Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: February 12, 2009

Y. Oiwa
H. Watanabe
H. Takagi
RCIS, AIST
H. Suzuki
Yahoo! Japan
August 11, 2008

Mutual Authentication Protocol for HTTP draft-oiwa-http-mutualauth-03

Status of this Memo

Copyright Notice

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

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 check validity of w_A, record the received w_A value, 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 check validity of w_B, and compute z and 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 computes 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 an 401-B0 message, it means the authentication has been failed, possibly due to that 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 or valid 200-B4, 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. In this case, the first (and only the first) response from the server MAY be a normal, unauthenticated message, and client MAY accept such messages.

Furthermore, if client owns a valid session ID (sid), the client MAY send a req-A3 message using existing sid. In such cases, the server MAY have thrown out the corresponding sessions, then the server SHOULD send a 401-B0-stale message as a response to req-A3 message, and C SHOULD retry from constructing req-A1 message.

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 [RFC5234]. 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*(" " / %x09 / %x0D.0A (" " / %x09))       ; LWSP
fields            = field *([spaces] "," spaces field)
field             = key "=" value
key               = extensive-token
extensive-token   = token / extension-token
extension-token   = token "@" token
token             = 1*(%x30-39 / %x41-5A / %x61-7A / "." / "-" / "_")
value             = extensive-token / integer / hex-integer
                  / hex-fixed-number
                  / base64-fixed-number / string
integer           = "0" / (%x31-39 *%x30-39)         ; no leading zeros
hex-integer       = "0"
                  / ((%x31-39 / %x41-46 / %x61-66)   ; no leading zeros
                     *(%x30-39 / %x41-46 / %x61-66))
hex-fixed-number = 1*(%x30-39 / %x41-46 / %x61-66)
base64-fixed-number = string
string            = %x22 *(%x20-21 / %x23-5B / %x5D-FF
                           / %x5C.22 / "\\" / "\,") %x22
```

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 extensible. 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 MUST have even characters (i.e. multiple of eight bits). When these are generated from 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. See Appendix B for information about the length of the fields used in this specification. 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 [RFC4648], 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. The type "algorithm-determined" means that the acceptable value type for the field is one of the types defined in Section 3, and is determined by the value of the "algorithm" field.

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 7, 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 9, or the tokens specified in other supplemental specification documentations.

auth-domain:
: (optional, string) specifies authentication domain, the set of hosts on which authentication credentials are valid. It MUST be one of the strings described in Section 8. If the value is omitted, it is assumed to be the host part of the requested URI.

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

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) = MD5(username | ":" | realm | ":" | p), the same value as MD5(A1) for "MD5" algorithm in [RFC2617].
- sha1: ph(p) = SHA1(p)

If omitted, the value "none" is assumed. The use of "none" is recommended.

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, realm="xxxx", user="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 algorithm=xxxx, validation=xxxx, realm="xxxx", sid=xxxx, wb=xxxx, nc-max=x, nc-window=x, time=x, path="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 C.

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 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 algorithm=xxxx, validation=xxxx, realm="xxxx", sid=xxxx, nc=x, oa=xxxx

The fields contained in the header is as follows:

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

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.) The labels on the steps are for informational purpose only.

Step 1 (step_new_request):

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 (step_send_a3_1):

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 (step_send_a1_1):

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-B1 message, go to Step 10.
