Internet Engineering Task Force (IETF)
Request for Comments: 8121
Category: Experimental
ISSN: 2070-1721

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

April 2017

Mutual Authentication Protocol for HTTP: Cryptographic Algorithms Based on the Key Agreement Mechanism 3 (KAM3)

Abstract

This document specifies cryptographic algorithms for use with the Mutual user authentication method for the Hypertext Transfer Protocol (HTTP).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies algorithms for use with the Mutual authentication protocol for the Hypertext Transfer Protocol (HTTP) [RFC8120] (hereafter referred to as the "core specification"). The algorithms are based on augmented password-based authenticated key exchange (augmented PAKE) techniques. In particular, it uses one of three key exchange algorithms defined in ISO 11770-4 ("Information technology - Security techniques - Key management - Part 4: Mechanisms based on weak secrets") [ISO.11770-4.2006] as its basis.

To briefly summarize, 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 point of view of literature related to cryptography, the original functionality of augmented PAKE is separated into the functions K_c1 and K_s1 as defined in this document, and the functions VK_c and VK_s, which are defined in Section 12.2 of [RFC8120] 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 non-negative integers (including zero) throughout this document.

This document treats both the input (domain) and the output (codomain) of hash functions as octet strings. When a natural-number output of hash function H is required, it will be notated, for example, as INT(H(s)).

2. Cryptographic Overview (Non-normative)

The cryptographic primitive used in this algorithm specification is based on a variant of augmented PAKE called "APKAS-AMP" (augmented password-authenticated key agreement scheme, version AMP), proposed by T. Kwon and originally submitted to [IEEE-1363.2_2008]. The general flow of the successful exchange is shown below for informative purposes only. The multiplicative notations are used for group operators, and all modulus operations for finite groups (mod q and mod r) are omitted.

```
      C: S_c1 = random
      C: K_c1 = g^(S_c1)
                        ----- ID, K_c1 ----->
      C: t_1 = H1(K_c1)               S: t_1 = H1(K_c1)
                                      S: fetch J = g^pi by ID
                                      S: S_s1 = random
                                      S: K_s1 = (J * K_c1^(t_1))^(S_s1)
                        <----- K_s1 -----
      C: t_2 = H2(K_c1, K_s1)         S: t_2 = H2(K_c1, K_s1)
      C: z = K_s1^((S_c1 + t_2) / (S_c1 * t_1 + pi))
                                      S: z' = (K_c1 * g^(t_2))^(S_s1)
      (assumption at this point: z = z' if authentication succeeded)

      C: VK_c = H4(K_c1, K_s1, z)     S: VK_c' = H4(K_c1, K_s1, z')
                        ----- VK_c ------->
                                      S: assert(VK_c = VK_c')

      C: VK_s' = H3(K_c1, K_s1, z)    S: VK_s = H3(K_c1, K_s1, z')
                        <----- VK_s ------
      C: assert(VK_s = VK_s')
```

Note that the concrete (binary) message formats (mapping to HTTP messages), as well as the formal definitions of equations for the latter two messages, are defined in the core specification [RFC8120]. The formal definitions for values corresponding to the first two messages are defined in the following sections.

3. Authentication Algorithms

This document specifies one family of algorithms based on APKAS-AMP, to be used with [RFC8120]. This family consists of four authentication algorithms, which differ only in their underlying mathematical groups and security parameters. These algorithms do not add any additional parameters. The tokens for these algorithms are as follows:

- o iso-kam3-dl-2048-sha256: for the 2048-bit discrete-logarithm setting with the SHA-256 hash function.

- o iso-kam3-dl-4096-sha512: for the 4096-bit discrete-logarithm setting with the SHA-512 hash function.

- o iso-kam3-ec-p256-sha256: for the 256-bit prime-field elliptic-curve setting with the SHA-256 hash function.

