4.1. 401-B0

Every 401-B0 message SHALL be a valid HTTP 401 (Authentication Required) message containing one (and only one: hereafter not explicitly noticed) "WWW-Authenticate" header of the following format.

WWW-Authenticate: Mutual algorithm=xxxx, validation=xxxx, realm=xxxx, stale=0

The header SHALL contain the fields with the following keys:

- **algorithm**: (extensive-token) specifies the authentication algorithm to be used. The value MUST be one of the tokens described in Section 6, or the tokens specified in other supplemental specification documentations.

- **validation**: (extensive-token) specifies the method of host validation. The value MUST be one of the tokens described in Section 7, or the tokens specified in other supplemental specification documentations.

- **realm**: (string) is a UTF-8 encoded name of the authentication domain inside the server.

- **pwd-hash**: (optional, extensive-token) specifies the hash algorithm (referred to by ph) used for additionally hashing the password. The valid tokens are
  * none: ph(p) = p
  * md5: ph(p) = MD5(p)
  * digest-md5: ph(p) defined as A1 in [RFC2617].
  * sha1: ph(p) = SHA1(p)

If omitted, the value "none" is assumed.

- **auth-domain**: (optional, string) MUST currently be equal to the host-part of the requested URI, and assumed to have that value if omitted. The triple of auth-domain, algorithm, and realm determines the "authentication realm" which defines the area where the same user-name and passwords are applicable.
stale: (token) MUST be "0".

Any additional fields SHOULD NOT be contained in the header, except those explicitly specified in supplement specifications of the "authentication algorithm".

The algorithm will determine the types and the values for w_A, w_B, o_A and o_B.

4.2. 401-B0-stale

A 401-B0-stale message is a variant of 401-B0 message, which means that the client has sent a request message which is not for any active session.

WWW-Authenticate: Mutual algorithm=xxxx, validation=xxxx, realm=xxxx, stale=1

The header MUST contain the same fields as in 401-B0, except that stale field holds the integer 1.

4.3. req-A1

Every req-A1 message SHALL be a valid HTTP request message containing a "Authorization" header of the following format.

Authorization: Mutual algorithm=xxxx, validation=xxxx, user=xxxx, realm=xxxx, wa=xxxx

The header SHALL contain the fields with the following keys:

algorithm, validation, auth-domain, realm: MUST be the same value as it is received from S.

user: (string) is the UTF-8 encoded name of the user.

wa: (algorithm-determined) is the value of w_A specified by the used algorithm.

4.4. 401-B1

Every 401-B1 message SHALL be a valid HTTP 401 (Authentication Required) message containing a "WWW-Authenticate" header of the following format.

WWW-Authenticate: Mutual sid=xxxx, wb=xxxx, nc-max=x, nc-window=x, time=x, path=xxxx
The header SHALL contain the fields with the following keys:

- **sid:** (hex-fixed-number) MUST be a session id, which is a random integer. The sid SHOULD have uniqueness of at least 80 bits or the square of the maximal estimated transactions concurrently available in the session table, whichever is larger. Sids are local to each authentication realm concerned: the same sids for different authentication realms SHOULD be treated as independent ones.

- **wb:** (algorithm-determined) is the value of w_B specified by the algorithm.

- **nc-max:** (hex-integer) is the maximal value of nonce counts which S accepts.

- **nc-window:** (hex-integer) the number of available nonce slots which S will accept. The value of nc-window is RECOMMENDED to be thirty-two ("20" in hex-integer) or more.

- **time:** (integer) represents the suggested time (in seconds) which C can reuse the session key represented by sid. It is RECOMMENDED to be at least 60. The value of this field is not directly linked to the duration that S keeps track of the session represented by sid.

- **path:** (optional, string) specifies for which path in the URI space the same authentication is expected to apply. The value is in the same format as it is specified in [RFC2617] for the Digest authentications, and clients are RECOMMENDED to recognize it. The all path elements contained in the field MUST be inside the specified auth-domain: if not, client SHOULD ignore such elements.

