Network Working Group
Internet Draft

Expires September, 1996

H. Krawczyk
M. Bellare
R. Canetti
March, 1996

HMAC-MD5: Keyed-MD5 for Message Authentication

<draft-ietf-ipsec-hmac-md5-00.txt>

Status of this Memo

Abstract

This document describes a keyed-MD5 mechanism (called HMAC-MD5) for use as a message authentication code (or, integrity check value). It is mainly intended for integrity verification of information transmitted over open networks (e.g., Internet) between parties that share a common secret key. The proposed mechanism combines the (key-less) MD5 hash function [RFC-1321] with a shared secret key. The description of HMAC-MD5 in this document is independent of its use in any particular protocol. An analogous mechanism can be used in combination with other iterative hash functions, e.g., SHA.

1. Introduction

Rivest introduced MD5 [RFC-1321] as a cryptographic hash function. It was originally designed and intended as a collision-resistant function as required for the hashing of information prior to application of a signature function (e.g., RSA). However, due to its relatively good performance in software implementation and its free availability the same function was adapted to provide functionalities different than the one for which MD5 was originally designed. In particular, MD5 (as well as other cryptographic hash functions) was proposed to provide message authentication by combining it with a secret key (see [Tsu]).

Only recently a close analysis of the security of these keyed MD5 modes has been initiated [BCK1,BCK2,KR,PV1,PV2]. In particular, Bellare, Canetti, and Krawczyk [BCK2] described a keyed-hash mechanism called HMAC which we adopt here for use with MD5. (An analogous mechanism can be used in combination with other iterative hash functions, e.g., SHA [SHA].) The description of this transform in this document is independent of its use in any particular protocol. It is intended to serve as a common mechanism for the many protocols (especially, IETF protocols) that require integrity verification based on a shared secret key (e.g., IP Authentication Header [RFC-1826]).

The proposed mechanism follows the same principles that guided some of the previous keyed-MD5 proposals, that is:

* it is based on MD5
* no change to the MD5 code required
* no degradation of MD5 speed
* simple key requirements
* replaceability of MD5 by other cryptographic hash functions

However, it improves on previous proposals relative to its security analysis. The present is the first construction (and the only one we are aware of) that can be fully analyzed based on relatively weak assumptions on the underlying hash function (MD5). It only requires MD5 to be collision-resistant in a weak sense, and its compression function to be weakly unpredictable. These requirements are weaker than the ones needed for other common applications of MD5, e.g., as hash functions for digital signatures and as "randomizers" (for pseudorandom generation, key derivation, etc.). In particular, we only require that collisions are hard to find when the "initial vector" of the function is random and secret, and that the output of the compression function is unpredictable when applied to partially unknown strings. The analytical results that back this particular construction are provided in [BCK2].

Note: The mechanism presented next differs from the one presented in draft-krawczyk-keyed-md5-txt.01 in the way padding is defined (see section 2).

2. Calculation

The definition and reference implementation of MD5 appears in [RFC-1321]. Let `text' denote the data to which HMAC-MD5 is to be applied and K be the message authentication secret key shared by the parties. The key K can be of any length up to the block length of the hash function, namely, 64 bytes for MD5 (however, 16 bytes is the minimal recommended length for keys -- see section 3). Applications that use longer than 64 byte keys will first hash the key using MD5 and then use the resultant 16 byte string as the actual key to HMAC-MD5.

We define two fixed and different strings ipad and opad as follows (the 'i' and 'o' are mnemonics for inner and outer):

```
ipad = the byte 0x36 repeated 64 times
opad = the byte 0x5C repeated 64 times.
```

To compute HMAC-MD5 over the data `text' we perform

```
MD5(K XOR opad, MD5(K XOR ipad, text))
```

Namely,

(1) append zeros to the end of K to create a 64 byte string (e.g., if K is of length 16 bytes it will be appended with 48 zero bytes 0x00)
(2) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with ipad
(3) append the data stream 'text' to the 64 byte string resulting from step (2)
(4) apply MD5 to the stream generated in step (3)
(5) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with opad
(6) append the MD5 result from step (4) to the 64 byte string resulting from step (5)
(7) apply MD5 to the stream generated in step (6) and output the result

For illustration purposes, sample code is provided as an appendix.

3. Keys

The key for HMAC-MD5 can be of any length (keys longer than 64 bytes are first hashed using MD5). However, less than 16 bytes is strongly discouraged as it would decrease the security strength of the function. Keys longer than 16 bytes are acceptable but the extra length would not significantly increase the function strength. (A longer key may be advisable if the randomness of the key is considered weak.)

Note: when using other cryptographic hash functions the length of the key should be chosen at least as the length of the hash output (e.g., 160 bits in the case of SHA).

Keys need to be chosen at random, or using a cryptographically strong pseudo-random generator seeded with a random seed.
We recommend that the key be changed periodically and as frequently as possible. (Current attacks do not indicate a specific recommended frequency for key changes as these attacks are practically infeasible. However, periodic key refreshment is a fundamental security practice that helps against potential weaknesses of the function and keys, and limits the damage of an exposed key.)

4. Implementation Note

Implementation of HMAC-MD5 requires the implementation of MD5 (see [RFC-1321]) and the calculation described in Section 2.
Notice that this calculation does not require any change in the definition or code of MD5. However, if desired, a performance improvement can be achieved by a simple adaptation of the MD5 code as presented next. (Choosing or not to implement HMAC-MD5 in this way is a decision of the local implementation and has no effect on inter-operability.)
The idea is that the intermediate results of MD5 on the 64-byte blocks (K XOR ipad) and (K XOR opad) can be precomputed only once at the time of generation of the key K, or before its first use. These intermediate results (called "contexts", and stored under MD5's structure MD5_CTX [RFC-1321]) are then used to initialize the MD5 function each time that a message needs to be authenticated. (This involves setting the MD5_CTX variable to the stored value and applying MD5Update to the data.) This method saves, for each authenticated message,
the application of the MD5's compression function (MD5Update) on two 64-byte blocks; a savings which may be significant when authenticating short streams of data. We stress that the stored contexts need to be treated and protected the same as secret keys.

5. Security

The security of the message authentication mechanism presented here depends on cryptographic properties of MD5: the resistance to collision finding (limited to the case where the initial value is secret and random, and where the output of the function is not explicitly available to the attacker), and the message authentication property of the compression function of MD5 when applied to single blocks (these blocks are partially unknown to an attacker as they contain the result of the internal MD5 computation and, in particular, cannot be fully chosen by the attacker).

These properties, and actually stronger ones, are commonly assumed for MD5. While significant weaknesses of MD5 regarding these properties could be discovered in the future, such weaknesses would rend MD5 useless for most (probably, all) cryptographic applications, including alternative message authentication schemes based on this function. Two important properties of the application of MD5 here are:

1. This construction is independent of the particular details of MD5 and then the latter can be replaced by any other secure (iterative) cryptographic hash function.

2. Message authentication, as opposed to encryption, has a "transient" effect. A published breaking of a message authentication scheme would lead to the replacement of that scheme, but would have no adversarial effect on information authenticated in the past. This is in sharp contrast with encryption, where information encrypted today may suffer from exposure in the future if, and when, the encryption algorithm is broken.

The strongest attack known against the proposed scheme is based on the frequency of collisions for MD5 ("birthday attack") [BCK1,PV] for which the attacker needs to acquire the correct message authentication tags computed on about 2**64 known plaintexts (and with the _same_ secret key K!). This would require the processing of 2^70 bytes under MD5, an impossible task in any realistic scenario (this would take 250,000 years in a continuous 1Gbit link, and without changing the secret key K all this time). This attack could become realistic only if serious flaws in the collision behavior of MD5 are discovered (e.g. collisions found after 2**30 messages). Such a discovery would determine the immediate replacement of MD5 (the effects of such failure would be far more severe for the traditional uses of MD5 in the context of digital signatures, public key certificates, etc.).

Note: this attack needs to be strongly contrasted with regular collision attacks on MD5 where no secret key is involved and where 2^64 off-line parallelizable (!) operations suffice to find collisions. The latter attack is approaching feasibility [VW] while the birthday attack on HMAC-MD5 is totally impractical.
(In all of the above discussion 64 should be replaced by 80 if one uses a 160 bit hash function as SHA.)

A correct implementation of the above construction, the choice of random (or cryptographically pseudorandom) keys, a secure key exchange mechanism, frequent key refreshments, and good secrecy protection of keys are all essential ingredients for the security of the integrity verification mechanism provided by HMAC-MD5.

Appendix -- Sample Code

The following sample code for the implementation of HMAC-MD5 and corresponding test vectors are provided for the sake of illustration only.

```
/*
** Function: hmac_md5
*/

void
hmac_md5(text, text_len, key, key_len, digest)
unsigned char*  text;                /* pointer to data stream */
int             text_len;            /* length of data stream */
unsigned char*  key;                 /* pointer to authentication key */
int             key_len;             /* length of authentication key */
caddr_t         digest;              /* caller digest to be filled in */

{
        MD5_CTX context;
        unsigned char k_ipad[65];     /* inner padding - key XORd with ipad */
        unsigned char k_opad[65];     /* outer padding - key XORd with opad */
        unsigned char tk[16];
        int i;

        /* sanity check parameters */
        if ((text == NULL) || (key == NULL) || (digest == NULL))
                /* most heinous, probably should log something */
                return;

        /* if key is longer than 64 bytes reset it to key=MD5(key) */
        if (key_len > 64) {

                MD5_CTX      tctx;

                MD5Init(&tctx);
                MD5Update(&tctx, key, key_len);
                MD5Final(tk, &tctx);

                key = tk;
                key_len = 16;
        }

        /*
         * the HMAC_MD5 transform looks like:
         *
         * MD5(K XOR opad, MD5(K XOR ipad, text))
         *
         * where K is an n byte key
         * ipad is the byte 0x36 repeated 64 times
         * opad is the byte 0x5c repeated 64 times
```

```
        * and text is the data being protected
        */
```

```
        /* start out by storing key in pads */
        bzero( k_ipad, sizeof k_ipad);
        bzero( k_opad, sizeof k_opad);
        bcopy( key, k_ipad, key_len);
        bcopy( key, k_opad, key_len);

        /* XOR key with ipad and opad values */
        for (i=0; i<64; i++) {
                k_ipad[i] ^= 0x36;
                k_opad[i] ^= 0x5c;
        }
        /*
         * perform inner MD5
         */
        MD5Init(&context);                   /* init context for 1st pass */
        MD5Update(&context, k_ipad, 64)      /* start with inner pad */
        MD5Update(&context, text, text_len); /* then text of datagram */
        MD5Final(digest, &context);          /* finish up 1st pass */
        /*
         * perform outer MD5
         */
        MD5Init(&context);                   /* init context for 2nd pass */
        MD5Update(&context, k_opad, 64);     /* start with outer pad */
        MD5Update(&context, digest, 16);     /* then results of 1st hash */
        MD5Final(digest, &context);          /* finish up 2nd pass */
}
```

Test Vectors:

```
  key =         0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b
  key_len =     16
  data =        "Hi There"
  digest =      0x9294727a3638bb1c13f48ef8158bfc9d


  key =         "Jefe"
  data =        "what do ya want for nothing?"
  digest =      0x750c783e6ab0b503eaa86e310a5db738


  key =         0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
  key_len       16
  data =        0xDDDDDDDDDDDDDDDDDDDD...
                ..DDDDDDDDDDDDDDDDDDDD...
                ..DDDDDDDDDDDDDDDDDDDD...
                ..DDDDDDDDDDDDDDDDDDDD...
                ..DDDDDDDDDDDDDDDDDDDD
  data_len =    50
  digest =      0x56be34521d144c88dbb8c733f0e8b3f6
```

Acknowledgments

Adi Shamir has suggested the use of the XOR-based padding technique as adopted in this draft instead of the previous concatenation pads. Pau-Chen Cheng, Jeff Kraemer, and Michael Oehler, have provided useful comments on the draft, and ran the first interoperability tests of this specification. Jeff and Pau-Chen kindly provided the sample code and test vectors that appear in the appendix.

References

[RFC-1826] R. Atkinson, "IP Authentication Header", RFC-1826, August 1995.

[BCK1] M. Bellare, R. Canetti, and H. Krawczyk, "Cascaded Pseudorandomness and Its Concrete Security", manuscript.

[BCK2] M. Bellare, R. Canetti, and H. Krawczyk, "Keyed Hash Functions and Message Authentication", presented at the 1996 RSA Data Security Conference, San Francisco, Jan. 1996. (http://www.research.ibm.com/security/keyed-md5.html)

[KR] B. Kaliski and M. Robshaw, "Message Authentication with MD5", RSA Labs' CryptoBytes, Vol. 1 No. 1, Spring 1995. (http://www.rsa.com/rsalabs/cryptobytes/)

[PV1] B. Prennel and P. van Oorschot, "Building fast MACs from hash functions", Advances in Cryptology -- CRYPTO'95 Proceedings, Lecture Notes in Computer Science, Springer-Verlag Vol.963, 1995, pp. 1-14.

[PV2] B. Prennel and P. van Oorschot, "On the security of two MAC algorithms", to appear Eurocrypt'96.

[RFC-1321] Ronald Rivest, "The MD5 Message-Digest Algorithm", RFC-1321, DDN Network Information Center, April 1992.

[SHA] NIST, FIPS PUB 180: Secure Hash Standard, May 1993.

[Tsu] G. Tsudik, "Message authentication with one-way hash functions", In Proceedings of Infocom~92, 1992.

[VW] P. van Oorschot and M. Wiener, "Parallel Collision Search with Applications to Hash Functions and Discrete Logarithms", Proceedings of the 2nd ACM Conf. Computer and Communications Security, Fairfax, VA, November 1994.

Authors Address

Hugo Krawczyk
IBM T.J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598

hugo@watson.ibm.com

Mihir Bellare
Dept of Computer Science and Engineering
Mail Code 0114
University of California at San Diego
9500 Gilman Drive
La Jolla, CA 92093

mihir@cs.ucsd.edu

Ran Canetti
Laboratory for Computer Science
MIT
545 Technology Square
Cambridge, Ma 02139

canetti@theory.lcs.mit.edu