- o 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 Modular Exponential (MODP) groups defined in [RFC3526]. See Appendix A for the exact specifications for the groups and associated parameters. Hash function H is SHA-256 for the 2048-bit group and SHA-512 for the 4096-bit group, respectively, as defined in FIPS PUB 180-4 [FIPS.180-4.2015]. 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, as specified in Appendix D.1.2 of the FIPS PUB 186-4 [FIPS.186-4.2013] specification. Hash function H is SHA-256 for the P-256 curve and SHA-512 for the P-521 curve, respectively. Cofactors of these curves are 1. The hash iteration count nIterPi is 16384. The representation of the parameters "kc1", "ks1", "vkc", and "vks" is hex-fixed-number.

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

3.1. Support Functions and Notations

The algorithm definitions use the support functions and notations defined below.

Decimal notations are used for integers in this specification by default. Integers in hexadecimal notations are prefixed with "0x".

In this document, the octet(), OCTETS(), and INT() functions are used as defined in the core specification [RFC8120].

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

All of the algorithms defined in this specification use the default functions defined in Section 12.2 of [RFC8120] for computing the values pi, VK_c, and VK_s.

3.2. Functions for Discrete-Logarithm Settings

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

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

$$J(pi) = g^{(pi)} \bmod q$$

The value of K_c1 is derived as

$$K_{c1} = g^{(S_{c1})} \bmod q$$

where S_c1 is a random integer within the 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 server MUST check the condition 1 < K_c1 < q-1 upon reception.

Let an intermediate value t_1 be

$$t_1 = INT(H(octet(1) \mid OCTETS(K_{c1})))$$

The value of K_s1 is derived from J(pi) and K_c1 as

$$K_{s1} = (J(pi) * K_{c1}^{(t_1)})^{(S_{s1})} \bmod q$$

where S_s1 is a random number within the 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 reject the exchange. The client MUST check this condition upon reception.

Let an intermediate value t_2 be

$$t_2 = INT(H(octet(2) \mid OCTETS(K_{c1}) \mid OCTETS(K_{s1})))$$

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

$$z = K_{s1}^{((S_{c1} + t_2) / (S_{c1} * t_1 + pi) \bmod r)} \bmod q$$

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

```
   z = (K_c1 * 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. This functionality is contained in the functions VK_c and VK_s.)

3.3. Functions for Elliptic-Curve Settings

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

- q: for the prime used to define the group.
- G: for the point defined with the underlying group called "the generator".
- h: for the cofactor of the group.
- 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 * 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] * p denotes an integer-multiplication of point p: it calculates p + p + ... (x times) ... + p. See the literature on elliptic-curve cryptography for the exact algorithms used for those functions (e.g., Section 3 of [RFC6090]; however, note that [RFC6090] uses different notations). 0_E represents the infinity point. The equation (x / y mod z) denotes a natural number w less than z that satisfies (w * y) mod z = x mod z.

The function J is defined as

```
   J(pi) = [pi] * G
```

The value of K_c1 is derived as

```
   K_c1 = P(K_c1'), where K_c1' = [S_c1] * G
```

where S_c1 is a random number within the range [1, r-1]. The server MUST check that (1) the value of received K_c1 represents a valid curve point and (2) [h] * K_c1' is not equal to 0_E.

Let an intermediate integer t_1 be

```
   t_1 = INT(H(octet(1) | 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) + [t_1] * K_c1'))
```

where S_s1 is a random number within the range [1, r-1]. The value of K_s1 MUST represent a valid curve point and satisfy [h] * P'(K_s1) <> 0_E. If this condition is not satisfied, the server MUST reject the exchange. The client MUST check this condition upon reception.

Let an intermediate integer t_2 be