### 4.5. req-A3

Every req-A3 message SHALL be a valid HTTP request message containing a "Authorization" header of the following format.

```
Authorization: Mutual sid=xxxx, nc=x, oa=xxxx
```

The fields contained in the header is as follows:
sid: (hex-fixed-number) MUST be one of the sid values which has been received from S.

nc: (hex-integer) is a nonce value which is unique among the requests sharing the same sid. The value of nc SHOULD satisfy the following properties:

* It is not larger than the nc-max value which has been sent from S in the session represented by the sid.
* C have not sent the same value in the same session.
* It is not smaller than (largest-nc - nc-window), where largest-nc is the maximal value of nc which has previously been sent in the session, and nc-window is the value of the nc-window field which has been sent from S in the session.

oa: (algorithm-determined) is the value of o_A specified by the algorithm.

4.6. 200-B4

Every 200-B1 message SHALL be a valid HTTP message which is not 401 (Authentication Required) type, containing an "Authentication-Info" header of the following format.

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

The fields contained in the header is as follows:

sid: (hex-fixed-number) MUST be the value received from C.

ob: (algorithm-determined) is the value of o_B specified by the algorithm.

logout-timeout: (optional, integer) is a number of seconds after which the client should re-validate the user’s password for the current authentication realm. As a special case, the value 0 means that the client SHOULD automatically forget the user-inputed password to the current authentication realm and revert to the unauthenticated state (i.e. server-initiated logout). This does not, however, mean that the long-term memories for the passwords (such as password reminders and auto fill-ins) should be removed. If a new value of timeout is received for the same authentication
realm, it overrides the previous timeout.

5. Decision procedure for the client

To securely implement the protocol, the user client must be careful to accepting authenticated responses from the server.

Clients SHOULD implement the decision procedure equivalent to the one shown below. (Unless implementers understand what is required for the security, they should not alter this.)

Step 1  If the client software needs to get a new Web resource, check whether the resource is expected to be inside some authentication realm for which the user has already authenticated. If yes, go to Step 2. Otherwise, go to Step 5.

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

Step 3  Send a req-A3 request.

* If you receive a 401-B0 message with a different authentication realm than expected, go to Step 6.

* If you receive a 401-B0-stale message, go to Step 9.

* If you receive a 401-B0 message, go to Step 13.

* If you receive a valid 200-B4 message, go to Step 14.

* If you receive a normal response (without Mutual-specific headers), go to Step 11.

Step 4  Send a req-A1 request.

* If you receive a 401-B0 message with a different authentication realm than expected, go to Step 6.

* If you receive a 401-B0-stale message, go to Step 9.

* If you receive a 401-B1 message, go to Step 10.

* If you receive a normal response (without Mutual-specific headers), go to Step 10.
Step 5  Send a request without any authentication headers.

  * If you receive a 401-B0 message, go to Step 6.
  * If you receive a normal response (without Mutual-specific headers), go to Step 11.

Step 6  Check whether you know the user’s password for the requested authentication realm. If yes, go to Step 7. Otherwise, go to Step 12.

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

Step 8  Send a req-A3 request.

  * If you receive a 401-B0 message with a different authentication realm than expected, go to Step 6.
  * If you receive a 401-B0-stale message, go to Step 9.
  * If you receive a 401-B0 message, go to Step 13.
  * If you receive a valid 200-B4 message, go to Step 14.

Step 9  Send a req-A1 request.

  * If you receive a 401-B1 message, go to Step 10.

Step 10  Send a req-A3 request.

  * If you receive a 401-B0 message, go to Step 13.
  * If you receive a valid 200-B4 message, go to Step 14.

Step 11  This case means that the resource requested is out of the authenticated area. The client will be in "UNAUTHENTICATED" status.

Step 12  This case means that the resource requested requires Mutual authentication, and the user is not authenticated yet. The client will be in "AUTH_REQUESTED" status, is RECOMMENDED to process the content sent from the server and ask user a username and password. If the user has input those, go to Step 9.
Step 13  This case means that in some reason the authentication failed: possibly the password or the username is invalid for the authenticated resource. Forget the password for the authentication realm and go to Step 12.