- If you receive a normal response (without Mutual-specific headers), go to Step 11.

Step 5 (step_send_normal_1):

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 (step_rcvd_b0):

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 (step_send_a3):

Send a req-A3 request.

- 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 (step_send_a1):

Send a req-A1 request.

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

Step 10 (step_rcvd_b1):

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 (step_rcvd_normal):

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

Step 12 (step_rcvd_b0_unknown):

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 (step_rcvd_b0_failed):

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 (step_rcvd_b4):

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.

Figure 2 shows the full client-side state diagram.

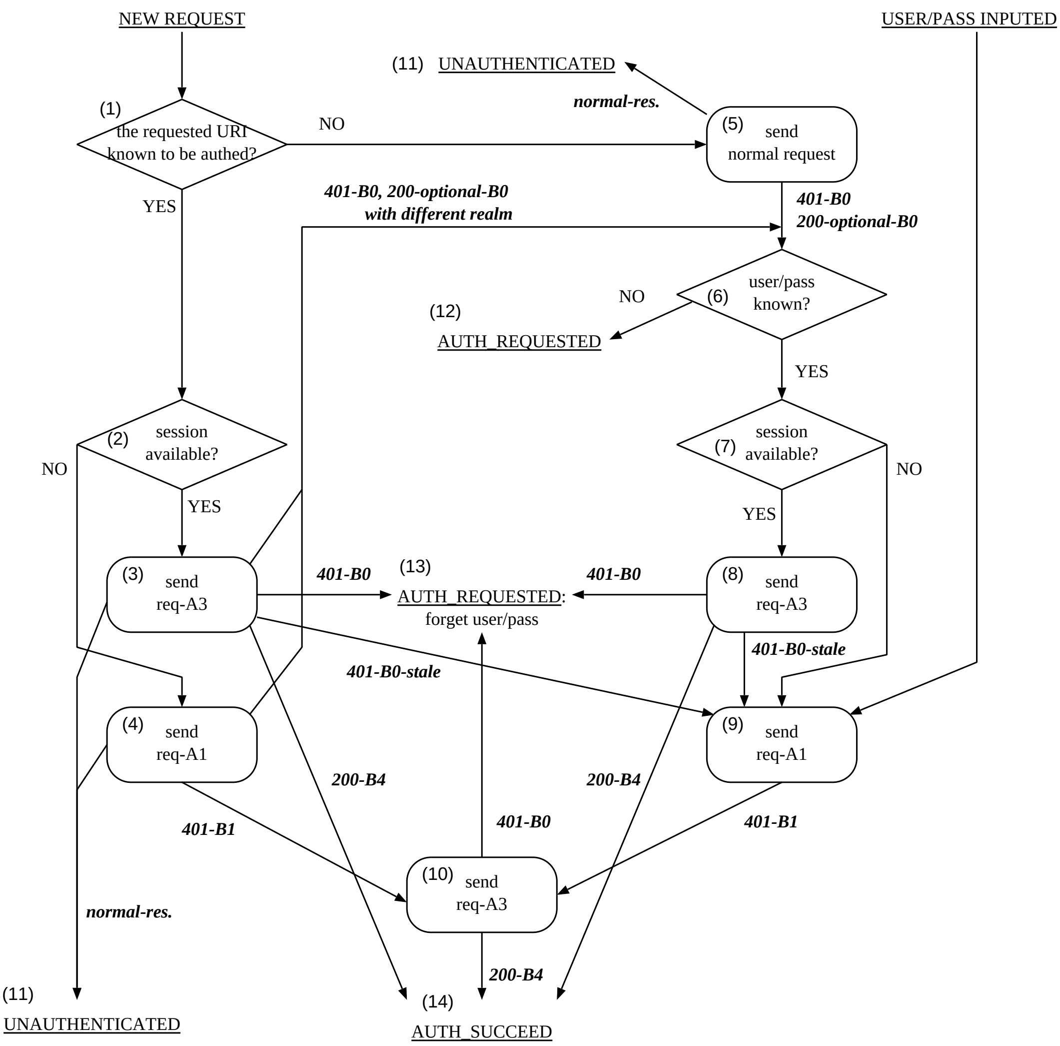

Figure 2: State diagram for clients

6. Decision procedure for the server

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

- When the server receives a normal request:
  - If the requested resource is not protected by Mutual Authentication, send a normal response.
  - If the resource is protected by Mutual Authentication, send a 401-B0 response.
  - If the resource is protected by Mutual Authentication with Optional Mutual Authentication extension (Section 11), send a 200-Optional-B0 response.
- When the server receives a req-A1 request:
  - If the requested resource is not protected by Mutual Authentication, send a normal response.
  - If the authentication realm specified in the req-A1 request is non-expected one, send a

    401-B0 (or 200-Optional-B0) response.
  - If the server cannot validate field wa, send a 401-B0 response.
  - If the received user name is invalid, send a fake 401-B1 response.
  - Otherwise, send a 401-B1 response.
- When the server receives a req-A3 request:
  - If the requested resource is not protected by Mutual Authentication, send a normal response.
  - If the authentication realm specified in the req-A3 request is non-expected one, send a 401-B0 (or 200-Optional-B0) response.
  - If the received sid is invalid, inactive or unknown, send a 401-B0-stale response.
  - If the receive oa is invalid, send a 401-B0 response.
  - If the receive oa is correct, send a 200-B4 response.

7. 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.

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-521, 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. See Appendix A for the exact specification of the group and associated parameters. 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 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.

7.1. Common functions

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

pi = H(VS(algorithm) | VS(auth-domain) | VS(realm) | VS(username) | VS(ph(password)).

The values of algorithm, realm and auth-domain are taken from the values contained in the 401-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) = octet(i) for i < 128, and VI(i) = octet(0x80 | (i >> 7)) | octet(i & 127) for 128 <= i < 16384. This encoding is the same as the one used 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)) | s

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

The function OCTETS converts an integer to corresponding radix-256 big-endian octet string having its natural length: See Section 3.2 for the definition of the "natural length". Note that this is different from the function GE2OS_x in [ISO.11770-4.2006], which takes the shortest representation.

The equations for J, w_A, T, z, and w_B are specified differently for the discrete-logarithm setting and the elliptic-curve setting based on [ISO.11770-4.2006]. These equations are defined later in this section.

The values o_A and o_B are derived by the following equation. Note that these equations are different from ones specified in [ISO.11770-4.2006].

o_A = H(octet(04) | OCTETS(w_A) | OCTETS(w_B) | OCTETS(z) | VI(nc) | VS(v))
o_B = H(octet(03) | OCTETS(w_A) | OCTETS(w_B) | OCTETS(z) | VI(nc) | VS(v))

7.2. Functions for discrete-logarithm settings

In this section, the equation (x / y mod z) denotes an natural number w less than z which satisfies (w * y) mod z = x mod z.

For the discrete-logarithm, we refer some of the domain parameters by the following symbols:

- q: for "the prime" of the group.
- g: for "the generator" associated with the group.
- r: for the order of the subgroup generated by g.

The function J is defined as

J(pi) = g^(pi) mod q,

where g and q are domain parameters of the underlying field.

The value of w_A is derived as

w_A = g^(s_A) mod q,

where s_A is a random integer within range [1, r-1] and r is the size of the subgroup generated by g. In addition, s_A MUST be larger than log(q)/log(g) (so that g^(s_A) > q).

