Requirements on Fast Message Authentication Codes
draft-irtf-cfrg-fast-mac-requirements-01

Status of this Memo

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

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

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

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

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

This Internet-Draft will expire on August 25, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Abstract

This memo outlines requirements guiding the development of a fast Message Authentication Code (MAC) algorithm suitable for use with many IETF protocols, but in particular routing protocols that use a MAC for packet authentication.

It is widely conjectured that secure MACs that are fast in software are possible, and many interesting MACs have appeared in the literature. Nevertheless, none of these MACs have seen broad
adoption, and none are a good match for the routing protocol case. We hope that this memo brings together MAC designers and MAC users for a fruitful discussion.

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 RFC 2119 [RFC2119].

Table of Contents

1. Introduction .................................................. 3
2. Interface ..................................................... 4
3. Throughput ..................................................... 5
4. Security ....................................................... 5
5. Progressive Verification ....................................... 6
6. Example Usage: Routing Authentication ........................ 6
7. Requirements for a fast Software MAC ............................ 7
8. Recent Advances in fast MAC algorithms .......................... 8
9. IANA Considerations ............................................ 9
10. Security Considerations ......................................... 9
11. Acknowledgements ............................................... 9
12. References ..................................................... 9
12.1. Normative References ....................................... 9
12.2. Informative References ..................................... 9
Authors’ Addresses ............................................... 11
Intellectual Property and Copyright Statements .................... 13
1. Introduction

There are a number of Message Authentication Codes (MACs) that are relatively fast, but these MACs do not meet all of the requirements of some applications. This memo outlines desirable attributes for a fast MAC, and describes a particularly demanding application in which no current MAC appears to be satisfactory. These desirable attributes are given as requirements. We invite comments on both the requirements and their relative priority.

We first review existing MACs. Unfortunately, some MACs are still in use even though they have known security deficiencies. The "TCP MD5 Signature Option" [RFC2385] construction is in this category.

HMAC [RFC2104] constructs a MAC out of a collision-resistant hash function, such as SHA-1. HMAC is a pseudorandom function, so it is also suitable for other purposes such as key derivation. It has proven popular, and it is in widespread use, though its design is not ideal for high-speed hardware, its performance is worse on short messages, and many cryptographers believe that more efficient designs are possible. Additionally, HMAC performance suffers when its underlying hash function changes to a slower hash function, so the current interest in replacing HMAC-MD5 and HMAC-SHA-1 with HMAC-SHA-256 and HMAC-SHA-512 is likely to motivate the consideration of alternative MACs.

There are many designs for MACs that use the Wegman-Carter "universal hashing" paradigm [WC81]. Informally speaking, this method minimizes computational cost by doing the bulk of the message-processing using functions that have good combinatorial properties. LFSR-hash [LFSR] is a hardware-oriented Wegman-Carter style MAC; most designs since then have been software-oriented, including UMAC [RFC4418], POLY1305 [POLY1305], VMAC [I-D.krovetz-vmac].

A number of block cipher modes of operation act as MACs, including CMAC [FIPS.800-38B], which is the modern version of CBC-MAC. Because the block cipher invocations in this MAC cannot be pipelined or parallelized, some other MAC modes have appeared, including XORMAC [XORMAC] and GMAC [FIPS.800-38D], the latter of which is a Wegman-Carter style MAC.

Many of the more recent MAC designs have MAC-generation algorithms that require that a nonce is provided as input, including LFSR-hash, UMAC (though an early version did not), POLY1305, GMAC, and VMAC. In this regard, they differ from HMAC.
2. Interface

It is desirable for a MAC to support the simplest possible interface, viz., one involving a message, a key, and a MAC value, and no other parameters. A new MAC that uses this interface could be used as a drop-in replacement for HMAC and "MD5 signature" (used by the TCP MD5 Signature Option [RFC2385]). Some MACs with impressive performance have not seen broad adoption. We suspect that one of the reasons for this is their use of nonces, which complicates their use as replacements for MACs with simpler interfaces. It is worth noting that a MAC with the simple interface can be built out of a MAC that requires a nonce, by incorporating nonce generation into the MAC-generation algorithm, and carrying the nonce within the MAC itself. This may prove fruitful in some cases, though it does have the undesirable property that it expands the size of the MAC. However, many current applications expect that the MAC verification procedure consists of computing the MAC of a message, then comparing that with a MAC provided with the message. That is, the applications themselves include a comparison for equality, rather than calling a MAC-verification algorithm. The inclusion of a nonce within the MAC value violates this assumption, and requires that the application be adapted to the new MAC. Alternatively, the nonce could be considered implicit state not distributed with the message. However, an implicit nonce may not be reliable for all applications. For example, if the nonce is incremented for each message, then messages must be processed strictly in order.

