Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2017

A. Malhotra
S. Goldberg
Boston University
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.

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

Network Working Group
Internet-Draft
Updates: 5905 (if approved)
Intended status: Standards Track
Expires: September 28, 2017

D. Franke
Akamai
A. Malhotra
Boston University

March 27, 2017

NTP Client Data Minimization
draft-dfranke-ntp-data-minimization-02

Abstract

This memo proposes backward-compatible updates to the Network Time Protocol to strip unnecessary identifying information from client requests and to improve resilience against blind spoofing of unauthenticated server responses.

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 September 28, 2017.

Table of Contents

1. Introduction

Network Time Protocol (NTP) packets, as specified by RFC 5905 [RFC5905], carry a great deal of information about the state of the NTP daemon which transmitted them. In the case of mode 4 packets (responses sent from server to client), as well as in broadcast (mode 5) and symmetric peering modes (mode 1/2), most of this information is essential for accurate and reliable time synchronizaton. However, in mode 3 packets (requests sent from client to server), most of these fields serve no purpose. Server implementations never need to inspect them, and they can achieve nothing by doing so. Populating these fields with accurate information is harmful to privacy of clients because it allows a passive observer to fingerprint clients and track them as they move across networks.

This memo updates RFC 5905 to redact unnecessary data from mode 3 packets. This is a fully backwards-compatible proposal. It calls for no changes on the server side, and clients which implement these updates will remain fully interoperable with existing servers.

2. 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].

3. Client Packet Format

In every client-mode packet sent by a Network Time Protocol [RFC5905] implementation:

The first octet, which contains the leap indicator, version number, and mode fields, SHOULD be set to 0x23 (LI = 0, VN = 4, Mode = 3).

The Transmit Timestamp field SHOULD be set uniformly at random, generated by a mechanism suitable for cryptographic purposes. [RFC4086] provides guidance on the generation of random values.

The Poll field MAY be set to the actual polling interval as specified by RFC 5905, or else MAY be set to zero.

All other header fields, specifically the Stratum, Precision, Root Delay, Root Dispersion, Reference ID, Reference Timestamp, Origin Timestamp, and Receive Timestamp, SHOULD be set to zero.

Servers MUST allow client packets to conform to the above recommendations. This requirement shall not be construed so as to prohibit servers from rejecting conforming packets for unrelated reasons, such as access control or rate limiting.

4. Security and Privacy Considerations

4.1. Data Minimization

Zeroing out unused fields in client requests prevents disclosure of information that can be used for fingerprinting [RFC6973].

While populating any of these fields with authentic data reveals at least some identifying information about the client, the Origin Timestamp and Receive Timestamp fields constitute a particularly severe information leak. RFC 5905 calls for clients to copy the transmit timestamp and destination timestamp of the server's most recent response into the origin timestamp and receive timestamp (respectively) of their next request to that server. Therefore, when a client moves between networks, a passive observer of both network paths can determine with high confidence that the old and new IP addresses belong to the same system by noticing that the transmit timestamp of a response sent to the old IP matches the origin timestamp of a request sent from the new one.

Zeroing the poll field is made optional (MAY rather than SHOULD) so as not to preclude future development of schemes wherein the server uses information about the client's current poll interval in order to recommend adjustments back to the client. Putting accurate information into this field has no significant impact on privacy since an observer can already obtain this information simply by observing the actual interval between requests.

4.2. Transmit Timestamp Randomization

While this memo calls for most fields in client packets to be set to zero, the transmit timestamp is randomized. This decision is motivated by security as well as privacy.

NTP servers copy the transmit timestamp from the client's request into the origin timestamp of the response; this memo calls for no change in this behavior. Clients discard any response whose origin timestamp does not match the transmit timestamp of any request currently in flight.

In the absence of cryptographic authentication, verification of origin timestamps is clients' primary defense against blind spoofing of NTP responses. It is therefore important that clients' transmit timestamps be unpredictable. Their role in this regard is closely analagous to that of TCP Initial Sequence Numbers [RFC6528].

The traditional behavior of the NTP reference implementation is to randomize only a few (typically 10-15 depending on the precision of the system clock) low-order bits of transmit timestamp, with all higher bits representing the system time, as measured just before the packet was sent. This is suboptimal, because with so few random bits, an adversary sending spoofed packets at high volume will have a good chance of correctly guessing a valid origin timestamp.

5. IANA Considerations

[RFC EDITOR: DELETE PRIOR TO PUBLICATION]

This memo introduces no new IANA considerations.

6. References

6.1. Normative References

[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>.

[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>.

6.2. Informative References

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February 2012, <http://www.rfc-editor.org/info/rfc6528>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <http://www.rfc-editor.org/info/rfc6973>.

Appendix A. Acknowledgements

The authors thank Prof. Sharon Goldberg and Miroslav Lichvar for calling attention to the issues addressed in this memo.

Authors' Addresses

Daniel Fox Franke
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA 02142
United States

Email: dafranke@akamai.com
URI: https://www.dfranke.us

Aanchal Malhotra
Boston University
111 Cummington St
Boston, MA 02215
United States

Email: aanchal4@bu.edu

Network Working Group
Internet-Draft
Intended status: Historic
Expires: April 20, 2017

D. Mills
University of Deleware
B. Haberman, Ed.
JHU
October 17, 2016

Control Messages Protocol for Use with Network Time Protocol Version 4
draft-haberman-ntpwg-mode-6-cmds-02

Abstract

This document describes the structure of the control messages used with the Network Time Protocol. These control messages can be used to monitor and control the Network Time Protocol application running on any IP network attached computer. The information in this document was originally described in Appendix B of RFC 1305. The goal of this document is to provide a historic description of the control messages.

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 April 20, 2017.

Table of Contents

1. Introduction

RFC 1305 [RFC1305] described a set of control messages for use within the Network Time Protocol (NTP) when a comprehensive network management solution was not available. The definitions of these control messages were not promulgated to RFC 5905 [RFC5905] when NTP version 4 was documented. These messages were intended for use only in systems where no other management facilities were available or appropriate, such as in dedicated-function bus peripherals. Support for these messages is not required in order to conform to RFC 5905 [RFC5905]. The control messages are described here as a historical record given their use within NTPv4.

1.1. Control Message Overview

The NTP Control Message has the value 6 specified in the mode field of the first octet of the NTP header and is formatted as shown in Figure 1. The format of the data field is specific to each command or response; however, in most cases the format is designed to be constructed and viewed by humans and so is coded in free-form ASCII. This facilitates the specification and implementation of simple management tools in the absence of fully evolved network-management facilities. As in ordinary NTP messages, the authenticator field follows the data field. If the authenticator is used the data field is zero-padded to a 32-bit boundary, but the padding bits are not considered part of the data field and are not included in the field count.

IP hosts are not required to reassemble datagrams larger than 576 octets; however, some commands or responses may involve more data than will fit into a single datagram. Accordingly, a simple reassembly feature is included in which each octet of the message data is numbered starting with zero. As each fragment is transmitted the number of its first octet is inserted in the offset field and the number of octets is inserted in the count field. The more-data (M) bit is set in all fragments except the last.

Most control functions involve sending a command and receiving a response, perhaps involving several fragments. The sender chooses a distinct, nonzero sequence number and sets the status field and R and E bits to zero. The responder interprets the opcode and additional information in the data field, updates the status field, sets the R bit to one and returns the three 32-bit words of the header along with additional information in the data field. In case of invalid message format or contents the responder inserts a code in the status field, sets the R and E bits to one and, optionally, inserts a diagnostic message in the data field.

Some commands read or write system variables and peer variables for an association identified in the command. Others read or write variables associated with a radio clock or other device directly connected to a source of primary synchronization information. To identify which type of variable and association a 16-bit association identifier is used. System variables are indicated by the identifier zero. As each association is mobilized a unique, nonzero identifier is created for it. These identifiers are used in a cyclic fashion, so that the chance of using an old identifier which matches a newly created association is remote. A management entity can request a list of current identifiers and subsequently use them to read and write variables for each association. An attempt to use an expired identifier results in an exception response, following which the list can be requested again.

Some exception events, such as when a peer becomes reachable or unreachable, occur spontaneously and are not necessarily associated with a command. An implementation may elect to save the event information for later retrieval or to send an asynchronous response (called a trap) or both. In case of a trap the IP address and port number is determined by a previous command and the sequence field is set as described below. Current status and summary information for the latest exception event is returned in all normal responses. Bits in the status field indicate whether an exception has occurred since the last response and whether more than one exception has occurred.

Commands need not necessarily be sent by an NTP peer, so ordinary access-control procedures may not apply; however, the optional mask/

match mechanism suggested elsewhere in this document provides the capability to control access by mode number, so this could be used to limit access for control messages (mode 6) to selected address ranges.

2. NTP Control Message Format

The format of the NTP Control Message header, which immediately follows the UDP header, is shown in Figure 1. Following is a description of its fields. Bit positions marked as zero are reserved and should always be transmitted as zero.

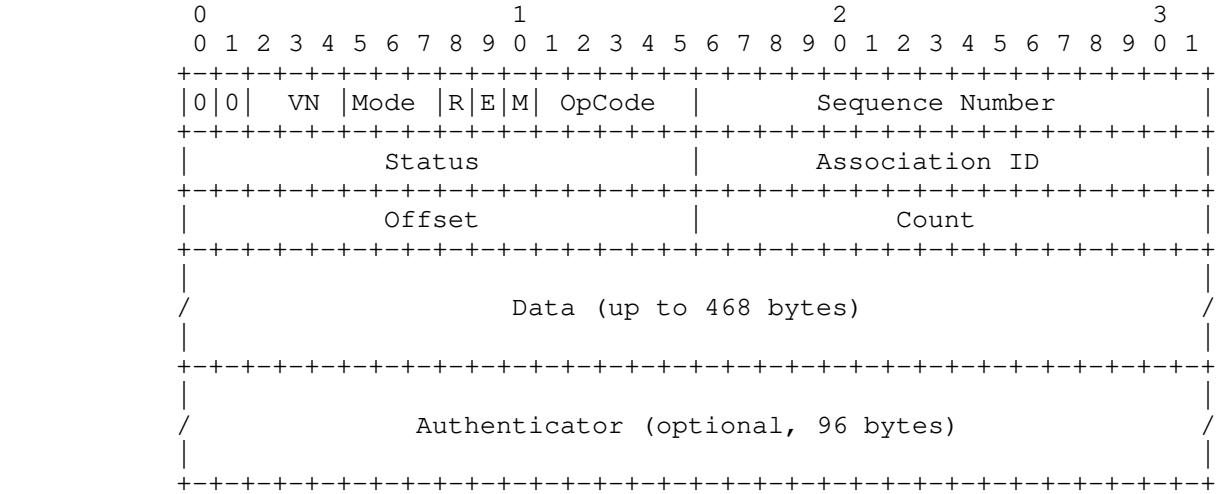


Figure 1: NTP Control Message Header

Version Number (VN): This is a three-bit integer indicating the NTP version number, currently four (4).

Mode: This is a three-bit integer indicating the mode. The value 6 indicates an NTP control message.

Response Bit (R): Set to zero for commands, one for responses.

Error Bit (E): Set to zero for normal response, one for error response.

More Bit (M): Set to zero for last fragment, one for all others.

Operation Code (OpCode): This is a five-bit integer specifying the command function. Values currently defined include the following:

Code	Meaning
0	reserved
1	read status command/response
2	read variables command/response
3	write variables command/response
4	read clock variables command/response
5	write clock variables command/response
6	set trap address/port command/response
7	trap response
8-31	reserved

Sequence Number: This is a 16-bit integer indicating the sequence number of the command or response.

Status: This is a 16-bit code indicating the current status of the system, peer or clock, with values coded as described in following sections.

Association ID: This is a 16-bit integer identifying a valid association.

Offset: This is a 16-bit integer indicating the offset, in octets, of the first octet in the data area.

Count: This is a 16-bit integer indicating the length of the data field, in octets.

Data: This contains the message data for the command or response. The maximum number of data octets is 468.

Authenticator (optional): When the NTP authentication mechanism is implemented, this contains the authenticator information defined in Appendix C of RFC 1305.

3. Status Words

Status words indicate the present status of the system, associations and clock. They are designed to be interpreted by network-monitoring programs and are in one of four 16-bit formats shown in Figure 2 and described in this section. System and peer status words are associated with responses for all commands except the read clock variables, write clock variables and set trap address/port commands. The association identifier zero specifies the system status word, while a nonzero identifier specifies a particular peer association. The status word returned in response to read clock variables and

write clock variables commands indicates the state of the clock hardware and decoding software. A special error status word is used to report malformed command fields or invalid values.

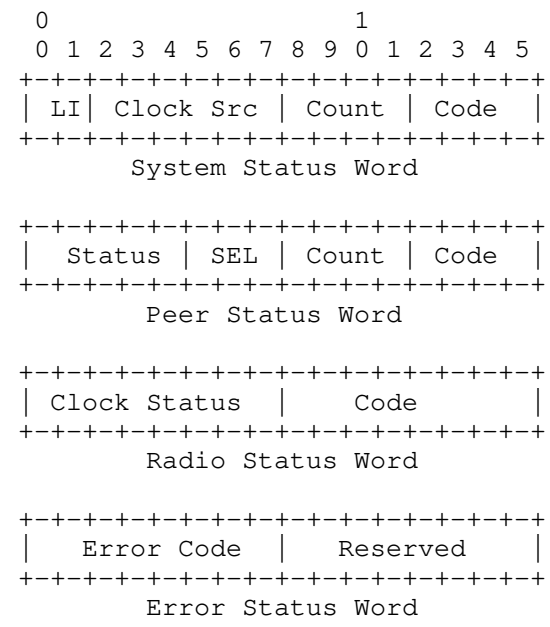


Figure 2: Status Word Formats

3.1. System Status Word

The system status word appears in the status field of the response to a read status or read variables command with a zero association identifier. The format of the system status word is as follows:

Leap Indicator (LI): This is a two-bit code warning of an impending leap second to be inserted/deleted in the last minute of the current day, with bit 0 and bit 1, respectively, coded as follows:

LI	Meaning
00	no warning
01	read status command/response
10	read variables command/response
11	write variables command/response

Clock Source (Clock Src): This is a six-bit integer indicating the current synchronization source, with values coded as follows:

Code	Meaning
0	unspecified or unknown
1	Calibrated atomic clock (e.g.,, HP 5061)
2	VLF (band 4) or LF (band 5) radio (e.g.,, OMEGA,, WWVB)
3	HF (band 7) radio (e.g.,, CHU,, MSF,, WWV/H)
4	UHF (band 9) satellite (e.g.,, GOES,, GPS)
5	local net (e.g.,, DCN,, TSP,, DTS)
6	UDP/NTP
7	UDP/TIME
8	eyeball-and-wristwatch
9	telephone modem (e.g.,, NIST)
10-63	reserved

System Event Counter (Count): This is a four-bit integer indicating the number of system exception events occurring since the last time the system status word was returned in a response or included in a trap message. The counter is cleared when returned in the status field of a response and freezes when it reaches the value 15.

System Event Code (Code): This is a four-bit integer identifying the latest system exception event, with new values overwriting previous values, and coded as follows:

Code	Meaning
0	unspecified
1	system restart
2	system or hardware fault
3	system new status word (leap bits or synchronization change)
4	system new synchronization source or stratum (sys.peer or sys.stratum change)
5	system clock reset (offset correction exceeds CLOCK.MAX)
6	system invalid time or date (see NTP specification)
7	system clock exception (see system clock status word)
8-15	reserved

3.2. Peer Status Word

A peer status word is returned in the status field of a response to a read status, read variables or write variables command and appears also in the list of association identifiers and status words returned by a read status command with a zero association identifier. The format of a peer status word is as follows:

Peer Status (Status): This is a five-bit code indicating the status of the peer determined by the packet procedure, with bits assigned as follows:

Peer Status	Meaning
0	configured (peer.config)
1	authentication enabled (peer.authenable)
2	authentication okay (peer.authentic)
3	reachability okay (peer.reach <F128M>?F255D> 0)
4	reserved

Peer Selection (SEL): This is a three-bit integer indicating the status of the peer determined by the clock-selection procedure, with values coded as follows:

Sel	Meaning
0	rejected
1	passed receive sanity checks
2	passed correctness check (intersection algorithm
3	passed candidate checks (if limit check implemented)
4	passed outlyer checks (cluster algorithm
5	current synchronization source; max distance exceeded (if limit check implemented)
6	current synchronization source; max distance okay
7	reserved

Peer Event Counter (Count): This is a four-bit integer indicating the number of peer exception events that occurred since the last time the peer status word was returned in a response or included in a trap message. The counter is cleared when returned in the status field of a response and freezes when it reaches the value 15.

Peer Event Code (Code): This is a four-bit integer identifying the latest peer exception event, with new values overwriting previous values, and coded as follows:

Peer Event Code	Meaning
0	unspecified
1	peer IP error
2	peer authentication failure (peer.authentic bit 1 --> 0
3	peer unreachable (peer.reach was nonzero now zero)
4	peer reachable (peer.reach was zero now nonzero)
5	peer clock exception (see peer clock status word)
6-15	reserved

3.3. Clock Status Word

There are two ways a reference clock can be attached to a NTP service host, as an dedicated device managed by the operating system and as a synthetic peer managed by NTP. As in the read status command, the association identifier is used to identify which one, zero for the system clock and nonzero for a peer clock. Only one system clock is supported by the protocol, although many peer clocks can be supported. A system or peer clock status word appears in the status field of the response to a read clock variables or write clock variables command. This word can be considered an extension of the system status word or the peer status word as appropriate. The format of the clock status word is as follows:

Clock Status: This is an eight-bit integer indicating the current clock status, with values coded as follows:

Clock Status	Meaning
0	clock operating within nominals
1	reply timeout
2	bad reply format
3	hardware or software fault
4	propagation failure
5	bad date format or value
6	bad time format or value
7-255	reserved

Clock Event Code (Code): This is an eight-bit integer identifying the latest clock exception event, with new values overwriting previous values. When a change to any nonzero value occurs in the radio status field, the radio status field is copied to the clock event code field and a system or peer clock exception event is declared as appropriate.

3.4. Error Status Word

An error status word is returned in the status field of an error response as the result of invalid message format or contents. Its presence is indicated when the E (error) bit is set along with the response (R) bit in the response. It consists of an eight-bit integer coded as follows:

Error Status	Meaning
0	unspecified
1	authentication failure
2	invalid message length or format
3	invalid opcode
4	unknown association identifier
5	unknown variable name
6	invalid variable value
7	administratively prohibited
8-255	reserved

4. Commands

Commands consist of the header and optional data field shown in Figure 2. When present, the data field contains a list of identifiers or assignments in the form <<identifier>>[=<<value>>],<<identifier>>[=<<value>>],... where <<identifier>> is the ASCII name of a system or peer variable specified in RFC 5905 and <<value>> is expressed as a decimal, hexadecimal or string constant in the syntax of the C programming language. Where no ambiguity exists, the <169>sys.<170> or <169>peer.<170> prefixes can be suppressed. Whitespace (ASCII nonprinting format effectors) can be added to improve readability for simple monitoring programs that do not reformat the data field. Internet addresses are represented as four octets in the form [n.n.n.n], where n is in decimal notation and the brackets are optional. Timestamps, including reference, originate, receive and transmit values, as well as the logical clock, are represented in units of seconds and fractions, preferably in hexadecimal notation, while delay, offset, dispersion and distance values are represented

in units of milliseconds and fractions, preferably in decimal notation. All other values are represented as-is, preferably in decimal notation.

Implementations may define variables other than those described in RFC 5905. Called extramural variables, these are distinguished by the inclusion of some character type other than alphanumeric or <169>.<170> in the name. For those commands that return a list of assignments in the response data field, if the command data field is empty, it is expected that all available variables defined in RFC 5905 will be included in the response. For the read commands, if the command data field is nonempty, an implementation may choose to process this field to individually select which variables are to be returned.

Commands are interpreted as follows:

Read Status (1): The command data field is empty or contains a list of identifiers separated by commas. The command operates in two ways depending on the value of the association identifier. If this identifier is nonzero, the response includes the peer identifier and status word. Optionally, the response data field may contain other information, such as described in the Read Variables command. If the association identifier is zero, the response includes the system identifier (0) and status word, while the data field contains a list of binary-coded pairs <<association identifier>> <<status word>>, one for each currently defined association.

Read Variables (2): The command data field is empty or contains a list of identifiers separated by commas. If the association identifier is nonzero, the response includes the requested peer identifier and status word, while the data field contains a list of peer variables and values as described above. If the association identifier is zero, the data field contains a list of system variables and values. If a peer has been selected as the synchronization source, the response includes the peer identifier and status word; otherwise, the response includes the system identifier (0) and status word.

Write Variables (3): The command data field contains a list of assignments as described above. The variables are updated as indicated. The response is as described for the Read Variables command.

Read Clock Variables (4): The command data field is empty or contains a list of identifiers separated by commas. The association identifier selects the system clock variables or peer clock variables in the same way as in the Read Variables command. The response

includes the requested clock identifier and status word and the data field contains a list of clock variables and values, including the last timecode message received from the clock.

Write Clock Variables (5): The command data field contains a list of assignments as described above. The clock variables are updated as indicated. The response is as described for the Read Clock Variables command.

Set Trap Address/Port (6): The command association identifier, status and data fields are ignored. The address and port number for subsequent trap messages are taken from the source address and port of the control message itself. The initial trap counter for trap response messages is taken from the sequence field of the command. The response association identifier, status and data fields are not significant. Implementations should include sanity timeouts which prevent trap transmissions if the monitoring program does not renew this information after a lengthy interval.

Trap Response (7): This message is sent when a system, peer or clock exception event occurs. The opcode field is 7 and the R bit is set. The trap counter is incremented by one for each trap sent and the sequence field set to that value. The trap message is sent using the IP address and port fields established by the set trap address/port command. If a system trap the association identifier field is set to zero and the status field contains the system status word. If a peer trap the association identifier field is set to that peer and the status field contains the peer status word. Optional ASCII-coded information can be included in the data field.

5. IANA Considerations

This document makes no request of IANA.

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

6. Security Considerations

A number of security vulnerabilities have been identified with these control messages.

NTP's control query interface allows reading and writing of system, peer, and clock variables remotely from arbitrary IP addresses using commands mentioned in Section 4. Traditionally, overwriting these variables, but not reading them, requires authentication by default. However, this document argues that an NTP host must authenticate all control queries and not just ones that overwrite these variables.

Alternatively, the host can use a whitelist to explicitly list IP addresses that are allowed to control query the clients. These access controls are required for the following reasons:

- NTP as a Distributed Denial-of-Service (DDoS) vector. NTP timing query and response packets (modes 1-2, 3-4, 5) are usually short in size. However, some NTP control queries generate a very long packet in response to a short query. As such, there is a history of use of NTP's control queries, which exhibit such behavior, to perform DDoS attacks. These off-path attacks exploit the large size of NTP control queries to cause UDP-based amplification attacks (e.g., mode 7 monlist command generates a very long packet in response to a small query (CVE-2013-5211)). These attacks only use NTP as a vector for DoS atacks on other protocols, but do not affect the time service on the NTP host itself.

- Time-shifting attacks through information leakage/overwriting. NTP hosts save important system and peer state variables. An off-path attacker who can read these variables remotely can leverage the information leaked by these control queries to perform time-shifting and DoS attacks on NTP clients. These attacks do affect time synchronization on the NTP hosts. For instance,

  * In the client/server mode, the client stores its local time when it sends the query to the server in its xmt peer variable. This variable is used to perform TEST2 to non-cryptographically authenticate the server, i.e., if the origin timestamp field in the corresponding server response packet matches the xmt peer variable, then the client accepts the packet. An off-path attacker, with the ability to read this variable can easily spoof server response packets for the client, which will pass TEST2, and can deny service or shift time on the NTP client. CVE-2015-8139 describes the specific attack.

  * The client also stores its local time when the server response is received in its rec peer variable. This variable is used for authentication in interleaved-pivot mode. An off-path attacker with the ability to read this state variable can easily shift time on the client by passing this test. CVE-2016-1548 describes the attack.

- Fast-Scanning. NTP mode 6 control messages are usually small UDP packets. Fast-scanning tools like ZMap can be used to spray the entire (potentially reachable) Internet with these messages within hours to identify vulnerable hosts. To make things worse, these attacks can be extremely low-rate, only requiring a control query for reconnaissance and a spoofed response to shift time on vulnerable clients. CVE-2016-1548 is one such example.

NTP best practices recommend configuring ntpd with the no-query parameter. The no-query parameter blocks access to all remote control queries. However, sometimes the nosts do not want to block all queries and want to give access for certain control queries remotely. This could be for the purpose of remote management and configuration of the hosts in certain scenarios. Such hosts tend to use firewalls or other middleboxes to blacklist certain queries within the network.

Recent work (reference needed) shows that significantly fewer hosts respond to mode 7 monlist queries as compared to other control queries because it is a well-known and exploited control query. These queries are likely blocked using blacklists on firewalls and middleboxes rather than the no-query option on NTP hosts. The remaining control queries that can be exploited likely remain out of the blacklist because they are undocumented in the current NTP specification [RFC5905].

This document describes all of the mode 6 control queries allowed by NTP and can help administrators make informed decisions on security measures to protect NTP devices from harmful queries and likely make those systems less vulnerable.

7. Acknowledgements

Tim Plunkett created the original version of this document. Aanchal Malhotra provided the initial version of the Security Considerations section.

8. Normative References

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, DOI 10.17487/RFC1305, March 1992, <http://www.rfc-editor.org/info/rfc1305>.

[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>.

Authors' Addresses

Dr. David L. Mills
University of Deleware

Email: mills@udel.edu

Brian Haberman (editor)
JHU

Email: brian@innovationslab.net

Internet Engineering Task Force
Internet-Draft
Intended status: Best Current Practice
Expires: June 16, 2019

D. Reilly, Ed.
Orolia USA
H. Stenn
Network Time Foundation
D. Sibold
PTB
December 13, 2018

Network Time Protocol Best Current Practices
draft-ietf-ntp-bcp-10

Abstract

The Network Time Protocol (NTP) is one of the oldest protocols on the Internet and has been widely used since its initial publication. This document is a collection of Best Practices for general operation of NTP servers and clients on the Internet. It includes recommendations for stable, accurate and secure operation of NTP infrastructure. This document is targeted at NTP version 4 as described in RFC 5905.

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 https://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 June 16, 2019.

Copyright Notice

Table of Contents

1. Introduction

NTP version 4 (NTPv4) has been widely used since its publication as [RFC5905]. This documentation is a collection of best practices for the operation of NTP clients and servers.

The recommendations in this document are intended to help operators distribute time on their networks more accurately and more securely. It is intended to apply generally to a broad range of networks. Some specific networks may have higher accuracy requirements that require additional techniques beyond what is documented here.

Among the best practices covered are recommendations for general network security, time protocol specific security, and NTP server and client configuration. NTP operation in embedded devices is also covered.

This document also contains information for protocol implementors who want to develop their own RFC 5905 compliant implementations.

1.1. Requirements Language

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. General Network Security Best Practices

2.1. BCP 38

Many network attacks rely on modifying the IP source address of a packet to point to a different IP address than the computer which originated it. UDP-based protocols such as NTP are generally more susceptible to spoofing attacks then other connection-oriented protocols. NTP control messages can generate a lot of data in response to a small query, which makes it more attractive as a vector for distributed denial-of-service attacks. (NTP Control messages are discussed further in Section 3.4). One documented instance of such an attack can be found here [1], and further discussion in [IMC14]

and [NDSS14]. Mitigating source address spoofing attacks should be a priority of anyone administering NTP.

BCP 38 [RFC2827] was approved in 2000 to address this. BCP 38 calls for filtering outgoing and incoming traffic to make sure that the source and destination IP addresses are consistent with the expected flow of traffic on each network interface. It is RECOMMENDED that large corporate networks (and ISP's of any size) implement ingress and egress filtering. More information is available at the BCP38 Info Web page [BCP38INFO] .

3. NTP Configuration Best Practices

This section provides Best Practices for NTP configuration and operation. Best Practices that are specific to the NTF implementation are compiled in Appendix A.

3.1. Keeping NTP up to date

Many network security mechanisms rely on time as part of their operation. If attackers can spoof the time, they may be able to bypass or neutralize other security elements. For example, incorrect time can disrupt the ability to reconcile logfile entries on the affected system with events on other systems. An application which is secure today could be insecure tomorrow once an unknown bug (or a known behavior) is exploited in the right way. Even our definition of what is secure has evolved over the years, so code which was considered secure when it was written may turn out to be insecure after some time.

There are multiple versions of the NTP protocol in use, and multiple implementations, on many different platforms. The practices in this document are meant to apply generally to any implementation of [RFC5905]. It is RECOMMENDED that that NTP users select an implementation that is actively maintained. Users should keep up to date on any known attacks on their selected implementation, and deploy updates containing security fixes as soon as practical.

3.2. Use enough time sources

An NTP implementation that is compliant with [RFC5905] takes the available sources of time and submits this timing data to sophisticated intersection, clustering, and combining algorithms to get the best estimate of the correct time. The description of these algorithms is beyond the scope of this document. Interested readers should read [RFC5905] or the detailed description of NTP in [MILLS2006].

- If there is only 1 source of time, the answer is obvious. It may not be a good source of time, but it's the only source of time that can be considered. Any issue with the time at the source will be passed on to the client.

- If there are 2 sources of time and they agree well enough, then the best time can be calculated easily. But if one source fails, then the solution degrades to the single-source solution outlined above. And if the two sources don't agree, then it's impossible to know which one is correct by simply looking at the time.

- If there are 3 sources of time, there is more data available to converge on the best calculated time, and this time is more likely to be accurate. And the loss of one of the sources (by becoming unreachable or unusable) can be tolerated. But at that point, the solution degrades to the 2 source solution.

- 4 or more sources of time is better, as long as the sources are diverse (Section 3.3). If one of these sources develops a problem there are still at least 3 other time sources.

Operators who are concerned with maintaining accurate time SHOULD use at least 4 independent, diverse sources of time. Four sources will provide sufficient backup in case one source goes down. If four sources are not available, operators MAY use fewer sources, subject to the risks outlined above.

But even with 4 or more sources of time, systemic problems can happen. For several hours before and after the June 2015 leap second, several operators implemented leap smearing while others did not, and many NTP end nodes could not determine an accurate time source because 2 of their 4 sources of time gave them consistent UTC/POSIX time, while the other 2 gave them consistent leap-smeared time. See Section 3.7.1 for more information.

Operators SHOULD monitor all of the time sources that are in use. If time sources do not generally agree, find out the cause and either correct the problems or stop using defective servers. See Section 3.5 for more information.

3.3. Use a diversity of Reference Clocks

When using servers with attached hardware reference clocks, it is RECOMMENDED that several different types of reference clocks be used. Having a diversity of sources with independent implementations means that any one issue is less likely to cause a service interruption.

Are all clocks on a network from the same vendor? They may have the same bugs. Even devices from different vendors may not be truly independent if they share common elements. Are they using the same base chipset? Are they all running the same version of firmware? Chipset and firmware bugs can happen, but they can be more difficult to diagnose than application software bugs. When having the correct time is of critical importance, it's ultimately up to operators to ensure that their sources are sufficiently independent, even if they are not under the operator's control.

A systemic problem with time from any satellite navigation service is possible and has happened. Sunspot activity can render satellite or radio-based time source unusable. If the time on your network has to be correct close to 100% of the time, then even if you are using a satellite-based system, operators need to plan for those rare instances when the system is unavailable (or wrong!).

3.4. Control Messages

Some implementations of NTPv4 provide the NTP Control Messages that were originally specified in Appendix B of [RFC1305] which defined NTPv3. These messages were never included the NTPv4 specification, but they are still used. Work is being done to formally document the structure of these control messages in [I-D.ietf-ntp-mode-6-cmds].

The NTP Control Messages are designed to permit monitoring and optionally authenticated control of NTP and its configuration. Used properly, these facilities provide vital debugging and performance information and control. Used improperly, these facilities can be an abuse vector. For this reason, it is RECOMMENDED that publicly-facing NTP servers should block mode 6 queries from outside their organization.

The ability to use Mode 6 beyond its basic monitoring capabilities SHOULD be limited to authenticated sessions that provide a 'controlkey'. It MAY also be limited through mechanisms outside of the NTP specification, such as Access Control Lists, that only allow access from approved IP addresses.

The NTP Control Messages responses are much larger than the corresponding queries. Thus, they can be abused in high-bandwidth DDoS attacks. To provide protection for such abuse NTP server operators on large networks SHOULD deploy ingress filtering in accordance with BCP 38 [RFC2827].

3.5. Monitoring

Operators SHOULD use their NTP implementation's remote monitoring capabilities to quickly identify servers which are out of sync, and ensure correctness of the service. Operators SHOULD also monitor system logs for messages so problems and abuse attempts can be quickly identified.

If a system starts getting unexpected time replies from its time servers, that can be an indication that the IP address of the system is being forged in requests to its time server. The goal of this attack is to convince the time server to stop serving time to the system whose address is being forged.

If a system is a broadcast client and its system log shows that it is receiving early time messages from its server, that is an indication that somebody may be forging packets from a broadcast server.

If a server's system log shows messages that indicates it is receiving timestamps that are earlier than the current system time, then either the system clock is unusually fast or somebody is trying to launch a replay attack against that server.

3.6. Using Pool Servers

It only takes a small amount of bandwidth and system resources to synchronize one NTP client, but NTP servers that can service tens of thousands of clients take more resources to run. Users who want to synchronize their computers SHOULD only synchronize to servers that they have permission to use.

The NTP pool project is a group of volunteers who have donated their computing and bandwidth resources to freely distribute time from primary time sources to others on the Internet. The time is generally of good quality, but comes with no guarantee whatsoever. If you are interested in using the pool, please review their instructions at http://www.pool.ntp.org/en/use.html [2].

If you are a vendor who wishes to provide time service to your customers or clients, consider joining the pool and providing a "vendor zone" through the pool project.

If you want to synchronize many computers, consider running your own NTP servers that are synchronized by the pool, and synchronizing your clients to your in-house NTP servers. This reduces the load on the pool.

3.7. Leap Second Handling

UTC is kept in agreement with the astronomical time UT1 [3] to within +/- 0.9 seconds by the insertion (or possibly a deletion) of a leap second. UTC is an atomic time scale whereas UT1 is based on the rotational rate of the earth. Leap seconds are not introduced at a fixed rate. They are announced by the International Earth Rotation and Reference Systems Service (IERS) in its Bulletin C [4] when necessary to keep UTC and UT1 aligned.

NTP time is based on the UTC timescale, and the protocol has the capability to broadcast leap second information. Some Global Navigation Satellite Systems (like GPS) or radio transmitters (like DCF77) broadcast leap second information, so if you are synced to an NTP server that is ultimately synced to a source that provides leap second notification you will get advance notification of impending leap seconds automatically.

Since the length of the UT1 day is generally slowly increasing [5], all leap seconds that have been introduced since the practice started in 1972 have been positive leap seconds, where a second is added to UTC. NTP also supports a negative leap second, where a second is removed from UTC, if that ever becomes necessary.

While earlier versions of NTP contained some ambiguity regarding when a leap second that is broadcast by a server should be applied by a client, RFC 5905 is clear that leap seconds are only applied on the last day of a month. However, because some older clients may apply it at the end of the current day, it is RECOMMENDED that NTP servers wait until the last day of the month before broadcasting leap seconds. Doing this will prevent older clients from applying a leap second at the wrong time. Note well that NTPv4's longest polling interval exceeds one day and thus a leap second announcement may be missed.

In circumstances where an NTP server is not receiving leap second information from an automated source, certain organizations maintain files which are updated every time a new leap second is announced:

NIST: ftp://time.nist.gov/pub/leap-seconds.list

US Navy (maintains GPS Time): ftp://tycho.usno.navy.mil/pub/ntp/leap-seconds.list

IERS (announces leap seconds):
https://hpiers.obspm.fr/iers/bul/bulc/ntp/leap-seconds.list

3.7.1. Leap Smearing

Some NTP installations make use of a technique called Leap Smearing. With this method, instead of introducing an extra second (or eliminating a second) on a leap second event, NTP time will be slewed in small increments over a comparably large window of time (called the smear interval) around the leap second event. The smear interval should be large enough to make the rate that the time is slewed small, so that clients will follow the smeared time without objecting. Periods ranging from 2 to 24 hours have been used successfully. During the adjustment window, all the NTP clients' times may be offset from UTC by as much as a full second, depending on the implementation. But at least all clients will generally agree on what time they think it is.

The purpose of Leap Smearing is to enable systems that don't deal with the leap second event properly to function consistently, at the expense of fidelity to UTC during the smear window. During a standard leap second event, that minute will have 61 (or possibly 59) seconds in it, and some applications (and even some OS's) are known to have problems with that.

Operators who have legal obligations or other strong requirements to be synchronized with UTC or civil time SHOULD NOT use leap smearing, because the distributed time cannot be guaranteed to be traceable to UTC during the smear interval.

Clients that are connected to leap smearing servers MUST NOT apply the standard NTP leap second handling. So these clients must never have a leap second file loaded, and the smearing servers must never advertise to clients that a leap second is pending.

Any use of leap smearing servers should be limited to within a single, well-controlled environment. Leap Smearing MUST NOT be used for public-facing NTP servers, as they will disagree with non-smearing servers (as well as UTC) during the leap smear interval, and there is no standardized way for a client to detect that a server is using leap smearing. However, be aware that some public-facing servers may be configured this way anyway in spite of this guidance.

System Administrators are advised to be aware of impending leap seconds and how the servers (inside and outside their organization) they are using deal with them. Individual clients MUST NOT be configured to use a mixture of smeared and non-smeared servers. If a client uses smeared servers, the servers it uses must all have the same leap smear configuration.

4. NTP Security Mechanisms

In the standard configuration NTP packets are exchanged unprotected between client and server. An adversary that is able to become a Man-In-The-Middle is therefore able to drop, replay or modify the content of the NTP packet, which leads to degradation of the time synchronization or the transmission of false time information. A threat analysis for time synchronization protocols is given in [RFC7384]. NTP provides two internal security mechanisms to protect authenticity and integrity of the NTP packets. Both measures protect the NTP packet by means of a Message Authentication Code (MAC). Neither of them encrypts the NTP's payload, because this payload information is not considered to be confidential.

4.1. Pre-Shared Key Approach

This approach applies a symmetric key for the calculation of the MAC, which protects authenticity and integrity of the exchanged packets for an association. NTP does not provide a mechanism for the exchange of the keys between the associated nodes. Therefore, for each association, keys SHOULD be exchanged securely by external means, and they SHOULD be protected from disclosure. It is RECOMMENDED that each association be protected by its own unique key. It is RECOMMENDED that participants agree to refresh keys periodically. However, NTP does not provide a mechanism to assist in doing so.

[RFC5905] specifies a hash which must be supported for calculation of the MAC, but other algorithms may be supported as well. The MD5 hash is now considered to be too weak. Implementations will soon be available based on AES-128-CMAC [I-D.ietf-ntp-mac], and users are encouraged to use that when it is available.

To use this approach the communication partners have to exchange the key, which consists of a keyid with a value between 1 and 65534, inclusive, and a label which indicates the chosen digest algorithm. Each communication partner adds this information to its own key file.

Some implementations store the key in clear text. Therefore it SHOULD only be readable by the NTP process. Different keys are added line by line to the key file.

An NTP client establishes a protected association by appending the key to the server statement in its configuration file. Note that the NTP process has to trust the applied key.

4.2. Autokey

Autokey was specified in 2010 to provide automated key management and authentication of NTP servers. However, security researchers have identified vulnerabilities [6] in the Autokey protocol.

Autokey SHOULD NOT be used.

4.3. Network Time Security

Work is in progress on an enhanced replacement for Autokey. Refer to [I-D.ietf-ntp-using-nts-for-ntp] for more information.

5. NTP Security Best Practices

This section lists some general NTP security practices, but these issues may (or may not) have been mitigated in particular versions of particular implementations. Contact the maintainers of your implementation for more information.

5.1. Minimizing Information Leakage

The base NTP packet leaks important information (including reference ID and reference time) that may be used in attacks [NDSS16], [CVE-2015-8138], [CVE-2016-1548]. A remote attacker can learn this information by sending mode 3 queries to a target system and inspecting the fields in the mode 4 response packet. NTP control queries also leak important information (including reference ID, expected origin timestamp, etc.) that may be used in attacks [CVE-2015-8139]. A remote attacker can learn this information by sending control queries to a target system and inspecting the response.

As such, mechanisms outside of the NTP protocol, such as Access Control Lists, SHOULD be used to limit the exposure of this information to allowed IP addresses, and keep it from remote attackers not on the list. Hosts SHOULD only respond to NTP control queries from authorized parties.

A host that is not supposed to act as an NTP server that provides timing information to other hosts MAY additionally log and drop incoming mode 3 timing queries from unexpected sources. Note well that the easiest way to monitor ntpd's status is to send it a mode 3 query. It is recommended that operators SHOULD filter mode 3 queries at the edge, or make sure mode 3 queries are allowed only from trusted systems or networks.

A "leaf-node host" is a host that is using NTP solely for the purpose of adjusting its own system time. Such a host is not expected to provide time to other hosts, and relies exclusively on NTP's basic mode to take time from a set of servers. (That is, the host sends mode 3 queries to its servers and receives mode 4 responses from these servers containing timing information.) To minimize information leakage, leaf-node hosts SHOULD drop all incoming NTP packets except mode 4 response packets that come from known sources. Note well that proper monitoring of an NTP server instance includes checking the time of that NTP server instance.

Please refer to [I-D.ietf-ntp-data-minimization] for more information.

5.2. Avoiding Daemon Restart Attacks

[RFC5905] says NTP clients should not accept time shifts greater than the panic threshold. Specifically, RFC 5905 says "PANIC means the offset is greater than the panic threshold PANICT (1000 s) and SHOULD cause the program to exit with a diagnostic message to the system log."

However, this behavior can be exploited by attackers as described in [NDSS16], when the following two conditions hold:

1. The operating system automatically restarts the NTP client when it quits. (Modern *NIX operating systems are replacing traditional init systems with process supervisors, such as systemd, which can be configured to automatically restart any daemons that quit. This behavior is the default in CoreOS and Arch Linux. It is likely to become the default behavior in other systems as they migrate legacy init scripts to process supervisors such as systemd.)

2. The NTP client is configured to ignore the panic threshold on all restarts.

In such cases, if the attacker can send the target an offset that exceeds the panic threshold, the client will quit. Then, when it restarts, it ignores the panic threshold and accepts the attacker's large offset.

Operators SHOULD be aware that when operating with the above two conditions, the panic threshold offers no protection from attacks. The natural solution is not to run hosts with these conditions. Specifically, operators SHOULD NOT ignore the panic threshold in all cold-start situations unless sufficient oversight and checking is in place to make sure that this type of attack cannot happen.

As an alternative, the following steps MAY be taken by operators to mitigate the risk of attack:

- Monitor the NTP system log to detect when the NTP daemon has quit due to a panic event, as this could be a sign of an attack.
- Request manual intervention when a timestep larger than the panic threshold is detected.
- Configure the ntp client to only ignore the panic threshold in a cold start situation.
- Add 'minsane' and 'minclock' parameters to the ntp.conf file so ntpd waits until enough trusted sources of time agree on the correct time.

In addition, implementations SHOULD prevent the NTP daemon from taking time steps that set the clock to a time earlier than the compile date of the NTP daemon.

5.3. Detection of Attacks Through Monitoring

Operators SHOULD monitor their NTP instances to detect attacks. Many known attacks on NTP have particular signatures. Common attack signatures include:

1. Bogus packets - A packet whose origin timestamp does not match the value that expected by the client.
2. Zero origin packet - A packet with an origin timestamp set to zero [CVE-2015-8138].
3. A packet with an invalid cryptographic MAC [CCR16].

The observation of many such packets could indicate that the client is under attack.

5.4. Kiss-o'-Death Packets

The "Kiss-o'-Death" (KoD) packet is a rate limiting mechanism where a server can tell a misbehaving client to reduce its query rate. It is RECOMMENDED that all NTP devices respect these packets and back off when asked to do so by a server. It is even more important for an embedded device, which may not have an exposed control interface for NTP.

That said, a client MUST only accept a KoD packet if it has a valid origin timestamp. Once a RATE packet is accepted, the client should

increase its poll interval value (thus decreasing its polling rate) up to a reasonable maximum. This maximum can vary by implementation but should not exceed a poll interval value of 13 (2 hours). The mechanism to determine how much to increase the poll interval value is undefined in [RFC5905]. If the client uses the poll interval value sent by the server in the KoD packet, it MUST NOT simply accept any value. Using large interval values may open a vector for a denial-of-service attack that causes the client to stop querying its server [NDSS16].

The KoD mechanism relies on clients behaving properly in order to be effective. Some clients ignore the KoD packet entirely, and other poorly-implemented clients might unintentionally increase their poll rate and simulate a denial of service attack. Server administrators SHOULD be prepared for this and take measures outside of the NTP protocol to drop packets from misbehaving clients when these clients are detected.

Also, Kiss-o'-Death (KoD) packets can be used in denial of service attacks. Thus, the observation of even just one KoD packet with a high poll value could be sign that the client is under attack.

5.5. Broadcast Mode Should Only Be Used On Trusted Networks

Per [RFC5905], NTP's broadcast mode is authenticated using symmetric key cryptography. The broadcast server and all of its broadcast clients share a symmetric cryptographic key, and the broadcast server uses this key to append a message authentication code (MAC) to the broadcast packets it sends.

Importantly, all broadcast clients that listen to this server have to know the cryptographic key. This mean that any client can use this key to send valid broadcast messages that look like they come from the broadcast server. Thus, a rogue broadcast client can use its knowledge of this key to attack the other broadcast clients.

For this reason, an NTP broadcast server and all its clients have to trust each other. Broadcast mode SHOULD only be run from within a trusted network.

5.6. Symmetric Mode Should Only Be Used With Trusted Peers

In symmetric mode, two peers Alice and Bob can both push and pull synchronization to and from each other using either ephemeral symmetric passive (mode 2) or persistent symmetric active (NTP mode 1) packets. The persistent association is preconfigured and initiated at the active peer but not preconfigured at the passive peer (Bob). Upon receipt of a mode 1 NTP packet from Alice, Bob

mobilizes a new ephemeral association if he does not have one already. This is a security risk for Bob because an arbitrary attacker can attempt to change Bob's time by asking Bob to become its symmetric passive peer.

For this reason, a host SHOULD only allow symmetric passive associations to be established with trusted peers. Specifically, a host SHOULD require each of its symmetric passive association to be cryptographically authenticated. Each symmetric passive association SHOULD be authenticated under a different cryptographic key.

6. NTP in Embedded Devices

As computing becomes more ubiquitous, there will be many small embedded devices that require accurate time. These devices may not have a persistent battery-backed clock, so using NTP to set the correct time on power-up may be critical for proper operation. These devices may not have a traditional user interface, but if they connect to the Internet they will be subject to the same security threats as traditional deployments.

6.1. Updating Embedded Devices

Vendors of embedded devices MUST pay attention to the current state of protocol security issues and bugs in their chosen implementation, because their customers don't have the ability to update their NTP implementation on their own. Those devices may have a single firmware upgrade, provided by the manufacturer, that updates all capabilities at once. This means that the vendor assumes the responsibility of making sure their devices have the latest NTP updates applied.

Vendors of embedded devices SHOULD also include the ability to update information regarding which NTP server to connect to on these devices.

There is a catalog of NTP server abuse incidents, some of which involve embedded devices, on the Wikipedia page for NTP Server Misuse and Abuse [7].

6.2. Server configuration

Vendors of embedded devices that need time synchronization SHOULD also carefully consider where they get their time from. There are several public-facing NTP servers available, but they may not be prepared to service requests from thousands of new devices on the Internet. Vendors SHOULD only synchronize to servers that they have permission to use.

Vendors are encouraged to invest resources into providing their own time servers for their devices to connect to.

Vendors should read [RFC4085], which advises against embedding globally-routable IP addresses in products, and offers several better alternatives.

6.2.1. NTP Pool Project Vendor Subdomains

The NTP Pool Project offers a program where vendors can obtain their own subdomain that is part of the NTP Pool. This offers vendors the ability to safely make use of the time distributed by the Pool for their devices. Vendors are encouraged to support the pool if they participate. For more information, visit http://www.pool.ntp.org/en/vendors.html [8] .

7. NTP over Anycast

Anycast is described in BCP 126 [RFC4786]. (Also see [RFC7094]). With anycast, a single IP address is assigned to multiple interfaces, and routers direct packets to the closest active interface.

Anycast is often used for Internet services at known IP addresses, such as DNS. Anycast can also be used in large organizations to simplify configuration of a large number of NTP clients. Each client can be configured with the same NTP server IP address, and a pool of anycast servers can be deployed to service those requests. New servers can be added to or taken from the pool, and other than a temporary loss of service while a server is taken down, these additions can be transparent to the clients.

Note well that using a single anycast address for NTP presents its own potential issues. It means each client will likely use a single time server source. A key element of a robust NTP deployment is each client using multiple sources of time. With multiple time sources, a client will analyze the various time sources, selecting good ones, and disregarding poor ones. If a single Anycast address is used, this analysis will not happen.

If clients are connected to an NTP server via anycast, the client does not know which particular server they are connected to. As anycast servers may arbitrarily enter and leave the network, the server a particular client is connected to may change. This may cause a small shift in time from the perspective of the client when the server it is connected to changes. It is RECOMMENDED that anycast only be deployed in environments where these small shifts can be tolerated.

Configuration of an anycast interface is independent of NTP. Clients will always connect to the closest server, even if that server is having NTP issues. It is RECOMMENDED that anycast NTP implementations have an independent method of monitoring the performance of NTP on a server. If the server is not performing to specification, it should remove itself from the Anycast network. It is also RECOMMENDED that each Anycast NTP server have an alternative method of access, such as an alternate Unicast IP address, so its performance can be checked independently of the anycast routing scheme.

One useful application in large networks is to use a hybrid unicast/ anycast approach. Stratum 1 NTP servers can be deployed with unicast interfaces at several sites. Each site may have several Stratum 2 servers with two ethernet interfaces, or a single interface which can support multiple addresses. One interface has a unique unicast IP address. The second has an anycast IP interface (with a shared IP address per location). The unicast interfaces can be used to obtain time from the Stratum 1 servers globally (and perhaps peer with the other Stratum 2 servers at their site). Clients at each site can be configured to use the shared anycast address for their site, simplifying their configuration. Keeping the anycast routing restricted on a per-site basis will minimize the disruption at the client if its closest anycast server changes. Each Stratum 2 server can be uniquely identified on their unicast interface, to make monitoring easier.

8. Acknowledgments

The authors wish to acknowledge the contributions of Sue Graves, Samuel Weiler, Lisa Perdue, Karen O'Donoghue, David Malone, Sharon Goldberg, Martin Burnicki, Miroslav Lichvar, Daniel Fox Franke, Robert Nagy, and Brian Haberman.

9. IANA Considerations

This memo includes no request to IANA.

10. Security Considerations

Time is a fundamental component of security on the internet. The absence of a reliable source of current time subverts many common web authentication schemes, e.g., by allowing the use of expired credentials or by allowing for replay of messages only intended to be processed once.

Much of this document directly addresses how to secure NTP servers. In particular, see Section 2, Section 4, and Section 5.

There are several general threats to time synchronization protocols which are discussed in [RFC7384].

[I-D.ietf-ntp-using-nts-for-ntp] specifies the Network Time Security (NTS) mechanism and applies it to NTP. Readers are encouraged to check the status of the draft, and make use of the methods it describes.

11. References

11.1. Normative References

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <https://www.rfc-editor.org/info/rfc2827>.

[RFC4085] Plonka, D., "Embedding Globally-Routable Internet Addresses Considered Harmful", BCP 105, RFC 4085, DOI 10.17487/RFC4085, June 2005, <https://www.rfc-editor.org/info/rfc4085>.

[RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <https://www.rfc-editor.org/info/rfc4786>.

[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, <https://www.rfc-editor.org/info/rfc5905>.

11.2. Informative References

[BCP38INFO] "BCP38 Info Page", <http://www.bcp38.info>.

[CCR16] Malhotra, A. and S. Goldberg, "Attacking NTP's Authenticated Broadcast Mode", SIGCOMM Computer Communications Review (CCR) , 2016.

[CVE-2015-8138] Van Gundy, M. and J. Gardner, "NETWORK TIME PROTOCOL ORIGIN TIMESTAMP CHECK IMPERSONATION VULNERABILITY", 2016, <http://www.talosintel.com/reports/TALOS-2016-0077>.

[CVE-2015-8139]
Van Gundy, M., "NETWORK TIME PROTOCOL NTPQ AND NTPDC ORIGIN TIMESTAMP DISCLOSURE VULNERABILITY", 2016, <http://www.talosintel.com/reports/TALOS-2016-0078>.

[CVE-2016-1548]
Gardner, J. and M. Lichvar, "Xleave Pivot: NTP Basic Mode to Interleaved", 2016, <http://blog.talosintel.com/2016/04/vulnerability-spotlight-further-ntpd_27.html>.

[I-D.ietf-ntp-data-minimization]
Franke, D. and A. Malhotra, "NTP Client Data Minimization", draft-ietf-ntp-data-minimization-03 (work in progress), September 2018.

[I-D.ietf-ntp-mac]
Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", draft-ietf-ntp-mac-05 (work in progress), October 2018.

[I-D.ietf-ntp-mode-6-cmds]
Haberman, B., "Control Messages Protocol for Use with Network Time Protocol Version 4", draft-ietf-ntp-mode-6-cmds-06 (work in progress), September 2018.

[I-D.ietf-ntp-using-nts-for-ntp]
Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", draft-ietf-ntp-using-nts-for-ntp-14 (work in progress), October 2018.

[IMC14] Czyz, J., Kallitsis, M., Gharaibeh, M., Papadopoulos, C., Bailey, M., and M. Karir, "Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks", Internet Measurement Conference , 2014.

[MILLS2006]
Mills, D., "Computer network time synchronization: the Network Time Protocol", CRC Press , 2006.

[NDSS14] Rossow, C., "Amplification Hell: Revisiting Network Protocols for DDoS Abuse", NDSS'14, San Diego, CA. , 2014.

[NDSS16] Malhotra, A., Cohen, I., Brakke, E., and S. Goldberg, "Attacking the Network Time Protocol", NDSS'16, San Diego, CA. , 2016, <https://eprint.iacr.org/2015/1020.pdf>.

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, DOI 10.17487/RFC1305, March 1992, <https://www.rfc-editor.org/info/rfc1305>.

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

[RFC7094] McPherson, D., Oran, D., Thaler, D., and E. Osterweil, "Architectural Considerations of IP Anycast", RFC 7094, DOI 10.17487/RFC7094, January 2014, <https://www.rfc-editor.org/info/rfc7094>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

11.3. URIs

[1] https://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack/

[2] http://www.pool.ntp.org/en/use.html

[3] https://en.wikipedia.org/wiki/Solar_time#Mean_solar_time

[4] https://www.iers.org/IERS/EN/Publications/Bulletins/bulletins.html

[5] https://en.wikipedia.org/wiki/Solar_time#Mean_solar_time

[6] https://lists.ntp.org/pipermail/ntpwg/2011-August/001714.html

[7] https://en.wikipedia.org/wiki/NTP_server_misuse_and_abuse

[8] http://www.pool.ntp.org/en/vendors.html

[9] http://bk1.ntp.org/ntp-stable/README.leapsmear?PAGE=anno

[10] https://support.ntp.org/bin/view/Support/ConfiguringNTP

Appendix A. NTP Implementation by the Network Time Foundation

The Network Time Foundation (NTF) provides the reference implementation of NTP, well-known under the name "ntpd". It is actively maintained and developed by NTF's NTP Project, with help from volunteers and NTF's supporters. This NTP software can be downloaded from <http://www.ntp.org/downloads.html>

A.1. Use enough time sources

In addition to the recommendation given in Section Section 3.2 the ntpd implementation provides the 'pool' directive. Starting with ntp-4.2.6, this directive will spin up enough associations to provide robust time service, and will disconnect poor servers and add in new servers as-needed. If you have good reason, you may use the 'minclock' and 'maxclock' options of the 'tos' command to override the default values of how many servers are discovered through the 'pool' directive.

A.2. NTP Control and Facility Messages

In addition to NTP Control Messages the ntpd implementation also offers the Mode 7 commands for monitoring and configuration.

If Mode 7 has been explicitly enabled to be used for more than basic monitoring it should be limited to authenticated sessions that provide a 'requestkey'.

As mentioned above, there are two general ways to use Mode 6 and Mode 7 requests. One way is to query ntpd for information, and this mode can be disabled with:

restrict ... noquery

The second way to use Mode 6 and Mode 7 requests is to modify ntpd's behavior. Modification of ntpd's configuration requires an authenticated session by default. If no authentication keys have been specified no modifications can be made. For additional protection, the ability to perform these modifications can be controlled with:

restrict ... nomodify

Users can prevent their NTP servers from considering query/ configuration traffic by default by adding the following to their ntp.conf file:

restrict default -4 nomodify notrap nopeer noquery

```
restrict default -6 nomodify notrap nopeer noquery

restrict source nomodify notrap noquery
# nopeer is OK if you don't use the 'pool' directive
```

A.3. Monitoring

The reference implementation of NTP allows remote monitoring. Access to this service is generally controlled by the "noquery" directive in NTP's configuration file (ntp.conf) via a "restrict" statement. The syntax reads:

```
restrict address mask address_mask noquery
```

If a system is using broadcast mode and is running ntp-4.2.8p6 or later, use the 4th field of the ntp.keys file to specify the IPs of machines that are allowed to serve time to the group.

A.4. Leap Second File

The use of leap second files requires ntpd 4.2.6 or later. After fetching the leap seconds file onto the server, add this line to ntpd.conf to apply and use the file:

```
leapfile "/path/to your/leap-file"
```

You may need to restart ntpd to apply this change.

ntpd servers with a manually configured leap second file will ignore leap second information broadcast from upstream NTP servers until the leap second file expires. If no valid leap second file is available then a leap second notification from an attached reference clock is always accepted by ntpd.

If no valid leap second file is available, a leap second notification may be accepted from upstream NTP servers. As of ntp-4.2.6, a majority of servers must provide the notification before it is accepted. Before 4.2.6, a leap second notification would be accepted if a single upstream server of a group of configured servers provided a leap second notification. This would lead to misbehavior if single NTP servers sent an invalid leap second warning, e.g. due to a faulty GPS receiver in one server, but this behavior was once chosen because in the "early days" there was a greater chance that leap second information would be available from a very limited number of sources.

A.5. Leap Smearing

Leap Smearing was introduced in ntpd versions 4.2.8.p3 and 4.3.47, in response to client requests. Support for leap smearing is not configured by default and must be added at compile time. In addition, no leap smearing will occur unless a leap smear interval is specified in ntpd.conf . For more information, refer to http://bk.ntp.org/ntp-stable/README.leapsmear?PAGE=anno [9].

A.6. Configuring ntpd

See https://support.ntp.org/bin/view/Support/ConfiguringNTP [10] for additional information on configuring ntpd.

A.7. Pre-Shared Keys

Each communication partner must add the keyid information to their key file in the form:

keyid label key

An ntpd client establishes a protected association by appending the option "key keyid" to the server statement in ntp.conf:

server address key keyid

A key is deemed trusted when its keyid is added to the list of trusted keys by the "trustedkey" statement in ntp.conf.

trustedkey keyid_1 keyid_2 ... keyid_n

Starting with ntp-4.2.8p7 the ntp.keys file accepts an optional 4th column, a comma-separated list of IPs that are allowed to serve time. Use this feature. Note, however, that an adversarial client that knows the symmetric broadcast key could still easily spoof its source IP to an IP that is allowed to serve time. (This is easy to do because the origin timestamp on broadcast mode packets is not validated by the client. By contrast, client/server and symmetric modes do require origin timestamp validation, making it more difficult to spoof packets [CCR16].

Authors' Addresses

Denis Reilly (editor)
Orolia USA
1565 Jefferson Road, Suite 460
Rochester, NY 14623
US

Email: denis.reilly@orolia.com

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 28, 2016

D. Sibold
K. Teichel
PTB
S. Roettger
Google Inc.
R. Housley
Vigil Security

February 25, 2016

Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS)

draft-ietf-ntp-cms-for-nts-message-06

Abstract

This document describes a convention for using the Cryptographic Message Syntax (CMS) to protect the messages in the Network Time Security (NTS) protocol. NTS provides authentication of time servers as well as integrity protection of time synchronization messages using Network Time Protocol (NTP) or Precision Time Protocol (PTP).

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].

Status of This Memo

Table of Contents

1. Introduction

This document provides details on how to construct NTS messages in practice. NTS provides secure time synchronization with time servers using Network Time Protocol (NTP) [RFC5905] or Precision Time Protocol (PTP) [IEEE1588]. Among other things, this document describes a convention for using the Cryptographic Message Syntax (CMS) [RFC5652] to protect messages in the Network Time Security (NTS) protocol. Encryption is used to provide confidentiality of secrets, and digital signatures are used to provide authentication and integrity of content.

Sometimes CMS is used in an exclusively ASN.1 [ASN1] environment. In this case, the NTS message may use any syntax that facilitates easy implementation.

2. CMS Conventions for NTS Message Protection

Regarding the usage of CMS, we differentiate between three archetypes according to which the NTS message types can be structured. They are presented below. Note that the NTS Message Object that is at the core of each structure does not necessarily contain all the data needed for the particular message type, but may contain only that data which needs to be secured directly with cryptographic operations using the CMS. Specific information about what is included can be found in Section 3.

NTS-Plain: This archetype is used for actual time synchronization messages (explicitly, the following message types: time_request, time_response, server_broad, see [I-D.ietf-ntp-network-time-security], Section 6) as well as for client-side messages of all unicast and broadcast bootstrapping exchanges (explicitly client_assoc, client_cook and client_bpar) as well as the broadcast keycheck exchange (client_keycheck and server_keycheck). This archetype does not make use of any CMS structures at all. Figure 1 illustrates this structure.

NTS-Encrypted-and-Signed: This archetype is used for secure transmission of the cookie (only for the server_cook message type,

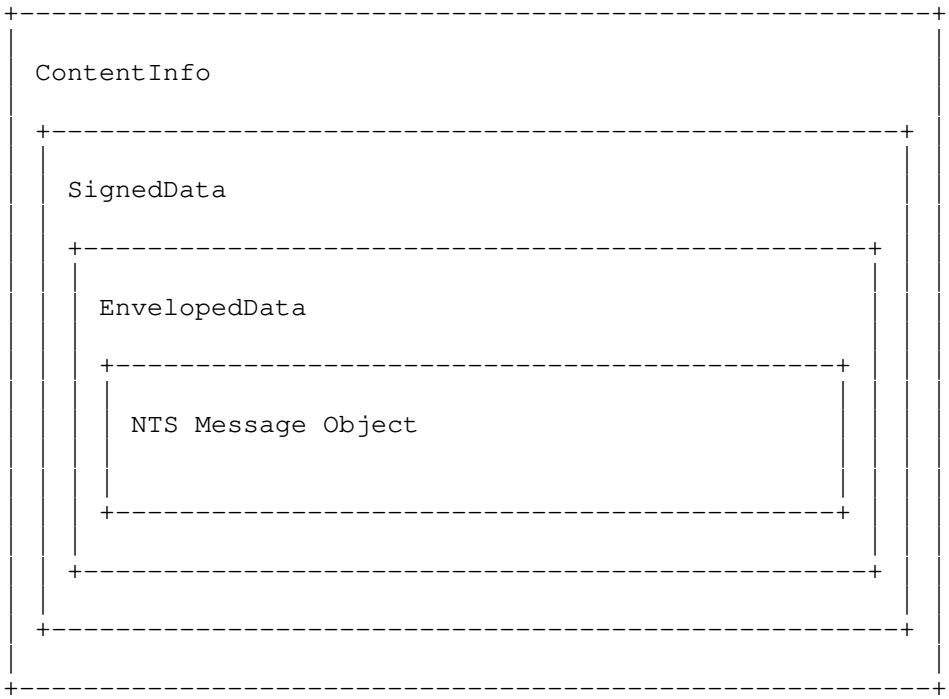
see [I-D.ietf-ntp-network-time-security], Section 6). For this, the following CMS structure is used:

First, the NTS message MUST be encrypted using the EnvelopedData content type. EnvelopedData supports nearly any form of key management. In the NTS protocol the client provides a certificate in an unprotected message, and the public key from this certificate, if it is valid, will be used to establish a pairwise symmetric key for the encryption of the protected NTS message.

Second, the EnvelopedData content MUST be digitally signed using the SignedData content type. SignedData supports nearly any form of digital signature, and in the NTS protocol the server will include its certificate within the SignedData content type.

Third, the SignedData content type MUST be encapsulated in a ContentInfo content type.

Figure 2 illustrates this structure.

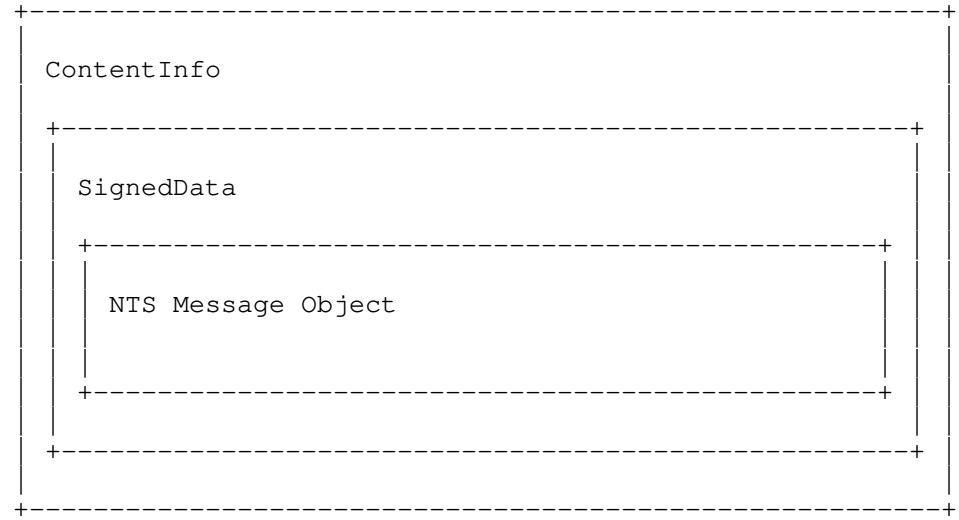


NTS-Signed: This archetype is used for server_assoc and server_bpar message types. It uses the following CMS structure:

First, the NTS message object MUST be wrapped in a SignedData content type. The messages MUST be digitally signed, and certificates included. SignedData supports nearly any form of digital signature, and in the NTS protocol the server will include its certificate within the SignedData content type.

Second, the SignedData content type MUST be encapsulated in a ContentInfo content type.

Figure 3 illustrates this structure.



2.1. Fields of the employed CMS Content Types

Overall, three CMS content types are used for NTS messages by the archetypes above. Explicitly, those content types are ContentInfo, SignedData and EnvelopedData. The following is a description of how the fields of those content types are used in detail.

2.1.1. ContentInfo

The ContentInfo content type is used in all archetypes except NTS-Plain. The fields of the ContentInfo content type are used as follows:

contentType -- indicates the type of the associated content. For all archetypes which use ContentInfo (these are NTS-Signed and

NTS-Encrypted-and-Signed), it MUST contain the object identifier for the SignedData content type:

```
id-signedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
   us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }
```

content -- is the associated content. For all archetypes using ContentInfo, it MUST contain the DER encoded SignedData content type.

2.1.2. SignedData

The SignedData content type is used in the NTS-Signed and NTS-Encrypted-and-Signed archetypes, but not in the NTS-Plain archetype. The fields of the SignedData content type are used as follows:

version -- the appropriate value depends on the optional items that are included. In the NTS protocol, the signer certificate MUST be included and other items MAY be included. The instructions in [RFC5652] Section 5.1 MUST be followed to set the correct value.

digestAlgorithms -- is a collection of message digest algorithm identifiers. In the NTS protocol, there MUST be exactly one algorithm identifier present. The instructions in Section 5.4 of [RFC5652] MUST be followed.

encapContentInfo -- this structure is always present. In the NTS protocol, it MUST follow these conventions:

eContentType -- is an object identifier. In the NTS protocol, for the NTS-Signed archetype, it MUST identify the type of the NTS message that was encapsulated. For the NTS-Encrypted-and-Signed archetype, it MUST contain the object identifier for the EnvelopedData content type:

```
id-envelopedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
       us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3 }.
```

eContent is the content itself, carried as an octet string. For the NTS-Signed archetype, it MUST contain the DER encoded encapsulated NTS message object. The instructions in Section 6.3 of [RFC5652] MUST be followed. For the NTS-Encrypted-and-Signed archetype, it MUST contain the DER encoded EnvelopedData content type.

certificates -- is a collection of certificates. In the NTS protocol, it MUST contain the DER encoded certificate [RFC5280] of

the sender. It is intended that the collection of certificates be sufficient for the recipient to construct a certification path from a recognized "root" or "top-level certification authority" to the certificate used by the sender.

crls -- is a collection of revocation status information. In the NTS protocol, it MAY contain one or more DER encoded CRLs [RFC5280]. It is intended that the collection contain information sufficient to determine whether the certificates in the certificates field are valid.

signerInfos -- is a collection of per-signer information. In the NTS protocol, for the NTS-Signed and the NTS-Encrypted-and-Signed archetypes, there MUST be exactly one SignerInfo structure present. The details of the SignerInfo type are discussed in Section 5.3 of [RFC5652]. In the NTS protocol, it MUST follow these conventions:

version -- is the syntax version number. In the NTS protocol, the SignerIdentifier is subjectKeyIdentifier, therefore the version MUST be 3.

sid -- identifies the signer's certificate. In the NTS protocol, the "sid" field contains the subjectKeyIdentifier from the signer's certificate.

digestAlgorithm -- identifies the message digest algorithm and any associated parameters used by the signer. In the NTS protocol, the identifier MUST match the single algorithm identifier present in the digestAlgorithms.

signedAttrs -- is a collection of attributes that are signed. In the NTS protocol, it MUST be present, and it MUST contain the following attributes:

Content Type -- see Section 11.1 of [RFC5652].

Message Digest -- see Section 11.2 of [RFC5652].

In addition, it MAY contain the following attributes:

Signing Time -- see Section 11.3 of [RFC5652].

Binary Signing Time -- see Section 3 of [RFC5652].

signatureAlgorithm -- identifies the signature algorithm and any associated parameters used by the signer to generate the digital signature.

signature is the result of digital signature generation using the message digest and the signer's private key. The instructions in Section 5.5 of [RFC5652] MUST be followed.

unsignedAttrs -- is an optional collection of attributes that are not signed. In the NTS protocol, it MUST be absent.

2.1.3. EnvelopedData

The EnvelopedData content type is used only in the NTS-Encrypted-and-Signed archetype. The fields of the EnvelopedData content type are used as follows:

version -- the appropriate value depends on the type of key management that is used. The instructions in [RFC5652] Section 6.1 MUST be followed to set the correct value.

originatorInfo -- this structure is present only if required by the key management algorithm. In the NTS protocol, it MUST be present when a key agreement algorithm is used, and it MUST be absent when a key transport algorithm is used. The instructions in Section 6.1 of [RFC5652] MUST be followed.

recipientInfos -- this structure is always present. In the NTS protocol, it MUST contain exactly one entry that allows the client to determine the key used to encrypt the NTS message. The instructions in Section 6.2 of [RFC5652] MUST be followed.

encryptedContentInfo -- this structure is always present. In the NTS protocol, it MUST follow these conventions:

contentType -- indicates the type of content. In the NTS protocol, it MUST identify the type of the NTS message that was encrypted.

contentEncryptionAlgorithm -- identifies the content-encryption algorithm and any associated parameters used to encrypt the content.

encryptedContent -- is the encrypted content. In the NTS protocol, it MUST contain the encrypted NTS message. The instructions in Section 6.3 of [RFC5652] MUST be followed.

unprotectedAttrs -- this structure is optional. In the NTS protocol, it MUST be absent.

3. Implementation Notes: ASN.1 Structures and Use of the CMS

This section presents some hints about the structures of the NTS message objects for the different message types when one wishes to implement the security mechanisms.

3.1. Preliminaries

The following ASN.1 coded data types "NTSAccessKey", "NTSNonce", and "NTSVersion" are needed for other types used below for NTS messages. "NTSAccessKey" specifies an access key, which is required for limitation of client association requests.

```
NTSAccessKey ::= OCTET STRING (SIZE(16))
```

"NTSNonce" specifies a 128 bit nonce as required in several message types.

```
NTSNonce ::= OCTET STRING (SIZE(16))
```

"NTSVersion" specifies a version number, which is required for version negotiation.

```
NTSVersion ::= INTEGER (0..255)
```

The following ASN.1 coded data types are also necessary for other types.

```
KeyEncryptionAlgorithmIdentifiers ::=
    SET OF KeyEncryptionAlgorithmIdentifier

ContentEncryptionAlgorithmIdentifiers ::=
    SET OF ContentEncryptionAlgorithmIdentifier
```

3.2. Unicast Messages

3.2.1. Access Messages

3.2.1.1. Message Type: "client_access"

This message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientAccessData" and structured as follows:

```
ClientAccessData ::= NULL
```

It is identified by the following object identifier:

```
id-ct-nts-clientAccess OBJECT IDENTIFIER ::= TBD1
```

3.2.1.2. Message Type: "server_access"

This message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ServerAccessData" and structured as follows:

```
ServerAccessData ::= SEQUENCE {
    accessKey         NTSAccessKey
}
```

It is identified by the following object identifier:

```
id-ct-nts-serverAccess OBJECT IDENTIFIER ::= TBD2
```

3.2.2. Association Messages

3.2.2.1. Message Type: "client_assoc"

This message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientAssocData" and structured as follows:

```
ClientAssocData ::= SEQUENCE {
    accessKey        NTSAccessKey,
    nonce            NTSNonce,
    minVersion       NTSVersion,
    hmacHashAlgos    DigestAlgorithmIdentifiers,
    keyEncAlgos      KeyEncryptionAlgorithmIdentifiers,
    contentEncAlgos  ContentEncryptionAlgorithmIdentifiers
}
```

It is identified by the following object identifier:

```
id-ct-nts-clientAssoc OBJECT IDENTIFIER ::= TBD3
```

3.2.2.2. Message Type: "server_assoc"

This message is structured according to the NTS-Signed archetype. It requires additional data besides that which is transported in the NTS message object, namely the signature and certificate chain are included in the appropriate fields of the "SignedData" CMS structure that the NTS message object is wrapped in. The NTS message object itself is an ASN.1 object of type "ServerAssocData" and structured as follows:

```
ServerAssocData ::= SEQUENCE {
    nonce                NTSNonce,
    proposedVersion      NTSVersion,
    hmacHashAlgos        DigestAlgorithmIdentifiers,
    choiceHmacHashAlgo   DigestAlgorithmIdentifier,
    keyEncAlgos          KeyEncryptionAlgorithmIdentifiers,
    choiceKeyEncAlgo     KeyEncryptionAlgorithmIdentifier,
    contentEncAlgos      ContentEncryptionAlgorithmIdentifiers,
    choiceContentEncAlgo ContentEncryptionAlgorithmIdentifier
}
```

It is identified by the following object identifier:

```
id-ct-nts-serverAssoc OBJECT IDENTIFIER ::= TBD4
```

3.2.3. Cookie Messages

3.2.3.1. Message Type: "client_cook"

This message is structured according to the NTS-Plain archetype. It requires no additional data besides that which is transported in the NTS message object. The NTS message object itself is an ASN.1 object of type "ClientCookieData" and structured as follows:

```
ClientCookieData ::= SEQUENCE {
    nonce         NTSNonce,
    signAlgo      SignatureAlgorithmIdentifier,
    hmacHashAlgo  DigestAlgorithmIdentifier,
    encAlgo       ContentEncryptionAlgorithmIdentifier,
    keyEncAlgo    KeyEncryptionAlgorithmIdentifier,
    certificates  CertificateSet
}
```

It is identified by the following object identifier:

```
id-ct-nts-clientCookie OBJECT IDENTIFIER ::= TBD5
```

3.2.3.2. Message Type: "server_cook"

This message is structured according to the "NTS-Encrypted-and-Signed" archetype. It requires additional data besides that which is transported in the NTS message object, namely the signature is included in the appropriate field of the "SignedData" CMS structure that the NTS message object is wrapped in. The NTS message object itself is an ASN.1 sequence of type "ServerCookieData" and structured as follows:

```
ServerCookieData ::= SEQUENCE {
    nonce     NTSNonce,
    cookie    OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier:

```
id-ct-nts-serverCookie OBJECT IDENTIFIER ::= TBD6
```

3.2.4. Time Synchronization Messages

3.2.4.1. Message Type: "time_request"

This message is structured according to the "NTS-Plain" archetype.

This message type requires additional data to that which is included in the NTS message object, namely it requires regular time synchronization data, as an unsecured packet from a client to a server would contain. Optionally, it requires the Message Authentication Code (MAC) to be generated over the whole rest of the packet (including the NTS message object) and transported in some way. The NTS message object itself is an ASN.1 object of type "TimeRequestSecurityData", whose structure is as follows:

```
TimeRequestSecurityData ::=
SEQUENCE {
    nonce           NTSNonce,
    hmacHashAlgo    DigestAlgorithmIdentifier,
    keyInputValue   OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier:

```
id-ct-nts-securityDataReq OBJECT IDENTIFIER ::= TBD7
```

3.2.4.2. Message Type: "time_response"

This message is structured according to the "NTS-Plain" archetype.

It requires two items of data in addition to that which is transported in the NTS message object. Like "time_request", it requires regular time synchronization data. Furthermore, it requires the Message Authentication Code (MAC) to be generated over the whole rest of the packet (including the NTS message object) and transported in some way. The NTS message object itself is an ASN.1 object of type "TimeResponseSecurityData", with the following structure:

```
TimeResponseSecurityData ::=
SEQUENCE {
    nonce    NTSNonce,
}
```

It is identified by the following object identifier:

```
id-ct-nts-securityDataResp OBJECT IDENTIFIER ::= TBD8
```

3.3. Broadcast Messages

3.3.1. Broadcast Parameter Messages

3.3.1.1. Message Type: "client_bpar"

This first broadcast message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "BroadcastParameterRequest" and structured as follows:

```
BroadcastParameterRequest ::=
SEQUENCE {
    nonce     NTSNonce,
    clientId  SubjectKeyIdentifier
}
```

It is identified by the following object identifier:

```
id-ct-nts-broadcastParamReq OBJECT IDENTIFIER ::= TBD9
```

3.3.1.2. Message Type: "server_bpar"

This message is structured according to "NTS-Signed". It requires additional data besides that which is transported in the NTS message object, namely the signature is included in the appropriate field of the "SignedData" CMS structure that the NTS message object is wrapped in. The NTS message object itself is an ASN.1 object of type "BroadcastParameterResponse" and structured as follows:

```
BroadcastParameterResponse ::=
SEQUENCE {
    nonce               NTSNonce,
    oneWayAlgo1         DigestAlgorithmIdentifier,
    oneWayAlgo2         DigestAlgorithmIdentifier,
    lastKey             OCTET STRING (SIZE (16)),
    intervalDuration    BIT STRING,
    disclosureDelay     INTEGER,
    nextIntervalTime    BIT STRING,
    nextIntervalIndex   INTEGER
}
```

It is identified by the following object identifier:

```
id-ct-nts-broadcastParamResp OBJECT IDENTIFIER ::= TBD10
```

3.3.2. Broadcast Time Synchronization Message

3.3.2.1. Message Type: "server_broad"

This message is structured according to the "NTS-Plain" archetype. It requires regular broadcast time synchronization data in addition to that which is carried in the NTS message object. Like "time_response", this message type also requires a MAC, generated over all other data, to be transported within the packet. The NTS message object itself is an ASN.1 object of type "BroadcastTime". It has the following structure:

```
BroadcastTime ::=
SEQUENCE {
    thisIntervalIndex   INTEGER,
    disclosedKey        OCTET STRING (SIZE (16)),
}
```

It is identified by the following object identifier:

```
id-ct-nts-broadcastTime OBJECT IDENTIFIER ::= TBD11
```

3.3.3. Broadcast Keycheck

3.3.3.1. Message Type: "client_keycheck"

This message is structured according to the "NTS-Plain" archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientKeyCheckSecurityData" and structured as follows:

```
ClientKeyCheckSecurityData ::=
SEQUENCE {
    nonce_k           NTSNonce,
    interval_number   INTEGER,
    hmacHashAlgo      DigestAlgorithmIdentifier,
    keyInputValue     OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier:

```
id-ct-nts-clientKeyCheck OBJECT IDENTIFIER ::= TBD12
```

3.3.3.2. Message Type: "server_keycheck"

This message is also structured according to "NTS-Plain". It requires only a MAC, generated over the NTS message object, to be included in the packet in addition to what the NTS message object itself contains. The latter is an ASN.1 object of type "ServerKeyCheckSecurityData", which is structured as follows:

```
ServerKeyCheckSecurityData ::=
SEQUENCE {
    nonce             NTSNonce,
    interval_number   INTEGER
}
```

It is identified by the following object identifier:

```
id-ct-nts-serverKeyCheck OBJECT IDENTIFIER ::= TBD13
```

4. Certificate Conventions

The syntax and processing rules for certificates are specified in [RFC5280]. In the NTS protocol, the server certificate MUST contain the following extensions:

Subject Key Identifier -- see Section 4.2.1.2 of [RFC5280].

Key Usage -- see Section 4.2.1.3 of [RFC5280].

Extended Key Usage -- see Section 4.2.1.12 of [RFC5280].

For a certificate issued to a time server, the Extended Key Usage extension MAY include the id-kp-ntsServerAuth object identifier. When a certificate issuer includes this object identifier in the extended key usage extension, it provides an attestation that the certificate subject is a time server that supports the NTS protocol. The extension MAY also include the id-kp-ntsServerAuthz object

identifier. When a certificate issuer includes this object identifier in the extended key usage extension, it provides an attestation that the certificate subject is a time server which the issuer believes to be willing and able to disseminate correct time (for example, this can be used to signal a server's authorization to disseminate legal time).

For a certificate issued to a time client, the Extended Key Usage extension MAY include the id-kp-ntsClientAuthz object identifier. When a certificate issuer includes this object identifier in the extended key usage extension, it provides an attestation that the certificate subject is a time client who has authorization from the issuer to access secured time synchronization (for example, this can be used to provide access in the case of a paid service for secure time synchronization).

5. IANA Considerations

5.1. SMI Security for S/MIME Module Identifier Registry

Within the "SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0)" table, add one module identifier:

Decimal	Description	References
TBD0	id-networkTimeSecurity-2015	[this doc]

5.2. SMI Security for S/MIME CMS Content Type Registry

Within the "SMI Security for S/MIME CMS Content Type (1.2.840.113549.1.9.16.1)" table, add thirteen content type identifiers:

Decimal	Description	References
TBD1	id-ct-nts-clientAccess	[this doc]
TBD2	id-ct-nts-serverAccess	[this doc]
TBD3	id-ct-nts-clientAssoc	[this doc]
TBD4	id-ct-nts-serverAssoc	[this doc]
TBD5	id-ct-nts-clientCookie	[this doc]
TBD6	id-ct-nts-serverCookie	[this doc]
TBD7	id-ct-nts-securityDataReq	[this doc]
TBD8	id-ct-nts-securityDataResp	[this doc]
TBD9	id-ct-nts-broadcastParamReq	[this doc]
TBD10	id-ct-nts-broadcastParamResp	[this doc]
TBD11	id-ct-nts-broadcastTime	[this doc]
TBD12	id-ct-nts-clientKeyCheck	[this doc]
TBD13	id-ct-nts-serverKeyCheck	[this doc]

5.3. SMI Security for PKIX Extended Key Purpose Registry

Within the "SMI Security for PKIX Extended Key Purpose Identifiers (1.3.6.1.5.5.7.3)" table, add three key purpose identifiers:

Decimal	Description	References
TBD14	id-kp-ntsServerAuth	[this doc]
TBD15	id-kp-ntsServerAuthz	[this doc]
TBD16	id-kp-ntsClientAuthz	[this doc]

6. Security Considerations

For authentication the server's certificate MAY contain an extended key purpose identifier (id-kp-ntsServerAuth). Additionally the identifiers id-kp-ntsServerAuthz and id-kp-ntsClientAuthz MAY be used to grant the associated roles to the certified entity in the time dissemination infrastructure (see also Appendix D in [I-D.ietf-ntp-network-time-security]).

7. Acknowledgements

The authors would like to thank Harlan Stenn, Richard Welty and Martin Langer for their technical review and specific text contributions to this document.

8. References

8.1. Normative References

[ASN1] International Telecommunication Union, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, November 2008.

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

8.2. Informative References

[I-D.ietf-ntp-network-time-security]
Sibold, D., Roettger, S., and K. Teichel, "Network Time Security", draft-ietf-ntp-network-time-security-13 (work in progress), February 2016.

[IEEE1588]
IEEE Instrumentation and Measurement Society. TC-9 Sensor Technology, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems", 2008.

[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>.

Appendix A. ASN.1 Module

The ASN.1 module contained in this appendix defines the id-kp-NTSserver object identifier.

```
NTSserverKeyPurpose
  { TBD }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

id-kp-NTSserver OBJECT IDENTIFIER ::= { TBD17 }

END
```

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

Stephen Roettger
Google Inc.

Email: stephen.roettger@googlemail.com

Russ Housley
Vigil Security
918 Spring Knoll Drive
Herndon, VA 20170

Email: housley@vigilsec.com

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 26, 2017

D. Sibold
PTB
S. Roettger
Google Inc.
K. Teichel
PTB

September 22, 2016

Network Time Security
draft-ietf-ntp-network-time-security-15

Abstract

This document describes Network Time Security (NTS), a collection of measures that enable secure time synchronization with time servers using protocols like the Network Time Protocol (NTP) or the Precision Time Protocol (PTP). Its design considers the special requirements of precise timekeeping which are described in Security Requirements of Time Protocols in Packet Switched Networks [RFC7384].

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].

Status of This Memo

Table of Contents

1. Introduction

Time synchronization protocols are increasingly utilized to synchronize clocks in networked infrastructures. Successful attacks against the time synchronization protocol can seriously degrade the reliable performance of such infrastructures. Therefore, time synchronization protocols have to be secured if they are applied in environments that are prone to malicious attacks. This can be

accomplished either by utilization of external security protocols, like IPsec or TLS, or by intrinsic security measures of the time synchronization protocol.

The two most popular time synchronization protocols, the Network Time Protocol (NTP) [RFC5905] and the Precision Time Protocol (PTP) [IEEE1588], currently do not provide adequate intrinsic security precautions. This document specifies generic security measures which enable these and possibly other protocols to verify the authenticity of the time server/master and the integrity of the time synchronization protocol packets. The utilization of these measures for a given specific time synchronization protocol has to be described in a separate document.

[RFC7384] specifies that a security mechanism for timekeeping must be designed in such a way that it does not degrade the quality of the time transfer. This implies that for time keeping the increase in bandwidth and message latency caused by the security measures should be small. Also, NTP as well as PTP work via UDP and connections are stateless on the server/master side. Therefore, all security measures in this document are designed in such a way that they add little demand for bandwidth, that the necessary calculations can be executed in a fast manner, and that the measures do not require a server/master to keep state of a connection.

2. Terminology

2.1. Terms and Abbreviations

MITM Man In The Middle

NTS Network Time Security

TESLA Timed Efficient Stream Loss-tolerant Authentication

MAC Message Authentication Code

2.2. Common Terminology for PTP and NTP

This document refers to different time synchronization protocols, in particular to both the PTP and the NTP. Throughout the document the term "server" applies to both a PTP master and an NTP server. Accordingly, the term "client" applies to both a PTP slave and an NTP client.

3. Security Threats

The document "Security Requirements of Time Protocols in Packet Switched Networks" [RFC7384] contains a profound analysis of security threats and requirements for time synchronization protocols.

4. Objectives

The objectives of the NTS specification are as follows:

- Authenticity: NTS enables the client to authenticate its time server(s).
- Integrity: NTS protects the integrity of time synchronization protocol packets via a message authentication code (MAC).
- Confidentiality: NTS does not provide confidentiality protection of the time synchronization packets.
- Authorization: NTS enables the client to verify its time server's authorization. NTS optionally enables the server to verify the client's authorization as well.
- Request-Response-Consistency: NTS enables a client to match an incoming response to a request it has sent. NTS also enables the client to deduce from the response whether its request to the server has arrived without alteration.
- Applicability to Protocols: NTS can be used to secure different time synchronization protocols, specifically at least NTP and PTP.
- Integration with Protocols: A client or server running an NTS-secured version of a time protocol does not negatively affect other participants who are running unsecured versions of that protocol.
- Server-Side Statelessness: All security measures of NTS work without creating the necessity for a server to keep state of a connection.
- Prevention of Amplification Attacks: All communication introduced by NTS offers protection against abuse for amplification denial-of-service attacks.

5. NTS Overview

NTS initially verifies the authenticity of the time server and exchanges a symmetric key, the so-called cookie, as well as a key input value (KIV). The KIV can be opaque for the client. After the cookie and the KIV are exchanged, the client then uses them to protect the authenticity and the integrity of subsequent unicast-type time synchronization packets. In order to do this, a Message Authentication Code (MAC) is attached to each time synchronization packet. The calculation of the MAC includes the whole time synchronization packet and the cookie which is shared between client and server.

The cookie is calculated according to:

cookie = MSB_<b> (MAC(server seed, KIV)),

with the server seed as the key, where KIV is the client's key input value, and where the application of the function MSB_<b> returns only the b most significant bits. The server seed is a random value of bit length b that the server possesses, which has to remain secret. The cookie deterministically depends on KIV as long as the server seed stays the same. The server seed has to be refreshed periodically in order to provide key freshness as required in [RFC7384]. See Section 7 for details on seed refreshing.

Since the server does not keep a state of the client, it has to recalculate the cookie each time it receives a unicast time synchronization request from the client. To this end, the client has to attach its KIV to each request (see Section 6.1).

Note: The communication of the KIV and the cookie can be performed between client and server directly, or via a third party key distribution entity.

For broadcast-type messages, authenticity and integrity of the time synchronization packets are also ensured by a MAC, which is attached to the time synchronization packet by the sender. Verification of the broadcast-type packets' authenticity is based on the TESLA protocol, in particular on its "not re-using keys" scheme, see Section 3.7.2 of [RFC4082]. TESLA uses a one-way chain of keys, where each key is the output of a one-way function applied to the previous key in the chain. The server securely shares the last element of the chain with all clients. The server splits time into intervals of uniform duration and assigns each key to an interval in reverse order. At each time interval, the server sends a broadcast packet appended by a MAC, calculated using the corresponding key, and the key of the previous disclosure interval. The client verifies the

MAC by buffering the packet until disclosure of the key in its associated disclosure interval occurs. In order to be able to verify the timeliness of the packets, the client has to be loosely time synchronized with the server. This has to be accomplished before broadcast associations can be used. For checking timeliness of packets, NTS uses another, more rigorous check in addition to just the clock lookup used in the TESLA protocol. For a more detailed description of how NTS employs and customizes TESLA, see Appendix C.

6. Protocol Messages

This section describes the types of messages needed for secure time synchronization with NTS.

For some guidance on how these message types can be realized in practice, and integrated into the communication flow of existing time synchronization protocols, see [I-D.ietf-ntp-cms-for-nts-message], a companion document for NTS. Said document describes ASN.1 encodings for those message parts that have to be added to a time synchronization protocol for security reasons.

6.1. Unicast Time Synchronisation Messages

In this message exchange, the usual time synchronization process is executed, with the addition of integrity protection for all messages that the server sends. This message exchange can be repeatedly performed as often as the client desires and as long as the integrity of the server's time responses is verified successfully.

6.1.1. Preconditions for the Unicast Time Synchronization Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- They MUST negotiate the algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.

- The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.

- Client and server MUST exchange the cookie (which depends on the KIV as described in section Section 5). Authenticity, confidentiality and integrity of the communication MUST be ensured.

One way of realizing these requirements is to use the Association and Cookie Message Exchanges described in Appendix B.

6.1.2. Goals of the Unicast Time Synchronization Exchange

The unicast time synchronization exchange:

- exchanges time synchronization data as specified by the appropriate time synchronization protocol,
- guarantees authenticity and integrity of the request to the server,
- guarantees authenticity and integrity of the response to the client,
- guarantees request-response-consistency to the client.

6.1.3. Message Type: "time_request"

This message is sent by the client when it requests a time exchange. It contains

- the NTS message ID "time_request",
- the negotiated version number,
- a nonce,
- the negotiated MAC algorithm,
- the client's key input value (for which the client knows the associated cookie),
- optional: a MAC (generated with the cookie as key) for verification of all of the above data.

6.1.4. Message Type: "time_response"

This message is sent by the server after it has received a time_request message. Prior to this the server MUST recalculate the client's cookie by using the received key input value and the transmitted MAC algorithm. The message contains

- the NTS message ID "time_response",
- the version number as transmitted in time_request,
- the server's time synchronization response data,
- the nonce transmitted in time_request,

- o a MAC (generated with the cookie as key) for verification of all of the above data.

6.1.5. Procedure Overview of the Unicast Time Synchronization Exchange

For a unicast time synchronization exchange, the following steps are performed:

1. The client sends a time_request message to the server. The client MUST save the included nonce and the transmit_timestamp (from the time synchronization data) as a correlated pair for later verification steps. Optionally, the client protects the request message with an appended MAC.

2. Upon receipt of a time_request message, the server performs the following steps:

    * It re-calculates the cookie.

    * If the request message contains a MAC the server re-calculates the MAC and compares this value with the MAC in the received data.

        + If the re-calculated MAC does not match the MAC in the received data the server MUST stop the processing of the request.

        + If the re-calculated MAC matches the MAC in the received data the server continues to process the request.

    * The server computes the necessary time synchronization data and constructs a time_response message as given in Section 6.1.4.

3. The client awaits a reply in the form of a time_response message. Upon receipt, it checks:

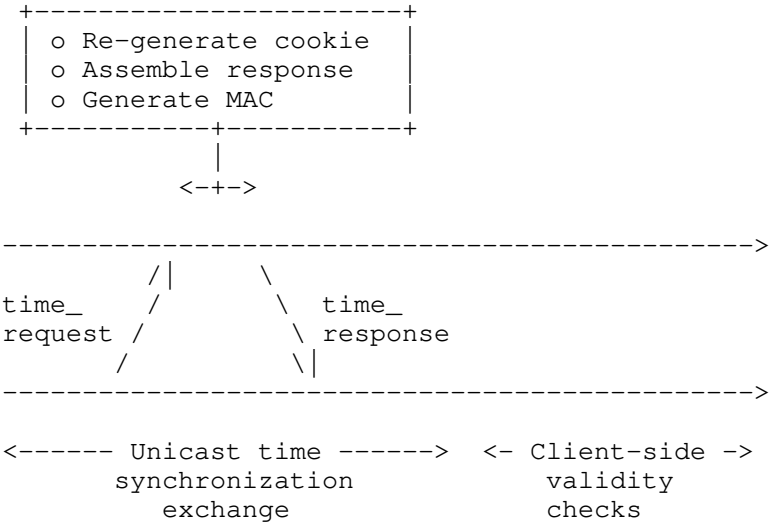
    * that the transmitted version number matches the one negotiated previously,

    * that the transmitted nonce belongs to a previous time_request message,

    * that the transmit_timestamp in that time_request message matches the corresponding time stamp from the synchronization data received in the time_response, and

* that the appended MAC verifies the received synchronization data, version number and nonce.

If at least one of the first three checks fails (i.e. if the version number does not match, if the client has never used the nonce transmitted in the time_response message, or if it has used the nonce with initial time synchronization data different from that in the response), then the client MUST ignore this time_response message. If the MAC is invalid, the client MUST do one of the following: abort the run or send another cookie request (because the cookie might have changed due to a server seed refresh). If both checks are successful, the client SHOULD continue time synchronization.



Procedure for unicast time synchronization exchange.

6.2. Broadcast Time Synchronization Exchange

6.2.1. Preconditions for the Broadcast Time Synchronization Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- The client MUST receive all the information necessary to process broadcast time synchronization messages from the server. This includes

  * the one-way functions used for building the key chain,

    * the last key of the key chain,

    * time interval duration,

    * the disclosure delay (number of intervals between use and disclosure of a key),

    * the time at which the next time interval will start, and

    * the next interval's associated index.

  o The communication of the data listed above MUST guarantee authenticity of the server, as well as integrity and freshness of the broadcast parameters to the client.

6.2.2. Goals of the Broadcast Time Synchronization Exchange

The broadcast time synchronization exchange:

- transmits (broadcast) time synchronization data from the server to the client as specified by the appropriate time synchronization protocol,
- guarantees to the client that the received synchronization data has arrived in a timely manner as required by the TESLA protocol and is trustworthy enough to be stored for later checks,
- additionally guarantees authenticity of a certain broadcast synchronization message in the client's storage.

6.2.3. Message Type: "server_broad"

This message is sent by the server over the course of its broadcast schedule. It is part of any broadcast association. It contains

- the NTS message ID "server_broad",
- the version number that the server is working under,
- time broadcast data,
- the index that belongs to the current interval (and therefore identifies the current, yet undisclosed, key),
- the disclosed key of the previous disclosure interval (current time interval minus disclosure delay),

- o a MAC, calculated with the key for the current time interval, verifying
  - * the message ID,
  - * the version number, and
  - * the time data.

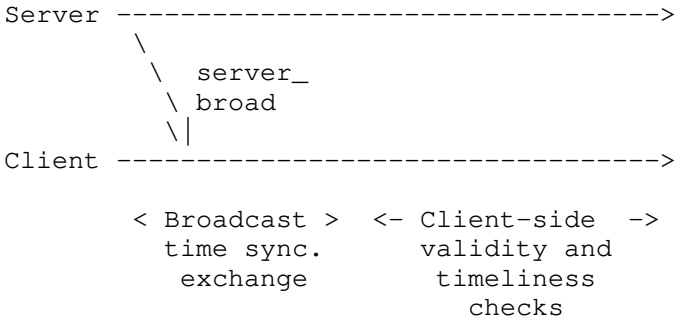
6.2.4. Procedure Overview of Broadcast Time Synchronization Exchange

A broadcast time synchronization message exchange consists of the following steps:

1. The server follows the TESLA protocol by regularly sending server_broad messages as described in Section 6.2.3, adhering to its own disclosure schedule.
2. The client awaits time synchronization data in the form of a server_broadcast message. Upon receipt, it performs the following checks:
   - * Proof that the MAC is based on a key that is not yet disclosed (packet timeliness). This is achieved via a combination of checks. First, the disclosure schedule is used, which requires loose time synchronization. If this is successful, the client obtains a stronger guarantee via a key check exchange (see below). If its timeliness is verified, the packet will be buffered for later authentication. Otherwise, the client MUST discard it. Note that the time information included in the packet will not be used for synchronization until its authenticity could also be verified.
   - * The client checks that it does not already know the disclosed key. Otherwise, the client SHOULD discard the packet to avoid a buffer overrun. If this check is successful, the client ensures that the disclosed key belongs to the one-way key chain by applying the one-way function until equality with a previous disclosed key is shown. If it is falsified, the client MUST discard the packet.
   - * If the disclosed key is legitimate, then the client verifies the authenticity of any packet that it has received during the corresponding time interval. If authenticity of a packet is verified, then it is released from the buffer and its time information can be utilized. If the verification fails, then authenticity is not given. In this case, the client MUST request authentic time from the server by means other than

broadcast messages. Also, the client MUST re-initialize the broadcast sequence with a "client_bpar" message if the one-way key chain expires, which it can check via the disclosure schedule.

See RFC 4082[RFC4082] for a detailed description of the packet verification process.



Procedure for broadcast time synchronization exchange.

6.3. Broadcast Keycheck

This message exchange is performed for an additional check of packet timeliness in the course of the TESLA scheme, see Appendix C.

6.3.1. Preconditions for the Broadcast Keycheck Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- They MUST negotiate the algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.

- The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.

- Client and server MUST exchange the cookie (which depends on the KIV as described in section Section 5). Authenticity, confidentiality and integrity of the communication MUST be ensured.

These requirements conform to those for the unicast time synchronization exchange. Accordingly, they too can be realized via the Association and Cookie Message Exchanges described in Appendix B (Appendix B).

6.3.2. Goals of the Broadcast Keycheck Exchange

The keycheck exchange:

- guarantees to the client that the key belonging to the respective TESLA interval communicated in the exchange had not been disclosed before the client_keycheck message was sent.

- guarantees to the client the timeliness of any broadcast packet secured with this key if it arrived before client_keycheck was sent.

6.3.3. Message Type: "client_keycheck"

A message of this type is sent by the client in order to initiate an additional check of packet timeliness for the TESLA scheme. It contains

- the NTS message ID "client_keycheck",

- the NTS version number negotiated during association,

- a nonce,

- an interval number from the TESLA disclosure schedule,

- the MAC algorithm negotiated during association,

- the client's key input value KIV, and

- optional: a MAC (generated with the cookie as key) for verification of all of the above data.

6.3.4. Message Type: "server_keycheck"

A message of this type is sent by the server upon receipt of a client_keycheck message during the broadcast loop of the server. Prior to this, the server MUST recalculate the client's cookie by using the received key input value and the transmitted MAC algorithm. It contains

- the NTS message ID "server_keycheck"

- the version number as transmitted in "client_keycheck,

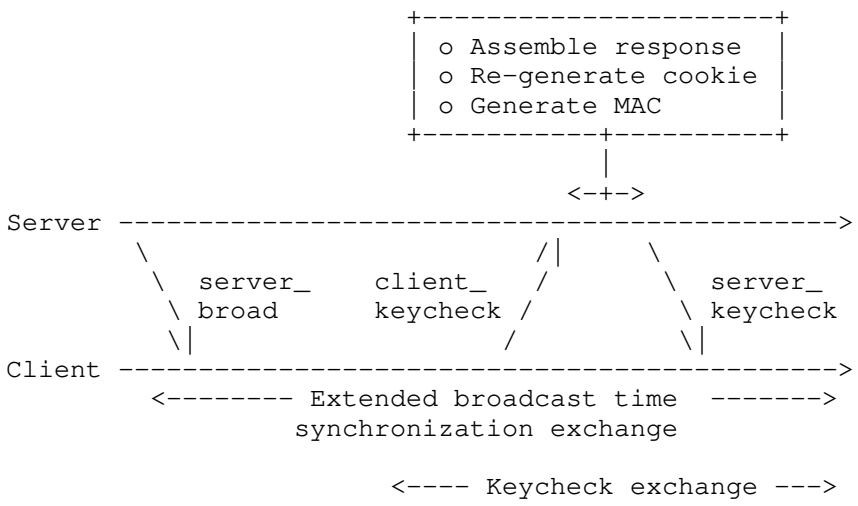
- the nonce transmitted in the client_keycheck message,

- the interval number transmitted in the client_keycheck message, and
- a MAC (generated with the cookie as key) for verification of all of the above data.

6.3.5. Procedure Overview of the Broadcast Keycheck Exchange

A broadcast keycheck message exchange consists of the following steps:

1. The client sends a client_keycheck message. It MUST memorize the nonce and the time interval number that it sends as a correlated pair.
2. Upon receipt of a client_keycheck message the server performs as follows: If the client_keycheck message contains a MAC the server re-calculates the MAC and compares this value with the MAC in the received data.
    * If the re-calculated MAC does not match the MAC in the received data the server MUST stop the processing of the request.
    * If the re-calculated MAC matches the MAC in the received data the server continues to process the request: It looks up whether it has already disclosed the key associated with the interval number transmitted in that message. If it has not disclosed it, it constructs and sends the appropriate server_keycheck message as described in Section 6.3.4. For more details, see also Appendix C.
3. The client awaits a reply in the form of a server_keycheck message. On receipt, it performs the following checks:
    * that the transmitted version number matches the one negotiated previously,
    * that the transmitted nonce belongs to a previous client_keycheck message,
    * that the TESLA interval number in that client_keycheck message matches the corresponding interval number from the server_keycheck, and
    * that the appended MAC verifies the received data.



Procedure for extended broadcast time synchronization exchange.

7. Server Seed, MAC Algorithms and Generating MACs

7.1. Server Seed

The server has to calculate a random seed which has to be kept secret. The server MUST generate a seed for each supported MAC algorithm, see Section 7.2.

According to the requirements in [RFC7384], the server MUST refresh each server seed periodically. Consequently, the cookie memorized by the client becomes obsolete. In this case, the client cannot verify the MAC attached to subsequent time response messages and has to respond accordingly by re-initiating the protocol with a cookie request (Appendix B.4).

7.2. MAC Algorithms

MAC algorithms are used for calculation of the cookie and the actual MAC. The client and the server negotiate a MAC algorithm during the association phase at the beginning. The selected algorithm MUST be used for all cookie and MAC creation processes in that run.

Note: Any MAC algorithm is prone to be compromised in the future. A successful attack on a MAC algorithm would enable any NTS client to derive the server seed from its own cookie. Therefore, the server MUST have separate seed values for its different supported MAC algorithms. This way, knowledge gained from an attack on a

MAC algorithm can at least only be used to compromise such clients who use this algorithm as well.

8. IANA Considerations

As mentioned, this document generically specifies security measures whose utilization for any given specific time synchronization protocol requires a separate document. Consequently, this document itself does not have any IANA actions (TO BE REVIEWED).

9. Security Considerations

Aspects of security for time synchronization protocols are treated throughout this document. For a comprehensive discussion of security requirements in time synchronization contexts, refer to [RFC7384]. See Appendix A for a tabular overview of how NTS deals with those requirements.

Additional NTS specific discussion of security issues can be found in the following subsections.

Note: Any separate document describing the utilization of NTS to a specific time synchronization protocol may additionally introduce discussion of its own specific security considerations.

9.1. Privacy

The payload of time synchronization protocol packets of two-way time transfer approaches like NTP and PTP consists basically of time stamps, which are not considered secret [RFC7384]. Therefore, encryption of the time synchronization protocol packet's payload is not considered in this document. However, an attacker can exploit the exchange of time synchronization protocol packets for topology detection and inference attacks as described in [RFC7624]. To make such attacks more difficult, that draft recommends the encryption of the packet payload. Yet, in the case of time synchronization protocols the confidentiality protection of time synchronization packet's payload is of secondary importance since the packet's meta data (IP addresses, port numbers, possibly packet size and regular sending intervals) carry more information than the payload. To enhance the privacy of the time synchronization partners, the usage of tunnel protocols such as IPsec and MACsec, where applicable, is therefore more suited than confidentiality protection of the payload.

9.2. Initial Verification of the Server Certificates

The client may wish to verify the validity of certificates during the initial association phase. Since it generally has no reliable time during this initial communication phase, it is impossible to verify the period of validity of the certificates. To solve this chicken-and-egg problem, the client has to rely on external means.

9.3. Revocation of Server Certificates

According to Section 7, it is the client's responsibility to initiate a new association with the server after the server's certificate expires. To this end, the client reads the expiration date of the certificate during the certificate message exchange (Appendix B.3.3). Furthermore, certificates may also be revoked prior to the normal expiration date. To increase security the client MAY periodically verify the state of the server's certificate via Online Certificate Status Protocol (OCSP) Online Certificate Status Protocol (OCSP) [RFC6960].

9.4. Mitigating Denial-of-Service for broadcast packets

TESLA authentication buffers packets for delayed authentication. This makes the protocol vulnerable to flooding attacks, causing the client to buffer excessive numbers of packets. To add stronger DoS protection to the protocol, the client and the server use the "not re-using keys" scheme of TESLA as pointed out in Section 3.7.2 of RFC 4082 [RFC4082]. In this scheme the server never uses a key for the MAC generation more than once. Therefore, the client can discard any packet that contains a disclosed key it already knows, thus preventing memory flooding attacks.

Discussion: Note that an alternative approach to enhance TESLA's resistance against DoS attacks involves the addition of a group MAC to each packet. This requires the exchange of an additional shared key common to the whole group. This adds additional complexity to the protocol and hence is currently not considered in this document.

9.5. Delay Attack

In a packet delay attack, an adversary with the ability to act as a MITM delays time synchronization packets between client and server asymmetrically [RFC7384]. This prevents the client from accurately measuring the network delay, and hence its time offset to the server [Mizrahi]. The delay attack does not modify the content of the exchanged synchronization packets. Therefore, cryptographic means do not provide a feasible way to mitigate this attack. However, several

non-cryptographic precautions can be taken in order to detect this attack.

1. Usage of multiple time servers: this enables the client to detect the attack, provided that the adversary is unable to delay the synchronization packets between the majority of servers. This approach is commonly used in NTP to exclude incorrect time servers [RFC5905].

2. Multiple communication paths: The client and server utilize different paths for packet exchange as described in the I-D [I-D.ietf-tictoc-multi-path-synchronization]. The client can detect the attack, provided that the adversary is unable to manipulate the majority of the available paths [Shpiner]. Note that this approach is not yet available, neither for NTP nor for PTP.

3. Usage of an encrypted connection: the client exchanges all packets with the time server over an encrypted connection (e.g. IPsec). This measure does not mitigate the delay attack, but it makes it more difficult for the adversary to identify the time synchronization packets.

4. For unicast-type messages: Introduction of a threshold value for the delay time of the synchronization packets. The client can discard a time server if the packet delay time of this time server is larger than the threshold value.

Additional provision against delay attacks has to be taken for broadcast-type messages. This mode relies on the TESLA scheme which is based on the requirement that a client and the broadcast server are loosely time synchronized. Therefore, a broadcast client has to establish time synchronization with its broadcast server before it starts utilizing broadcast messages for time synchronization.

One possible way to achieve this initial synchronization is to establish a unicast association with its broadcast server until time synchronization and calibration of the packet delay time is achieved. After that, the client can establish a broadcast association with the broadcast server and utilizes TESLA to verify integrity and authenticity of any received broadcast packets.

An adversary who is able to delay broadcast packets can cause a time adjustment at the receiving broadcast clients. If the adversary delays broadcast packets continuously, then the time adjustment will accumulate until the loose time synchronization requirement is violated, which breaks the TESLA scheme. To mitigate this vulnerability the security condition in TESLA has to be supplemented

by an additional check in which the client, upon receipt of a broadcast message, verifies the status of the corresponding key via a unicast message exchange with the broadcast server (see Appendix C.4 for a detailed description of this check). Note that a broadcast client should also apply the above-mentioned precautions as far as possible.

9.6. Random Number Generation

At various points of the protocol, the generation of random numbers is required. The employed methods of generation need to be cryptographically secure. See [RFC4086] for guidelines concerning this topic.

10. Acknowledgements

The authors would like to thank Tal Mizrahi, Russ Housley, Steven Bellovin, David Mills, Kurt Roeckx, Rainer Bermbach, Martin Langer and Florian Weimer for discussions and comments on the design of NTS. Also, thanks go to Harlan Stenn and Richard Welty for their technical review and specific text contributions to this document.

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>.

[RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, DOI 10.17487/RFC4082, June 2005, <http://www.rfc-editor.org/info/rfc4082>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <http://www.rfc-editor.org/info/rfc7384>.

11.2. Informative References

[I-D.ietf-ntp-cms-for-nts-message]
Sibold, D., Teichel, K., Roettger, S., and R. Housley, "Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS)", draft-ietf-ntp-cms-for-nts-message-04 (work in progress), July 2015.

[I-D.ietf-tictoc-multi-path-synchronization]
Shpiner, A., Tse, R., Schelp, C., and T. Mizrahi, "Multi-Path Time Synchronization", draft-ietf-tictoc-multi-path-synchronization-02 (work in progress), April 2015.

[IEEE1588]
IEEE Instrumentation and Measurement Society. TC-9 Sensor Technology, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems", 2008.

[Mizrahi] Mizrahi, T., "A game theoretic analysis of delay attacks against time synchronization protocols", in Proceedings of Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2012, pp. 1-6, September 2012.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[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>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <http://www.rfc-editor.org/info/rfc6960>.

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <http://www.rfc-editor.org/info/rfc7624>.

[Shpiner] Shpiner, A., Revah, Y., and T. Mizrahi, "Multi-path Time Protocols", in Proceedings of Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2013, pp. 1-6, September 2013.

Appendix A. (informative) TICTOC Security Requirements

The following table compares the NTS specifications against the TICTOC security requirements [RFC7384].

Section	Requirement from RFC 7384	Requirement level	NTS
5.1.1	Authentication of Servers	MUST	OK
5.1.1	Authorization of Servers	MUST	OK
5.1.2	Recursive Authentication of Servers (Stratum 1)	MUST	OK
5.1.2	Recursive Authorization of Servers (Stratum 1)	MUST	OK
5.1.3	Authentication and Authorization of Clients	MAY	Optional, Limited
5.2	Integrity protection	MUST	OK
5.3	Spoofing Prevention	MUST	OK
5.4	Protection from DoS attacks against the time protocol	SHOULD	OK
5.5	Replay protection	MUST	OK
5.6	Key freshness	MUST	OK
	Security association	SHOULD	OK
	Unicast and multicast associations	SHOULD	OK
5.7	Performance: no degradation in quality of time transfer	MUST	OK
	Performance: lightweight computation	SHOULD	OK

	Performance: storage	SHOULD	OK
	Performance: bandwidth	SHOULD	OK
5.8	Confidentiality protection	MAY	NO
5.9	Protection against Packet Delay and Interception Attacks	MUST	Limited*)
5.10	Secure mode	MUST	OK
	Hybrid mode	SHOULD	-

*) See discussion in Section 9.5.

Comparison of NTS specification against Security Requirements of Time Protocols in Packet Switched Networks (RFC 7384)

Appendix B. (normative) Inherent Association Protocol Messages

This appendix presents a procedure that performs the association, the cookie, and also the broadcast parameter message exchanges between a client and a server. This procedure is one possible way to achieve the preconditions listed in Sections Section 6.1.1, Section 6.2.1, and Section 6.3.1 while taking into account the objectives given in Section Section 4.

B.1. Overview of NTS with Inherent Association Protocol

This inherent association protocol applies X.509 certificates to verify the authenticity of the time server and to exchange the cookie. This is done in two separate message exchanges, described below. An additional required exchange in advance serves to limit the amplification potential of the association message exchange.

A client needs a public/private key pair for encryption, with the public key enclosed in a certificate. A server needs a public/ private key pair for signing, with the public key enclosed in a certificate. If a participant intends to act as both a client and a server, it MUST have two different key pairs for these purposes.

If this protocol is employed, the hash value of the client's certificate is used as the client's key input value, i.e. the cookie is calculated according to:

cookie = MSB_<b> (MAC(server seed, H(certificate of client))),

Where the hash function H is the one used in the MAC algorithm. The client's certificate contains the client's public key and enables the server to identify the client, if client authorization is desired.

B.2. Access Message Exchange

This message exchange serves only to prevent the next (association) exchange from being abusable for amplification denial-of-service attacks.

B.2.1. Goals of the Access Message Exchange

The access message exchange:

- transfers a secret value from the server to the client (initiator),
- the secret value permits the client to initiate an association message exchange.

B.2.2. Message Type: "client_access"

This message is sent by a client who intends to perform an association exchange with the server in the future. It contains:

- the NTS message ID "client_access".

B.2.3. Message Type: "server_access"

This message is sent by the server on receipt of a client_access message. It contains:

- the NTS message ID "server_access",
- an access key.

B.2.4. Procedure Overview of the Access Exchange

For an access exchange, the following steps are performed:

1. The client sends a client_access message to the server.
2. Upon receipt of a client_access, the server calculates the access key. It then sends a reply in the form of a server_access message. The server must either memorize the access key or alternatively apply a means by which it can reconstruct the

access key. Note that in both cases the access key must be correlated with the address of the requester. Note also that if the server memorizes the access key for a requester, it has to keep state for a certain amount of time.

3. The client waits for a response in the form of a server_access message. Upon receipt of one, it MUST memorize the included access key.

B.3. Association Message Exchange

In this message exchange, the participants negotiate the MAC and encryption algorithms that are used throughout the protocol. In addition, the client receives the certification chain up to a trusted anchor. With the established certification chain the client is able to verify the server's signatures and, hence, the authenticity of future NTS messages from the server is ensured.

B.3.1. Goals of the Association Exchange

The association exchange:

- enables the client to verify any communication with the server as authentic,
- lets the participants negotiate NTS version and algorithms,
- guarantees authenticity and integrity of the negotiation result to the client,
- guarantees to the client that the negotiation result is based on the client's original, unaltered request.

B.3.2. Message Type: "client_assoc"

This message is sent by the client if it wants to perform association with a server. It contains

- the NTS message ID "client_assoc",
- a nonce,
- the access key obtained earlier via an access message exchange,
- the version number of NTS that the client wants to use (this SHOULD be the highest version number that it supports),
- a selection of accepted MAC algorithms, and

- a selection of accepted encryption algorithms.

B.3.3. Message Type: "server_assoc"

This message is sent by the server upon receipt of client_assoc. It contains

- the NTS message ID "server_assoc",
- the nonce transmitted in client_assoc,
- the client's proposal for the version number, selection of accepted MAC algorithms and selection of accepted encryption algorithms, as transmitted in client_assoc,
- the version number used for the rest of the protocol (which SHOULD be determined as the minimum over the client's suggestion in the client_assoc message and the highest supported by the server),
- the server's choice of algorithm for encryption and for MAC creation, all of which MUST be chosen from the client's proposals,
- a signature, calculated over the data listed above, with the server's private key and according to the signature algorithm which is also used for the certificates that are included (see below), and
- a chain of certificates, which starts at the server and goes up to a trusted authority; each certificate MUST be certified by the one directly following it.

B.3.4. Procedure Overview of the Association Exchange

For an association exchange, the following steps are performed:

1. The client sends a client_assoc message to the server. It MUST keep the transmitted values for the version number and algorithms available for later checks.
2. Upon receipt of a client_assoc message, the server checks the validity of the included access key. If it is not valid, the server MUST abort communication. If it is valid, the server constructs and sends a reply in the form of a server_assoc message as described in Appendix B.3.3. Upon unsuccessful negotiation for version number or algorithms the server_assoc message MUST contain an error code.

3. The client waits for a reply in the form of a server_assoc message. After receipt of the message it performs the following checks:

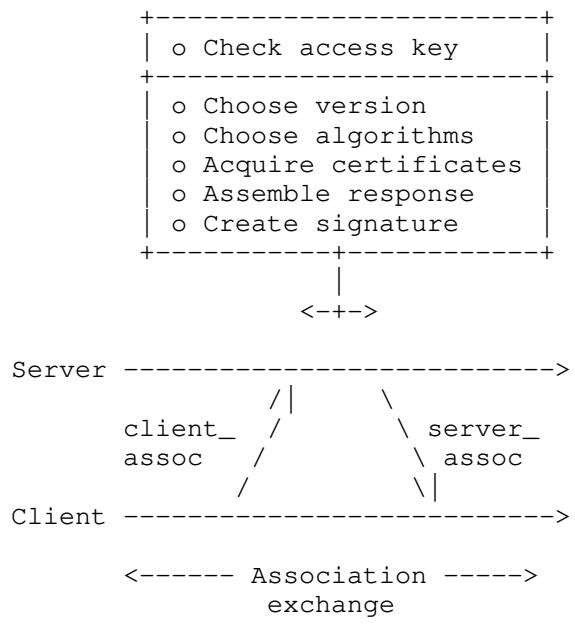
   * The client checks that the message contains a conforming version number.

   * It checks that the nonce sent back by the server matches the one transmitted in client_assoc,

   * It also verifies that the server has chosen the encryption and MAC algorithms from its proposal sent in the client_assoc message and that this proposal was not altered.

   * Furthermore, it performs authenticity checks on the certificate chain and the signature.

   If one of the checks fails, the client MUST abort the run.



Procedure for association and cookie exchange.

B.4. Cookie Message Exchange

During this message exchange, the server transmits a secret cookie to the client securely. The cookie will later be used for integrity protection during unicast time synchronization.

B.4.1. Goals of the Cookie Exchange

The cookie exchange:

- enables the server to check the client's authorization via its certificate (optional),
- supplies the client with the correct cookie and corresponding KIV for its association to the server,
- guarantees to the client that the cookie originates from the server and that it is based on the client's original, unaltered request.
- guarantees that the received cookie is unknown to anyone but the server and the client.

B.4.2. Message Type: "client_cook"

This message is sent by the client upon successful authentication of the server. In this message, the client requests a cookie from the server. The message contains

- the NTS message ID "client_cook",
- a nonce,
- the negotiated version number,
- the negotiated signature algorithm,
- the negotiated encryption algorithm,
- the negotiated MAC algorithm,
- the client's certificate.

B.4.3. Message Type: "server_cook"

This message is sent by the server upon receipt of a client_cook message. The server generates the hash (the used hash function is the one used for the MAC algorithm) of the client's certificate, as conveyed during client_cook, in order to calculate the cookie according to Section 5. This message contains

- the NTS message ID "server_cook"
- the version number as transmitted in client_cook,

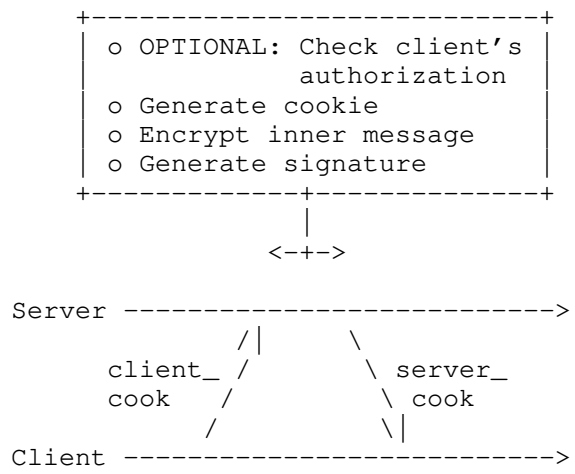
- a concatenated datum which is encrypted with the client's public key, according to the encryption algorithm transmitted in the client_cook message. The concatenated datum contains
  - the nonce transmitted in client_cook, and
  - the cookie.
- a signature, created with the server's private key, calculated over all of the data listed above. This signature MUST be calculated according to the transmitted signature algorithm from the client_cook message.

B.4.4. Procedure Overview of the Cookie Exchange

For a cookie exchange, the following steps are performed:

1. The client sends a client_cook message to the server. The client MUST save the included nonce until the reply has been processed.
2. Upon receipt of a client_cook message, the server checks whether it supports the given cryptographic algorithms. It then calculates the cookie according to the formula given in Section 5. The server MAY use the client's certificate to check that the client is authorized to use the secure time synchronization service. With this, it MUST construct a server_cook message as described in Appendix B.4.3.
3. The client awaits a reply in the form of a server_cook message; upon receipt it executes the following actions:
   - It verifies that the received version number matches the one negotiated beforehand.
   - It verifies the signature using the server's public key. The signature has to authenticate the encrypted data.
   - It decrypts the encrypted data with its own private key.
   - It checks that the decrypted message is of the expected format: the concatenation of a nonce and a cookie of the expected bit lengths.
   - It verifies that the received nonce matches the nonce sent in the client_cook message.

   If one of those checks fails, the client MUST abort the run.



Procedure for association and cookie exchange.

B.4.5. Broadcast Parameter Messages

In this message exchange, the client receives the necessary information to execute the TESLA protocol in a secured broadcast association. The client can only initiate a secure broadcast association after successful association and cookie exchanges and only if it has made sure that its clock is roughly synchronized to the server's.

See Appendix C for more details on TESLA.

B.4.5.1. Goals of the Broadcast Parameter Exchange

The broadcast parameter exchange

- provides the client with all the information necessary to process broadcast time synchronization messages from the server, and
- guarantees authenticity, integrity and freshness of the broadcast parameters to the client.

B.4.5.2. Message Type: "client_bpar"

This message is sent by the client in order to establish a secured time broadcast association with the server. It contains

- the NTS message ID "client_bpar",

- the NTS version number negotiated during association,
- a nonce, and
- the signature algorithm negotiated during association.

B.4.5.3. Message Type: "server_bpar"

This message is sent by the server upon receipt of a client_bpar message during the broadcast loop of the server. It contains

- the NTS message ID "server_bpar",
- the version number as transmitted in the client_bpar message,
- the nonce transmitted in client_bpar,
- the one-way functions used for building the key chain, and
- the disclosure schedule of the keys. This contains:
  - the last key of the key chain,
  - time interval duration,
  - the disclosure delay (number of intervals between use and disclosure of a key),
  - the time at which the next time interval will start, and
  - the next interval's associated index.
- The message also contains a signature signed by the server with its private key, verifying all the data listed above.

B.4.5.4. Procedure Overview of the Broadcast Parameter Exchange

A broadcast parameter exchange consists of the following steps:

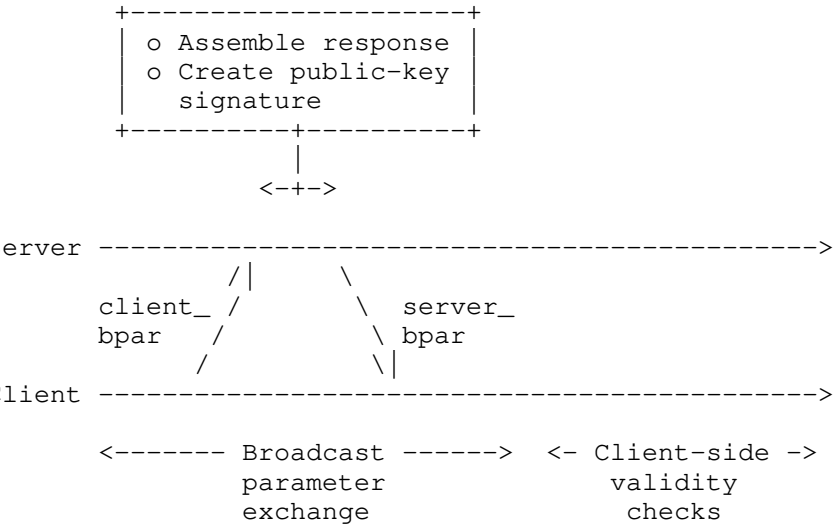
1. The client sends a client_bpar message to the server. It MUST remember the transmitted values for the nonce, the version number and the signature algorithm.
2. Upon receipt of a client_bpar message, the server constructs and sends a server_bpar message as described in Appendix B.4.5.3.
3. The client waits for a reply in the form of a server_bpar message, on which it performs the following checks:

* The message must contain all the necessary information for the TESLA protocol, as listed in Appendix B.4.5.3.

* The message must contain a nonce belonging to a client_bpar message that the client has previously sent.

* Verification of the message's signature.

If any information is missing or if the server's signature cannot be verified, the client MUST abort the broadcast run. If all checks are successful, the client MUST remember all the broadcast parameters received for later checks.



Procedure for unicast time synchronization exchange.

Appendix C. (normative) Using TESLA for Broadcast-Type Messages

For broadcast-type messages, NTS adopts the TESLA protocol with some customizations. This appendix provides details on the generation and usage of the one-way key chain collected and assembled from [RFC4082]. Note that NTS uses the "not re-using keys" scheme of TESLA as described in Section 3.7.2. of [RFC4082].

C.1. Server Preparation

Server setup:

1. The server determines a reasonable upper bound B on the network delay between itself and an arbitrary client, measured in milliseconds.

2. It determines the number n+1 of keys in the one-way key chain. This yields the number n of keys that are usable to authenticate broadcast packets. This number n is therefore also the number of time intervals during which the server can send authenticated broadcast messages before it has to calculate a new key chain.

3. It divides time into n uniform intervals I_1, I_2, ..., I_n. Each of these time intervals has length L, measured in milliseconds. In order to fulfill the requirement 3.7.2. of RFC 4082, the time interval L has to be shorter than the time interval between the broadcast messages.

4. The server generates a random key K_n.

5. Using a one-way function F, the server generates a one-way chain of n+1 keys K_0, K_1, ..., K_{n} according to

   K_i = F(K_{i+1}).

6. Using another one-way function F', it generates a sequence of n MAC keys K'_0, K'_1, ..., K'_{n-1} according to

   K'_i = F'(K_i).

7. Each MAC key K'_i is assigned to the time interval I_i.

8. The server determines the key disclosure delay d, which is the number of intervals between using a key and disclosing it. Note that although security is provided for all choices d>0, the choice still makes a difference:

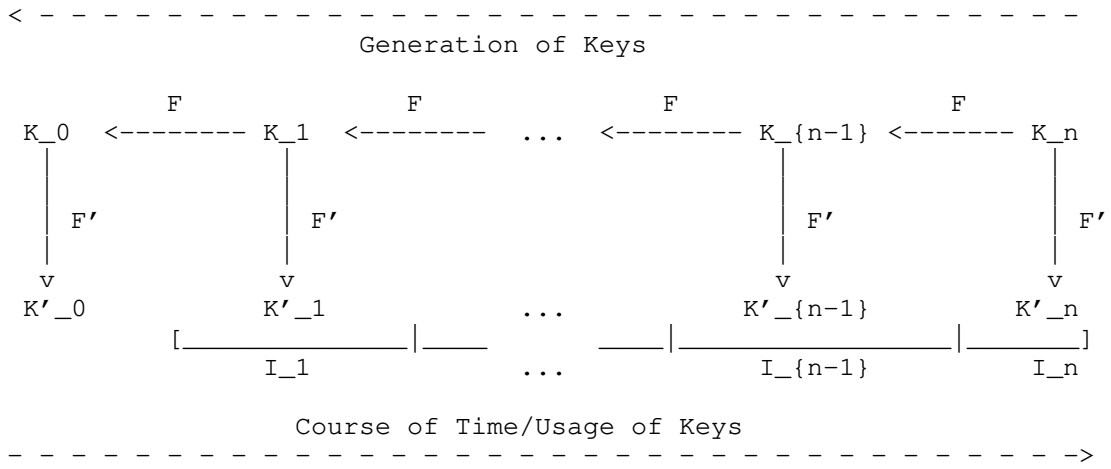
   * If d is chosen too short, the client might discard packets because it fails to verify that the key used for its MAC has not yet been disclosed.

   * If d is chosen too long, the received packets have to be buffered for an unnecessarily long time before they can be verified by the client and be subsequently utilized for time synchronization.

   It is RECOMMENDED that the server calculate d according to

   d = ceil(2*B / L) + 1,

where ceil yields the smallest integer greater than or equal to its argument.



A schematic explanation of the TESLA protocol's one-way key chain

C.2. Client Preparation

A client needs the following information in order to participate in a TESLA broadcast:

- One key K_i from the one-way key chain, which has to be authenticated as belonging to the server. Typically, this will be K_0.
- The disclosure schedule of the keys. This consists of:
  - the length n of the one-way key chain,
  - the length L of the time intervals I_1, I_2, ..., I_n,
  - the starting time T_i of an interval I_i. Typically this is the starting time T_1 of the first interval;
  - the disclosure delay d.
- The one-way function F used to recursively derive the keys in the one-way key chain,
- The second one-way function F' used to derive the MAC keys K'_0, K'_1, ... , K'_n from the keys in the one-way chain.

- An upper bound D_t on how far its own clock is "behind" that of the server.

Note that if D_t is greater than (d - 1) * L, then some authentic packets might be discarded. If D_t is greater than d * L, then all authentic packets will be discarded. In the latter case, the client SHOULD NOT participate in the broadcast, since there will be no benefit in doing so.

C.3. Sending Authenticated Broadcast Packets

During each time interval I_i, the server sends at most one authenticated broadcast packet P_i. Such a packet consists of:

- a message M_i,
- the index i (in case a packet arrives late),
- a MAC authenticating the message M_i, with K'_i used as key,
- the key K_{i-d}, which is included for disclosure.

C.4. Authentication of Received Packets

When a client receives a packet P_i as described above, it first checks that it has not already received a packet with the same disclosed key. This is done to avoid replay/flooding attacks. A packet that fails this test is discarded.

Next, the client begins to check the packet's timeliness by ensuring that according to the disclosure schedule and with respect to the upper bound D_t determined above, the server cannot have disclosed the key K_i yet. Specifically, it needs to check that the server's clock cannot read a time that is in time interval I_{i+d} or later. Since it works under the assumption that the server's clock is not more than D_t "ahead" of the client's clock, the client can calculate an upper bound t_i for the server's clock at the time when P_i arrived. This upper bound t_i is calculated according to

$$t_i = R + D_t,$$

where R is the client's clock at the arrival of P_i. This implies that at the time of arrival of P_i, the server could have been in interval I_x at most, with

$$x = \text{floor}((t_i - T_1) / L) + 1,$$

where floor gives the greatest integer less than or equal to its argument. The client now needs to verify that

x < i+d

is valid (see also Section 3.5 of [RFC4082]). If it is falsified, it is discarded.

If the check above is successful, the client performs another more rigorous check: it sends a key check request to the server (in the form of a client_keycheck message), asking explicitly if K_i has already been disclosed. It remembers the time stamp t_check of the sending time of that request as well as the nonce it used correlated with the interval number i. If it receives an answer from the server stating that K_i has not yet been disclosed and it is able to verify the HMAC on that response, then it deduces that K_i was undisclosed at t_check and therefore also at R. In this case, the client accepts P_i as timely.

Next the client verifies that a newly disclosed key K_{i-d} belongs to the one-way key chain. To this end, it applies the one-way function F to K_{i-d} until it can verify the identity with an earlier disclosed key (see Clause 3.5 in RFC 4082, item 3).

Next the client verifies that the transmitted time value s_i belongs to the time interval I_i, by checking

T_i =< s_i, and

s_i < T_{i+1}.

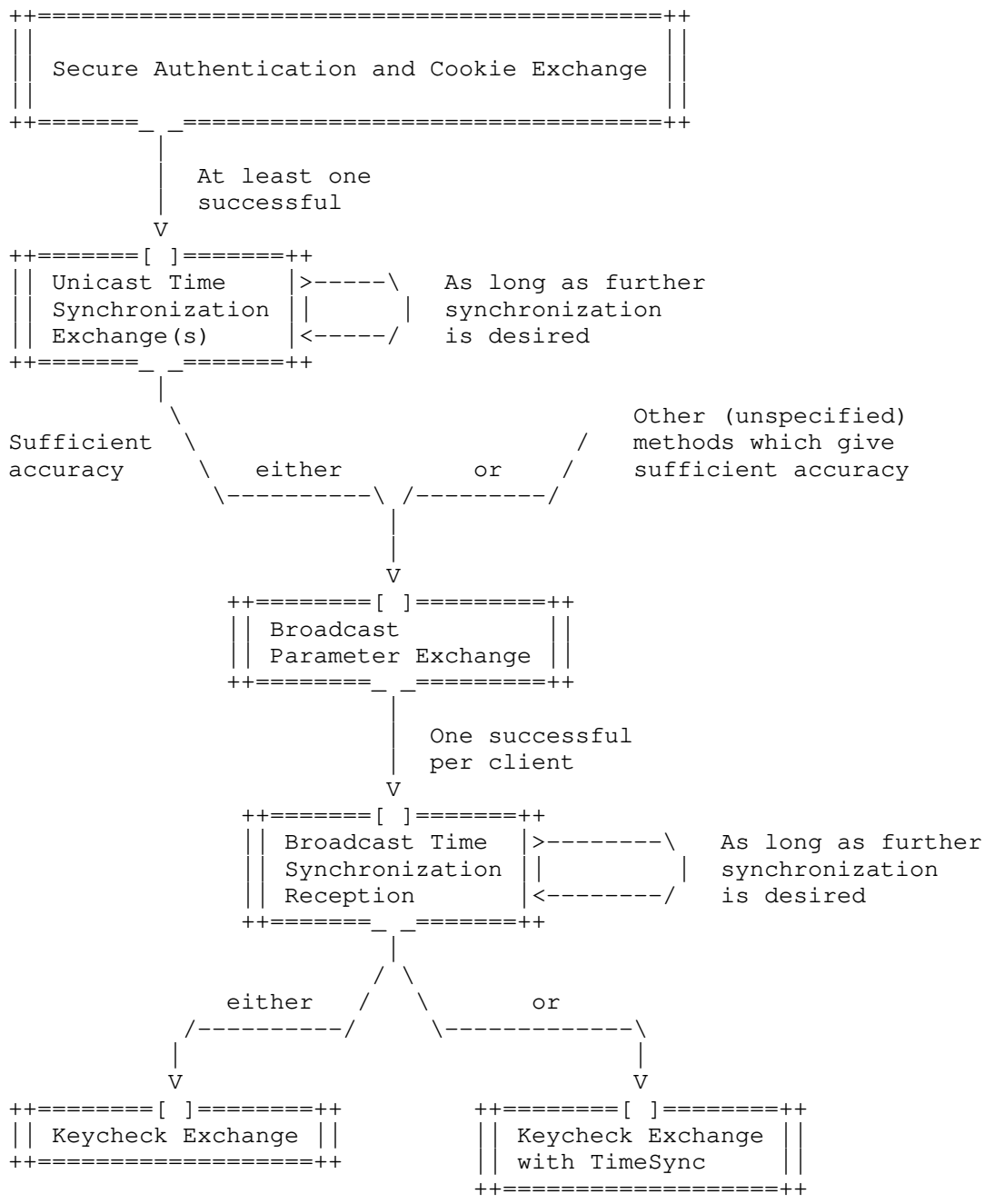
If it is falsified, the packet MUST be discarded and the client MUST reinitialize its broadcast module by performing time synchronization by other means than broadcast messages, and it MUST perform a new broadcast parameter exchange (because a falsification of this check yields that the packet was not generated according to protocol, which suggests an attack).

If a packet P_i passes all the tests listed above, it is stored for later authentication. Also, if at this time there is a package with index i-d already buffered, then the client uses the disclosed key K_{i-d} to derive K'_{i-d} and uses that to check the MAC included in package P_{i-d}. Upon success, it regards M_{i-d} as authenticated.

Appendix D. (informative) Dependencies

Issuer	Type	Owner	Description
Server PKI	private key (signature)	server	Used for server_assoc, server_cook, server_bpar. The server uses the private key to sign these messages. The client uses the public key to verify them.
	public key (signature)	client	
	certificate	server	The certificate is used in server_assoc messages, for verifying authentication and (optionally) authorization.
Client PKI	private key (encryption)	client	The server uses the client's public key to encrypt the content of server_cook messages. The client uses the private key to decrypt them. The certificate is sent in client_cook messages, where it is used for transportation of the public key as well as (optionally) for verification of client authorization.
	public key (encryption)	server	
	certificate	client	

This table shows the kind of cryptographic resources that NTS participants of server and client role should have ready before NTS communication starts.



This diagram shows the dependencies between the different message exchanges and procedures which NTS offers.

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Stephen Roettger
Google Inc.

Email: stephen.roettger@googlemail.com

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 7, 2019

H. Stenn
Network Time Foundation
S. Goldberg
Boston University

October 4, 2018

Network Time Protocol REFID Updates
draft-ietf-ntp-refid-updates-04

Abstract

RFC 5905 [RFC5905], section 7.3, "Packet Header Variables", defines the value of the REFID, the system peer for the responding host. In the past, for IPv4 associations the IPv4 address is used, and for IPv6 associations the first four octets of the MD5 hash of the IPv6 are used. There are two recognized shortcomings to this approach, and this proposal addresses them. One is that knowledge of the system peer is "abusable" information and should not be generally available. The second is that the four octet hash of the IPv6 address looks very much like an IPv4 address, and this is confusing.

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 https://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 April 7, 2019.

Table of Contents

1. Introduction

1.1. The REFID

The interpretation of a REFID is based on the stratum, as documented in RFC 5905 [RFC5905], section 7.3, "Packet Header Variables". The core reason for the REFID in the NTP Protocol is to prevent a degree-one timing loop, where server B decides to follow A as its time source, and A then decides to follow B as its time source.

At Stratum 2+, which will be the case if two servers A and B are exchanging timing information, then if server B follows A as its time source, A's address will be B's REFID. When A uses IPv4, the default REFID is A's IPv4 address. When A uses IPv6, the default REFID is a four-octet digest of A's IPv6 address. Now, if A queries B for its time, then A will learn that B is using A as its time source by observing A's address in the REFID field of the response packet sent by B. Thus, A will not select B as a potential time source, as this would cause a timing loop.

1.2. NOT-YOU REFID

The traditional REFID mechanism, however, also allows a third-party C to learn that A is the time source that is being used by B. When A is using IPv4, C can learn this by querying B for its time, and observing that the REFID in B's response is the IPv4 address of A. Meanwhile, when A is using IPv6, then C can again query B for its time, and then can use an offline dictionary attack to attempt to determine the IPv6 address that corresponds to the digest value in the response sent by B. C could construct the necessary dictionary by compiling a list of publicly accessible IPv6 servers. Remote attackers can use this technique to attempt to identify the time sources used by a target, and then send spoofed packets to the target or its time source in an attempt to disrupt time service, as was done e.g., in [NDSS16] or [CVE-2015-8138].

The REFID thus unnecessarily leaks information about a target's time server to remote attackers. The best way to mitigate this vulnerability is to decouple the IP address of the time source from the REFID. To do this, a system can use an otherwise-impossible value for its REFID, called the "not-you" value, when it believes that a querying system is not its time source.

The NOT-YOU REFID proposal is backwards-compatible. It can be implemented by one peer in an NTP association without any changes to the other peer.

The NOT-YOU REFID proposal does have a small risk, in that a system that might return NOT-YOU does not have perfect information and it is possible that the remote system peer is contacting "us" via a different network interface. In this case, the remote system might choose us as their system peer, and a degree-one timing loop will occur. In this case, however, the two systems will spiral into worsening stratum positions with increasing root distances, and eventually the loop will break. If any other systems are available as time servers, one of them may become the new system peer. However, unless or until this happens the two spiraling systems will have degraded time quality.

1.3. IPv6 REFID

In an environment where all time queries made to a server can be trusted, an operator might well choose to expose the real REFID. RFC 5905 [RFC5905], section 7.3, "Packet Header Variables", explains how a remote system peer is converted to a REFID. It says:

> If using the IPv4 address family, the identifier is the four-octet IPv4 address. If using the IPv6 family, it is the first four octets of the MD5 hash of the IPv6 address. ...

However, the MD5 hash of an IPv6 address often looks like a valid IPv4 address. When this happens, an operator cannot tell if the REFID refers to an IPv6 address or and IPv4. Specifically, the NTP Project has received a report where the generated IPv6 hash decoded to the IPv4 address of a different machine on the system peer's network.

This proposal offers a way for a system to generate a REFID for a IPv6 system peer that does not conflict with an IPv4-based REFID.

This proposal is not fully backwards-compatible. It SHOULD be implemented by both peers in an NTP association. In the scenario where A and B are peering using IPv6, where A is the system peer and does not understand IPv6 REFID, and B is subordinate and is using IPv6 REFID, A will not be able to determine that B is using A as its system peer and a degree-one timing loop can form.

If both peers implement the IPv6 REFID this situation cannot happen.

[If at least one of the peers implements the proposed I-DO protocol this situation cannot happen.]

1.4. 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. The NOT-YOU REFID

2.1. Proposal

When enabled, this proposal allows the one-degree loop detection to work and useful diagnostic information to be provided to trusted partners while keeping potentially abusable information from being disclosed to ostensibly uninterested parties. It does this by returning the normal REFID to queries that come from trusted addresses or from an address that the current system believes is its time source (aka its "system peer"), and otherwise returning one of two special IP addresses that is interpreted to mean "not you". The "not you" IP addresses are 127.127.127.127 and 127.127.127.128. If an IPv6 query is received from an address whose four-octet hash equals one of these two addresses and we believe the querying host is

not our system peer, the other NOT-YOU address is returned as the REFID.

This mechanism is correct and transparent when the system responding with a NOT-YOU can accurately detect when it's getting a timing query from its system peer. A querying system that uses IPv4 continues to check that its IPv4 address does not appear in the REFID before deciding whether to take time from the current system. A querying system that uses IPv6 continues to check that the four-octet hash of its IPv6 address does not appear in the REFID before deciding whether to take time from the current system. However...

Use of the NOT-YOU REFID proposal will hide the current system's system peer from querying systems that the current system believes are not the current system's system peer. Should the current system return the "not you" REFID to a query from its system peer, for example in the case where the system peer sends its query from an unexpected IP address, a one-degree timing loop can occur. Put another way, the responding system has imperfect knowledge about whether or not the sender is its system peer and there are cases where it will offer a NOT-YOU response to its system peer, which can then produce a degree-one timing loop.

Note that this mechanism fully supports degree-one loop detection in the case where the responding NOT-YOU system can accurately detect when it's getting a request from its system peer, and otherwise provides the most basic diagnostic information to third parties.

3. Augmenting the IPv6 REFID Hash

3.1. Background

In a trusted network, the S2+ REFID is generated based on the network system peer. RFC 5905 [RFC5905] says:

> If using the IPv4 address family, the identifier is the four-octet IPv4 address. If using the IPv6 family, it is the first four octets of the MD5 hash of the IPv6 address. ...

This means that the IPv4 representation of the IPv6 hash would be: b1.b2.b3.b4 . This proposal is that the system MAY also use 255.b2.b3.b4 as its REFID. This reduces the risk of ambiguity, since addresses beginning with 255 are "reserved", and thus will not collide with valid IPv4 on the network.

When using the REFID to check for a timing loop for an IPv6 association, if the code that checks the first four-octets of the

hash fails to match then the code must check again, using 0xFF as the first octet of the hash.

3.2. Potential Problems

There is a 1 in 16,777,216 chance that the REFID hashes of two IPv6 addresses will be identical, producing a false-positive loop detection. With a sufficient number of servers, the risk of this problem becomes a non-issue. [The use of the NOT-YOU REFID and/or the proposed "REFID Suggestion" or "I-DO" extension fields are ways to mitigate this potential situation.]

Unrealistically, if only two instances of NTP are communicating via IPv6 and system A implements this new IPv6 REFID hash and system B does not, system B will not be able to detect this loop condition. In this case, the two machines will slowly increase their Stratum until they reach S16 and become unsynchronized. This situation is considered to be unrealistic because, for this to happen, each system would have to have only the other system available as a time source, for example, in a misconfigured "orphan mode" setup. There is no risk of this happening in an NTP network with 3 or more time sources, or in a properly-configured "time island" setup.

3.3. Questions

Should we reference the REFID Suggestion and I-DO proposals here?