The value of w_A SHALL satisfy 1 < w_A < q-1. The server MUST check this condition upon reception.

The value of w_B is derived from J(pi) and w_A as:

w_B = (J(pi) * w_A^(H(octet(1) | OCTETS(w_A))))^s_B mod q,

where s_B is a random number within range [1, r-1]. The value of w_B MUST satisfy 1 < w_B < q-1. If this condition is not hold, the server MUST retry with another value of s_B. The client MUST check this condition upon reception.

The value z in the client side is derived by the following equation:

z = w_B^((s_A + H(octet(2) | OCTETS(w_A) | OCTETS(w_B))) / (s_A * H(octet(1) | w_A) + pi) mod r) mod q.

The value z in the server side is derived by the following equation:

z = (w_A * g^(H(octet(2) | OCTETS(w_A) | OCTETS(w_B))))^s_B mod q.

7.3. Functions for elliptic-curve settings

For the elliptic-curve setting, we refer some of the domain parameters by the following symbols:

- q: for the prime used to define the field,
- G: for the defined point called the generator,
- r: for the order of the subfield generated by G.

The function P(p) converts a curve point p to an integer representing the point p, by computing x * 2 + (y mod 2), where (x, y) are the coordinates of the point p. P'(z) is the inverse of function P, that is, it converts an integer z to a point p which satisfies P(p) = z. If such p is exist, it is uniquely defined. Otherwise, z does not represent a valid curve point. The operation [x] * p denotes an integer-multiplication of point p: it calculates p + p + ... (x times) ... + p. See literatures on elliptic-curve cryptography for the exact algorithms for those. 0_E represents the infinity point. The equation (x / y mod z) denotes an natural number w less than z which satisfies (w * y) mod z = x mod z.

the function J is defined as

J(pi) = [pi] * G.

The value of w_A is derived as

w_A = P(W_A), where W_A = [s_A] x G.

where s_A is a random number within range [1, r-1]. The value of w_A MUST represent a valid curve point, and W_A SHALL NOT be 0_E. The server MUST check this condition upon reception.

The value of w_B is derived from J(pi) and W_A = P’(w_A) as:

w_B = P(W_B), where W_B = [s_B] * (J(pi) + [H(octet(1) | OCTETS(w_A))] * W_A).

where s_B is a random number within range [1, r-1]. The value of w_B MUST represent a valid curve point and satisfy [4] * P’(w_B) <> 0_E. If this condition is not hold, the server MUST retry with another value of s_B. The client MUST check this condition upon reception.

The value z in the client side is derived by the following equation:

z = P([(s_A + H(octet(2) | OCTETS(w_A) | OCTETS(w_B))) / (s_A * H(octet(1) | OCTETS(w_A)) + pi) mod r] * W_B), where W_B = P’(w_B).

The value z in the server side is derived by the following equation:

z = P([s_B] * (W_A + [H(octet(2) | OCTETS(w_A) | OCTETS(w_B))] * G)), where W_A = P’(w_A).

8. Authentication Realms

In this protocol, "authentication realm" is defined as the set of resources (URIs) for which the same set of user names and passwords is valid for. If the server requests authentication for the authentication realm which the client is already authenticated, the client will automatically perform authentication using the already-known secrets. On the contrary, for the different authentication realms, clients SHOULD NOT automatically reuse the usernames and passwords for another realm.

Just like Basic and Digest access authentication protocol, Mutual authentication protocol supports multiple, separate authentication realms to be set up inside each hosts. Furthermore, the protocol supports that a single authentication realm spans over several hosts in the same Internet domain.

Each authentication realm is defined and distinguished by the triple of an "authentication algorithm", an "authentication domain", a "realm" parameter. Server operators are NOT RECOMMENDED to use the same pair of an authentication domain and a realm for different authentication algorithms, however.

Authentication algorithms are defined in Section 4 and Section 7. Realm parameters are just a string, as defined in Section 4. Authentication domains are described in the rest of this section.

An authentication domain specifies the range of hosts which the authentication realm spans over. In the protocol, it MUST currently be one of the following strings.

- the string in format "<scheme>://<host>:<port>", where scheme, host and port are the URI parts of the requested URI. Even if the request-URI does not have a port part, the string will include the one (i.e. 80 for http and 443 for https). Use this when authentication is only valid for specific protocol (such as https).