Step 14  This case means that the mutual authentication has been succeeded. The client will be in "AUTH_SUCCEEDED" status.

All other kind of responses than shown in above procedure SHOULD be interpreted as fatal communication error, and in such cases user clients MUST NOT process any data (contents and other content-related headers) sent from the server.

The client software SHOULD show the three client status to the end-user.

6.  Authentication Algorithms

This document specifies only one family of the authentication algorithm. The family consists of four authentication algorithms, which only differ in underlying mathematical groups and security parameters. The algorithms do not add any additional fields. The tokens for algorithms are

- "iso11770-4-ec-p256" for the 256-bit prime-field elliptic-curve setting.
- "iso11770-4-ec-p521" for the 521-bit prime-field elliptic-curve setting.
- "iso11770-4-dl-2048" for the 2048-bit discrete-logarithm setting.
- "iso11770-4-dl-4096" for the 4096-bit discrete-logarithm setting.

The clients SHOULD support at least "iso11770-4-dl-2048" algorithm, and are advised to support all of the above four algorithms whenever possible. The server software implementations SHOULD support at least "iso11770-4-dl-2048" algorithm, unless it is known that users will not use it.

This algorithm uses Key Agreement Mechanism 3 (KAM3) defined in Section 6.3 of ISO/IEC-11770-4 [ISO.11770-4.2006] as a basis.

For the elliptic-curve settings, the underlying fields and the curves used for elliptic-curve cryptography are the prime field and the Curve P-256 and P-511, respectively, specified in the appendix of FIPS PUB 186-2 [FIPS.186-2.2000] specification. The hash functions H
are SHA-256 for P-256 curve and SHA-512 for P-521 curve, respectively, defined in FIPS PUB 180-2 [FIPS.180-2.2002]. The representation of fields wa, wb, oa, and ob is hex-fixed-number.

For discrete-logarithm settings, the underlying groups are 2048-bit and 4096-bit MODP groups defined in [RFC3526] respectively. The hash functions H are SHA-256 for the 2048-bit field and SHA-512 for the 4096-bit field, respectively. The representation of fields wa, wb, oa, and ob is base64-fixed-number.

The password-based string pi used by this authentication is derived in the following manner:

\[ pi = H(VS(algorithm) \mid VS(auth-domain) \mid VS(realm) \mid VS(username) \mid VS(ph(password))). \]

The values of algorithm, realm and auth-domain are taken from the values contained in the 401-B0 message. When pi is used in the context of an octet string, it SHALL have the natural length derived from the size of the output of function H (e.g. 32 octets for SHA-256). The function ph is defined by the value of the pwd-hash field given in a 401-B0 message.

The function VI encodes natural numbers into octet strings in the following manner: integers are represented in big-endian radix-128 string, where each digit is represented by a octet 0x80-0xff except the last digit represented by 0x00-0x7f. The first octet MUST NOT be 0x80. For example, \( VI(i) = \text{octet}(i) \) for \( i < 128 \), and \( VI(i) = \text{octet}(0x80 \mid (i >> 7)) \mid \text{octet}(i \& 127) \) for \( 128 \leq i < 16384 \). This encoding is the same as the one used in the length field in the ASN.1 encoding [ITU.X690.1994].

The function VS encodes variable-length octet string into decodable octet string, as in the following manner:

\[ VS(s) = VI(length(s)) \mid s \]

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

The equations for \( J, w_A, T, z, K_i, \) and \( w_B \) are those specified for KAM3 in [ISO.11770-4.2006], given that pi is derived in the above equation.

The values \( o_A (o'_A) \) and \( o_B (o'_B) \) are derived by the following equation, instead of ones specified in [ISO.11770-4.2006].

\[ o_A = H(hex(04) \mid GEtoOS_x(\bar{w}_A) \mid GEtoOS_x(\bar{W}_B) \mid GEtoOS_x(z) \mid GEtoOS_x(g_1) \mid VI(nc) \mid VS(v)) \]
7. Validation Methods

The "validation method" specifies a method to "relate" the mutual authentication processed by this protocol with other authentications already performed in the underlying layers and to prevent man-in-the-middle attacks. It decides the value of v which is an input to authentication protocols.

The valid tokens for the validation field and corresponding values of v are as follows:

host: hostname validation: v will be the ASCII string in the following format: "scheme://host:port". The scheme and host are lower-case, and the port is in a shortest decimal representation. Even if the request-URI does not have a port part, v will include the one.

tls-cert: TLS certificate validation: v will be the octet string of the fingerprint of the public key certificate used in underlying TLS/SSL [RFC4346] connection.

tls-key: TLS shared-key validation: v will be the octet string of the shared master secret negotiated in underlying TLS/SSL connection.

If the HTTP protocol is used on unencrypted channel, the validation type MUST be "host". If HTTP/TLS [RFC2818] (https) protocol is used with server certificates, the validation type MUST be either "tls-cert" or "tls-key". If HTTP/TLS protocol is used with anonymous Diffie-Hellman key exchange, the validation type MUST be "tls-key" (but see the note below).

The client MUST validate this field upon reception of 401-B0 messages.

However, when the protocol is used on web browsers with any scripting capabilities, the anonymous Diffie-Hellman family of TLS/SSL cipher-suite MUST NOT be used even if "tls-key" validated Mutual authentication has been employed, and the certificate shown in TLS/SSL negotiation MUST be verified using PKI. For other systems, if the "tls-key" validation is used on TLS/SSL protocol without certificate verification using PKI, those systems MUST ensure that all transactions with authenticated peer servers MUST use and be validated by the Mutual authentication protocol, regardless of the
existence of the 401-B0 responses.

8. Session Management

By the first 4 messages (first request, 401-B0, req-A1 and 401-B1), a session represented by a sid is generated. This session can be used for 1 or more requests for resources protected by the same realm in the same server.

The server SHOULD accept at least one req-A3 request for each session, given that the request reaches the server in a time window specified by the timeout field in the 401-B1 message, and that there are no emergent reasons (such as flooding attacks) to forget the sessions. After that, the server MAY discard any session at any time and MAY send 401-B0-stale messages for any req-A3 requests.

The client MAY send more than one requests using a single session specified by the sid. However, for all such requests, the values of the nonce-counter (nc field) MUST be different from each other. The server MUST check for duplication of the received nonces, and if any duplication is detected, the server MUST discard the session and respond by a 401-B1-stale message.

In addition, for each sessions, if the client has already sent a request with nonce value x, it SHOULD NOT send requests with a nonce value not larger than (x - nc-window). The server MAY reject any requests with nonces violating this rule with 401-B1-stale responses.

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

9. Extension 1: Optional Mutual Authentication

In several Web applications, users can access the same contents both as a guest user and as a authenticated users. In usual Web applications, it is implemented using Cookies and custom form-based authentications. The extension described in this section provides a replacement for those authentication systems. The support for this
extension is RECOMMENDED.

Servers MAY send HTTP successful responses (response code 200, 206 and others) containing the Optional-WWW-Authenticate header, when it is allowed to send 401-B0 responses and the requests do not contain Authentication-Info: headers. Such responses are hereafter called 200-Optional-B0 responses.

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

The fields contained in the Optional-WWW-Authenticate header is the same as the 401-B0 message described in Section 4.1. The client software supporting the mutual authentication protocol receiving a 200-Optional-B0 message will process the contents of the message and enables an authentication input field.

When the user input the username and password, the client resends the request with req-A1 header. The server MUST respond with a 401-B1 message. In terms of the state management in Section 5, 200-Optional-B0 responses are treated as if it is 401-B0 response: these messages SHOULD NOT be sent as a response to req-A1 and req-A3 messages, unless the authentication realm sent from the client or indicated by sid is different from the one which the server expects.

Servers requesting optional mutual authentication SHOULD send the path field in 401-B1 messages with an appropriate value. Client software supporting optional mutual authentication MUST recognize the field, and MUST send either req-A1 or req-A3 request for the URI space inside the specified paths, instead of unauthenticated requests.