In applications in which there is no key management protocol available, it is particularly undesirable to use a MAC that requires that each invocation of the MAC generation algorithm to use a distinct nonce value. This is because, in many situations, it is difficult to maintain the state needed to generate and maintain unique nonces. It may be acceptable in these applications to use a random nonce. In some other applications, it is desirable to minimize the number of bytes required by a MAC. In these cases, it is unappealing to incorporate a nonce into the MAC.

In most applications of message authentication, there is a need for anti-replay protection. The data used to provide this security service may be used directly as a nonce. However, not all applications fall into this category. In particular, environments in which there are multiple senders that share a single MAC key may have trouble coordinating nonce values to ensure their uniqueness. The methods that are used to coordinate anti-replay information may be insufficiently secured from the point of view of nonce management. In these environments MACs that do not require nonces are desirable.

Another useful property for MACs is incrementality [BGG95]; i.e., the
ability to take a MAC computed for a given message and to generate a MAC for a slightly different message more efficiently than just running the MAC-generation algorithm on the new message. XORMAC can be used as an incremental MAC, as can GMAC [GMAC-INCR].

3. Throughput

In most applications it is desirable to minimize the computational cost of a MAC. Ideally, performance should be good across all implementation environments: hardware circuits, field-programmable gate arrays, and software implementations on all sorts of processors. This goal is hard to meet, especially considering the diversity of processors. Modern high-end processors typically support operations on 64-bit data values with multiple pipelines, and sometimes provide circuits for performing special operations in parallel (usually ones related to video processing). Often multiple processors work together in parallel. However, many software environments lack this sort of high-performance hardware support. Processors that operate on 32 bits at a time, or even 8 bits at a time, are still common. Some environments lack support for floating-point operations. Also, the speed and size of random access memory varies considerably in different environments.

It seems to be difficult to design a MAC that provides excellent performance in both hardware and software. The Galois/Message Authentication Code (GMAC) is efficient in pipelined hardware, and provides good performance in software, though only by trading off some storage for better performance. Nonetheless, it may be possible to provide considerably better performance in some software environments by taking advantage of fast integer multiplications. Clearly it is best for a MAC to perform well in all implementation environments. However, we believe that a message authentication code that performs exceptionally well in software, but only adequately well in hardware, would prove useful. Ideally, the computational cost of evaluating a MAC should be proportional to the length of the message, so that the cost of processing ten 16 byte messages is equal to the cost of processing a single 160 byte message, for example.

4. Security

Some MACs offer a performance-security tradeoff, for example by having a variable output size. It is an open question as to when it is acceptable to use relatively low security levels. We guess that 32-bit and 64-bit MACs are acceptable in some situations, though such MACs could probably have relatively limited applicability. If a short MAC is used, the MAC should retain a high level of security.
even if an adversary is able to craft a successful forgery. More specifically, the attacker’s probability of successfully forging a message should not significantly change after the attacker succeeds in forging one or more messages. This property is needed to ensure that an n-bit MAC can be safely used to process a number of messages close to $2^n$ (or perhaps greater than that number). MACs that use the LFSR-hash approach, such as GMAC, UMAC, and VMAC, do not have this property. In applications that lack dedicated key management, it is desirable that a MAC provide a high security level even after it processes a very large amount of data.

5. Progressive Verification

Some devices (e.g., sensors) would benefit from the ability to dynamically trade-off throughput and security. Progressive Verification is the idea that the verifier of a MAC could choose the security level of a validation as part of such a performance-security tradeoff [F03]. The ACSA Project included progressive verification using multiple authentication tags [ABCHS00] where, with little overhead, the sender can compute two or more authentication tags with different strength performance levels for the same message packet. For example, a device under DoS attack could choose low-security verification as a means of reducing CPU utilization. A low-security verification could be used as a "filtering" mechanism, to be followed by a high-security check. One motivation for the evolution of UMAC was to allow a receiver to verify a tag to various forgery probabilities (as described in Section 9 of [BHKKR99]). This has the interesting property that receivers of different security policies or computational capabilities can perform a verification using a single tag. Note that a MAC progressive verification is distinct from an incremental MAC, as described in Section 2.