- The "host" part of the requested URI. This is the default value. Authentication realms in this kind of authentication domain will span over several protocols (i.e. http and https) and ports, but not over different hosts.
- String in format "*.<domain-postfix>", where "domain-postfix" is either the host part of the requested URI, or any domain in which the requested host is included (this means that the specification "*.example.com" is valid for all of hosts "www.example.com", "web.example.com" and "example.com"). The domain-postfix must be equal to or included in a valid Internet domain assigned to specific organization: if the clients can know by some way (such as blacklists for

HTTP cookies) that the specified domain is not to be assigned to any specific organization (e.g. "*.com" or "*.jp"), the client is RECOMMENDED to reject the authentication request.

In the above specifications, every "scheme", "host" and "domain" MUST in lower-case. and IDNs MUST be represented in puny-code [RFC3492]. All "port"s MUST be in the shortest, unsigned, decimal number notation. Not obeying these requirements will cause failure of authentication attempts.

8.1. Resolving ambiguities

In the above definition of authentication domains, several domains will overwrap each other. Depending on the "path" parameters given in the "401-B1" message (see Section 4), There may be several candidate when the client is to send a request with authentication credentials included (at the Steps 3 and 4 of the decision procedure shown in Section 5).

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

- If the client has previously sent a request to the same URI, and it remembers the authentication realm requested by 401-B0 messages at that time, use that realm.
- In other cases, use one of authentication realms which specific most-specific authentication domains. In the list of possible domain specifications shown above, one described earlier has priority over ones described after that.
  If there are several choices with different domain-postfix specifications, the ones which has longer domain possible has priority over ones with shorter domain-postfix.
- If there are realms with the same specifications of authentication domain, there is not defined priority: client can choose any one of possible choices.

If possible, server operators are recommended to avoid such ambiguities by setting "path" parameters properly.

9. 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", where scheme, host and port are the URI parts correspond to the currently accessing resource. 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 hash value of the public key certificate used in underlying TLS [RFC4346] (or SSL) connection. The hash value is defined as the value of the whole signed certificate (specified as "Certificate" in [RFC5280]), hashed by the hash algorithm specified by the authentication algorithm used.

tls-key:
: TLS shared-key validation: v will be the octet string of the shared master secret negotiated in underlying TLS (or 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 (or SSL) cipher-suite MUST NOT be used even if "tls-key" validated Mutual authentication has been employed, and the certificate shown in TLS (or SSL) negotiation MUST be verified using PKI. For other systems, if the "tls-key" validation is used on TLS (or 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.

The protocol defines two variants for validation on TLS connections. The method "tls-key" method is the more secure, so it is recommended to use tls-key when applicable. However, there are some situations where tls-cert is more preferable.

- When TLS accelerating proxies are used. In this case, it is difficult for the authenticating server to acquire the TLS key information which are used between the client and the proxy. It is not the case for client-side "tunneling" proxies using CONNECT method extension of HTTP.
- When a black-box implementation of the TLS protocol is used on either peer.

10. 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. Note that the 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 will automatically reestablish another session without telling it to the users.

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-B0-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-B0-stale responses. This restriction enables servers to implement duplicated nonce detection in a constant memory.

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.

11. 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, unless an authentication is mandatory for some specific applications.

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.

12. 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.

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.

13. IANA Considerations

The tokens used for authentication-algorithm, pwd-hash, and validation fields MUST be allocated by IANA. To acquire registered tokens, a specification for the use of such tokens MUST be available as an RFC, as outlined in [RFC5226].

[More formal declarations will be added in future drafts to meet RFC 5226 requirements.]

14. Security Considerations

14.1. General Assumptions

- The protocol is secure against passive eavesdropping and replay attacks. However, the protocol relies on transport security including DNS security for active attacks. HTTP/TLS SHOULD be used where transport security is not assured and data secrecy is important.
- 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 valid TLS server certificates MUST be presented (Section 9).

14.2. 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: 100-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.

14.3. Usage Considerations

- The user-names inputted by user may be sent automatically to any servers sharing the same auth-domain. This means that when host-type auth-domain is used for authentication in HTTPS site, and when an HTTP server on the same host requests Mutual authentication with the same realm, the client will send the user-name in a clear text. If user-names have to kept secret against eavesdropping, the server must use full-scheme-type auth-domain parameter. On the contrary, passwords are not exposed to eavesdroppers even on HTTP requests.

- "Pwd_hash" field is only provided for backward compatibility for password databases, and using "none" function is the mostly secure choice and RECOMMENDED. If values other than "none" is used, you must ensure that the hash values of the passwords were not exposed to the public. Note that hashed password databases for plain-text authentications are usually not considered secret.
- If the server provides several ways of storing server-side password database, it is advised to store the values encrypted by one-way function J(pi), instead of the real passwords, those hashed by ph, or pi.

15. 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 and its status, please 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.

16. Acknowledgement

We gratefully acknowledge Lepidum, Co. Ltd. for support on design and trial implementation of this protocol.

17. References

17.1. Normative References

[FIPS.180-2.2002] National Institute of Standards and Technology, “Secure Hash Standard,” FIPS PUB 180-2, August 2002.

[FIPS.186-2.2000] National Institute of Standards and Technology, “Digital Signature Standard (DSS),” FIPS PUB 186-2, January 2000.

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

[RFC2818] Rescorla, E., “HTTP Over TLS,” RFC 2818, May 2000.

[RFC3526] Kivinen, T. and M. Kojo, “More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE),” RFC 3526, May 2003.

[RFC3629] Yergeau, F., “UTF-8, a transformation format of ISO 10646,” STD 63, RFC 3629, November 2003.

[RFC4346] Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.1,” RFC 4346, April 2006.

[RFC4648] Josefsson, S., “The Base16, Base32, and Base64 Data Encodings,” RFC 4648, October 2006.

[RFC5234] Crocker, D. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” STD 68, RFC 5234, January 2008.

17.2. Informative References

[I-D.altman-tls-channel-bindings] Altman, J. and N. Williams, “Unique Channel Bindings for TLS,” draft-altman-tls-channel-bindings-03 (work in progress), November 2007.

[ISO.10646-1.1993] International Organization for Standardization, “Information Technology - Universal Multiple-octet coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane,” ISO Standard 10646-1, May 1993.

[ISO.11770-4.2006] International Organization for Standardization, “Information technology – Security techniques – Key management – Part 4: Mechanisms based on weak secrets,” ISO Standard 11770-4, May 2006.

[ITU.X690.1994] International Telecommunications Union, “Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER),” ITU-T Recommendation X.690, 1994.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2616, June 1999 (TXT, PS, PDF, HTML, XML).

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” RFC 2617, June 1999 (TXT, HTML, XML).

[RFC3492] Costello, A., “Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA),” RFC 3492, March 2003.

[RFC5226] Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” BCP 26, RFC 5226, May 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” RFC 5280, May 2008.

Appendix A. Group parameters for discrete-logarithm based algorithms

The MODP group used for the iso11770-4-dl-2048 algorithm is defined by the following parameters.

The prime is:

```
q = 0xFFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
      29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
      EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
      E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
      EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
      C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
      83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
      670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
      E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
      DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
      15728E5A 8AACAA68 FFFFFFFF FFFFFFFF.
```

The generator is:

```
g = 2.
```

The size of the subgroup generated by g is:

```
r = (q - 1) / 2 =
    0x7FFFFFFF FFFFFFFF E487ED51 10B4611A 62633145 C06E0E68
      94812704 4533E63A 0105DF53 1D89CD91 28A5043C C71A026E
      F7CA8CD9 E69D218D 98158536 F92F8A1B A7F09AB6 B6A8E122
      F242DABB 312F3F63 7A262174 D31BF6B5 85FFAE5B 7A035BF6
      F71C35FD AD44CFD2 D74F9208 BE258FF3 24943328 F6722D9E
      E1003E5C 50B1DF82 CC6D241B 0E2AE9CD 348B1FD4 7E9267AF
      C1B2AE91 EE51D6CB 0E3179AB 1042A95D CF6A9483 B84B4B36
      B3861AA7 255E4C02 78BA3604 650C10BE 19482F23 171B671D
      F1CF3B96 0C074301 CD93C1D1 7603D147 DAE2AEF8 37A62964
      EF15E5FB 4AAC0B8C 1CCAA4BE 754AB572 8AE9130C 4C7D0288
      0AB9472D 45565534 7FFFFFFF FFFFFFFF.
```

The MODP group used for the iso11770-4-dl-4096 algorithm is defined by the following parameters.

The prime is:

```
q = 0xFFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1
      29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD
      EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245
      E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
      EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D
      C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F
      83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
      670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
      E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9
      DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510
      15728E5A 8AAAC42D AD33170D 04507A33 A85521AB DF1CBA64
      ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
      ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B
      F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
      BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31
      43DB5BFC E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7
      88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA
```

```
    2583E9CA 2AD44CE8 DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6
    287C5947 4E6BC05D 99B2964F A090C3A2 233BA186 515BE7ED
    1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
    93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34063199
    FFFFFFFF FFFFFFFF.
```

The generator is:

```
g = 2.
```

The size of the subgroup generated by g is:

```
r = (q - 1) / 2 =
  0x7FFFFFFF FFFFFFFF E487ED51 10B4611A 62633145 C06E0E68
    94812704 4533E63A 0105DF53 1D89CD91 28A5043C C71A026E
    F7CA8CD9 E69D218D 98158536 F92F8A1B A7F09AB6 B6A8E122
    F242DABB 312F3F63 7A262174 D31BF6B5 85FFAE5B 7A035BF6
    F71C35FD AD44CFD2 D74F9208 BE258FF3 24943328 F6722D9E
    E1003E5C 50B1DF82 CC6D241B 0E2AE9CD 348B1FD4 7E9267AF
    C1B2AE91 EE51D6CB 0E3179AB 1042A95D CF6A9483 B84B4B36
    B3861AA7 255E4C02 78BA3604 650C10BE 19482F23 171B671D
    F1CF3B96 0C074301 CD93C1D1 7603D147 DAE2AEF8 37A62964
    EF15E5FB 4AAC0B8C 1CCAA4BE 754AB572 8AE9130C 4C7D0288
    0AB9472D 45556216 D6998B86 82283D19 D42A90D5 EF8E5D32
    767DC282 2C6DF785 457538AB AE83063E D9CB87C2 D370F263
    D5FAD746 6D8499EB 8F464A70 2512B0CE E771E913 0D697735
    F897FD03 6CC50432 6C3B0139 9F643532 290F958C 0BBD9006
    5DF08BAB BD30AEB6 3B84C460 5D6CA371 047127D0 3A72D598
    A1EDADFE 707E8847 25C16890 54908400 8D391E09 53C3F36B
    C438CD08 5EDD2D93 4CE1938C 357A711E 0D4A341A 5B0A85ED
    12C1F4E5 156A2674 6DDDE16D 826F477C 97477E0A 0FDF6553
    143E2CA3 A735E02E CCD94B27 D04861D1 119DD0C3 28ADF3F6
    8FB094B8 67716BD7 DC0DEEBB 10B8240E 68034893 EAD82D54
    C9DA754C 46C7EEE0 C37FDBEE 48536047 A6FA1AE4 9A0318CC
    FFFFFFFF FFFFFFFF.
```

Appendix B. Derived numerical values

This section gives several numerical values for implementing this protocol, derived from the above specifications. The values shown in this section are for informative purpose only.

	dl-2048	**dl-4096**	**ec-p256**	**ec-p521**	
Size of w_A etc.	2048	4096	257	522	(bits)
Size of H(...)	256	512	256	512	(bits)
length of OCTETS(w_A) etc.	256	512	33	66	(octets)
length of wa, wb field values.	346 *	686 *	66	132	(octets)
length of oa, ob field values.	46 *	90 *	64	128	(octets)
minimum allowed s_A	2048	4096	1	1	

(The numbers marked with * include enclosing quotation marks.)

Appendix C. Draft Remarks from the Authors

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

- Whether to use "TLS channel binding" [I-D.altman-tls-channel-bindings] for "tls-key" verification (Section 9). Note that existing implementations of TLS should be considered to determine this.

Appendix D. Draft Change Log

D.1. Changes in revision 03

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

D.2. Changes in revision 02

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

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

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).