10. Methods to extend this protocol

If a non-standard extension to the this protocol is implemented, it MUST use the extension-tokens defined in Section 3 to avoid conflicts with this protocol and other extensions.

Authentication algorithms other than those defined in this document MAY use other representations for keys "wa", "wb", "oa" and "ob", replace those keys, and/or add fields to the messages containing those fields by supplemental specifications. If those specifications use keys other than shown above, it is RECOMMENDED to use extension-tokens to avoid any key-name conflict with the future extension of this protocol.
11. IANA Considerations

The tokens used for authentication-algorithm, pwd-hash, and validation fields MUST be allocated by IANA. To acquire registered token, IESG Approval outlined in [RFC2434] is required. Extension-tokens MAY be freely used for any non-standard, private and/or experimental uses for those fields provided that the domain part in the token is appropriately used.

12. Security Considerations

12.1. General Assumptions

- The protocol, by itself, does not assure any secrecy and relies on transport security including DNS security. HTTP/TLS SHOULD be used where transport security is not assured and data secrecy is important. The protocol is secure against passive eavesdropping and replay attacks, though.

- When used with HTTP/TLS, the protocol gives true protection against active man-in-the-middle attacks for each HTTP request/response pair, even when the server certificate is not used or is unreliable. However, in such cases, JavaScript or similar scripting facilities can be used to affect Mutually-authenticated contents from those not protected by this authentication mechanism. This is why this memo requires that sane TLS server certificates MUST be presented.

12.2. Implementation Considerations

- To securely implement the protocol, the Authentication-Info headers in the 200-B4 messages MUST always be validated by the client. If the validation is failed, the client MUST NOT process any content sent with the message, including the body part. Non-compliance to this will enable phishing attacks.

- The authentication status on the client-side SHOULD be visible to the users of the client. In addition, the method for asking user’s name and passwords SHOULD be carefully designed so that (1) the user can easily distinguish request of this authentication methods from other existing authentication methods such as Basic and Digest methods, and (2) the Web contents cannot imitate the user-interfaces of this protocol.

An informational memo regarding user-interface considerations and recommendations for implementing this protocol will be separately published.
For HTTP/TLS communications, when a web form is submitted from Mutually-authenticated pages with the validation methods of "tls-cert" to a URI which is protected by the same realm (so indicated by the path field), if server certificate has been changed since the pages has been received, the peer is RECOMMENDED to be revalidated using a req-A1 message with an "Expect: continue" header. The same applies when the page is received with the validation methods of "tls-key", and when the TLS session has been expired.

Server-side storages of user passwords are advised to have the values encrypted by one-way function J(pi), instead of the real passwords, those hashed by ph, or pi.

13. Notice on intellectual properties

The National Institute of Advanced Industrial Science and Technology (AIST) and Yahoo! Japan, Inc. has jointly submitted a patent application about the protocol proposed in this documentation to the Patent Office of Japan. The patent is intended to be open to any implementors of this protocol and its variants under non-exclusive royalty-free manner once the protocol is accepted as an Internet standard. For the detail of the patent application, contact the author of this document.

The elliptic-curve based authentication algorithms might involve several existing patents of third-parties. The authors of the document take no position regarding the validity or scope of such patents, and other patents as well.

14. References

14.1. Normative References

[FIPS.180-2.2002]

[FIPS.186-2.2000]

[ISO.10646-1.1993]


14.2. Informative References


Authors’ Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Research Center for Information Security
Akihabara Daibiru #1102
1-18-13 Sotokanda
Chiyoda-ku, Tokyo
JP
Phone: +81 3-5298-4722
Email: mutual-auth-contact@m.aist.go.jp

Hajime Watanabe
National Institute of Advanced Industrial Science and Technology

Hiromitsu Takagi
National Institute of Advanced Industrial Science and Technology

Hirofumi Suzuki
Yahoo! Japan, Inc.
Roppongi Hills Mori Tower
6-10-1 Roppongi
Minato-ku, Tokyo
JP
Phone: +81 3-6440-6290