6. Example Usage: Routing Authentication

Routing protocols typically use a MAC to provide message integrity. One common method is the TCP MD5 Signature Option[RFC2385], which uses "MD5 Signature"; standards work is underway with the goal of replacing this option with a more secure and easier to use option [I-D.bellovin-tcpsec] [I-D.ietf-tcpm-tcp-auth-opt]. MACs currently in use are based on "MD5 Signature", HMAC-SHA1 or HMAC-MD5. Some modes of AES (e.g., AES-CMAC) could also be used. However, all of these algorithms are computationally expensive in software, which limits the number of secure connections that a particular router can support. In some cases, the MAC verification process itself can be used by an attacker to propagate a Denial of Service (DoS) attack. A strong MAC algorithm that requires less computation by a router is
Routers often implement routing protocols in software in such a way that hardware acceleration of a MAC algorithm is not optimal. In some cases (e.g., TCP), efficient processing of the option containing the MAC requires that it be done on a route processor rather than in an ASIC or FPGA. As a consequence, routers often implement the MAC algorithm in software, resulting in the over-subscription of the CPU and the filling of the input queues to the route processor. A faster MAC algorithm could greatly increase the number of secure connections to routing peers that a router can support. A faster MAC algorithm not requiring a nonce would use fewer protocol header bytes, which leaves more space for routing protocol data to be included in each packet. Such a method should be suitable for use as a TCP option, as well as the OSPF, EIGRP, IS-IS, and other routing protocols. We estimate that a MAC algorithm suitable for use by routing protocols would be at least an order of magnitude faster than HMAC-MD5, and would not use more than 1K of memory per key (including internal key scheduling tables and/or expanded key space).

7. Requirements for a fast Software MAC

R1. The MAC MUST have a simple interface; in particular, it SHOULD NOT require a nonce.

R2. The MAC SHOULD provide throughput at least five times that of HMAC-MD5 in common software environments.

R3. The MAC SHOULD be at least as efficient as HMAC-MD5 in common hardware environments.

R4. The MAC SHOULD be compact; that is, it should have a forgery probability close to $1/2^n$, where $n$ is the number of bits in the MAC.

R5. The MAC MAY provide a security/performance tradeoff.

R6. If the MAC can have a short output value, then MAC SHOULD be difficult to forge even after an adversary has succeeded in perpetrating one or more forgeries.

R7. Efficient software implementations of the MAC SHOULD NOT require more than one kilobyte of RAM for each key that is in use.

R8. The MAC SHOULD include "progressive verification"
R9. A MAC that uses a nonce SHOULD maintain some reasonable security level even if nonces are repeated, non-random, and/or chosen by an adversary. If the MAC requires that distinct nonces be provided to its MAC-generation algorithm, it MUST describe the loss of security that entails when this requirement is not met. The loss of security SHOULD be minimal.

R10. The MAC SHOULD support inputs up to at least $2^{48}$-1 bytes

R11. The MAC SHOULD be able to protect up to $2^{48}$ messages, or a total of $2^{64}$ bytes, with a single key.

R12. The MAC SHOULD support tags of 96 and/or 128 bits, and MAY support 32, 64, 192, or 256 bits.

Of the MACs described in this memo, the only ones that do not require a nonce (R1) are HMAC and CMAC, neither of which meet the aggressive performance goals that we identified (R2 and R3). None of the fastest MACs (UMAC, POLY1305-AES[GH05], VMAC, GMAC) meet the goal that repeat forgeries are unlikely when the MAC is short (R6).

8. Recent Advances in fast MAC algorithms

Several exciting new results have appeared in pre-publication form since the initial version of this draft was published.

Dan Bernstein has found a way to halve the number of multiplication operations needed in the evaluation of a polynomial hash, while using the same amount of memory [B07]. Roughly speaking, his observation makes it possible to double the speed of some MACs, making it easier to simultaneously satisfy requirements R2, R3, and R7. However, the new method requires the multiplication of arbitrary values, and thus it seems incompatible with the implementation strategy of using precomputation with a fixed multiplicand, as is sometimes used with Horner’s method evaluations of polynomial hash functions. (This strategy is used in many software implementations of GMAC, for example.)

John Black and Martin Cochran show that many MACs fail to meet requirement R6 (because there exists an attack that allows efficient forgeries after the first one is obtained), and they developed a MAC that meets this requirement [BC07]. It requires a nonce, and thus does not meet requirement R1, but it should be noted that it resists nonce misuse, and thus addresses some of the concerns behind that stipulation. It appears that it meets the other functional requirements, though, including R8.
9. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

10. Security Considerations

This memo describes requirements for the development of a cryptographic algorithm. Security considerations are included throughout the memo.

11. Acknowledgements

The authors wish to thank Ran Canetti and Jack Lloyd for their insightful comments.

12. References

12.1. Normative References


12.2. Informative References


[BGG95] Bellare, M., Goldreich, O., and S. Goldwasser, "Incremental cryptography and application to virus


(work in progress), November 2007.

[I-D.krovetz-vmac]

[LFSR]

[POLY1305]

[RFC2104]

[RFC2385]

[RFC4418]

[WC81]

[XORMAC]
Authors’ Addresses

David A. McGrew
Cisco Systems
170 W. Tasman Drive
San Jose, California 95134-1706
USA

Phone: +1 408 525 8651
Fax:
Email: mcgrew@cisco.com
URI:

Brian Weis
Cisco Systems
170 W. Tasman Drive
San Jose, California 95134-1706
USA

Phone: +1 408 526 4796
Fax:
Email: bew@cisco.com
URI: