Mutual Authentication Protocol for HTTP: 
KAM3-based Cryptographic Algorithms 
draft-ietf-httpauth-mutual-algo-00

Abstract
This document specifies some cryptographic algorithms which will be used for the Mutual user authentication method for the Hyper-text Transport Protocol (HTTP).

Status of this Memo
This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 5, 2015.

Copyright Notice
Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

This document specifies some algorithms for Mutual authentication protocol for Hyper-Text Transport Protocol (HTTP) [I-D.ietf-httpauth-mutual]. The algorithms are based on so-called "Augumented Password-based Authenticated Key Exchange" (Augumented PAKE) techniques. In particular, it uses one of three key exchange algorithm defined in the ISO 11770-4: "Key management - Mechanisms based on weak secrets" [ISO.11770-4.2006] as a basis.

In very brief summary, the Mutual authentication protocol exchanges four values, \( K_{c1} \), \( K_{s1} \), \( VK_c \) and \( VK_s \), to perform authenticated key exchanges, using the password-derived secret \( pi \) and its "augmented version" \( J(pi) \). This document defines the set of functions \( K_{c1} \), \( K_{s1} \), and \( J \) for a specific algorithm family.

Please note that, from the view of cryptographic literatures, the original functionality of Augumented PAKE is separated into the functions \( K_{c1} \) and \( K_{s1} \) defined in this draft, and the functions \( VK_c \) and \( VK_s \) defined in Section 11 of [I-D.ietf-httpauth-mutual] as "default functions". For the purpose of security analysis, please also refer to these functions.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
The term "natural numbers" refers to the non-negative integers (including zero) throughout this document.

This document treats target (codomain) of hash functions to be octet strings. The notation \( \text{INT}(H(s)) \) gives a natural-number output of hash function \( H \) applied to string \( s \).

2. Authentication Algorithms

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

- iso-kam3-dl-2048-sha256: for the 2048-bit discrete-logarithm setting with the SHA-256 hash function.
- iso-kam3-dl-4096-sha512: for the 4096-bit discrete-logarithm setting with the SHA-512 hash function.
- iso-kam3-ec-p256-sha256: for the 256-bit prime-field elliptic-curve setting with the SHA-256 hash function.
- iso-kam3-ec-p521-sha512: for the 521-bit prime-field elliptic-curve setting with the SHA-512 hash function.

For discrete-logarithm settings, the underlying groups are the 2048-bit and 4096-bit MODP groups defined in [RFC3526], respectively. See Appendix A for the exact specifications of the groups and associated parameters. The hash functions \( H \) are SHA-256 for the 2048-bit group and SHA-512 for the 4096-bit group, respectively, defined in FIPS PUB 180-2 [FIPS.180-2.2002]. The hash iteration count \( nIterPi \) is 16384. The representation of the parameters \( kc1, ks1, vkc, \) and \( vks \) is base64-fixed-number.

For the elliptic-curve settings, the underlying groups are the elliptic curves over the prime fields P-256 and P-521, respectively, specified in the appendix D.1.2 of FIPS PUB 186-3 [FIPS.186-3.2009] specification. The hash functions \( H \), which are referenced by the core document, are SHA-256 for the P-256 curve and SHA-512 for the P-521 curve, respectively. The hash iteration count \( nIterPi \) is 16384. The representation of the parameters \( kc1, ks1, vkc, \) and \( vks \) is hex-fixed-number.

[[Editorial Note: remove before submission] We should take a care on recent hot discussion topic on the choice of elliptic curves for cryptography. If the discussion leads to some recommendation on "default choice of curves" before publishing this document as an RFC, the authors will reflect that change to the paragraph above (reference to FIPS 186-3). If not, we may need to update this document with specifying a new set of algorithm identifier tokens for use of new curves.]

Note: This algorithm is based on the Key Agreement Mechanism 3 (KAM3) defined in Section 6.3 of ISO/IEC 11770-4 [ISO.11770-4.2006] with a few modifications/improvements. However, implementers should use this document as the normative reference, because the algorithm has been changed in several minor details as well as major improvements.

2.1. Support Functions and Notations

The algorithm definitions use several support functions and notations defined below:

The integers in the specification are in decimal, or in hexadecimal when prefixed with "0x".
The functions named octet(), OCTETS(), and INT() are those defined in the core specification [I-D.ietf-httpauth-mutual].

Note: The definition of OCTETS() is different from the function GE2OS_x in the original ISO specification, which takes the shortest representation without preceding zeros.

