Internet Engineering Task Force A. Malhotra
Internet-Draft S. Goldberg
Intended status: Standards Track Boston University
Expires: April 28, 2017 October 25, 2016

Message Authentication Codes for the Network Time Protocol
draft-aanchal4-ntp-mac-02

Abstract

The Network Time Protocol (NTP) RFC 5905 [RFC5905] uses a message authentication code (MAC) to cryptographically authenticate its UDP packets. Currently, NTP packets are authenticated by appending a 128-bit key to the NTP data, and hashing the result with MD5 to obtain a 128-bit tag. However, as discussed in [BCK] and [RFC6151], this is not a secure MAC. As such, this draft considers different secure MAC algorithms for use with NTP, evaluates their performance, and recommends the use of CMAC-AES [RFC4493]. We also suggest deprecating the use of MD5 as defined in [RFC5905] for authenticating NTP packets.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

NTP uses a message authentication code (MAC) to authenticate its packets. Currently, NTP packets are authenticated by appending a 128-bit key to the NTP data, and hashing the result with MD5 to obtain a 128-bit tag. However, as discussed in [BCK] and [RFC6151], this not a secure MAC. As such, this draft considers different secure MAC algorithms for use with NTP, evaluates their performance, and recommends the use of CMAC-AES [RFC4493]. We also suggest deprecating the use of MD5, as defined in [RFC5905], for authenticating NTP packets.

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 RFC 2119 [RFC2119].

2. MAC Algorithms

We consider five diverse MAC algorithms, which encompass hash-based HMAC-MD5 and HMAC-SHA224 [RFC2104], block cipher-based CMAC-AES [RFC4493], and universal hashing-based Galois MAC (GMAC) [RFC4543] and Poly1305(ChaCha20) as in section 2.6 of [RFC7539]. For completeness we also benchmark the legacy MD5(key||message) from [RFC5905].

Algorithm	Input Key Length (Bytes)	Output Tag Length (Bytes)
legacy MD5	16	16
HMAC-MD5	16	16
HMAC-SHA224	16	16
CMAC(AES)	16	16
GMAC(AES)	16	16
Poly1305(ChaCha20)	32	16

The choice of algorithms evaluated here is motivated, in part, by standardization and availablity of their open source implementations. All algorithms we consider, other than the plain MD5, are standardized. Four out of five algorithms are at least available in the OpenSSL library, while Poly1305(ChaCha20) is implemented in LibreSSL (a fork of OpenSSL) and also in BoringSSL (Google's implementation of OpenSSL).

The output tag length for HMAC-SHA224 is 28 bytes, but we truncate it to 16 bytes as in section 4 of [RFC7630] to fit into the NTP packet. As noted in section 6 of [RFC2104] it is safe to truncate the output of MACs as long as the truncated length is greater than 80-bits and not less than half the length of the hash output.

3. Requirements

3.1. Performance Requirements

In order to accurately compute the time, NTP ideally requires MAC algorithms to have a constant computational latency. However, this is generally not possible, since latency depends on the CPU load, temperature, and other uncontrollable factors. Instead, a MAC algorithm that requires fewer clock cycles for computation is prefered over one that requires more clock cycles, as this directly translates to a reduction in jitter (i.e., the variance of the latency for computing the MAC).

Throughput is another important consideration. NTP servers may have to deal with thousands of client requests per second. A study [NIST] on the usage analysis of NIST's NTP stratum 1 servers shows that these servers cater to 28,000 requests/second on an average, per server.

Most of the Internet is served by stratum 2 and stratum 3 servers, some of which are a part of voluntary NTP pool. These machines may be running old hardware. Generally, while benchmarking MAC algorithms, several optimization techniques on custom specialized hardware are used to get the best results. However, for the reason stated above we choose to benchmark performance on a range of software and hardware platforms with and without optimizations.

3.2. Security Requirements

There are several more constraints specific to NTP that need to be taken into account.

1. NTP servers are stateless, i.e. they do not keep per client state.

2. Per [RFC5905], NTP uses a pre-shared symmetric key. This makes key management difficult because there is no in-band mechanism for distributing keys. As such, to simplify key management, some deployments use the same pre-shared key at many servers (typically at the same stratum). In other words, the same key is used for several client/server associations.

3. [RFC5905] also has no in-band mechanism to refresh keys.

4. Performance Results

The NTP header is 48 bytes long. We therefore consider the latency and throughput for several secure MAC algorithms when computed over 48-byte messages.

We customize the in-built speed utility of OpenSSL-1.0.2g (03 May 2016) version to compute the latency and throughput for each MAC as shown in the tables below. OpenSSL, however, does not implement stream-cipher ChaCha20-based Poly1305 MAC algorithm. To speed test this MAC, we use LibreSSL 2.3.1, a fork of OpenSSL implementation. OpenSSL and LibreSSL are the most widely used cryptographic libraries and are used by the current NTP implementations.

Since the introduction of New Instruction (NI) set for hardware support in Intel chips, certain MACs like CMAC and GMAC have performance advantage on such machines. Based on this, we perform

two different benchmarks: one with AES-NI enabled and the other with it disabled. Benchmarks were taken on an x86_64, Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz with one core CPU.

This table shows throughput in terms of number of 48-byte NTP payload processed per second.

Algorithm	with AES-NI	without AES-NI
legacy MD5	3118K	3165K
HMAC-MD5	2742K	2749K
HMAC-SHA224	1265K	1267K
CMAC(AES)	7567K	4388K
GMAC(AES)	16612K	4627K
Poly1305(ChaCha20)	2598K	2398K

This table shows latency in terms of number of CPU cycles per byte (cpb) when processing a 48-byte NTP payload.

Algorithm	with AES-NI	without AES-NI
legacy MD5	16.0	15.7
HMAC-MD5	18.2	18.1
HMAC-SHA224	39.4	39.0
CMAC(AES)	6.6	11.3
GMAC(AES)	3.0	10.8
Poly1305(ChaCha20)	14.4	15.0

5. Other Hardware Platforms

We also perform tests on the following ARM CPU cores and PowerPC(PPC)core with OpenSSL 1.0.2h released May 2016. These cores are most commonly used in consumer products and Industrial Control Systems (ICS) components. We select these cores to cover the ARM architecture versions 5/6 to 8. The results vary depending on the availability of CPU specific optimizations. For example the used Cortex-A9 and Cortex-A53 CPU have a NEON unit and OpenSSL can utilize it to accelerate AES.

1. Freescale/Apple PPC74xx 1.5GHz

2. NXP i.MX6 1GHz (dual core) ARM Cortex-A9

3. Broadcom BCM2837 1.2GHz (quad core) ARM Cortex-A53

4. Marvell 88F6281 1.2GHz 88FR131 (ARMv5te compliant)

The table below shows throughput in terms of number of 48-byte NTP payload processed per second.

Algorithm	PPC74xx	ARM Cortex-A9	ARM Cortex A-53	Marvell
legacy MD5	600K	543K	748K	383k
HMAC-MD5	463K	415K	864K	438k
HMAC-SHA224	276K	245K	357K	150k
CMAC(AES)	576K	412K	614K	246k
GMAC(AES)	681K	1362K	2193K	453k
Poly1305(ChaCha20)	335K	379K	580K	273k

The table below shows latency in terms of number of CPU cycles per byte (cpb) when processing a 48-byte NTP payload.

Algorithm	PPC74xx	ARM Cortex-A9	ARM Cortex A-53	Marvell
legacy MD5	52.1	38.4	33.4	65.3
HMAC-MD5	67.4	50.3	29.0	57.1
HMAC-SHA224	113.3	85.2	70.1	166.5
CMAC(AES)	54.2	50.5	40.7	101.2
GMAC(AES)	50.0	15.3	11.4	55.1
Poly1305(ChaCha20)	93.1	55.0	43.1	91.7

6. Security Considerations

The MD5 (key||message) "message authentication code" specified in [RFC5905] is vulnerable to length extension attacks, and uses the insecure MD5 hash function, and therefore MUST be deprecated.

Therefore, we consider hash-based MACs (HMAC-MD5, HMAC-SHA224), and cipher-based MACs (CMAC-AES, Poly1305 (ChaCha20)). The upper bound on the security level provided by any MAC against brute-force attacks is min (key-length, tag-length). The security of these MACs can be worse but not better than this bound. All MAC algorithms we consider have comparable key-lengths and output tag-lengths. So the advantage of an adversary that wishes to forge a MAC is lower-bounded by 1/2^{128}.

Assume that an adversary can obtain a valid MAC for q distinct messages. Then the table below describes the advantage of an adversary that wishes to forge a MAC in terms of number of queries (q) it launches.

Algorithm	Advantage
HMAC-MD5 [MB]	q^2/2^{128}
HMAC-SHA224 [BCK]	q^2/2^{224}
CMAC(AES)[IK]	q^2/2^{128}
GMAC(AES) [IOM]	q^2/2^{128}
Poly1305(ChaCha20) [DJB]	{e^{{q^2}/{2^{129}}}}/2^{103}

Poly1305 can easily handle up to q=2^{64} but security degrades pretty rapidly after that.

However, the bounds in the table above are somewhat optimistic, for the following reasons.

1. GMAC has an initialization vector (IV) that [RFC4106] allows to be 1 <= len(IV) <= 2^{64}-1. Per [RFC4106], implementations are optimized to handle a 12-octet IV. With a 12-octet IV, the total number of message invocations is bound to 2^{48}. Moreover, if the IV is reused even once (for the same secret authentication key and different input messages), then [Joux] shows that the secret authetication key can easily be recovered by the adversary. Notice that this attack is even stronger than a message forgery because it recovers the authentication key. This is known as nonce-reuse vulnerability.

2. The other three algorithms evaluated here do not suffer from nonce reuse vulnerabilities where an adversary can recover the authentication key if the nonce is reused just once.

3. The table above suggests that for CMAC, the total number of invocations of the MAC is limited to 2^{64}. However, [NIST-CMAC] recommends, to be on the safe side, that the total number of invocations of the block cipher algorithm during the lifetime of the key is limited to 2^{48}.

6.1. Why is GMAC not suitable for NTP?

[Joux] showed that for GMAC-AES, if the IV is repeated just once, then the authentication key can be fully recovered. None of the other algorithms evaluated here have this vulnerability. Thus, for

GMAC-AES to be secure, we need to make sure that IV is never repeated.

[NIST-GMAC] recommends constructing the 12-byte IV used in GMAC by concatenating a fixed 4-byte salt value concatenate with a variable 8-byte nonce i.e. IV = (salt|| nonce). Here salt is an implicit value established when a session is established, remains fixed for all exchanges in a session (i.e. for all invocations that use the same authenication key) between the sender and the receiver. Meanwhile, the nonce is freshly generated for each authenticated message.

Because NTP servers do not keep per-client state, the nonce can not be a sequential value. Instead, this nonce must be randomly generated 8-bytes value chosen freshly for each authenticated message. According to birthday bound, the nonce value will be repeated, with high probability, after 2^{32} messages sent in a given association. This leads to a repeated IV value and to [Joux]'s attack. Thus, to prevent repeated nonces, we would need to require the authentication key to be refreshed for the association after 2^{32} messages.

On one hand, 2^{32} is a lot of queries for an honest client, assuming that the client queries once per minute (which is NTP's minimum polling interval [RFC5905]). On the other hand, a man-in-the-middle (MiTM) can quickly and easily exhaust this number by replaying old authenticated queries to the NTP server.

The main problem here is that NTP lacks an explict in-band key refresh mechanism that can be invoked automatically (without operator intervention). And a key refresh mechanism is unlikely to be adopted as it would allow denial-of-service (DoS) attacks. The state less nature makes NTP resilient against DoS attacks.

Even if there was a method by which key-refresh could be performed, there is an additional problem. An NTP server does not keep per-client state. Therefore, it cannot keep track of the number of messages it sent in a given association. One idea is to have the client keep this state, and then send an authenicated request for a key refresh. However, a man-in-the-middle could replay old authenticated queries to the NTP server, and then intercept the server's' response before they reach the legitimate client. In this case, the client would never know when to ask for a key refresh.

Alternatively, the server could maintain a global counter (since it can't afford to keep per client counter). And after 2^{32} messages, it can refresh the keys with all its clients. However, a man-in-the-

middle could exhaust this number quickly and the server will have to refresh keys with all the clients very frequently.

Thus, we conclude that a scheme that requires refreshing the key after 2^{32} client queries is not a good idea at all.

Even in the absence of a man-in-the-middle, there is also the problem of multiple servers using the same authentication key. The salt could be used to distinguish IVs across different client/server associations that use the same authenication key. However, this brings us back to the original key management problem. One way to deal with this is to choose the 4-byte salt at random. However, this gives rise to a birthday bound of 2^{16} = 65,000 unique IVs. If we consider 20,000 stratum 3 clients synchronizing to three stratum 2 servers each, all of which are in the same organization and share the same symmetric key, we get very close to the birthday bound. This is another disadvantage of using GMAC with NTP.

7. Use HMAC or CMAC instead

1. CMAC seems to be the next best choice. Leaving out GMAC, it has the best performance with and without hardware support. It is not vulnerable to nonce misuse issues.

2. HMACs are inherently slower because of their structure and also in some cases because of lack of built-in hardware support.

3. On the other hand, it is much easier to get the right implementation for HMAC compared to CMAC.

8. GMAC-SIV - Another Potential MAC Candidate

GMAC-SIV is another possible MAC candidate, which claims to be nonce-misuse resistant [SIV]. There is an IETF Internet draft for the standardization of GCM-SIV AEAD mode.

In terms of security, GCM-SIV (AEAD) achieves usual notion of nonce-based security of an authenticated encryption mode as long as a unique nonce is used per authentication key per message. If, however, the nonce is reused authenticity is still retained (unlike in GMAC).

But there is not many implementations for GCM-SIV available except for the one from the authors. We customized this code for authentication only mode GMAC-SIV and run it on an x86_64, Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz with one core CPU with AES-NI enabled. GMAC-SIV takes ~5.9 CPU cycles/byte to generate a tag of length 16 bytes on a 48-byte NTP payload. The performance efficiency

is far less than GMAC, but is slightly better than CMAC. CMAC, on the other hand is a standardized mode of operation and has several open source implementations.

9. Recommendations

From the tables we clearly see that GMAC(AES) has the best latency and throughput performance in both hardware and software implementations. It is freely available, and there is a flexibilty of changing the underlying block-cipher. However there are several security problems surrounding the use of this mode, as highlighted above, so it is not recommended.

CMAC, on the other hand, is the next best choice in terms of performance and security. So we recommend the use of CMAC (AES).

10. Acknowledgements

The authors wish to acknowledge useful discussions with Leen Alshenibr, Daniel Franke, Ethan Heilman, Kenny Paterson, Leonid Reyzin, Harlan Stenn, Mayank Varia.

11. References

11.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[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>.

[RFC4106] Viega, J. and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)", RFC 4106, DOI 10.17487/RFC4106, June 2005, <http://www.rfc-editor.org/info/rfc4106>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <http://www.rfc-editor.org/info/rfc4493>.

[RFC4543] McGrew, D. and J. Viega, "The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH", RFC 4543, DOI 10.17487/RFC4543, May 2006, <http://www.rfc-editor.org/info/rfc4543>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <http://www.rfc-editor.org/info/rfc6151>.

[RFC7539] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015, <http://www.rfc-editor.org/info/rfc7539>.

[RFC7630] Merkle, J., Ed. and M. Lochter, "HMAC-SHA-2 Authentication Protocols in the User-based Security Model (USM) for SNMPv3", RFC 7630, DOI 10.17487/RFC7630, October 2015, <http://www.rfc-editor.org/info/rfc7630>.

11.2. Informative References

[BCK] Bellare, M., Canetti, R., and H. Krawczyk, "Keyed Hash Functions and Message Authentication", in Proceedings of Crypto'96, 1996.

[DJB] Bernstein, D., "The Poly1305-AES message-authentication code", in Fast Software Encryption, 2005.

[GK] Gueron, S. and V. Krasnov, "The fragility of AES-GCM authentication algorithm", in Proceedings of 11th International Conference on Information Technology: New Generations 2014, 2014.

[IK] Iwata, T. and K. Kurosawa, "Keyed Hash Functions and Message Authentication", in Progress in Cryptology-INDOCRYPT 2003, 2003.

[IOM] Iwata, T., Ohashi, K., and K. Minematsu, "Breaking and Repairing GCM Security Proofs", in Proceedings of CRYPTO 2012, 2012.

[Joux] Joux, A., "Authentication Failures in NIST version of GCM", <http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/GCM/Joux_comments.pdf>.

[MB] Bellare, M., "New Proofs for NMAC and HMAC:Security without Collision-Resistance", in Proceedings of Crypto'96, 1996.

[NIST] Sherman, J. and J. Levine, "Usage Analysis of the NIST Internet Time Service", in Journal of Research of the National Institute of Standards and Technology, 2016.

[NIST-CMAC] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication", in NIST Special Publication 800-38B, 2005.

[NIST-GMAC] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", in NIST Special Publication 800-38D, 2007.

[SIV] Gueron, S., Langley, A., and Y. Lindell, "https://tools.ietf.org/html/draft-gueron-gcmsiv-00", in Work in Progress, 2016.

Authors' Addresses

Aanchal Malhotra
Boston University
111 Cummington St
Boston, MA 02215
US

Email: aanchal4@bu.edu

Sharon Goldberg
Boston University
111 Cummington St
Boston, MA 02215
US

Email: goldbe@cs.bu.edu