```
   t_2 = INT(H(octet(2) | OCTETS(K_c1) | 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))
```

4. IANA Considerations

This document defines four new tokens that have been added to the "HTTP Mutual Authentication Algorithms" registry:

Token	Description	Reference
iso-kam3-dl-2048-sha256	ISO-11770-4 KAM3, 2048-bit DL	RFC 8121
iso-kam3-dl-4096-sha512	ISO-11770-4 KAM3, 4096-bit DL	RFC 8121
iso-kam3-ec-p256-sha256	ISO-11770-4 KAM3, 256-bit EC	RFC 8121
iso-kam3-ec-p521-sha512	ISO-11770-4 KAM3, 521-bit EC	RFC 8121

5. Security Considerations

Please refer to the Security Considerations section of the core specification [RFC8120] for algorithm-independent considerations.

5.1. General Implementation Considerations

- o During the exchange, the value VK_s, defined in [RFC8120], MUST only be sent when the server has received a correct (expected) value of VK_c. This is a cryptographic requirement, as stated in [ISO.11770-4.2006].

- o All random numbers used in these algorithms MUST be cryptographically secure against forward and backward guessing attacks.

- o To prevent timing-based side-channel 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.

5.2. Cryptographic Assumptions and Considerations

The notes in this subsection are for those who analyze the security of this algorithm and those who might want to make a derived work from this algorithm specification.

- The treatment of an invalid K_s1 value in the exchange has been changed from the method defined in the original ISO specification, which specifies that the sender should retry with another random S_s1 value. We specify that the exchange must be rejected. This is due to an observation that this condition is less likely to result from a random error caused by an unlucky choice of S_s1 but is more likely the result of a systematic failure caused by an invalid J(pi) value (even implying possible denial-of-service attacks).

- The usual construction of authenticated key exchange algorithms consists of a key exchange phase and a key verification phase. To avoid security risks or vulnerabilities caused by mixing values from two or more key exchanges, the latter usually involves some kinds of exchange transactions to be verified. In the algorithms defined in this document, such verification steps are provided in the generalized definitions of VK_c and VK_s in [RFC8120]. If the algorithm defined above is used in other protocols, this aspect MUST be given careful consideration.

- The domain parameters chosen and specified in this document are based on a few assumptions. In the discrete-logarithm setting, q has to be a safe prime ([(q - 1) / 2] must also be prime), and r should be the largest possible value [(q - 1) / 2]. In the elliptic-curve setting, r has to be prime. Implementers defining a variation of this algorithm using a different domain parameter SHOULD be attentive to these conditions.

6. References

6.1. Normative References

[FIPS.180-4.2015]
National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[FIPS.186-4.2013]
National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-4, DOI 10.6028/NIST.FIPS.186-4, July 2013, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, DOI 10.17487/RFC3526, May 2003, <http://www.rfc-editor.org/info/rfc3526>.

[RFC8120] Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "Mutual Authentication Protocol for HTTP", RFC 8120, DOI 10.17487/RFC8120, April 2017, <http://www.rfc-editor.org/info/rfc8120>.

6.2. Informative References

[IEEE-1363.2_2008]
IEEE, "IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques", IEEE 1363.2-2008, DOI 10.1109/ieeestd.2009.4773330, <http://ieeexplore.ieee.org/servlet/opac?punumber=4773328>.

[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, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39723>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <http://www.rfc-editor.org/info/rfc6090>.

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

The MODP group used for the iso-kam3-dl-2048-sha256 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 iso-kam3-dl-4096-sha512 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. (Informative) Derived Numerical Values

This section provides several numerical values for implementing this protocol. These values are derived from the specifications provided in Section 3. The values shown in this section are for informative purposes only.

	dl-2048	dl-4096	ec-p256	ec-p521	
Size of K_c1, etc.	2048	4096	257	522	(bits)
hSize, size of H(...)	256	512	256	512	(bits)
Length of OCTETS(K_c1), etc.	256	512	33	66	(octets)
Length of kc1, ks1 param. values	344*	684*	66	132	(octets)
Length of vkc, vks param. values	44*	88*	64	128	(octets)
Minimum allowed S_c1	2048	4096	1	1	

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

Authors' Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Information Technology Research Institute
Tsukuba Central 1
1-1-1 Umezono
Tsukuba-shi, Ibaraki
Japan
Email: y.oiwa@aist.go.jp

Hajime Watanabe
National Institute of Advanced Industrial Science and Technology
Information Technology Research Institute
Tsukuba Central 1
1-1-1 Umezono
Tsukuba-shi, Ibaraki
Japan
Email: h-watanabe@aist.go.jp

Hiromitsu Takagi
National Institute of Advanced Industrial Science and Technology
Information Technology Research Institute
Tsukuba Central 1
1-1-1 Umezono
Tsukuba-shi, Ibaraki
Japan
Email: takagi.hiromitsu@aist.go.jp

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

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

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