All of the algorithms defined in this specification use the default functions defined in the core specification (defined in Section 11 of [I-D.ietf-httpauth-mutual]) for computing the values pi, VKc and VKs.

2.2. Functions for Discrete-Logarithm Settings

In this section, an equation \((x / y \mod z)\) denotes a natural number \(w\) less than \(z\) that satisfies \((w \ast y) \mod z = x \mod z\).

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

- **q**: for "the prime" defining the MODP 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
\[ J(pi) = g^{pi} \mod q. \]

The value of \(K_{c1}\) is derived as
\[ K_{c1} = g^{S_{c1}} \mod q, \]
where \(S_{c1}\) is a random integer within range \([1, r-1]\) and \(r\) is the size of the subgroup generated by \(g\). In addition, \(S_{c1}\) MUST be larger than \(\log(q)/\log(g)\) (so that \(g^{S_{c1}} > q\)).

The value of \(K_{c1}\) SHALL satisfy \(1 < K_{c1} < q-1\). The server MUST check this condition upon reception.

Let an intermediate value \(t_1\) be
\[ t_1 = \text{INT}(H(\text{octet}(1) \mid \text{OCTETS}(K_{c1}))), \]
the value of \(K_{s1}\) is derived from \(J(pi)\) and \(K_{c1}\) as:
\[ K_{s1} = (J(pi) \ast K_{c1}^{t_1})^{S_{s1}} \mod q, \]
where \(S_{s1}\) is a random number within range \([1, r-1]\). The value of \(K_{s1}\) MUST satisfy \(1 < K_{s1} < q-1\). If this condition is not held, the server MUST retry using another value for \(S_{s1}\). The client MUST check this condition upon reception.

Let an intermediate value \(t_2\) be
\[ t_2 = \text{INT}(H(\text{octet}(2) \mid \text{OCTETS}(K_{c1}) \mid \text{OCTETS}(K_{s1})) ), \]

the value \( z \) on the client side is derived by the following equation:

\[ z = K_{s1} \wedge ((S_{c1} + t_2) / (S_{c1} \ast t_1 + \pi) \mod r) \mod q. \]

The value \( z \) on the server side is derived by the following equation:

\[ z = (K_{c1} \ast g^{(t_2)})^{(S_{s1})} \mod q. \]

(Note: the original ISO specification contained a message pair containing verification of value \( z \) along with the "transcript" of the protocol exchange. The functionality of this kind is contained in the functions \( VK_c \) and \( VK_s \).

2.3. Functions for Elliptic-Curve Settings

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

- \( q \): for the prime used to define the group.
- \( G \): for the defined point called the generator.
- \( r \): for the order of the subgroup generated by \( G \).

The function \( P(p) \) converts a curve point \( p \) into an integer representing point \( p \), by computing \( x \ast 2 + (y \mod 2) \), where \( (x, y) \) are the coordinates of point \( p \). \( P'(z) \) is the inverse of function \( P \), that is, it converts an integer \( z \) to a point \( p \) that satisfies \( P(p) = z \). If such \( p \) exists, it is uniquely defined. Otherwise, \( z \) does not represent a valid curve point. The operator + indicates the elliptic-curve group operation, and the operation \([x] \ast p\) denotes an integer-multiplication of point \( p \): it calculates \( p + p + \ldots \) (\( x \) times) \( + p \). See the literatures on elliptic-curve cryptography for the exact algorithms used for those functions (e.g. Section 3 of \([\text{RFC6090}]\), which uses different notations, though.) \( 0_E \) represents the infinity point. The equation \( (x / y \mod z) \) denotes a natural number \( w \) less than \( z \) that satisfies \( (w \ast y) \mod z = x \mod z \).

The function \( J \) is defined as

\[ J(\pi) = [\pi] \ast G. \]

The value of \( K_{c1} \) is derived as

\[ K_{c1} = P(K_{c1}') \), where \( K_{c1}' = [S_{c1}] \ast G, \]

where \( S_{c1} \) is a random number within range \([1, r-1]\). The value of \( K_{c1} \) MUST represent a valid curve point, and \( K_{c1}' \) SHALL NOT be \( 0_E \). The server MUST check this condition upon reception.

Let an intermediate integer \( t_1 \) be

\[ t_1 = \text{INT}(H(\text{octet}(1) \mid \text{OCTETS}(K_{c1}) ), \]

the value of \( K_{s1} \) is derived from \( J(\pi) \) and \( K_{c1}' = P'(K_{c1}) \) as:
\[ K_{s1} = P([S_{s1}] * (J(\pi_i) + [t_1] * K_{c1}'))) \]

where \( S_{s1} \) is a random number within range \([1, r-1]\). The value of \( K_{s1} \) MUST represent a valid curve point and satisfy \([4] * P'(K_{s1}) \not\equiv 0 \mod{E}\). If this condition is not satisfied, the server MUST retry using another value for \( S_{s1} \). The client MUST check this condition upon reception.

Let an intermediate integer \( t_2 \) be

\[ t_2 = \text{INT}(H(\text{octet}(2) | \text{OCTETS}(K_{c1}) | \text{OCTETS}(K_{s1}))), \]

the value \( z \) on the client side is derived by the following equation:

\[ z = P([(S_{c1} + t_2) / (S_{c1} * t_1 + \pi) \mod{r}] * P'(K_{s1})). \]

The value \( z \) on the server side is derived by the following equation:

\[ z = P([S_{s1}] * (P'(K_{c1}) + [t_2] * G)). \]

3. IANA Considerations

Four tokens iso-kam3-dl-2048-sha256, iso-kam3-dl-4096-sha512, iso-kam3-ec-p256-sha256 and iso-kam3-ec-p521-sha512 shall be allocated and registered according to the provision of the core documentation when this document is promoted to an RFC.

Note: More formal declarations will be added in the future drafts to meet the RFC 5226 requirements.

4. Security Considerations

Refer the corresponding section of the core specification for algorithm-independent, generic considerations.

- All random numbers used in these algorithms MUST be at least cryptographically computationally secure against forward and backward guessing attacks.
- Computation times of all numerical operations on discrete-logarithm group elements and elliptic-curve points MUST be normalized and made independent of the exact values, to prevent timing-based side-channel attacks.

The usual construction of authenticated key exchange algorithms are build from a key-exchange period and a key verification period, and the latter usually involving some kind of exchange transaction to be verified, to avoid security risks or vulnerabilities caused from mixing of values from two or more key exchanges. In the design of the algorithms in this document, such a functionality is defined in generalized manner in the core specification [I-D.ietf-httpauth-mutual] (see definitions of \( VK_c \) and \( VK_s \)). If any attempts to reuse the algorithm defined above with any other protocols exist, care MUST be taken on that aspect.

5. Notice on intellectual properties

The National Institute of Advanced Industrial Science and Technology (AIST) and Yahoo! Japan, Inc. has jointly submitted a patent application on 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. For the details of the patent application and its status, please contact the author of this document.

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

6. References

6.1. Normative References


6.2. Informative References


Appendix A. (Informative) Group Parameters for Discrete-Logarithm Based Algorithms

The MODP group used for the iso-kam3-dl-2048-sha256 algorithm is defined by the following parameters.

The prime is:
\begin{align*}
q &= 0xFFFFFFFF \text{ FFFFFFFF C90FDAA2 2168C234 C466228B 80DC1CD1} \\
   & \quad 29024E08 8A67CC74 02B0BEAE 3B139B22 514A0879 8E3404DD \\
EF9519B3 \text{ CD3A431B 302B0A6D F251437 4EF1356D 6D51C245} \\
E485B576 \text{ 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED} \\
EE386BFB \text{ 5A899FA5 AE9F2411 7C4B1F6E 9286651 ECE45B3D} \\
C2007CB8 \text{ A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F} \\
83655D23 \text{ DCA3AD96 1C62F356 208552BB 9ED52907 709696ED} \\
670C354E \text{ 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B} \\
E39E772C \text{ 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9} \\
DE2BCBF6 \text{ 95581718 3995497C EA956AE5 15D22618 98FA0510} \\
15728E5A \text{ 8AACA668 FFFFEFF FFFFFFFF}. \\
\end{align*}

The generator is:

\[ g = 2. \]

The size of the subgroup generated by \( g \) is:

\[ r = \frac{(q - 1)}{2} = \\
\begin{align*}
& 0x7FFFFFFF \text{ FFFFFFFF E487ED51 10B4611A 62633145 C060E686} \\
& \quad 94812704 453E63A 0105DF53 1D89CD91 28A05043 C71A026E \\
& \quad F7CA8CD9 E69D218D 98158536 F929FA1B A7F90AB6 B6A8E122 \\
& \quad F242DAB3 312F36F8 7A262174 D31BF65B 85FA2EB5 7A035BF6 \\
& \quad 7F1C35FD AD44C92D D74F9208 BE258FF3 24943238 F67229DE \\
& \quad E100E5EC 50BD1F82 CC6D241B 0E2A9EDC 34BB1FD4 7E9267AF \\
& \quad C1B2AE91 E51D6CB 0E3179AB 1042A95D CF6A4943 B84B4B36 \\
& \quad B3861A97 25BE4C02 78BA3604 650C10BE 19482F23 171B671D \\
& \quad F1CF3896 0C074301 CD93C1D1 76031D47 DAE2AE9F 37A62964 \\
& \quad EF155E8F 4AAC08BC 1CCAA4BE 754AB572 8AE9130C 4C7D0288 \\
& \quad 0AB9472D 45565534 7FFFFFFF FFFFFFFF. \\
\end{align*}

The MODP group used for the iso-kam3-dl-4096-sha512 algorithm is defined by the following parameters.

The prime is:

\begin{align*}
q &= 0xFFFFFFFF \text{ FFFFFFFF C90FDAA2 2168C234 C466228B 80DC1CD1} \\
   & \quad 29024E08 8A67CC74 02B0BEAE 3B139B22 514A0879 8E3404DD \\
EF9519B3 \text{ CD3A431B 302B0A6D F251437 4EF1356D 6D51C245} \\
E485B576 \text{ 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED} \\
EE386BFB \text{ 5A899FA5 AE9F2411 7C4B1F6E 9286651 ECE45B3D} \\
C2007CB8 \text{ A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F} \\
83655D23 \text{ DCA3AD96 1C62F356 208552BB 9ED52907 709696ED} \\
670C354E \text{ 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B} \\
E39E772C \text{ 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9} \\
DE2BCBF6 \text{ 95581718 3995497C EA956AE5 15D22618 98FA0510} \\
15728E5A \text{ 8AACA668 FFFFEFF FFFFFFFF}. \\
\end{align*}
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
F71C55FD AD44CDF2 D74F9208 BE258FF3 24943328 F6722D9E
E1003E5C 5081DF82 CC6D241B OE2AE9CD 34BB1FD4 7E9267AF
C1B2A291 EE51D6CB 0E3179AB 1042A85D CF6A4983 B84B4B36
B3861AA7 255E4C02 78BA3604 650C10BE 19482F23 171B671D
F1CF89B6 0C074301 CD93C1D1 7603D147 DAE2AEF8 37A62964

Appendix B. (Informative) Derived Numerical Values

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

<table>
<thead>
<tr>
<th></th>
<th>dl-2048</th>
<th>dl-4096</th>
<th>ec-p256</th>
<th>ec-p521</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of ( K_{c1} ) etc.</td>
<td>2048</td>
<td>4096</td>
<td>257</td>
<td>522</td>
</tr>
<tr>
<td>hSize, Size of ( H(...) )</td>
<td>256</td>
<td>512</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>length of OCTETS(( K_{c1} )) etc.</td>
<td>256</td>
<td>512</td>
<td>33</td>
<td>66</td>
</tr>
<tr>
<td>length of ( k_{c1}, k_{s1} ) param. values.</td>
<td>344 *</td>
<td>684 *</td>
<td>66</td>
<td>132</td>
</tr>
<tr>
<td>length of ( v_{c1}, v_{s1} ) param. values.</td>
<td>44 *</td>
<td>88 *</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>minimum allowed ( S_{c1} )</td>
<td>2048</td>
<td>4096</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(The numbers marked with an * do not include any enclosing quotation marks.)

Appendix C. (Informative) Draft Change Log
C.1. Changes in HTTPAUTH-WG revision 00

- Added a note on the choice of elliptic curves.

C.2. Changes in HTTPAUTH revision 02

- Added nIterPi parameter to adjust to the changes to the core draft.
- Added a note on the verification of exchange transaction.

C.3. Changes in HTTPAUTH revision 01

- Notation change: integer output of hash function will be notated as INT(H(*)), changed from H(*).

C.4. Changes in revision 02

- Implementation hints in appendix changed (number of characters for base64-fixed-number does not contain double-quotes).

C.5. Changes in revision 01

- Parameter names renamed.
- Some expressions clarified without changing the value.

C.6. Changes in revision 00

The document is separated from the revision 08 of the core documentation.

Authors’ Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
3-11-46 Nakouji
Amagasaki, Hyogo
JP
Email: mutual-auth-contact-ml@aist.go.jp

Hajime Watanabe
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
Tsukuba Central 2
1-1-1 Umezono
Tsukuba-shi, Ibaraki
JP

Hiromitsu Takagi
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
Tsukuba Central 2
1-1-1 Umezono
Tsukuba-shi, Ibaraki
JP

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

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

Yuichi Ioku
Individual