4. Acknowledgements

For the "not-you" REFID, we acknowledge useful discussions with Aanchal Malhotra and Matthew Van Gundy.

For the IPv6 REFID, we acknowledge Dan Mahoney (and perhaps others) for suggesting the idea of using an "impossible" first-octet value to indicate an IPv6 refid hash.

5. IANA Considerations

This memo requests IANA to allocate a pseudo Extension Field Type of 0xFFFF so the proposed "I-Do" exchange can report whether or not the "IPv6 REFID Hash" is supported.

6. Security Considerations

Many systems running NTP are configured to return responses to timing queries by default. These responses contain a REFID field, which generally reveals the address of the system's time source if that source is an IPv4 address. This behavior can be exploited by remote

attackers who wish to first learn the address of a target's time source, and then attack the target and/or its time source. As such, the "not-you" REFID proposal is designed to harden NTP against these attacks by limiting the amount of information leaked in the REFID field.

Systems running NTP should reveal the identity of their system in peer in their REFID only when they are on a trusted network. The IPv6 REFID proposal provides one way to do this, when the system peer uses addresses in the IPv6 family.

7. References

7.1. Normative References

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

[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, <https://www.rfc-editor.org/info/rfc5905>.

7.2. Informative References

[CVE-2015-8138] Van Gundy, M. and J. Gardner, "Network Time Protocol Origin Timestamp Check Impersonation Vulnerability (CVE-2015-8138)", in TALOS VULNERABILITY REPORT (TALOS-2016-0077), 2016.

[NDSS16] Malhotra, A., Cohen, I., Brakke, E., and S. Goldberg, "Attacking the Network Time Protocol", in ISOC Network and Distributed System Security Symposium 2016 (NDSS'16), 2016.

Authors' Addresses

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

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

Email: goldbe@cs.bu.edu

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2019

D. Franke
Akamai
D. Sibold
K. Teichel
PTB
M. Dansarie

R. Sundblad
Netnod

October 22, 2018

Network Time Security for the Network Time Protocol
draft-ietf-ntp-using-nts-for-ntp-14

Abstract

This memo specifies Network Time Security (NTS), a mechanism for using Transport Layer Security (TLS) and Authenticated Encryption with Associated Data (AEAD) to provide cryptographic security for the client-server mode of the Network Time Protocol (NTP).

NTS is structured as a suite of two loosely coupled sub-protocols. The first (NTS-KE) handles initial authentication and key establishment over TLS. The second handles encryption and authentication during NTP time via extension fields in the NTP packets, and holds all required state only on the client via opaque cookies.

Table of Contents

1. Introduction

This memo specifies Network Time Security (NTS), a cryptographic security mechanism for network time synchronization. A complete specification is provided for application of NTS to the client-server mode of the Network Time Protocol (NTP) [RFC5905].

1.1. Objectives

The objectives of NTS are as follows:

- Identity: Through the use of the X.509 public key infrastructure, implementations may cryptographically establish the identity of the parties they are communicating with.

- Authentication: Implementations may cryptographically verify that any time synchronization packets are authentic, i.e., that they were produced by an identified party and have not been modified in transit.

- Confidentiality: Although basic time synchronization data is considered non-confidential and sent in the clear, NTS includes support for encrypting NTP extension fields.

- Replay prevention: Client implementations may detect when a received time synchronization packet is a replay of a previous packet.

- Request-response consistency: Client implementations may verify that a time synchronization packet received from a server was sent in response to a particular request from the client.

- Unlinkability: For mobile clients, NTS will not leak any information additional to NTP which would permit a passive adversary to determine that two packets sent over different networks came from the same client.

- Non-amplification: Implementations (especially server implementations) may avoid acting as distributed denial-of-service (DDoS) amplifiers by never responding to a request with a packet larger than the request packet.

- Scalability: Server implementations may serve large numbers of clients without having to retain any client-specific state.

1.2. Protocol Overview

The Network Time Protocol includes many different operating modes to support various network topologies. In addition to its best-known and most-widely-used client-server mode, it also includes modes for synchronization between symmetric peers, a control mode for server monitoring and administration, and a broadcast mode. These various modes have differing and partly contradictory requirements for security and performance. Symmetric and control modes demand mutual authentication and mutual replay protection. Additionally, for certain message types control mode may require confidentiality as well as authentication. Client-server mode places more stringent requirements on resource utilization than other modes, because servers may have vast number of clients and be unable to afford to maintain per-client state. However, client-server mode also has more relaxed security needs, because only the client requires replay protection: it is harmless for stateless servers to process replayed packets. The security demands of symmetric and control modes, on the

other hand, are in conflict with the resource-utilization demands of client-server mode: any scheme which provides replay protection inherently involves maintaining some state to keep track of what messages have already been seen.

This memo specifies NTS exclusively for the client-server mode of NTP. To this end, NTS is structured as a suite of two protocols:

The "NTS Extensions for NTPv4" define a collection of NTP extension fields for cryptographically securing NTPv4 using previously-established key material. They are suitable for securing client-server mode because the server can implement them without retaining per-client state. All state is kept by the client and provided to the server in the form of an encrypted cookie supplied with each request. On the other hand, the NTS Extension Fields are suitable *only* for client-server mode because only the client, and not the server, is protected from replay.

The "NTS Key Establishment" protocol (NTS-KE) is a mechanism for establishing key material for use with the NTS Extension Fields for NTPv4. It uses TLS to exchange keys, provide the client with an initial supply of cookies, and negotiate some additional protocol options. After this exchange, the TLS channel is closed with no per-client state remaining on the server side.

The typical protocol flow is as follows: The client connects to an NTS-KE server on the NTS TCP port and the two parties perform a TLS handshake. Via the TLS channel, the parties negotiate some additional protocol parameters and the server sends the client a supply of cookies along with a list of one or more IP addresses to NTP servers for which the cookies are valid. The parties use TLS key export [RFC5705] to extract key material which will be used in the next phase of the protocol. This negotiation takes only a single round trip, after which the server closes the connection and discards all associated state. At this point the NTS-KE phase of the protocol is complete. Ideally, the client never needs to connect to the NTS-KE server again.

Time synchronization proceeds with one of the indicated NTP servers over the NTP UDP port. The client sends the server an NTP client packet which includes several extension fields. Included among these fields are a cookie (previously provided by the key exchange server) and an authentication tag, computed using key material extracted from the NTS-KE handshake. The NTP server uses the cookie to recover this key material and send back an authenticated response. The response includes a fresh, encrypted cookie which the client then sends back

in the clear in a subsequent request. (This constant refreshing of cookies is necessary in order to achieve NTS's unlinkability goal.)

Figure 1 provides an overview of the high-level interaction between the client, the NTS-KE server, and the NTP server. Note that the cookies' data format and the exchange of secrets between NTS-KE and NTP servers are not part of this specification and are implementation dependent. However, a suggested format for NTS cookies is provided in Section 6.

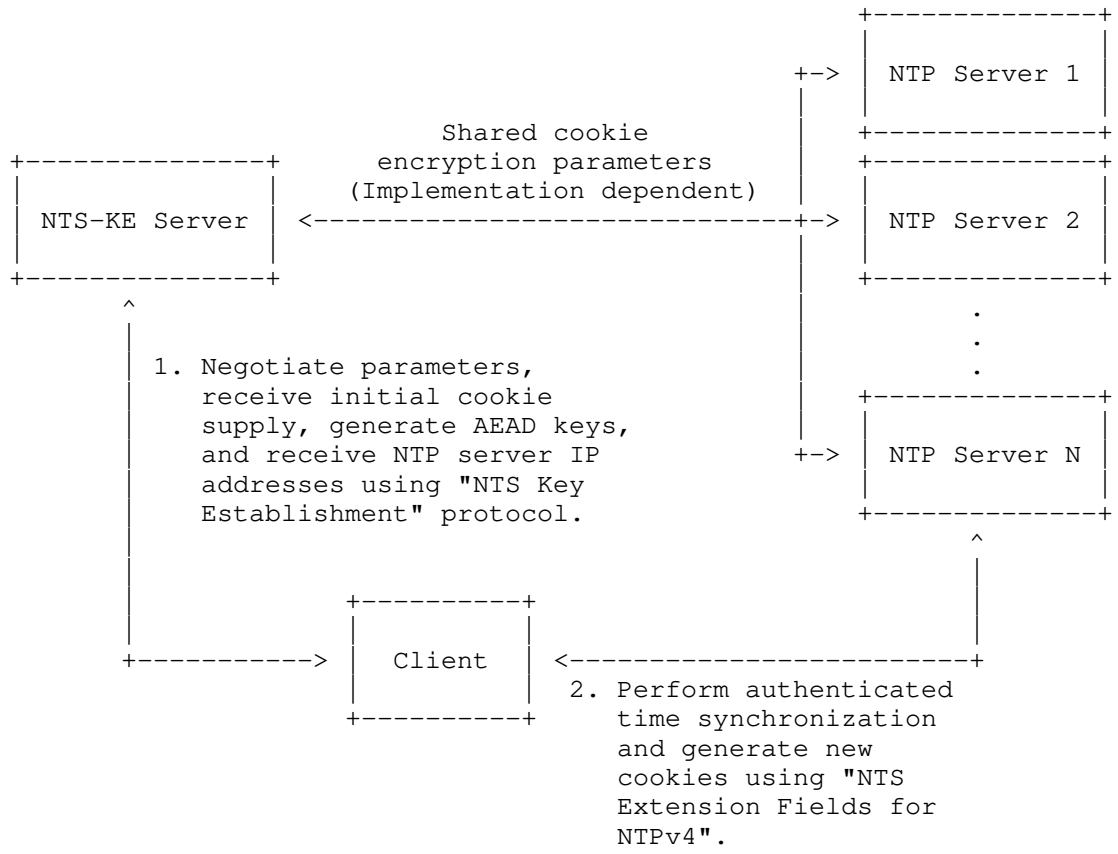


Figure 1: Overview of High-Level Interactions in NTS

2. Requirements Language

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

BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. TLS profile for Network Time Security

Network Time Security makes use of TLS for NTS key establishment.

Since the NTS protocol is new as of this publication, no backward-compatibility concerns exist to justify using obsolete, insecure, or otherwise broken TLS features or versions. Implementations MUST conform with [RFC7525] or with a later revision of BCP 195. Furthermore:

Implementations MUST NOT negotiate TLS versions earlier than 1.2, SHOULD negotiate TLS 1.3 [RFC8446] or later when possible, and MAY refuse to negotiate any TLS version which has been superseded by a later supported version.

Use of the Application-Layer Protocol Negotiation Extension [RFC7301] is integral to NTS and support for it is REQUIRED for interoperability.

4. The NTS Key Establishment Protocol

The NTS key establishment protocol is conducted via TCP port [[TBD1]]. The two endpoints carry out a TLS handshake in conformance with Section 3, with the client offering (via an ALPN [RFC7301] extension), and the server accepting, an application-layer protocol of "ntske/1". Immediately following a successful handshake, the client SHALL send a single request as Application Data encapsulated in the TLS-protected channel. Then, the server SHALL send a single response followed by a TLS "Close notify" alert and then discard the channel state.

The client’s request and the server’s response each SHALL consist of a sequence of records formatted according to Figure 2. Requests and non-error responses each SHALL include exactly one NTS Next Protocol Negotiation record. The sequence SHALL be terminated by a "End of Message" record. The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting.

Clients and servers MAY enforce length limits on requests and responses, however, servers MUST accept requests of at least 1024 octets and clients SHOULD accept responses of at least 65536 octets.

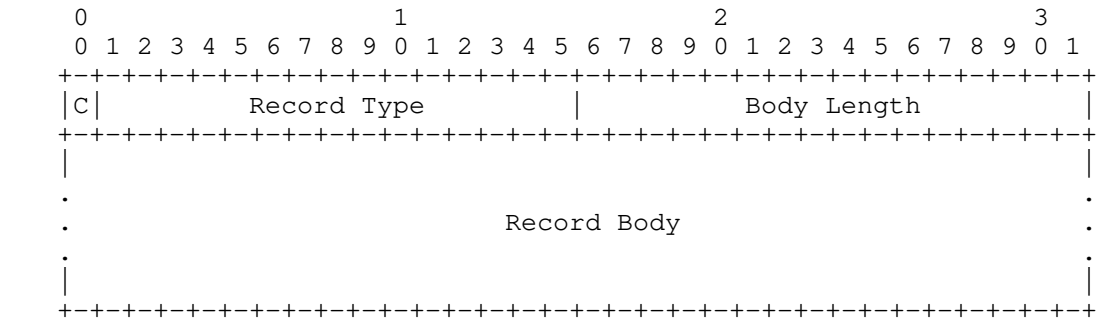


Figure 2: NTS-KE Record Format

The fields of an NTS-KE record are defined as follows:

C (Critical Bit): Determines the disposition of unrecognized Record Types. Implementations which receive a record with an unrecognized Record Type MUST ignore the record if the Critical Bit is 0 and MUST treat it as an error if the Critical Bit is 1.

Record Type Number: A 15-bit integer in network byte order. The semantics of record types 0-6 are specified in this memo. Additional type numbers SHALL be tracked through the IANA Network Time Security Key Establishment Record Types registry.

Body Length: The length of the Record Body field, in octets, as a 16-bit integer in network byte order. Record bodies MAY have any representable length and need not be aligned to a word boundary.

Record Body: The syntax and semantics of this field SHALL be determined by the Record Type.

For clarity regarding bit-endianness: the Critical Bit is the most-significant bit of the first octet. In C, given a network buffer ‘unsigned char b[]‘ containing an NTS-KE record, the critical bit is ‘b[0] >> 7‘ while the record type is ‘((b[0] & 0x7f) << 8) + b[1]‘.

Figure 3 provides a schematic overview of the key exchange. It displays the protocol steps to be performed by the NTS client and server and record types to be exchanged.

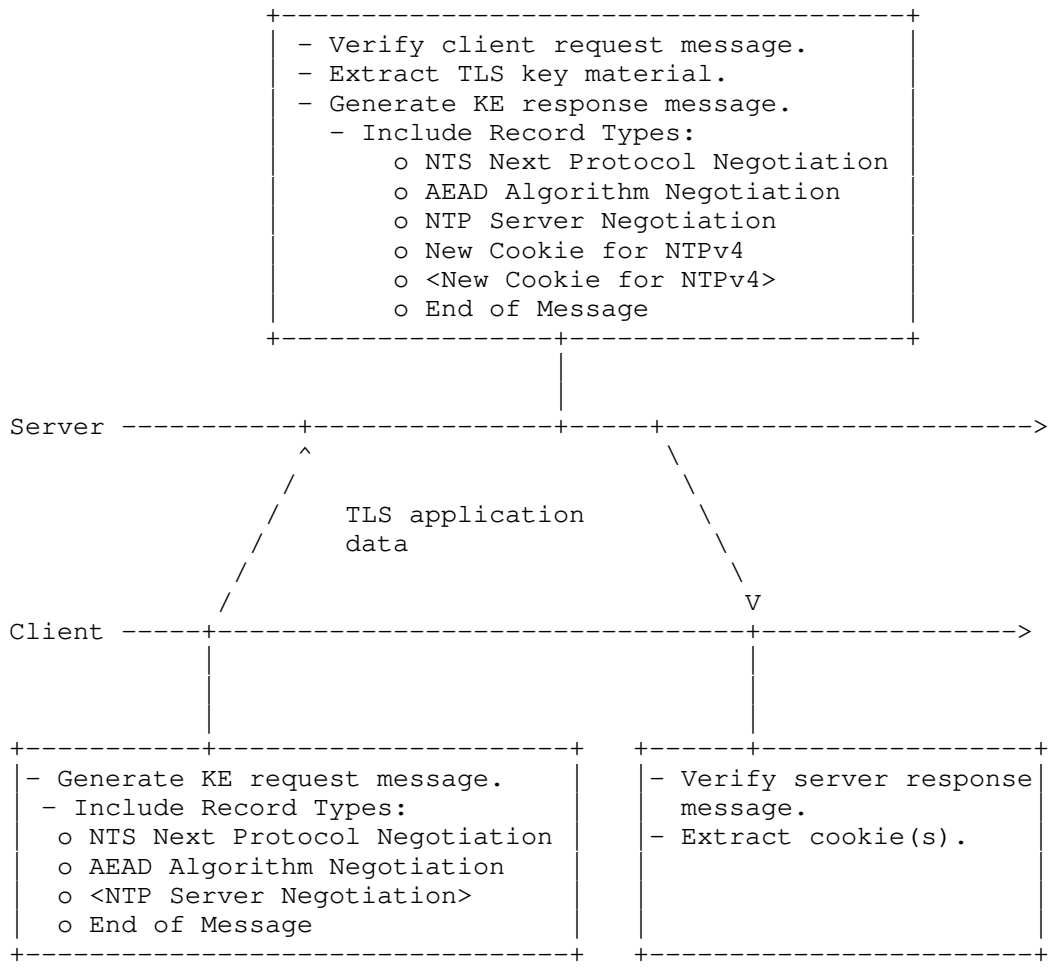


Figure 3: NTS Key Exchange Messages

4.1. NTS-KE Record Types

The following NTS-KE Record Types are defined:

4.1.1. End of Message

The End of Message record has a Record Type number of 0 and a zero-length body. It MUST occur exactly once as the final record of every NTS-KE request and response. The Critical Bit MUST be set.

4.1.2. NTS Next Protocol Negotiation

The NTS Next Protocol Negotiation record has a Record Type number of 1. It MUST occur exactly once in every NTS-KE request and response. Its body consists of a sequence of 16-bit unsigned integers in network byte order. Each integer represents a Protocol ID from the IANA Network Time Security Next Protocols registry. The Critical Bit MUST be set.

The Protocol IDs listed in the client's NTS Next Protocol Negotiation record denote those protocols which the client wishes to speak using the key material established through this NTS-KE session. The Protocol IDs listed in the server's response MUST comprise a subset of those listed in the request and denote those protocols which the server is willing and able to speak using the key material established through this NTS-KE session. The client MAY proceed with one or more of them. The request MUST list at least one protocol, but the response MAY be empty.

4.1.3. Error

The Error record has a Record Type number of 2. Its body is exactly two octets long, consisting of an unsigned 16-bit integer in network byte order, denoting an error code. The Critical Bit MUST be set.

Clients MUST NOT include Error records in their request. If clients receive a server response which includes an Error record, they MUST discard any negotiated key material and MUST NOT proceed to the Next Protocol.

The following error codes are defined:

Error code 0 means "Unrecognized Critical Record". The server MUST respond with this error code if the request included a record which the server did not understand and which had its Critical Bit set. The client SHOULD NOT retry its request without modification.

Error code 1 means "Bad Request". The server MUST respond with this error if, upon the expiration of an implementation-defined timeout, it has not yet received a complete and syntactically well-formed request from the client.

4.1.4. Warning

The Warning record has a Record Type number of 3. Its body is exactly two octets long, consisting of an unsigned 16-bit integer in

network byte order, denoting a warning code. The Critical Bit MUST be set.

Clients MUST NOT include Warning records in their request. If clients receive a server response which includes a Warning record, they MAY discard any negotiated key material and abort without proceeding to the Next Protocol. Unrecognized warning codes MUST be treated as errors.

This memo defines no warning codes.

4.1.5. AEAD Algorithm Negotiation

The AEAD Algorithm Negotiation record has a Record Type number of 4. Its body consists of a sequence of unsigned 16-bit integers in network byte order, denoting Numeric Identifiers from the IANA AEAD registry [RFC5116]. The Critical Bit MAY be set.

If the NTS Next Protocol Negotiation record offers Protocol ID 0 (for NTPv4), then this record MUST be included exactly once. Other protocols MAY require it as well.

When included in a request, this record denotes which AEAD algorithms the client is willing to use to secure the Next Protocol, in decreasing preference order. When included in a response, this record denotes which algorithm the server chooses to use. It is empty if the server supports none of the algorithms offered. In requests, the list MUST include at least one algorithm. In responses, it MUST include at most one. Honoring the client's preference order is OPTIONAL: servers may select among any of the client's offered choices, even if they are able to support some other algorithm which the client prefers more.

Server implementations of NTS extension fields for NTPv4 (Section 5) MUST support AEAD_AES_SIV_CMAC_256 [RFC5297] (Numeric Identifier 15). That is, if the client includes AEAD_AES_SIV_CMAC_256 in its AEAD Algorithm Negotiation record and the server accepts Protocol ID 0 (NTPv4) in its NTS Next Protocol Negotiation record, then the server's AEAD Algorithm Negotiation record MUST NOT be empty.

4.1.6. New Cookie for NTPv4

The New Cookie for NTPv4 record has a Record Type number of 5. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 6 for a suggested construction.

Clients MUST NOT send records of this type. Servers MUST send at least one record of this type, and SHOULD send eight of them, if the Next Protocol Negotiation response record contains Protocol ID 0 (NTPv4) and the AEAD Algorithm Negotiation response record is not empty. The Critical Bit SHOULD NOT be set.

4.1.7. NTPv4 Server Negotiation

The NTPv4 Server Negotiation record has a Record Type number of 6. Its body consists of an ASCII-encoded [ANSI.X3-4.1986] string conforming to the syntax of the Host subcomponent of a URI ([RFC3986]). IPv6 addresses MUST NOT include zone identifiers [RFC6874].

When NTPv4 is negotiated as a Next Protocol and this record is sent by the server, the body specifies the hostname or IP address of the NTPv4 server with which the client should associate and which will accept the supplied cookies. If no record of this type is sent, the client SHALL interpret this as a directive to associate with an NTPv4 server at the same IP address as the NTS-KE server. Servers MUST NOT send more than one record of this type.

When this record is sent by the client, it indicates that the client wishes to associate with the specified NTP server. The NTS-KE server MAY incorporate this request when deciding what NTPv4 Server Negotiation records to respond with, but honoring the client's preference is OPTIONAL. The client MUST NOT send more than one record of this type.

Servers MAY set the Critical Bit on records of this type; clients SHOULD NOT.

4.1.8. NTPv4 Port Negotiation

The NTPv4 Port Negotiation record has a Record Type number of 7. Its body consists of a 16-bit unsigned integer in network byte order, denoting a UDP port number.

When NTPv4 is negotiated as a Next Protocol and this record is sent by the server, the body specifies the port number of the NTPv4 server with which the client should associate and which will accept the supplied cookies. If no record of this type is sent, the client SHALL assume a default of 123 (the registered port number for NTP).

When this record is sent by the client in conjunction with a NTPv4 Server Negotiation record, it indicates that the client wishes to associate with the NTP server at the specified port. The NTS-KE server MAY incorporate this request when deciding what NTPv4 Server

Negotiation and NTPv4 Port Negotiation records to respond with, but honoring the client's preference is OPTIONAL.

Servers MAY set the Critical Bit on records of this type; clients SHOULD NOT.

4.2. Key Extraction (generally)

Following a successful run of the NTS-KE protocol, key material SHALL be extracted according to RFC 5705 [RFC5705]. Inputs to the exporter function are to be constructed in a manner specific to the negotiated Next Protocol. However, all protocols which utilize NTS-KE MUST conform to the following two rules:

The disambiguating label string MUST be "EXPORTER-network-time-security/1".

The per-association context value MUST be provided and MUST begin with the two-octet Protocol ID which was negotiated as a Next Protocol.

5. NTS Extension Fields for NTPv4

5.1. Key Extraction (for NTPv4)

Following a successful run of the NTS-KE protocol wherein Protocol ID 0 (NTPv4) is selected as a Next Protocol, two AEAD keys SHALL be extracted: a client-to-server (C2S) key and a server-to-client (S2C) key. These keys SHALL be computed according to RFC 5705 [RFC5705], using the following inputs.

The disambiguating label string SHALL be "EXPORTER-network-time-security/1".

The per-association context value SHALL consist of the following five octets:

The first two octets SHALL be zero (the Protocol ID for NTPv4).

The next two octets SHALL be the Numeric Identifier of the negotiated AEAD Algorithm in network byte order.

The final octet SHALL be 0x00 for the C2S key and 0x01 for the S2C key.

Implementations wishing to derive additional keys for private or experimental use MUST NOT do so by extending the above-specified syntax for per-association context values. Instead, they SHOULD use

their own disambiguating label string. Note that RFC 5705 [RFC5705] provides that disambiguating label strings beginning with "EXPERIMENTAL" MAY be used without IANA registration.

5.2. Packet Structure Overview

In general, an NTS-protected NTPv4 packet consists of:

The usual 48-octet NTP header which is authenticated but not encrypted.

Some extension fields which are authenticated but not encrypted.

An extension field which contains AEAD output (i.e., an authentication tag and possible ciphertext). The corresponding plaintext, if non-empty, consists of some extension fields which benefit from both encryption and authentication.

Possibly, some additional extension fields which are neither encrypted nor authenticated. These are discarded by the receiver.

Always included among the authenticated or authenticated-and-encrypted extension fields are a cookie extension field and a unique identifier extension field. The purpose of the cookie extension field is to enable the server to offload storage of session state onto the client. The purpose of the unique identifier extension field is to protect the client from replay attacks.

5.3. The Unique Identifier Extension Field

The Unique Identifier extension field provides the client with a cryptographically strong means of detecting replayed packets. It has a Field Type of [[TBD2]]. When the extension field is included in a client packet (mode 3), its body SHALL consist of a string of octets generated uniformly at random. The string MUST be at least 32 octets long. When the extension field is included in a server packet (mode 4), its body SHALL contain the same octet string as was provided in the client packet to which the server is responding. All server packets generated by NTS-implementing servers in response to client packets containing this extension field MUST also contain this field with the same content as in the client's request. The field's use in modes other than client-server is not defined.

This extension field MAY also be used standalone, without NTS, in which case it provides the client with a means of detecting spoofed packets from off-path attackers. Historically, NTP's origin timestamp field has played both these roles, but for cryptographic purposes this is suboptimal because it is only 64 bits long and,

depending on implementation details, most of those bits may be predictable. In contrast, the Unique Identifier extension field enables a degree of unpredictability and collision resistance more consistent with cryptographic best practice.

5.4. The NTS Cookie Extension Field

The NTS Cookie extension field has a Field Type of [[TBD3]]. Its purpose is to carry information which enables the server to recompute keys and other session state without having to store any per-client state. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 6 for a suggested construction. The NTS Cookie extension field MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

5.5. The NTS Cookie Placeholder Extension Field

The NTS Cookie Placeholder extension field has a Field Type of [[TBD4]]. When this extension field is included in a client packet (mode 3), it communicates to the server that the client wishes it to send additional cookies in its response. This extension field MUST NOT be included in NTP packets whose mode is other than 3.

Whenever an NTS Cookie Placeholder extension field is present, it MUST be accompanied by an NTS Cookie extension field. The body length of the NTS Cookie Placeholder extension field MUST be the same as the body length of the NTS Cookie extension field. This length requirement serves to ensure that the response will not be larger than the request, in order to improve timekeeping precision and prevent DDoS amplification. The contents of the NTS Cookie Placeholder extension field's body are undefined and, aside from checking its length, MUST be ignored by the server.

5.6. The NTS Authenticator and Encrypted Extension Fields Extension Field

The NTS Authenticator and Encrypted Extension Fields extension field is the central cryptographic element of an NTS-protected NTP packet. Its Field Type is [[TBD5]]. It SHALL be formatted according to Figure 4 and include the following fields:

Nonce length: Two octets in network byte order, giving the length of the Nonce field, excluding any padding, interpreted as an unsigned integer.

Ciphertext Length: Two octets in network byte order, giving the length of the Ciphertext field, excluding any padding, interpreted as an unsigned integer.

Nonce: A nonce as required by the negotiated AEAD Algorithm. The field is zero-padded to a word (four octets) boundary.

Ciphertext: The output of the negotiated AEAD Algorithm. The structure of this field is determined by the negotiated algorithm, but it typically contains an authentication tag in addition to the actual ciphertext. The field is zero-padded to a word (four octets) boundary.

Additional Padding: Clients which use a nonce length shorter than the maximum allowed by the negotiated AEAD algorithm may be required to include additional zero-padding. The necessary length of this field is specified below.

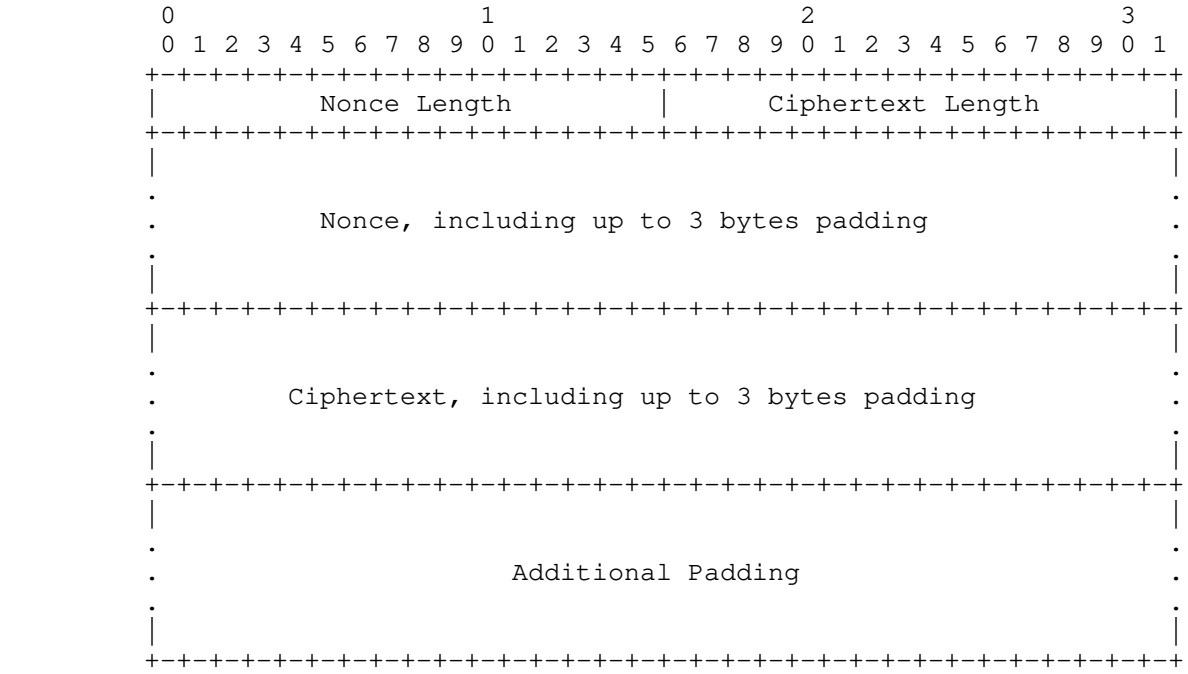


Figure 4: NTS Authenticator and Encrypted Extension Fields Extension Field Format

The Ciphertext field SHALL be formed by providing the following inputs to the negotiated AEAD Algorithm:

K: For packets sent from the client to the server, the C2S key SHALL be used. For packets sent from the server to the client, the S2C key SHALL be used.

A: The associated data SHALL consist of the portion of the NTP packet beginning from the start of the NTP header and ending at the end of the last extension field which precedes the NTS Authenticator and Encrypted Extension Fields extension field.

P: The plaintext SHALL consist of all (if any) NTP extension fields to be encrypted. The format of any such fields SHALL be in accordance with RFC 7822 [RFC7822]. If multiple extension fields are present they SHALL be joined by concatenation.

N: The nonce SHALL be formed however required by the negotiated AEAD algorithm.

The purpose of the Additional Padding field is to ensure that servers can always choose a nonce whose length is adequate to ensure its uniqueness, even if the client chooses a shorter one, and still ensure that the overall length of the server's response packet. does not exceed the length of the request. For mode 4 (server) packets, no Additional Padding field is ever required. For mode 3 (client) packets, the length of the Additional Padding field SHALL be computed as follows. Let 'N_LEN' be the padded length of the the Nonce field. Let 'N_MAX' be, as specified by RFC 5116 [RFC5116], the maximum permitted nonce length for the negotiated AEAD algorithm. Let 'N_REQ' be the lesser of 16 and N_MAX, rounded up to the nearest multiple of 4. If N_LEN is greater than or equal to N_REQ, then no Additional Padding field is required. Otherwise, the Additional Padding field SHALL be at least N_REQ - N_LEN octets in length. Servers MUST enforce this requirement by discarding any packet which does not conform to it.

The NTS Authenticator and Encrypted Extension Fields extension field MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

5.7. Protocol Details

A client sending an NTS-protected request SHALL include the following extension fields as displayed in Figure 5:

Exactly one Unique Identifier extension field which MUST be authenticated, MUST NOT be encrypted, and whose contents MUST NOT duplicate those of any previous request.

Exactly one NTS Cookie extension field which MUST be authenticated and MUST NOT be encrypted. The cookie MUST be one which has been previously provided to the client; either from the key exchange server during the NTS-KE handshake or from the NTP server in response to a previous NTS-protected NTP request. To protect the client's privacy, the same cookie SHOULD NOT be included in multiple requests. If the client does not have any cookies that it has not already sent, it SHOULD initiate a re-run the NTS-KE protocol.

Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using an AEAD Algorithm and C2S key established through NTS-KE.

The client MAY include one or more NTS Cookie Placeholder extension fields which MUST be authenticated and MAY be encrypted. The number of NTS Cookie Placeholder extension fields that the client includes SHOULD be such that if the client includes N placeholders and the server sends back N+1 cookies, the number of unused cookies stored by the client will come to eight. The client SHOULD NOT include more than seven NTS Cookie Placeholder extension fields in a request. When both the client and server adhere to all cookie-management guidance provided in this memo, the number of placeholder extension fields will equal the number of dropped packets since the last successful volley.

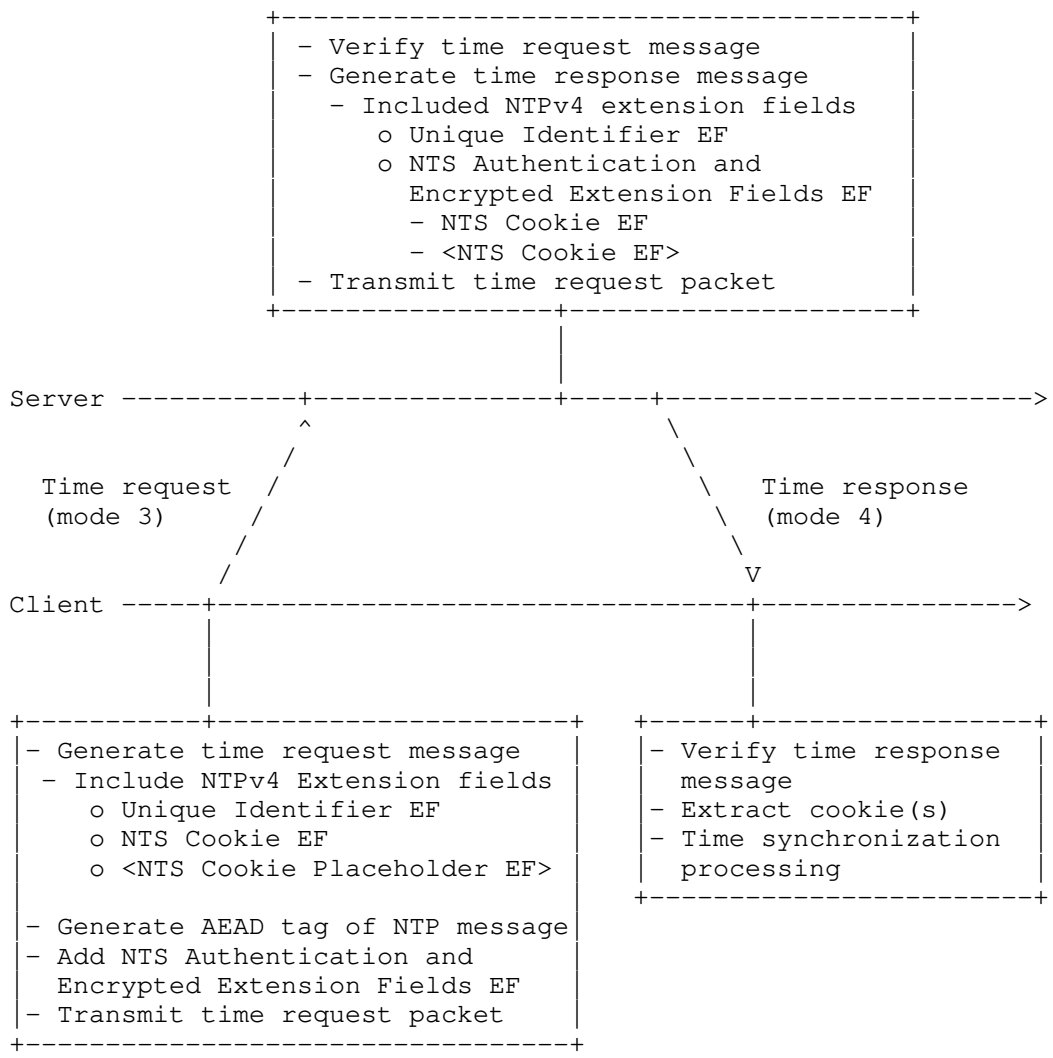


Figure 5: NTS Time Synchronization Messages

The client MAY include additional (non-NTS-related) extension fields which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension fields (therefore authenticated but not encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). In general, however, the server MUST discard any unauthenticated extension fields and process the packet as though they were not present. Servers MAY implement exceptions to this requirement for

particular extension fields if their specification explicitly provides for such.

Upon receiving an NTS-protected request, the server SHALL (through some implementation-defined mechanism) use the cookie to recover the AEAD Algorithm, C2S key, and S2C key associated with the request, and then use the C2S key to authenticate the packet and decrypt the ciphertext. If the cookie is valid and authentication and decryption succeed, the server SHALL include the following extension fields in its response:

- Exactly one Unique Identifier extension field which MUST be authenticated, MUST NOT be encrypted, and whose contents SHALL echo those provided by the client.

- Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using the AEAD algorithm and S2C key recovered from the cookie provided by the client.

- One or more NTS Cookie extension fields which MUST be authenticated and encrypted. The number of NTS Cookie extension fields included SHOULD be equal to, and MUST NOT exceed, one plus the number of valid NTS Cookie Placeholder extension fields included in the request. The cookies returned in those fields MUST be valid for use with the NTP server that sent them. They MAY be valid for other NTP servers as well, but there is no way for the server to indicate this.

We emphasize the contrast that NTS Cookie extension fields MUST NOT be encrypted when sent from client to server, but MUST be encrypted from sent from server to client. The former is necessary in order for the server to be able to recover the C2S and S2C keys, while the latter is necessary to satisfy the unlinkability goals discussed in Section 10.1. We emphasize also that "encrypted" means encapsulated within the the NTS Authenticator and Encrypted Extensions extension field. While the body of an NTS Cookie extension field will generally consist of some sort of AEAD output (regardless of whether the recommendations of Section 6 are precisely followed), this is not sufficient to make the extension field "encrypted".

The server MAY include additional (non-NTS-related) extension fields which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension field (therefore authenticated but not encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). In general, however, the client MUST discard any unauthenticated extension fields and process the packet as though they were not present. Clients MAY implement exceptions to this requirement for

particular extension fields if their specification explicitly provides for such.

Upon receiving an NTS-protected response, the client MUST verify that the Unique Identifier matches that of an outstanding request, and that the packet is authentic under the S2C key associated with that request. If either of these checks fails, the packet MUST be discarded without further processing.

If the server is unable to validate the cookie or authenticate the request, it SHOULD respond with a Kiss-o'-Death (KoD) packet (see RFC 5905, Section 7.4 [RFC5905]) with kiss code "NTSN", meaning "NTS negative-acknowledgment (NAK)". It MUST NOT include any NTS Cookie or NTS Authenticator and Encrypted Extension Fields extension fields.

If the NTP server has previously responded with authentic NTS-protected NTP packets (i.e., packets containing the NTS Authenticator and Encrypted Extension Fields extension field), the client MUST verify that any KoD packets received from the server contain the Unique Identifier extension field and that the Unique Identifier matches that of an outstanding request. If this check fails, the packet MUST be discarded without further processing. If this check passes, the client MUST comply with RFC 5095, Section 7.4 [RFC5905] where required. A client MAY automatically re-run the NTS-KE protocol upon forced disassociation from an NTP server. In that case, it MUST be able to detect and stop looping between the NTS-KE and NTP servers.

Upon reception of the NTS NAK kiss code, the client SHOULD wait until the next poll for a valid NTS-protected response and if none is received, initiate a fresh NTS-KE handshake to try to renegotiate new cookies, AEAD keys, and parameters. If the NTS-KE handshake succeeds, the client MUST discard all old cookies and parameters and use the new ones instead. As long as the NTS-KE handshake has not succeeded, the client SHOULD continue polling the NTP server using the cookies and parameters it has.

The client MAY reuse cookies in order to prioritize resilience over unlinkability. Which of the two that should be prioritized in any particular case is dependent on the application and the user's preference. Section 10.1 describes the privacy considerations of this in further detail.

To allow for NTP session restart when the NTS-KE server is unavailable and to reduce NTS-KE server load, the client SHOULD keep at least one unused but recent cookie, AEAD keys, negotiated AEAD algorithm, and other necessary parameters on persistent storage.

This way, the client is able to resume the NTP session without performing renewed NTS-KE negotiation.

6. Suggested Format for NTS Cookies

This section is non-normative. It gives a suggested way for servers to construct NTS cookies. All normative requirements are stated in Section 4.1.6 and Section 5.4.

The role of cookies in NTS is closely analogous to that of session cookies in TLS. Accordingly, the thematic resemblance of this section to RFC 5077 [RFC5077] is deliberate and the reader should likewise take heed of its security considerations.

Servers should select an AEAD algorithm which they will use to encrypt and authenticate cookies. The chosen algorithm should be one such as AEAD_AES_SIV_CMAC_256 [RFC5297] which resists accidental nonce reuse. It need not be the same as the one that was negotiated with the client. Servers should randomly generate and store a master AEAD key 'K'. Servers should additionally choose a non-secret, unique value 'I' as key-identifier for 'K'.

Servers should periodically (e.g., once daily) generate a new pair (I,K) and immediately switch to using these values for all newly-generated cookies. Immediately following each such key rotation, servers should securely erase any keys generated two or more rotation periods prior. Servers should continue to accept any cookie generated using keys that they have not yet erased, even if those keys are no longer current. Erasing old keys provides for forward secrecy, limiting the scope of what old information can be stolen if a master key is somehow compromised. Holding on to a limited number of old keys allows clients to seamlessly transition from one generation to the next without having to perform a new NTS-KE handshake.

The need to keep keys synchronized between NTS-KE and NTP servers as well as across load-balanced clusters can make automatic key rotation challenging. However, the task can be accomplished without the need for central key-management infrastructure by using a ratchet, i.e., making each new key a deterministic, cryptographically pseudo-random function of its predecessor. A recommended concrete implementation of this approach is to use HKDF [RFC5869] to derive new keys, using the key's predecessor as Input Keying Material and its key identifier as a salt.

To form a cookie, servers should first form a plaintext 'P' consisting of the following fields:

The AEAD algorithm negotiated during NTS-KE.

The S2C key.

The C2S key.

Servers should then generate a nonce `N` uniformly at random, and form AEAD output `C` by encrypting `P` under key `K` with nonce `N` and no associated data.

The cookie should consist of the tuple `(I,N,C)`.

To verify and decrypt a cookie provided by the client, first parse it into its components `I`, `N`, and `C`. Use `I` to look up its decryption key `K`. If the key whose identifier is `I` has been erased or never existed, decryption fails; reply with an NTS NAK. Otherwise, attempt to decrypt and verify ciphertext `C` using key `K` and nonce `N` with no associated data. If decryption or verification fails, reply with an NTS NAK. Otherwise, parse out the contents of the resulting plaintext `P` to obtain the negotiated AEAD algorithm, S2C key, and C2S key.

7. IANA Considerations

7.1. Service Name and Transport Protocol Port Number Registry

IANA is requested to allocate the following entry in the Service Name and Transport Protocol Port Number Registry [RFC6335]:

Service Name: ntske

Transport Protocol: tcp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Network Time Security Key Exchange

Reference: [[this memo]]

Port Number: [[TBD1]], selected by IANA from the User Port range

[[RFC EDITOR: Replace all instances of [[TBD1]] in this document with the IANA port assignment.]]

7.2. TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs Registry

IANA is requested to allocate the following entry in the TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs registry [RFC7301]:

Protocol: Network Time Security Key Establishment, version 1

Identification Sequence:
0x6E 0x74 0x73 0x6B 0x65 0x2F 0x31 ("ntske/1")

Reference: [[this memo]], Section 4

7.3. TLS Exporter Labels Registry

IANA is requested to allocate the following entry in the TLS Exporter Labels Registry [RFC5705]:

Value	DTLS-OK	Recommended	Reference	Note
EXPORTER-network-time-security/1	Y	Y	[[this memo]], Section 4.2	

7.4. NTP Kiss-o'-Death Codes Registry

IANA is requested to allocate the following entry in the registry of NTP Kiss-o'-Death Codes [RFC5905]:

Code	Meaning	Reference
NTSN	Network Time Security (NTS) negative-acknowledgment (NAK)	[[this memo]], Section 5.7

7.5. NTP Extension Field Types Registry

IANA is requested to allocate the following entries in the NTP Extension Field Types registry [RFC5905]:

Field Type	Meaning	Reference
[[TBD2]]	Unique Identifier	[[this memo]], Section 5.3
[[TBD3]]	NTS Cookie	[[this memo]], Section 5.4
[[TBD4]]	NTS Cookie Placeholder	[[this memo]], Section 5.5
[[TBD5]]	NTS Authenticator and Encrypted Extension Fields	[[this memo]], Section 5.6

[[RFC EDITOR: Replace all instances of [[TBD2]], [[TBD3]], [[TBD4]], and [[TBD5]] in this document with the respective IANA assignments.

7.6. Network Time Security Key Establishment Record Types Registry

IANA is requested to create a new registry entitled "Network Time Security Key Establishment Record Types". Entries SHALL have the following fields:

Record Type Number (REQUIRED): An integer in the range 0-32767 inclusive.

Description (REQUIRED): A short text description of the purpose of the field.

Reference (REQUIRED): A reference to a document specifying the semantics of the record.

The policy for allocation of new entries in this registry SHALL vary by the Record Type Number, as follows:

0-1023: IETF Review

1024-16383: Specification Required

16384-32767: Private and Experimental Use

Applications for new entries SHALL specify the contents of the Description, Set Critical Bit, and Reference fields as well as which of the above ranges the Record Type Number should be allocated from. Applicants MAY request a specific Record Type Number and such requests MAY be granted at the registrar’s discretion.

The initial contents of this registry SHALL be as follows:

Record Type Number	Description	Reference
0	End of Message	[[this memo]], Section 4.1.1
1	NTS Next Protocol Negotiation	[[this memo]], Section 4.1.2
2	Error	[[this memo]], Section 4.1.3
3	Warning	[[this memo]], Section 4.1.4
4	AEAD Algorithm Negotiation	[[this memo]], Section 4.1.5
5	New Cookie for NTPv4	[[this memo]], Section 4.1.6
6	NTPv4 Server Negotiation	[[this memo]], Section 4.1.7
7	NTPv4 Port Negotiation	[[this memo]], Section 4.1.8
16384-32767	Reserved for Private & Experimental Use	[[this memo]]

7.7. Network Time Security Next Protocols Registry

IANA is requested to create a new registry entitled "Network Time Security Next Protocols". Entries SHALL have the following fields:

Protocol ID (REQUIRED): An integer in the range 0-65535 inclusive, functioning as an identifier.

Protocol Name (REQUIRED): A short text string naming the protocol being identified.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries in these registries SHALL vary by their Protocol ID, as follows:

0-1023: IETF Review

1024-32767: Specification Required

32768-65535: Private and Experimental Use

The initial contents of this registry SHALL be as follows:

Protocol ID	Protocol Name	Reference
0	Network Time Protocol version 4 (NTPv4)	[[this memo]]
32768-65535	Reserved for Private or Experimental Use	Reserved by [[this memo]]

7.8. Network Time Security Error and Warning Codes Registries

IANA is requested to create two new registries entitled "Network Time Security Error Codes" and "Network Time Security Warning Codes". Entries in each SHALL have the following fields:

Number (REQUIRED): An integer in the range 0-65535 inclusive

Description (REQUIRED): A short text description of the condition.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries in these registries SHALL vary by their Number, as follows:

0-1023: IETF Review

1024-32767: Specification Required

32768-65535: Private and Experimental Use

The initial contents of the Network Time Security Error Codes Registry SHALL be as follows:

Number	Description	Reference
0	Unrecognized Critical Extension	[[this memo]], Section 4.1.3
1	Bad Request	[[this memo]], Section 4.1.3
32768-65535	Reserved for Private or Experimental Use	Reserved by [[this memo]]

The Network Time Security Warning Codes Registry SHALL initally be empty except for the reserved range, i.e.:

Number	Description	Reference
32768-65535	Reserved for Private or Experimental Use	Reserved by [[this memo]]

8. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

8.1. Implementation PoC 1

Organization: Ostfalia University of Applied Science

Implementor: Martin Langer

Maturity: Proof-of-Concept Prototype

This implementation was used to verify consistency and to ensure completeness of this specification. It also demonstrate interoperability with NTP’s client-server mode messages.

8.1.1. Coverage

This implementation covers the complete specification.

8.1.2. Licensing

The code is released under a Apache License 2.0 license.

The source code is available at: https://gitlab.com/MLanger/nts/

8.1.3. Contact Information

Contact Martin Langer: mart.langer@ostfalia.de

8.1.4. Last Update

The implementation was updated 3rd May 2018.

8.2. Implementation PoC 2

Organization: tbd

Implementor: Daniel Fox Franke

Maturity: Proof-of-Concept Prototype

This implementation was used to verify consistency and to ensure completeness of this specification.

8.2.1. Coverage

This implementation provides the client and the server for the initial TLS handshake and NTS key exchange. It provides the the client part of the NTS protected NTP messages.

8.2.2. Licensing

Public domain.

The source code is available at: https://github.com/dfoxfranke/nts-hackathon

8.2.3. Contact Information

Contact Daniel Fox Franke: dfoxfranke@gmail.com

8.2.4. Last Update

The implementation was updated 16th March 2018.

8.3. Interoperability

The Interoperability tests distinguished between NTS key exchange and NTS time exchange messages. For the NTS key exchange, interoperability between the two implementations has been verified successfully. Interoperability of NTS time exchange messages has been verified successfully for the case that PoC 1 represents the server and PoC 2 the client.

These tests successfully demonstrate that there are at least two running implementations of this draft which are able to interoperate.

9. Security Considerations

9.1. Sensitivity to DDoS attacks

The introduction of NTS brings with it the introduction of asymmetric cryptography to NTP. Asymmetric cryptography is necessary for initial server authentication and AEAD key extraction. Asymmetric cryptosystems are generally orders of magnitude slower than their symmetric counterparts. This makes it much harder to build systems that can serve requests at a rate corresponding to the full line speed of the network connection. This, in turn, opens up a new possibility for DDoS attacks on NTP services.

The main protection against these attacks in NTS lies in that the use of asymmetric cryptosystems is only necessary in the initial NTS-KE phase of the protocol. Since the protocol design enables separation of the NTS-KE and NTP servers, a successful DDoS attack on an NTS-KE server separated from the NTP service it supports will not affect NTP users that have already performed initial authentication, AEAD key extraction, and cookie exchange.

NTS users should also consider that they are not fully protected against DDoS attacks by on-path adversaries. In addition to dropping packets and attacks such as those described in Section 9.4, an on-path attacker can send spoofed kiss-o'-death replies, which are not authenticated, in response to NTP requests. This could result in significantly increased load on the NTS-KE server. Implementers have to weigh the user's need for unlinkability against the added resilience that comes with cookie reuse in cases of NTS-KE server unavailability.

9.2. Avoiding DDoS Amplification

Certain non-standard and/or deprecated features of the Network Time Protocol enable clients to send a request to a server which causes the server to send a response much larger than the request. Servers

which enable these features can be abused in order to amplify traffic volume in DDoS attacks by sending them a request with a spoofed source IP. In recent years, attacks of this nature have become an endemic nuisance.

NTS is designed to avoid contributing any further to this problem by ensuring that NTS-related extension fields included in server responses will be the same size as the NTS-related extension fields sent by the client. In particular, this is why the client is required to send a separate and appropriately padded-out NTS Cookie Placeholder extension field for every cookie it wants to get back, rather than being permitted simply to specify a desired quantity.

Due to the RFC 7822 [RFC7822] requirement that extensions be padded and aligned to four-octet boundaries, response size may still in some cases exceed request size by up to three octets. This is sufficiently inconsequential that we have declined to address it.

9.3. Initial Verification of Server Certificates

NTS's security goals are undermined if the client fails to verify that the X.509 certificate chain presented by the NTS-KE server is valid and rooted in a trusted certificate authority. RFC 5280 [RFC5280] and RFC 6125 [RFC6125] specify how such verification is to be performed in general. However, the expectation that the client does not yet have a correctly-set system clock at the time of certificate verification presents difficulties with verifying that the certificate is within its validity period, i.e., that the current time lies between the times specified in the certificate's notBefore and notAfter fields. It may be operationally necessary in some cases for a client to accept a certificate which appears to be expired or not yet valid. While there is no perfect solution to this problem, there are several mitigations the client can implement to make it more difficult for an adversary to successfully present an expired certificate:

Check whether the system time is in fact unreliable. If the system clock has previously been synchronized since last boot, then on operating systems which implement a kernel-based phase-locked-loop API, a call to ntp_gettime() should show a maximum error less than NTP_PHASE_MAX. In this case, the clock SHOULD be considered reliable and certificates can be strictly validated.

Allow the system administrator to specify that certificates should *always* be strictly validated. Such a configuration is appropriate on systems which have a battery-backed clock and which can reasonably prompt the user to manually set an approximately-correct time if it appears to be needed.

Once the clock has been synchronized, periodically write the current system time to persistent storage. Do not accept any certificate whose notAfter field is earlier than the last recorded time.

Do not process time packets from servers if the time computed from them falls outside the validity period of the server's certificate.

Use multiple time sources. The ability to pass off an expired certificate is only useful to an adversary who has compromised the corresponding private key. If the adversary has compromised only a minority of servers, NTP's selection algorithm (RFC 5905 section 11.2.1 [RFC5905]) will protect the client from accepting bad time from the adversary-controlled servers.

9.4. Delay Attacks

In a packet delay attack, an adversary with the ability to act as a man-in-the-middle delays time synchronization packets between client and server asymmetrically [RFC7384]. Since NTP's formula for computing time offset relies on the assumption that network latency is roughly symmetrical, this leads to the client to compute an inaccurate value [Mizrahi]. The delay attack does not reorder or modify the content of the exchanged synchronization packets. Therefore, cryptographic means do not provide a feasible way to mitigate this attack. However, the maximum error that an adversary can introduce is bounded by half of the round trip delay.

RFC 5905 [RFC5905] specifies a parameter called MAXDIST which denotes the maximum round-trip latency (including not only the immediate round trip between client and server, but the whole distance back to the reference clock as reported in the Root Delay field) that a client will tolerate before concluding that the server is unsuitable for synchronization. The standard value for MAXDIST is one second, although some implementations use larger values. Whatever value a client chooses, the maximum error which can be introduced by a delay attack is MAXDIST/2.

Usage of multiple time sources, or multiple network paths to a given time source [Shpiner], may also serve to mitigate delay attacks if the adversary is in control of only some of the paths.

9.5. Random Number Generation

At various points in NTS, the generation of cryptographically secure random numbers is required. Whenever this draft specifies the use of random numbers, cryptographically secure random number generation

MUST be used. RFC 4086 [RFC4086] contains guidelines concerning this topic.

10. Privacy Considerations

10.1. Unlinkability

Unlinkability prevents a device from being tracked when it changes network addresses (e.g. because said device moved between different networks). In other words, unlinkability thwarts an attacker that seeks to link a new network address used by a device with a network address that it was formerly using, because of recognizable data that the device persistently sends as part of an NTS-secured NTP association. This is the justification for continually supplying the client with fresh cookies, so that a cookie never represents recognizable data in the sense outlined above.

NTS's unlinkability objective is merely to not leak any additional data that could be used to link a device's network address. NTS does not rectify legacy linkability issues that are already present in NTP. Thus, a client that requires unlinkability must also minimize information transmitted in a client query (mode 3) packet as described in the draft [I-D.ietf-ntp-data-minimization].

The unlinkability objective only holds for time synchronization traffic, as opposed to key exchange traffic. This implies that it cannot be guaranteed for devices that function not only as time clients, but also as time servers (because the latter can be externally triggered to send authentication data).

It should also be noted that it could be possible to link devices that operate as time servers from their time synchronization traffic, using information exposed in (mode 4) server response packets (e.g. reference ID, reference time, stratum, poll). Also, devices that respond to NTP control queries could be linked using the information revealed by control queries.

Note that the unlinkability objective does not prevent a client device to be tracked by its time servers.

10.2. Confidentiality

NTS does not protect the confidentiality of information in NTP's header fields. When clients implement [I-D.ietf-ntp-data-minimization], client packet headers do not contain any information which the client could conceivably wish to keep secret: one field is random, and all others are fixed. Information in server packet headers is likewise public: the origin

timestamp is copied from the client's (random) transmit timestamp, and all other fields are set the same regardless of the identity of the client making the request.

Future extension fields could hypothetically contain sensitive information, in which case NTS provides a mechanism for encrypting them.

11. Acknowledgements

The authors would like to thank Richard Barnes, Steven Bellovin, Patrik Faeltstroem (Faltstrom), Scott Fluhrer, Sharon Goldberg, Russ Housley, Martin Langer, Miroslav Lichvar, Aanchal Malhotra, Dave Mills, Danny Mayer, Karen O'Donoghue, Eric K. Rescorla, Stephen Roettger, Kurt Roeckx, Kyle Rose, Rich Salz, Brian Sniffen, Susan Sons, Douglas Stebila, Harlan Stenn, Joachim Stroembergsson (Strombergsson), Martin Thomson, and Richard Welty for contributions to this document and comments on the design of NTS.

12. References

12.1. Normative References

[ANSI.X3-4.1986]
American National Standards Institute, "Coded Character Set - 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, DOI 10.17487/RFC5297, October 2008, <https://www.rfc-editor.org/info/rfc5297>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[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, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC7507] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", RFC 7507, DOI 10.17487/RFC7507, April 2015, <https://www.rfc-editor.org/info/rfc7507>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7822] Mizrahi, T. and D. Mayer, "Network Time Protocol Version 4 (NTPv4) Extension Fields", RFC 7822, DOI 10.17487/RFC7822, March 2016, <https://www.rfc-editor.org/info/rfc7822>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

12.2. Informative References

[I-D.ietf-ntp-data-minimization] Franke, D. and A. Malhotra, "NTP Client Data Minimization", draft-ietf-ntp-data-minimization-02 (work in progress), July 2018.

[Mizrahi] Mizrahi, T., "A game theoretic analysis of delay attacks against time synchronization protocols", in Proceedings of Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2012, pp. 1-6, September 2012.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/info/rfc5077>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[Shpiner] "Multi-path Time Protocols", in Proceedings of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS), September 2013.

Appendix A. Terms and Abbreviations

AEAD Authenticated Encryption with Associated Data [RFC5116]

ALPN Application-Layer Protocol Negotiation [RFC7301]

C2S Client-to-server

DDoS Distributed Denial-of-Service

EF Extension Field [RFC5905]

HKDF Hashed Message Authentication Code-based Key Derivation Function [RFC5869]

IANA Internet Assigned Numbers Authority

IP Internet Protocol

KoD Kiss-o'-Death [RFC5905]

NTP Network Time Protocol [RFC5905]

NTS Network Time Security

NTS-KE Network Time Security Key Exchange

S2C Server-to-client

SCSV Signaling Cipher Suite Value [RFC7507]

TCP Transmission Control Protocol [RFC0793]

TLS Transport Layer Security [RFC8446]

UDP User Datagram Protocol [RFC0768]

Authors' Addresses

Daniel Fox Franke
Akamai Technologies
150 Broadway
Cambridge, MA 02142
United States

Email: dafranke@akamai.com
URI: https://www.dfranke.us

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-4471
Email: kristof.teichel@ptb.de

Marcus Dansarie

Email: marcus@dansarie.se

Ragnar Sundblad
Netnod

Email: ragge@netnod.se

Internet Working Group
Internet Draft
Intended status: Standards Track
Expires: April 2017

Y. Jiang, Ed.
X. Liu
J. Xu
Huawei
R. Cummings, Ed.
National Instruments
October 20, 2016

YANG Data Model for IEEE 1588v2
draft-ietf-tictoc-1588v2-yang-00

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and 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 April 20, 2017.

Copyright Notice

Abstract

This document defines a YANG data model for the configuration of IEEE 1588-2008 devices and clocks, and also retrieval of the configuration information, data set and running states of IEEE 1588-2008 clocks.

Table of Contents

1. Introduction

As a synchronization protocol, IEEE 1588-2008 (also known as IEEE 1588v2) [IEEE1588] is widely supported in the carrier networks, industrial networks, automotive networks, and many other applications. It can provide high precision time synchronization as high as nano-seconds. The protocol depends on a Precision Time Protocol (PTP) engine to decide its state automatically, and a PTP transportation layer to carry the PTP timing and various quality messages. The configuration parameters and state data sets of IEEE 1588-2008 are numerous.

According to the concepts described in [RFC3444], IEEE 1588-2008 itself provides an information model in its normative

specifications for the data sets (in IEEE 1588-2008 clause 8). Some standardization organizations including the IETF have specified data models in MIBs (Management Information Bases) for IEEE 1588-2008 data sets (e.g. [PTP-MIB], [IEEE8021AS]). Since these MIBs are typically focused on retrieval of state data using the Simple Network Management Protocol (SNMP), configuration is not considered.

Some service providers and applications require that the management of the IEEE 1588-2008 synchronization network be flexible and more Internet-based (typically overlaid on their transport networks). Software Defined Network (SDN) is another driving factor which demands an improved configuration capability of synchronization networks.

YANG [RFC6020] is a data modeling language used to model configuration and state data manipulated by network management protocols like the Network Configuration Protocol (NETCONF) [RFC6241]. A small set of built-in data types are defined in [RFC6020], and a collection of common data types are further defined in [RFC6991]. Advantages of YANG include Internet based configuration capability, validation, roll-back and so on. All of these characteristics make it attractive to become another candidate modeling language for IEEE 1588-2008.

This document defines a YANG [RFC6020] data model for the configuration of IEEE 1588-2008 devices and clocks, and also retrieval of the state data of IEEE 1588-2008 clocks. The data model is based on the PTP data sets as specified in [IEEE1588]. The router specific IEEE 1588-2008 information is out of scope of this document.

When used in practice, network products in support of synchronization typically conform to one or more IEEE 1588-2008 profiles. Each profile specifies how IEEE 1588-2008 is used in a given industry (e.g. telecom, automotive) and application. A profile can require features that are optional in IEEE 1588-2008, and it can specify new features that use IEEE 1588-2008 as a foundation.

It is expected that the IEEE 1588-2008 YANG module will be used as follows:

o The IEEE 1588-2008 YANG module can be used as-is for products that conform to one of the default profiles specified in IEEE 1588-2008.

o When the IEEE 1588 standard is revised (e.g. the IEEE 1588 revision in progress scheduled to be published in 2017), it will add some new optional features to its data sets. The YANG module of this document can be revised and extended to add the new features (e.g. of IEEE 1588-2017). The YANG "revision" can be used to indicate changes to the YANG module.

o A profile standard based on IEEE 1588-2008 may create a dedicated YANG module for its profile. The profile's YANG module may use YANG "import" to import the IEEE 1588-2008 YANG module as its foundation. Then the profile's YANG module can use YANG "augment" to add any profile-specific enhancements.

o A product that conforms to a profile standard can also create its own YANG module. The product's YANG module can "import" the profile's module, and then use YANG "augment" to add any product-specific enhancements.

1.1. Conventions used in this document

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 [RFC2119].

1.2. Terminology

Terminologies used in this document are extracted from [IEEE1588] and [PTP-MIB].

BC Boundary Clock

DS Data Set

E2E End-to-End

EUI Extended Unique Identifier.

GPS Global Positioning System

IANA Internet Assigned Numbers Authority

IP Internet Protocol

NIST National Institute of Standards and Technology

NTP Network Time Protocol

OC	Ordinary Clock
P2P	Peer-to-Peer
PTP	Precision Time Protocol
TAI	International Atomic Time
TC	Transparent Clock
UTC	Coordinated Universal Time

2. IEEE 1588-2008 YANG Model hierarchy

This section describes the hierarchy of IEEE 1588-2008 YANG module. Query and configuration of device wide or port specific configuration information and clock data set is described for this version.

Query and configuration of clock information include:

- Clock data set attributes in a clock node, including: current-ds, parent-ds, default-ds, time-properties-ds, and transparentClock-default-ds.

- Port specific data set attributes, including: port-ds and transparentClock-port-ds.

A simplified graphical representation of the data model is typically used by YANG modules as described in [REST-CONF]. This document uses the same representation and the meaning of the symbols in these diagrams is as follows:

o Brackets "[" and "]" enclose list keys.

o Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

o Ellipsis ("...") stands for contents of subtrees that are not shown.

```
   module: ietf-ptp-dataset
      +--rw instance-list* [instance-number]
      |  +--rw instance-number                      uint8
      |  +--rw default-ds
      |  |  +--rw two-step-flag?    boolean
      |  |  +--rw clock-identity?   binary
      |  |  +--rw number-ports?     uint16
      |  |  +--rw clock-quality
      |  |  |  +--rw clock-class?                  uint8
      |  |  |  +--rw clock-accuracy?               uint8
      |  |  |  +--rw offset-scaled-log-variance?   uint16
      |  |  +--rw priority1?        uint8
      |  |  +--rw priority2?        uint8
      |  |  +--rw domain-number     uint8
      |  |  +--rw slave-only?       boolean
      |  +--rw current-ds
      |  |  +--rw steps-removed?        uint16
      |  |  +--rw offset-from-master?   binary
      |  |  +--rw mean-path-delay?      binary
      |  +--rw parent-ds
      |  |  +--rw parent-port-identity
      |  |  |  +--rw clock-identity?    binary
      |  |  |  +--rw port-number?       uint16
      |  |  +--rw parent-stats?         boolean
      |  |  +--rw observed-parent-offset-scaled-log-variance?   uint16
      |  |  +--rw observed-parent-clock-phase-change-rate?      int32
      |  |  +--rw grandmaster-identity?                         binary
      |  |  +--rw grandmaster-clock-quality
      |  |  |  +--rw grandmaster-clock-class?                   uint8
      |  |  |  +--rw grandmaster-clock-accuracy?                uint8
      |  |  |  +--rw grandmaster-offset-scaled-log-variance?    uint16
      |  |  +--rw grandmaster-priority1?                        uint8
      |  |  +--rw grandmaster-priority2?                        uint8
      |  +--rw time-properties-ds
      |  |  +--rw current-utc-offset-valid?   boolean
      |  |  +--rw current-utc-offset?         uint16
      |  |  +--rw leap59?                     boolean
      |  |  +--rw leap61?                     boolean
      |  |  +--rw time-traceable?             boolean
      |  |  +--rw frequency-traceable?        boolean
      |  |  +--rw ptp-timescale?              boolean
      |  |  +--rw time-source?                uint8
      |  +--rw port-ds-list* [port-number]
      |     +--rw port-number         -> ../port-identity/port-number
      |     +--rw port-identity
      |     |  +--rw clock-identity?   binary
      |     |  +--rw port-number?      uint16
```

```
|      +--rw port-state?                    uint8
|      +--rw log-min-delay-req-interval?    int8
|      +--rw peer-mean-path-delay?          int64
|      +--rw log-announce-interval?         int8
|      +--rw announce-receipt-timeout?      uint8
|      +--rw log-sync-interval?             int8
|      +--rw delay-mechanism?               enumeration
|      +--rw log-min-pdelay-req-interval?   int8
|      +--rw version-number?                uint8
+--rw transparent-clock-default-ds
|  +--rw clock-identity?    binary
|  +--rw number-ports?      uint16
|  +--rw delay-mechanism?   enumeration
|  +--rw primary-domain?    uint8
+--rw transparent-clock-port-ds-list* [port-number]
   +--rw port-number         -> ../port-identity/port-number
   +--rw port-identity
   |  +--rw clock-identity?   binary
   |  +--rw port-number?      uint16
   +--rw log-min-pdelay-req-interval?   int8
   +--rw faulty-flag?                   boolean
   +--rw peer-mean-path-delay?          int64
```

2.1. Interpretations from IEEE 1588 Working Group

The preceding model and the associated YANG module have some subtle differences from the data set specifications of IEEE Std 1588-2008. These differences are based on interpretation from the IEEE 1588 Working Group, and are intended to provide compatibility with future revisions of the IEEE 1588 standard.

In IEEE Std 1588-2008, a physical product can implement multiple PTP clocks (i.e. ordinary, boundary, or transparent clock). As specified in 1588-2008 subclause 7.1, each of the multiple clocks operates in an independent domain. However, the organization of multiple PTP domains was not clear in the data sets of IEEE Std 1588-2008. This document introduces the concept of PTP instance as described in the new revision of IEEE 1588. The instance concept is used exclusively to allow for optional support of multiple domains. The instance number has no usage within PTP messages.

Based on statements in IEEE 1588-2008 subclauses 8.3.1. and 10.1, most transparent clock products have interpreted the transparent clock data sets to reside as a singleton at the root level of the managed product. Since 1588-2008 transparent clocks are domain independent, the instance concept is not applicable for them.

3. IEEE 1588-2008 YANG Module

<CODE BEGINS> file "ietf-ptp-dataset@2016-10-20"

```
module ietf-ptp-dataset{
  namespace "urn:ietf:params:xml:ns:yang:ietf-ptp-dataset";
  prefix "ptp-dataset";
  organization "IETF TICTOC WG";
  contact
      "WG Web:   http://tools.ietf.org/wg/tictoc/
       WG List:  <mailto:tictoc@ietf.org>
       WG Chair: Karen O'Donoghue
                 <mailto:odonoghue@isoc.org>
       WG Chair: Yaakov Stein
                 <mailto: Yaakov_s@rad.com>
       Editor:   Yuanlong Jiang
                 <mailto:jiangyuanlong@huawei.com>
       Editor:   Rodney Cummings
                 <mailto:rodney.cummings@ni.com>";
  description
    "This YANG module defines a data model for the configuration
    of IEEE 1588-2008 clocks, and also retrieval of the state
    data of IEEE 1588-2008 clocks.";
  revision "2016-10-20" {
    description "Original version.";
    reference "draft-ietf-tictoc-1588v2-yang";
   }

  grouping default-ds-entry {
    description
      "Collection of members of the default data set.";

    leaf two-step-flag {
      type boolean;
      description
        "The flag indicates whether the Two Step process is
         used.";
    }
    leaf clock-identity {
      type binary {
        length "8";
      }
      description
        "The clockIdentity of the local clock";
    }

    leaf number-ports {
```

```
        type uint16;
        description
          "The number of PTP ports on the device.";
      }

      container clock-quality {
        description
          "The clockQuality of the local clock. It contains
         clockClass, clockAccuracy and offsetScaledLogVariance.";

        leaf clock-class {
          type uint8;
          default 248;
          description
            "The clockClass denotes the traceability of the time
            or frequency distributed by the grandmaster clock.";
        }
        leaf clock-accuracy {
          type uint8;
          description
            "The clockAccuracy indicates the expected accuracy
             of a clock when it is the grandmaster.";
        }
        leaf offset-scaled-log-variance {
          type uint16;
          description
            "An estimate of the variations of the local clock
            from a linear timescale when it is not synchronized
            to another clock using the protocol.";
        }
      }

      leaf priority1 {
        type uint8;
        description
          "The priority1 attribute of the local clock.";
      }
      leaf priority2{
        type uint8;
        description
          "The priority2 attribute of the local clock. ";
      }

      leaf domain-number {
        type uint8;
        description
```

```
          "The domain number of the current syntonization
     domain.";
       }

       leaf slave-only {
         type boolean;
         description
           "Indicates whether the clock is a slave-only clock.";

       }
     }

     grouping current-ds-entry {
       description
         "Collection of members of current data set.";

       leaf steps-removed {
         type uint16;
         default 0;
         description
           "The number of communication paths traversed
           between the local clock and the grandmaster clock.";

       }
       leaf offset-from-master {
         type binary {
           length "1..255";
         }
         description
           "An implementation-specific representation of the
           current value of the time difference between a master
           and a slave clock as computed by the slave.";
       }
       leaf mean-path-delay {
         type binary {
           length "1..255";
         }
         description
           "An implementation-specific representation of the
            current value of the mean propagation time between a
            master and slave clock as computed by the slave.";

       }
     }

     grouping parent-ds-entry {
       description
```

```
        "Collection of members of the parent data set.";

      container parent-port-identity {
        description
          "The portIdentity of the port on the master.
          It contains two members: clockIdentity and portNumer.";

        leaf clock-identity {
          type binary {
            length "8";
          }
          description
            "The clockIdentity of the master clock.";
        }

        leaf port-number {
          type uint16;
          description
            "The portNumber for the port on the specific
             master.";
        }
      }
      leaf parent-stats {
        type boolean;
        default false;
        description
          "Indicates whether the values of
           observedParentOffsetScaledLogVariance and
           observedParentClockPhaseChangeRate of parentDS
           have been measured and are valid.";
      }
      leaf observed-parent-offset-scaled-log-variance {
        type uint16;
        default 0xFFFF;
        description
          "An estimate of the parent clock's PTP variance
           as observed by the slave clock.";
      }
      leaf observed-parent-clock-phase-change-rate {
        type int32;
        description
          "An estimate of the parent clock's phase change rate
           as observed by the slave clock.";
      }
      leaf grandmaster-identity {
        type binary{
          length "8";
```

```
        }
        description
         "The clockIdentity attribute of the grandmaster clock.";

      }
      container grandmaster-clock-quality {
        description
          "The clockQuality of the grandmaster clock. It contains
         clockClass, clockAccuracy and offsetScaledLogVariance.";

        leaf grandmaster-clock-class {
          type uint8;
          default 248;
          description
           "The clockClass attribute of the grandmaster clock.";

        }
        leaf grandmaster-clock-accuracy {
          type uint8;
          description
            "The clockAccuracy attribute of the grandmaster
             clock.";
        }
        leaf grandmaster-offset-scaled-log-variance {
          type uint16;
          description
            "The offsetScaledLogVariance of the grandmaster
             clock.";
        }
      }
      leaf grandmaster-priority1 {
        type uint8;
        description
          "The priority1 attribute of the grandmaster clock.";

      }
      leaf grandmaster-priority2 {
        type uint8;
        description
          "The priority2 attribute of the grandmaster clock.";

      }

    }

    grouping time-properties-ds-entry {
      description
```

```
        "Collection of members of the timeProperties data set.";

      leaf current-utc-offset-valid {
        type boolean;
        description
          "Indicates whether current UTC offset is valid.";
      }
      leaf current-utc-offset {
        type uint16;
        description
          "The offset between TAI and UTC when the epoch of the
           PTP system is the PTP epoch, otherwise the value has
           no meaning.";
      }
      leaf leap59 {
        type boolean;
        description
          "Indicates whether the last minute of the current UTC
           day contains 59 seconds.";
      }
      leaf leap61 {
        type boolean;
        description
          "Indicates whether the last minute of the current UTC
           day contains 61 seconds.";
      }
      leaf time-traceable {
        type boolean;
        description
          "Indicates whether the timescale and the
           currentUtcOffset are traceable to a primary
           reference.";
      }
      leaf frequency-traceable {
        type boolean;
        description
          "Indicates whether the frequency determining the
           timescale is traceable to a primary reference.";
      }
      leaf PTP-timescale {
        type boolean;
        description
          "Indicates whether the clock timescale
           of the grandmaster clock is PTP.";
      }
      leaf time-source {
        type uint8;
```

```
        description
          "The source of time used by the grandmaster clock.";

      }
    }

    grouping port-ds-entry {
      description
        "Collection of members of the port data set.";

      container port-identity {
        description
          "The PortIdentity attribute of the local port.
           It contains two members: clockIdentity and
           portNumber.";

        leaf clock-identity {
          type binary {
            length "8";
          }
          description
            "The clockIdentity of the local clock.";
        }

        leaf port-number {
          type uint16;
          description
            "The portNumber for a port on the local clock.";

        }
      }

      leaf port-state {
        type uint8;
        default 1;
        description
          "Current state associated with the port.";
      }

      leaf log-min-delay-req-interval {
        type int8;
        description
          "The logarithm to the base 2 of the minDelayReqInterval
           (the minimum permitted mean time interval between
           successive Delay_Req messages).";
      }
```

```
      leaf peer-mean-path-delay {
        type int64;
        default 0;
        description
          "An estimate of the current one-way propagation delay
           on the link when the delayMechanism is P2P, otherwise
           it is zero.";
      }

      leaf log-announce-interval {
        type int8;
        description
          "The logarithm to the base 2 of the of the mean
           announceInterval (mean time interval between
           successive Announce messages).";
      }

      leaf announce-receipt-timeout {
        type uint8;
        description
          "The number of announceInterval that have to pass
           without receipt of an announce message before the
           occurrence of the event ANNOUNCE_RECEIPT_TIMEOUT_
           EXPIRES.";
      }

      leaf log-sync-interval {
        type int8;
        description
          "The logarithm to the base 2 of the mean SyncInterval
           for multicast messages.  The rates for unicast
           transmissions are negotiated separately on a per port
           basis.";
      }

      leaf delay-mechanism {
        type enumeration {
          enum E2E {
            value 01;
            description
              "The port uses the delay request-response
               mechanism.";
          }
          enum P2P {
            value 02;
            description
              "The port uses the peer delay mechanism.";
```

```
          }
          enum DISABLED {
            value 254;
            description
              "The port does not implement the delay
               mechanism.";
          }
        }
        description
          "The propagation delay measuring option used by the
           port in computing meanPathDelay.";
      }

      leaf log-min-Pdelay-req-interval {
        type int8;
        description
          "The logarithm to the base 2 of the
           minPdelayReqInterval (minimum permitted mean time
           interval between successive Pdelay_Req messages).";

      }

      leaf version-number {
        type uint8;
        description
          "The PTP version in use on the port.";
      }
    }

    grouping transparent-clock-default-ds-entry {
      description
        "Collection of members of the transparentClockDefault data
          set (default data set for a transparent clock).";

      leaf clock-identity {
        type binary {
          length "8";
        }
        description
          "The clockIdentity of the transparent clock.";
      }
      leaf number-ports {
        type uint16;
        description
          "The number of PTP ports on the device.";
      }
      leaf delay-mechanism {
```

```
        type enumeration {
          enum E2E {
            value 1;
            description
              "The port uses the delay request-response
               mechanism.";
          }
          enum P2P {
            value 2;
            description
              "The port uses the peer delay mechanism.";
          }
          enum DISABLED {
            value 254;
            description
              "The port does not implement the delay
               mechanism.";
          }
        }
        description
          "The propagation delay measuring option
           used by the transparent clock.";
      }
      leaf primary-domain {
        type uint8;
        default 0;
        description
         "The domainNumber of the primary syntonization domain.";

      }
    }

    grouping transparent-clock-port-ds-entry {
      description
        "Collection of members of the transparentClockPort data
         set (port data set for a transparent clock).";

      container port-identity {
        description
          "This object specifies the portIdentity of the local
           port.";

        leaf clock-identity {
          type binary {
            length "8";
          }
          description
```

```
            "The clockIdentity of the transparent clock.";
        }

        leaf port-number {
          type uint16;
          description
            "The portNumber for a port on the transparent
             clock.";
        }
      }
      leaf log-min-pdelay-req-interval {
        type int8;
        description
          "The logarithm to the base 2 of the
           minPdelayReqInterval (minimum permitted mean time
           interval between successive Pdelay_Req messages).";
      }
      leaf faulty-flag {
        type boolean;
        default false;
        description
          "Indicates whether the port is faulty.";
      }
      leaf peer-mean-path-delay {
        type int64;
        default 0;
        description
          "An estimate of the current one-way propagation delay
           on the link when the delayMechanism is P2P, otherwise
           it is zero.";
      }
    }

    list instance-list {

      key "instance-number";

      description
        "List of one or more PTP datasets in the device,
         one for each domain-number (see IEEE 1588-2008 subclause
         6.3)";

      leaf instance-number {
        type uint8;
        description
          "The instance number of the current PTP instance";
      }
```

```
        container default-ds {
          description
            "The default data set of the clock.";
          uses default-ds-entry;
        }

        container current-ds {
          description
            "The current data set of the clock.";
          uses current-ds-entry;
        }

        container parent-ds {
          description
            "The parent data set of the clock.";
          uses parent-ds-entry;
        }

        container time-properties-ds {
          description
            "The timeProperties data set of the clock.";
          uses time-properties-ds-entry;
        }

        list port-ds-list {
          key "port-number";
          description
            "List of port data sets of the clock.";
          leaf port-number{
            type leafref{
              path "../port-identity/port-number";
            }
            description
              "Refers to the portNumber memer of
              portDS.portIdentity.";
          }
          uses port-ds-entry;
        }
      }


      container transparent-clock-default-ds {
        description
          "The members of the transparentClockDefault Data Set";
        uses transparent-clock-default-ds-entry;
      }
```

```
    list transparent-clock-port-ds-list {
      key "port-number";
      description
        "List of transparentClockPort data sets
         of the transparent clock.";
      leaf port-number {
        type leafref {
          path "../port-identity/port-number";
        }
         description
           "Refers to the portNumber memer
            of transparentClockPortDS.portIdentity.";
       }
       uses transparent-clock-port-ds-entry;
     }
  }
  <CODE ENDS>
```

4. Security Considerations

YANG modules are designed to be accessed via the NETCONF protocol [RFC6241], thus security considerations in [RFC6241] apply here. Security measures such as using the NETCONF over SSH [RFC6242] and restricting its use with access control [RFC6536] can further improve its security, avoid injection attacks and misuse of the protocol.

Some data nodes defined in this YANG module are writable, and any changes to them may adversely impact a synchronization network.

5. IANA Considerations

This document registers a URI in the IETF XML registry, and the following registration is requested to be made:
URI: urn:ietf:params:xml:ns:yang:ietf-ptp-dataset

This document registers a YANG module in the YANG Module Names: name: ietf-ptp-dataset namespace: urn:ietf:params:xml:ns:yang:ietf-ptp-dataset

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) ", RFC 6020, October 2010

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013

[IEEE1588] IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE Std 1588-2008, July 2008

6.2. Informative References

[IEEE8021AS] IEEE, "Timing and Synchronizations for Time-Sensitive Applications in Bridged Local Area Networks", IEEE 802.1AS-2001, 2011

[PTP-MIB] Shankarkumar, V., Montini, L., Frost, T., and Dowd, G., "Precision Time Protocol Version 2 (PTPv2) Management Information Base", draft-ietf-tictoc-ptp-mib-11, Work in progress

[REST-CONF] Bierman, A., Bjorklund, M., and Watsen, K., "RESTCONF protocol", draft-ietf-netconf-restconf-17, Work in progress

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, January 2003

[RFC4663] Harrington, D., "Transferring MIB Work from IETF Bridge MIB WG to IEEE 802.1 WG", RFC 4663, September 2006

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012

7. Acknowledgments

The authors would like to thank reviews and suggestions from Mahesh Jethanandani and Tal Mizrahi.

Appendix A Transferring YANG Work to IEEE 1588 WG (Informational)

This appendix describes a future plan to transition responsibility for IEEE 1588 YANG modules from the IETF TICTOC Working Group (WG) to the IEEE 1588 WG, which develops the time synchronization technology that the YANG modules are designed to manage.

This appendix is forward-looking with regard to future standardization roadmaps in IETF and IEEE. Since those roadmaps cannot be predicted with significant accuracy, this appendix is informational, and it does not specify imperatives or normative specifications of any kind.

The IEEE 1588-2008 YANG module of this standard represents a cooperation between IETF (for YANG) and IEEE (for 1588). For the initial standardization of IEEE-1588 YANG modules, the information model is relatively clear (i.e. IEEE 1588 data sets), but expertise in YANG is required, making IETF an appropriate location for the standards. The TICTOC WG has expertise with IEEE 1588, making it the appropriate location within IETF.

The IEEE 1588 WG anticipates future changes to its standard on an ongoing basis. As IEEE 1588 WG members gain practical expertise with YANG, the IEEE 1588 WG will become more appropriate for standardization of its YANG modules. As the IEEE 1588 standard is revised and/or amended, IEEE 1588 members can more effectively synchronize the revision of this YANG module with future versions of the IEEE 1588 standard.

This appendix is meant to establish some clear expectations between IETF and IEEE about the future transfer of IEEE 1588 YANG modules to the IEEE 1588 WG. The goal is to assist in making the future transfer as smooth as possible. As the transfer takes place, some case-by-case situations are likely to arise, which can be handled by discussion on the IETF TICTOC WG mailing lists and/or appropriate liaisons.

This appendix obtained insight from [RFC4663], an informational memo that described a similar transfer of MIB work from the IETF Bridge MIB WG to the IEEE 802.1 WG.

A.1. Assumptions for the Transfer

For the purposes of discussion in this appendix, assume that the IETF TICTOC WG has approved a standard YANG module for a published IEEE 1588 standard. As of this writing, this is IEEE Std 1588-2008, but it is possible that YANG for subsequent 1588 revisions could be published from the IETF TICTOC WG. For discussion in this appendix, we use the phrase "last IETF 1588 YANG" to refer to most recently published 1588 YANG from the IETF TICTOC WG.

The IEEE-SA Standards Board New Standards Committee (NesCom) handles new Project Authorization Requests (PARs) (see http://standards.ieee.org/board/nes/). PARs are roughly the equivalent of IETF Working Group Charters and include information concerning the scope, purpose, and justification for standardization projects.

Assume that IEEE 1588 has an approved PAR that explicitly specifies development of a YANG module. The transfer of YANG work will occur in the context of this IEEE 1588 PAR. For discussion in this appendix, we use the phrase "first IEEE 1588 YANG" to refer to the first IEEE 1588 standard for YANG.

Assume that as part of the transfer of YANG work, the IETF TICTOC WG agrees to cease all work on standard YANG modules for IEEE 1588.

Assume that the IEEE 1588 WG has participated in the development of the last IETF 1588 YANG module, such that the first IEEE 1588 YANG module will effectively be a revision of it. In other words, the transfer of YANG work will be relatively clean.

The actual conditions for the future transfer can be such that the preceding assumptions do not hold. Exceptions to the assumptions will need to be addressed on a case-by-case basis at the time of

the transfer. This appendix describes topics that can be addressed based on the preceding assumptions.

A.2. Intellectual Property Considerations

During review of the legal issues associated with transferring Bridge MIB WG documents to the IEEE 802.1 WG (Section 3.1 and Section 9 of [RFC4663]), it was concluded that the IETF does not have sufficient legal authority to make the transfer to IEEE without the consent of the document authors.

If the last IETF 1588 YANG is published as a RFC, the work is required to be transferred from the IETF to the IEEE, so that IEEE 1588 WG can begin working on the first IEEE 1588 YANG.

When work on the first IEEE YANG module begins in the IEEE 1588 WG, that work derives from the last IETF YANG module of this RFC, requiring a transfer of that work from the IETF to the IEEE. In order to avoid having the transfer of that work be dependent on the availability of this RFC's authors at the time of its publication, the IEEE Standards Association department of Risk Management and Licensing provided the appropriate forms and mechanisms for this document's authors to assign a non-exclusive license for IEEE to create derivative works from this document. Those IEEE forms and mechanisms will be updated as needed during the development of this document and any future IETF YANG modules for IEEE 1588. This will help to make the future transfer of work from IETF to IEEE occur as smoothly as possible.

As stated in the initial "Status of this Memo", the YANG module in this document conforms to the provisions of BCP 78. The IETF will retain all the rights granted at the time of publication in the published RFCs.

A.3. Namespace and Module Name

As specified in the "IANA Considerations" section, the YANG module in this document uses IETF as the root of its URN namespace and YANG module name.

Use of IETF as the root of these names implies that the YANG module is standardized in a Working Group of IETF, using the IETF processes. If the IEEE 1588 Working Group were to continue using these names rooted in IETF, the IEEE 1588 YANG standardization would need to continue in the IETF. The goal of transferring the

YANG work is to avoid this sort of dependency between standards organizations.

IEEE 802 has an active PAR (IEEE P802d) for creating a URN namespace for IEEE use (see http://standards.ieee.org/develop/project/802d.html). It is likely that this IEEE 802 PAR will be approved and published prior to the transfer of YANG work to the IEEE 1588 WG. If so, the IEEE 1588 WG can use the IEEE URN namespace for the first IEEE 1588 YANG module, such as:

urn:ieee:Std:1588:yang:ieee1588-ptp-dataset

where "ieee1588-ptp-dataset" is the registered YANG module name in the IEEE.

Under the assumptions of section A.1, the first IEEE 1588 YANG module prefix can be the same as the last IETF 1588 YANG module prefix (i.e. "ptp-dataset"), since the nodes within both YANG modules are compatible.

The result of these name changes are that for complete compatibility, a server (i.e. IEEE 1588 node) can choose to implement a YANG module for the last IETF 1588 YANG module (with IETF root) as well as the first IEEE 1588 YANG module (with IEEE root). Since the content of the YANG module transferred are the same, the server implementation is effectively common for both.

From a client's perspective, a client of the last IETF 1588 YANG module (or earlier) looks for the IETF-rooted module name; and a client of the first IEEE 1588 YANG module (or later) looks for the IEEE-rooted module name.

A.4. IEEE 1588 YANG Modules in ASCII Format

Although IEEE 1588 can certainly decide to publish YANG modules only in the PDF format that they use for their standard documents, without publishing an ASCII version, most network management systems cannot import the YANG module directly from the PDF. Thus, not publishing an ASCII version of the YANG module would negatively impact implementers and deployers of YANG modules and would make potential IETF reviews of YANG modules more difficult.

This appendix recommends that the IEEE 1588 WG consider future plans for:

- Public availability of the ASCII YANG modules during project development. These ASCII files allow IETF participants to access these documents for pre-standard review purposes.

- Public availability of the YANG portion of published IEEE 1588 standards, provided as an ASCII file for each YANG module. These ASCII files are intended for use of the published IEEE 1588 standard.

As an example of public availability during project development, IEEE 802 uses the same repository that IETF uses for YANG module development (see https://github.com/YangModels/yang). IEEE branches are provided for experimental work (i.e. pre-PAR) as well as standard work (post-PAR drafts). IEEE-SA has approved use of this repository for project development, but not for published standards.

As an example of public availability of YANG modules for published standards, IEEE 802.1 provides a public list of ASCII files for MIB (see http://www.ieee802.org/1/files/public/MIBs/ and http://www.ieee802.org/1/pages/MIBS.html), and analogous lists are planned for IEEE 802.1 YANG files.

Authors' Addresses

Yuanlong Jiang (Editor)
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
Email: jiangyuanlong@huawei.com

Xian Liu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
lene.liuxian@huawei.com

Jinchun Xu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
xujinchun@huawei.com

Rodney Cummings (Editor)
National Instruments
11500 N. Mopac Expwy
Bldg. C
Austin, TX 78759-3504
Email: Rodney.Cummings@ni.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 5, 2019

H. Stenn
Network Time Foundation

October 2, 2018

Network Time Protocol Extended Information Extension Field
draft-stenn-ntp-extended-information-03

Abstract

The network packet format used by NTP has changed very little between NTPv1, defined by RFC 958 [RFC0958] in 1985, and NTPv4, defined by RFC 5905 [RFC5905]. The core network packet used by NTP has no spare bits available for reporting additional state information and no larger data areas available for larger amounts of information. This proposal offers a new extension field that would contains this additional information.

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 https://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 April 5, 2019.

Copyright Notice

Table of Contents

1. Introduction

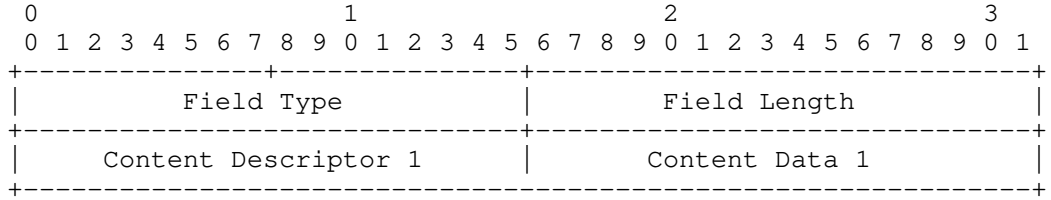
The core NTP packet format has changed little since RFC 958 [RFC0958] was published in 1985. Since then, there has been demonstrated need to convey additional information about NTP's state in an NTP packet but no backward-compatible way to usurp the few otherwise potentially available bits has been found, and no larger data areas are available in the core packet structure. This proposal offers a new extension field that would contain this additional information.

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. The Extended Information Extension Field

The Field Type of the Extended Information EF includes a version specification, to make it easier to evolve this specification.



NTP Extension Field: Extended Information

Field Type: TBD (Recommendation for IANA: 0x0009 (Extended-Information), 0x0109 (Extended-Information, Version 1)

Field Length: as needed

Payload: For Version 1, a two octet Content Descriptor field and a two octet Content Data field, as described below.

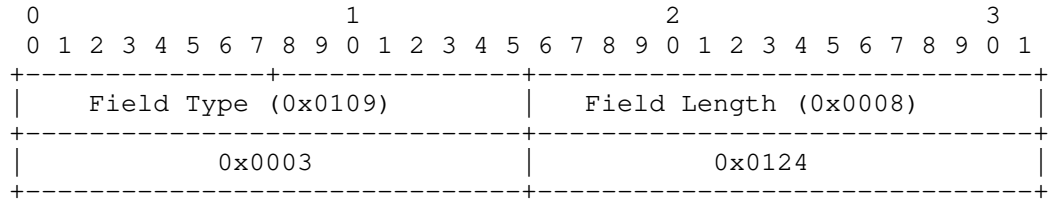
Version 1 Content fields.

Content Descriptor 1	Content Data 1
0x0001	TAI offset in the low-order 8 bits, 24-31
0x0002	Interleave Mode indicator in Bit 23
0xFFFD	Reserved (Zeroes)

Interleave Mode: 1 if the sender is in interleave mode, 0 otherwise

NTP Extension Field: Extended Information, Version 1 Content Fields

Example: A system that wants to convey an offset to TAI of 36 seconds, and show it is in interleave mode.



NTP Extension Field: Extended Information V1, Example

3. Acknowledgements

The author wishes to acknowledge the contributions of Martin Burnicki.

4. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types

0x0009 (Extended-Information)

0x0109 (Extended-Information, Version 1)

for this proposal.

5. Security Considerations

Additional information TBD

6. Normative References

[RFC0958] Mills, D., "Network Time Protocol (NTP)", RFC 958, DOI 10.17487/RFC0958, September 1985, <https://www.rfc-editor.org/info/rfc958>.

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

[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, <https://www.rfc-editor.org/info/rfc5905>.

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

Internet Engineering Task Force
Internet-Draft
Obsoletes: 7822 (if approved)
Intended status: Standards Track
Expires: April 5, 2019

H. Stenn
D. Mills
Network Time Foundation

October 2, 2018

Network Time Protocol Version 4 (NTPv4) Extension Fields
draft-stenn-ntp-extension-fields-08

Abstract

Network Time Protocol version 4 (NTPv4) defines the optional usage of extension fields. An extension field, as defined in RFC 5905 [RFC5905] and RFC 5906 [RFC5906], resides after the end of the NTP header and supplies optional capabilities or information that cannot be conveyed in the basic NTP packet. This document updates RFC 5905 [RFC5905] by clarifying some points regarding NTP extension fields and their usage with legacy Message Authentication Codes (MACs), and removes wasteful requirements added by RCF 7822 [RFC7822].

This proposal deprecates RFC 7822 [RFC7822].

Table of Contents

1. Introduction

An NTP packet consists of a set of fixed fields that may be followed by optional fields. Two types of optional fields are defined: extension fields (EFs) as defined in Section 7.5 of RFC 5905 [RFC5905], and legacy Message Authentication Codes (legacy MACs).

If a legacy MAC is used, it resides at the end of the packet. This field can be either a 4-octet crypto-NAK or data that has traditionally been 16, 20 or 24 octets long.

Additional information about the content of a MAC is specified in RFC 5906 [RFC5906], but since that RFC is Informational an implementor that was not planning to provide Autokey would likely never read that document. The result of this would be interoperability problems, at least. To address this problem this proposal also copies and clarifies some of the content of RFC 5906, putting it into RFC 5905. Because there is a reasonable expectation that RFC 5906 will be

deprecated, this document does not propose changes or updates to RFC 5906.

NTP extension fields are defined in RFC 5905 [RFC5905] as a generic mechanism that allows the addition of future extensions and features without modifying the NTP header format (Section 16 of RFC 5905 [RFC5905]).

With the knowledge and experience we have gained over time, it has become clear that simplifications, clarifications, and improvements can be made to the NTP specification around EFs and MACs.

This proposal adjusts and clarifies the requirements around EFs and MACs, allows EFs to be on 4-octet boundaries of any acceptable length, and provides methods to disambiguate packet parsing in the unexpected and unlikely case where an implementation would choose to send a packet that could be ambiguously parsed by the receiver.

This proposal deprecates RFC 7822 [RFC7822].

Implementations are still free to send EFs that are padded to longer lengths that otherwise follow the requirements below.

This document better specifies and clarifies extension fields as well as the requirements and parsing of a legacy MAC, with changes to address errors found after the publication of RFC 5905 [RFC5905] with respect to extension fields. Specifically, this document updates Section 7.5 of RFC 5905 [RFC5905], clarifying the relationship between extension fields and MACs, and expressly defines the behavior of a host that receives an unknown extension field.

2. Conventions Used in This Document

2.1. 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].

2.2. Terms and Abbreviations

EF - Extension Field

MAC - Message Authentication Code

NTPv4 - Network Time Protocol, Version 4 RFC 5905 [RFC5905]

3. NTP MAC - RFC 5906 Update

This document copies and updates some information in RFC 5906 [RFC5906] and puts it in to RFC 5905, as follows:

3.1. RFC5906 Section 4. - Autokey Cryptography

This section describes some of the cryptography aspects of Autokey. The third paragraph describes the use of 128- and 160-bit message digests. The enumeration of 128- and 160-bit message digests is not meant to be limiting - other message digest lengths MAY be implemented. This paragraph also describes some of the expected semantic ranges of the key ID. This information belongs in RFC 5905. The key ID value is particularly significant because it provides additional detection and disambiguation protection when deciding if the next data portion is either a legacy MAC or an extension field. [This is additional evidence that although RFC 5906 is Informational, parts of its content are REQUIRED for proper behavior of RFC 5905.]

3.2. RFC5906 Section 10. - Autokey Protocol Messages

This section describes the extension field format, including initial flag bits, a Code field, and 8-bit Field Type, and the 16-bit Length. This proposal expands and clarifies this information and puts it into RFC 5905.

This section says "The reference implementation discards any packet with a field length of more than 1024 characters." but this is no longer true.

3.3. RFC5906 Section 11.5. - Error Recovery

This section describes the crypto-NAK, which should be described in RFC 5905. A crypto-NAK is used by RFC 5905 as well. [This is additional evidence that even though RFC 5906 was Informational, some of its content is REQUIRED for proper behavior for RFC 5095.]

3.4. RFC5906 Section 13. - IANA Consideration

This section lists the Autokey-related Extension Field Types, including Flag Bits, Codes, and Field Types, which should be described in RFC 5905, or perhaps in some other document. [This is additional evidence that even though RFC 5906 is Informational, some of its content is REQUIRED for proper behavior for RFC 5905.]

4. NTP Extension Fields - RFC 5905 Update

This document updates Section 7.5 of RFC 5905 [RFC5905] as follows:

4.1. OLD: 'RFC5905 7.5 - NTP Extension Field Format'

In NTPv4, one or more extension fields can be inserted after the header and before the MAC, which is always present when an extension field is present. Other than defining the field format, this document makes no use of the field contents. An extension field contains a request or response message in the format shown in Figure 14.

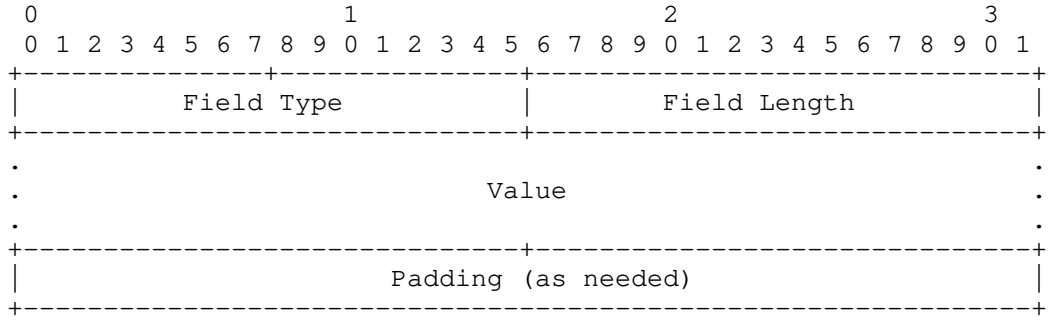


Figure 14: Extension Field Format

All extension fields are zero-padded to a word (four octets) boundary. The Field Type field is specific to the defined function and is not elaborated here. While the minimum field length containing required fields is four words (16 octets), a maximum field length remains to be established.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including the Padding field.

4.2. NEW: 'RFC5905 Section 7.5 - NTP Extension Field Format'

In NTPv4, one or more extension fields can be inserted after the header and before the possibly optional legacy MAC. A MAC SHOULD be present when an extension field is present. A MAC is always present in some form when NTP packets are authenticated. This MAC SHOULD be either a legacy MAC or a MAC-EF. It MAY be both. Other than defining the field format, this document makes no use of the field contents. An extension field contains a request or response message in the format shown in Figure 14.

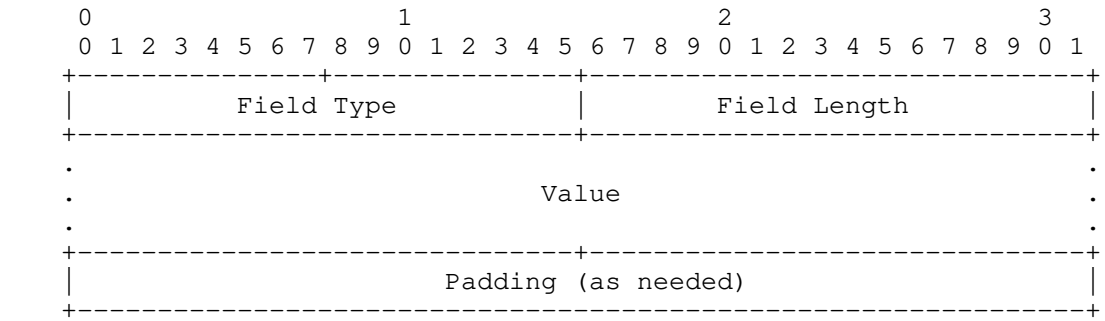


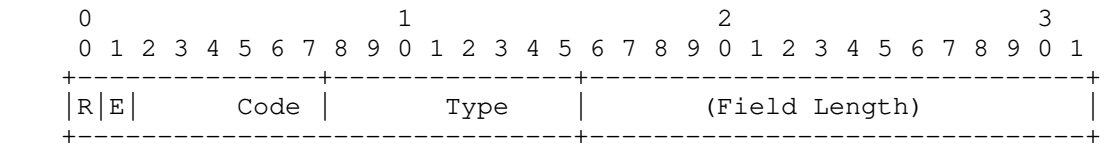
Figure 14: Extension Field Format

The four octets that comprise the Field Type and Field Length are called the Extension Field Header. Octets beyond the Extension Field Header are called the Extension Field Body, or the Extension Field Payload. The EF Body (EF Payload) MAY be null in some cases.

All extension fields are zero-padded to a word (four octet) boundary. The Field Type is specific to the defined functionality and detailed information about the Field Type is not elaborated here. The minimum size of an Extension Field is a 32-bit word (4 octets), and while the maximum extension field size MUST be 65532 octets or less, an NTP packet SHOULD NOT exceed the network MTU.

The Field Length is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including any Padding octets. The bottom two bits of the Field Length SHOULD be zero, and the size of the extension field SHOULD end on a 32-bit (4 octet) boundary. [RFC5905 Section 7.5 says "All extension fields are zero-padded to a word (four octets) boundary." but does not use 'MUST' language. Is it overkill to reiterate this requirement here? Should we use SHOULD or MUST regarding the bottom two bits or the boundary of the EF? It is possible, down the road, that we might find some use for those bottom 2 bits, even if we require a 32-bit boundary on the last octet of an EF.]

The Field Type contains the following sub-elements:



Extension Field Header Format

Where the following Field Type flags are defined:

R: 0 for "Information/Query", 1 for a "Response"

E: 0 for "OK", 1 for an "Error". Unused, and will be deprecated.

[The 'R' flag is currently used by Autokey, and by the proposed I-DO extension field. This flag is used after the packet is accepted.]

[The 'E' flag was proposed for use by Autokey, after the packet was accepted. As it was never used and no other use-cases have been identified, we are recommending this flag be deprecated at some point in the future.]

[The EF Code subtype is currently used by RFC 5906, Autokey [RFC5906], by the proposed Extended Information EF proposal, and is expected to be used by the NTS Extension Field, at least.]

The Autokey EF currently uses the most Code values - 10 of them, which equates to the least-significant 4 bits of the high-order octet. It is possible that additional flag bits will be allocated; in the past, the high-order 2 bits were reserved, and for a time two additional bits were proposed. Make no assumptions about the unused bits in this octet.

The EF Header and Body fields (the Flags, Code, Type, and Length, and any Value or Padding) are specific to the defined functionality and are not elaborated here; appropriate Field Type Flags, the EF Code, and EF Type values are defined in an IANA registry, and the Length, Value, and Padding values are defined by the document referred to by the registry. If a host receives an extension field with an unknown Field Type, the host SHOULD ignore the extension field and MAY drop the packet altogether, depending on local policy.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including any Padding.

While the minimum field length of an EF that contains no value or padding fields is one word (four octets), and the minimum field length of an EF that contains required fields is two words (8 octets), the maximum field length MUST NOT be longer than 65532 octets due to the maximum size of the data represented by the Length field, and SHOULD be small enough that the size of the NTP packet received by the client does not exceed the smallest MTU between the sender and the recipient. The bottom two bits of the Field Length SHOULD be zero and the EF data SHOULD be aligned to a 32-bit (4 octet) boundary.

4.3. NEW: 'RFC5905 Section 7.5.1 - Extension Fields and MACs'

With the inclusion of additional Extension Fields, there is now a potential that a poorly-designed implementation would produce an ambiguous parsing in the presence of a legacy MAC. What follows are two possibly independent ways to prevent this situation from ever happening.

Note well that to-date, there are only two defined Extension Field Types: Autokey, defined by RFC 5906 [RFC5906], and the Experimental UDP Checksum Complement in the Network Time Protocol, defined by RFC 7821 [RFC7821].

In spite of its known serious problems, Autokey is still in use by some and is a legacy case that is easily supported. Old systems will still work. An old system will still be able to open a properly-configured Autokey association to a new system, a new system will still be able to open a properly-configured Autokey association with an old system, and two new systems will be able to open a properly-configured Autokey association.

The UDP Checksum Complement extension field forbids the use of a legacy MAC, so any packet that uses it CANNOT be using a legacy MAC. [We could list the detailed and specific reasons why traffic using this EF is immune to EF/legacy MAC problems, but I fear that would just be confusing to most people.]

The first and best way to prevent ambiguous parsing is to use the I-DO extension field.

By definition any NTP client or server that handles any other Extension Fields is "new code" and can completely prevent ambiguity by the initiating side sending a packet containing an I-DO extension field followed by an optional MAC-EF followed by an optional legacy MAC. The inclusion of any MAC would be dictated by the authentication requirements of the association.

Note that NTP traffic works perfectly well without using any other extension fields. Newer extension fields offer additional capabilities, but these capabilities are not required for operation. [Even in the case of NTS or SNT, we're talking about "new code" that can be expected to be aware of issues with new extension fields an legacy MACs.]

If the initiating side sends an I-DO packet and gets no response, it operates as if the other side cannot handle new extension fields and simply continues the association without sending any new extension

fields. At any point in the future a packet can be sent with an I-DO extension field to see if the other side will respond.

An NTP implementation that receives a packet with an I-DO extension field may respond with a packet that may or may not contain an I-DO Response. If it does not respond, the other side SHOULD assume that the receiver does not understand new EFs. If it responds without sending an I-DO Response extension field, the sending side knows it should not send any new extension fields to this server. If the system that receives an I-DO extension field responds with an I-DO Response, it's telling the sender exactly what capabilities it is currently willing to exchange.

The second way to prevent ambiguous parsing is to use the LAST-EF extension field.

By definition, if I-DO is used and each side agrees to support LAST-EF then LAST-EF will prevent any ambiguity.

If, however, I-DO is not used then one side can simply send a packet with a LAST-EF. The LAST-EF extension field could be four-octet extension field, it could be a 28 octet extension field, or some other length that ends on a 32-bit boundary. If the other side responds appropriately then all is well. If the other side does not respond appropriately the sender should proceed without sending any new extension fields.

Parties interested in additional reasons for and approaches to understanding why there is no reason to be concerned about potential ambiguities with new code that would use new extension fields and legacy MACs can look at the the drafts that preceded this document.

4.4. OLD: 'RFC5905 Section 9.2. - Peer Process Operations'

...

FXMIT. ... This message includes the normal NTP header data shown in Figure 8, but with a MAC consisting of four octets of zeros. ...

4.5. NEW: 'RFC5905 Section 9.2. - Peer Process Operations'

...

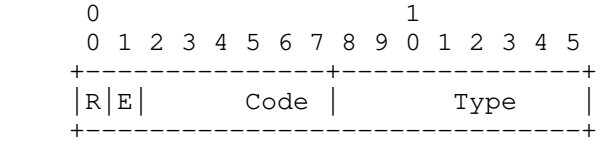
FXMIT. ... This message includes the normal NTP header data shown in Figure 8, but with a MAC consisting of four octets of zeros. This MAC can be a legacy MAC or a MAC-EF. If it's a MAC-EF, the crypto-NAK MUST be the only MAC in the MAC-EF payload. ...

5. Acknowledgements

The authors wish to acknowledge the contributions of Sam Weiler, Danny Mayer, and Tal Mizrahi.

6. IANA Considerations

This memo requests IANA to update the NTP Extension Field Types table in the NTP Parameters document as follows. The following is expected to be a functional superset of the existing information:



NTP Extension Field Type Format

Where the following Field Type flags are defined:

R: 0 for "Information/Query", 1 for a "Response"

E: 0 for "OK", 1 for an "Error". Unused, and will be deprecated.

Field Type	Meaning
0x0000	crypto-NAK (with Field Length of 0)
0x0000	RESERVED: Permanently Unassigned
0x0001	RESERVED: Unassigned
0x0002	Autokey: No-Operation Request
0x8002	Autokey: No-Operation Response
0x0102	Autokey: Association Message Request
0x8102	Autokey: Association Message Response
0x0202	Autokey: Certificate Message Request
0x8202	Autokey: Certificate Message Response
0x0302	Autokey: Cookie Message Request
0x8302	Autokey: Cookie Message Response
0x0402	Autokey: Autokey Message Request
0x8402	Autokey: Autokey Message Response
0x0502	Autokey: Leapseconds Value Message Request
0x8502	Autokey: Leapseconds Value Message Response
0x0602	Autokey: Sign Message Request
0x8602	Autokey: Sign Message Response
0x0702	Autokey: IFF Identity Message Request
0x8702	Autokey: IFF Identity Message Response
0x0802	Autokey: GQ Identity Message Request
0x8802	Autokey: GQ Identity Message Response
0x0902	Autokey: MV Identity Message Request
0x8902	Autokey: MV Identity Message Response
0x0005	Checksum Complement
0x2005	Checksum Complement (deprecated flag 0x2000)

Current Extension Fields

7. Security Considerations

Additional information TBD, as needed.

8. Normative References

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

[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, <https://www.rfc-editor.org/info/rfc5905>.

[RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <https://www.rfc-editor.org/info/rfc5906>.

[RFC7821] Mizrahi, T., "UDP Checksum Complement in the Network Time Protocol (NTP)", RFC 7821, DOI 10.17487/RFC7821, March 2016, <https://www.rfc-editor.org/info/rfc7821>.

[RFC7822] Mizrahi, T. and D. Mayer, "Network Time Protocol Version 4 (NTPv4) Extension Fields", RFC 7822, DOI 10.17487/RFC7822, March 2016, <https://www.rfc-editor.org/info/rfc7822>.

Authors' Addresses

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

David L. Mills
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: mills@udel.edu

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 5, 2019

H. Stenn
Network Time Foundation

October 2, 2018

Network Time Protocol I-Do Extension Field
draft-stenn-ntp-i-do-05

Abstract

The first implementation of NTPv4 was released in 2003. NTPv4 is defined by RFC 5905 [RFC5905]. It contains a public-key security protocol, Autokey, which is defined by RFC 5906 [RFC5906]. Until very recently, Autokey has been the only defined "user" of NTP packet Extension Fields. New proposals for extension fields are being written and there is currently no convenient way to learn if a remote instance of NTP supports any extension fields or not. This proposal contains a method to tell a remote instance of NTP what we (are willing to admit we) support, and ask what they (are willing to admit they) support.

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 https://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 April 5, 2019.

Table of Contents

1. Introduction

The first implementation of NTPv4 was released in 2003. NTPv4 is defined by RFC 5905 [RFC5905]. It contains a public-key security protocol, Autokey, which is defined by RFC 5906 [RFC5906]. Until very recently, Autokey has been the only defined "user" of NTP packet Extension Fields. New proposals for extension fields are being written and there is currently no convenient way to learn if a remote instance of NTP supports any extension fields or not. This proposal contains a method to tell a remote instance of NTP what we (are willing to admit we) support, and ask what they (are willing to admit they) support.

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. The I-Do Extension Field

The purpose of the I-DO EF is to provide information to the remote side about our capabilities.

If an incoming packet contains an unrecognized extension field, one of several things will happen. While that unrecognized extension field SHOULD be ignored, an implementation MAY choose to drop the entire packet. If any extension field is present there ordinarily SHOULD be a MAC following the extension field, but an older conforming NTP implementation would assume that any EF MUST be followed by a MAC. Some extension fields are unable to be "signed" by a MAC, regardless of whether or not that MAC is a traditional MAC

or an extension field MAC. In the final case, the receiving system will interpret the unrecognized EF as a legacy MAC, and return a crypto-NAK.

If the remote system replies with a crypto-NAK, that is a good indication that it is running older software that does not recognize EFs and thinks we have sent an invalid MAC. In this case, we should behave accordingly with regard to the remote system.

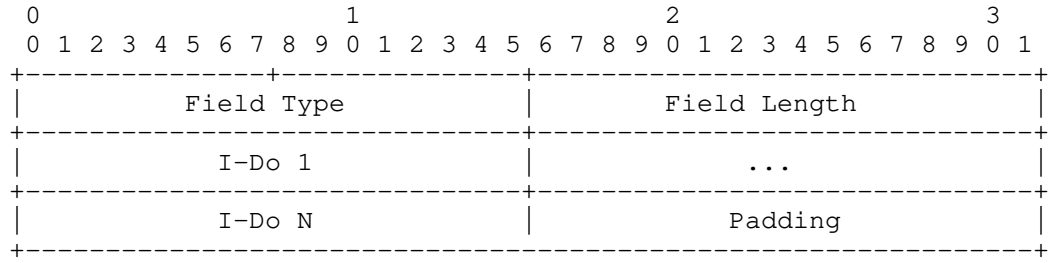
If the remote system replies without including an I-DO-RESPONSE EF, we at least know they can handle EFs, but they either don't understand I-DO or are not willing to tell us anything.

If the remote system replies with a packet that includes an I-DO-RESPONSE EF, then we SHOULD remember what they told us, and use that information appropriately.

In client/server mode, it makes sense for the client to send an I-DO to the server, and notice how the server responds. It likely does not make sense for the server to send an I-DO EF in response to a client request.

In symmetric mode, either side may initiate sending an I-DO EF, and the receiving side SHOULD reply with an I-DO-RESPONSE EF.

In broadcast mode, the broadcast server MAY send broadcast packets that include an I-DO EF, but note that if, counter to recommended practice, these packets are unauthenticated they MAY cause client machines to misinterpret the packet as having invalid authentication. In this situation, the broadcast server SHOULD alternate sending broadcast server packets with and without an I-DO EF, to insure that all clients receive time packets they will accept. Note that if, as recommended, broadcast packets are authenticated, a conforming client SHOULD have no difficulty in receiving a broadcast (mode 5) packet from a server that includes an I-DO EF.



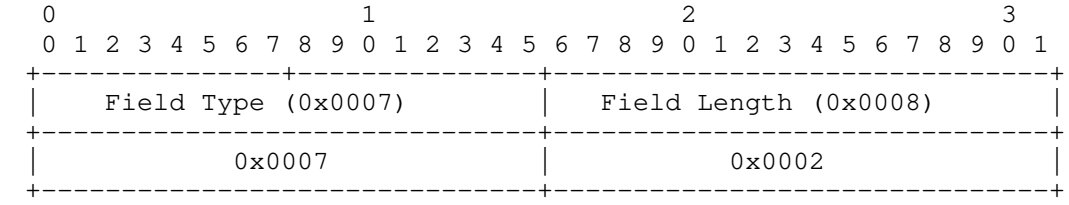
NTP Extension Field: REFID Suggestion

Field Type: TBD (Recommendation for IANA: 0x0007 (I-Do), 0x8007 (I-Do Response))

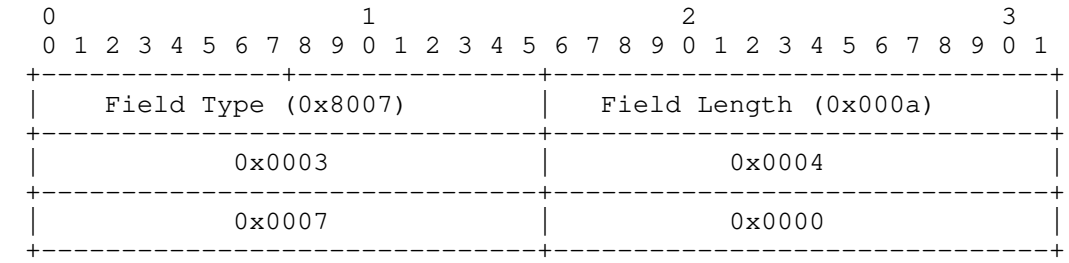
Field Length: as needed

Payload: An enumeration of the supported base Field Types, followed by any padding, 0x0000, needed to fill the payload to the desired 32-bit boundary.

Example: A system that wants to advertise support for Autokey and I-Do, sending to a system that responds with support for I-Do, NTS, and MAC-As-Extension-Field



NTP Extension Field: I-Do



NTP Extension Field: I-Do Response

The sender of any I-Do extension field MUST send an extension field with a Field Type of 0x0007 (I-Do) and SHOULD include a payload with any 0x0000 padding values after enumerating the supported base Extension Field Types. If the responding system recognizes the I-Do extension field, its response MUST include an extension field with a Field Type of 0x8007 (I-Do Response), and SHOULD include a payload with any 0x0000 padding values after enumerating the supported base Extension Field Types.

Any system that receives an I-Do extension field as either an "offer" or a "response" SHOULD scan the entire payload looking for nonzero values that specify the capabilities of the remote association.

Any system that receives an I-Do "offer", 0x0007, SHOULD reply with an I-Do "response", 0x8007.

Any system that sends an I-Do "offer" or "response" may send as few or as many of its supported Field Types as it chooses. At any subsequent time, either side may re-negotiate the list of supported field types it is prepared to accept from the other system by sending a new I-Do extension field.

The most-recently received I-Do list replaces any previous I-Do list.

3. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types:

0x0007 (I-DO)

0x8007 (I-DO Response)

and I-DO types:

0xFFFE (I-DO Leap Smear REFIDs)

0xFFFF (I-DO IPv6 REFID hash)

for this proposal.

4. Security Considerations

Additional information TBD

5. Normative References

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

[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, <https://www.rfc-editor.org/info/rfc5905>.

[RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <https://www.rfc-editor.org/info/rfc5906>.

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: May 2, 2017

H. Stenn
D. Mayer
Network Time Foundation

October 29, 2016

Network Time Protocol MAC/Last Extension Fields
draft-stenn-ntp-mac-last-ef-00

Abstract

NTPv4 is defined by RFC 5905 [RFC5905], and it and earlier versions of the NTP Protocol have supported symmetric private key Message Authentication Code (MAC) authentication. MACs were first described in Appendix C of RFC 1305 [RFC1305] and are further described in RFC 5905 [RFC5905]. As the number of Extension Fields grows there is an increasing chance of a parsing ambiguity when deciding if the "next" set of data is an Extension Field or a legacy MAC. This proposal defines two new Extension Fields to avoid this ambiguity. One is used to signifiy that it is the last Extension Field in the packet. If present, any subsequent data MUST be considered to be a legacy MAC. The other allows one or more MACs to be encapsulated in an Extension Field. If all parties in an association support MAC-EF, the use of a legacy MAC may be avoided.

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 May 2, 2017.

Copyright Notice

Table of Contents

1. Introduction

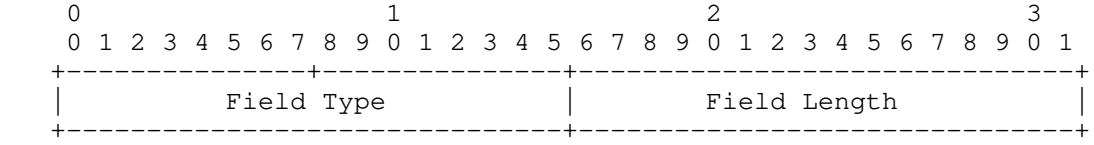
NTPv4 is defined by RFC 5905 [RFC5905], and it and earlier versions of the NTP Protocol have supported symmetric private key Message Authentication Code (MAC) authentication. MACs were first described in Appendix C of RFC 1305 [RFC1305] and are further described in RFC 5905 [RFC5905]. As the number of Extension Fields grows there is an increasing chance of a parsing ambiguity when deciding if the "next" set of data is an Extension Field or a legacy MAC. This proposal defines two new Extension Fields to avoid this ambiguity. One is used to signifiy that it is the last Extension Field in the packet. If present, any subsequent data MUST be considered to be a legacy MAC. The other allows one or more MACs to be encapsulated in an Extension Field. If all parties in an association support MAC-EF, the use of a legacy MAC may be avoided.

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. The Last Extension Field Extension Field

Now that multiple extension fields are a possibility, additional packet data could be either an Extension Field or a legacy MAC. Having a means to indicate that there are no more Extension Fields in an NTP packet and any subsequent data MUST be something else, almost certainly a legacy MAC, is a valuable facility.



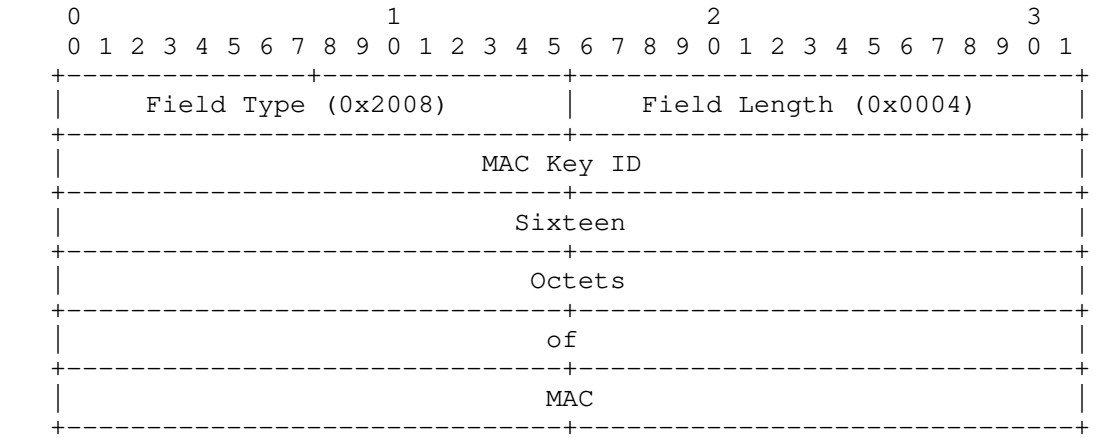
NTP Extension Field: Last Extension Field

Field Type: TBD (Recommendation for IANA: 0x2008 (Last Extension Field, MAC OPTIONAL))

Field Length: 4

Payload: None.

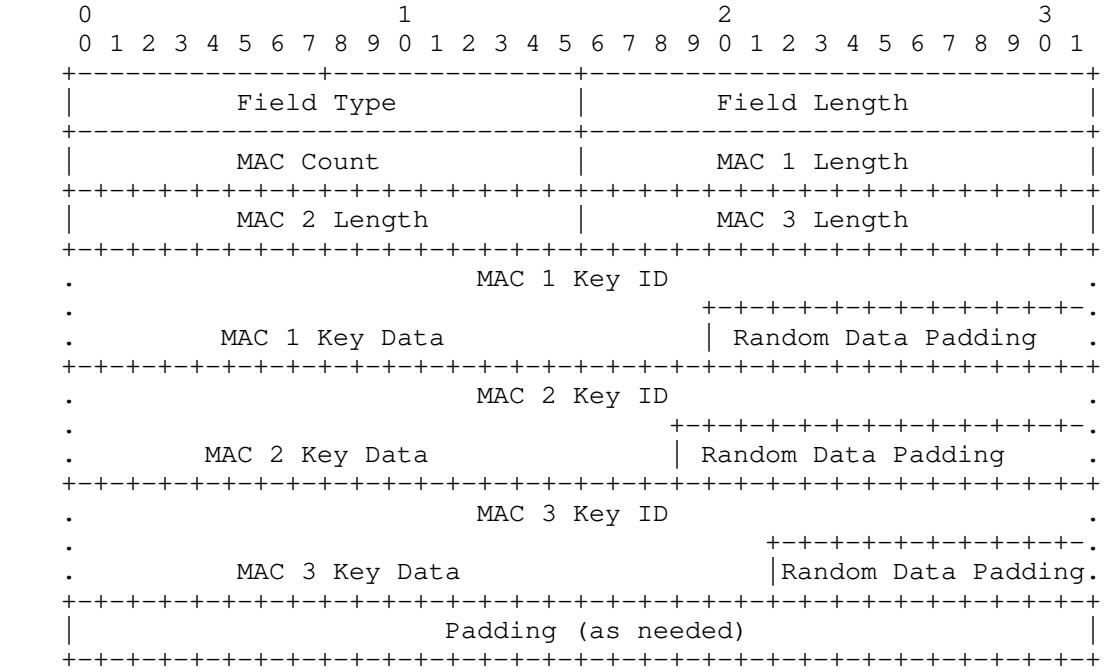
Example:



Example: NTP Extension Field: Last Extension Field, followed by a Legacy MAC

3. MAC Extension Field

Now that multiple extension fields are a possibility, there is a chance that additional packet data could be either an Extension Field or a legacy MAC. There is benefit to encapsulating the MAC in an extension field. By encapsulating the MAC in an EF, we also have the option to include multiple MACs in a packet, which may be of use in broadcast scenarios, for example.



NTP Extension Field: MAC EF Format

Field Type: TBD (Recommendation for IANA: 0x1003 (MAC-EF, MAC INCLUDED), 0x3003 (MAC-EF, MAC OPTIONAL, MAC INCLUDED))

Field Length: As needed

Payload: As described.

A Field Type of 0 and a Length of 0 means this extension field is a CRYPTO-NAK, as defined by RFC5905. Otherwise, a Field Type value of TBD (0x1003 is suggested) identifies this extension field as a MAC Extension field. The MAC Count is an unsigned 16-bit field, as is each MAC length field. If there are an even number of MACs specified there is an unused 16-bit field which SHOULD be 0x0000 at the end of

the set of MAC length values so that the subsequent MAC data is longword (4-octet) aligned. Each MAC SHALL be padded so that any subsequent MAC starts on a 4-octet boundary.

A MAC SHOULD not be present if there is a crypto-NAK present in the packet.

Each MAC within the extension field consists of a 32-bit key identifier which SHOULD be unique to the set of key identifiers in this MAC extension field followed by ((MAC Length) - 4) octets of data, optionally followed by random octets to pad the key data to the length specified earlier in the extension field. That key identifier is a shared secret which defines the algorithm to be used and a cookie or secret to be used in generating the digest. The MAC digest is produced by hashing the data from the beginning of the NTP packet up to but not including the start of the MAC extension field. The calculation of the digest SHOULD be a hash of this data concatenated with the 32-bit keyid (in network-order), and the key. When sending or receiving a key identifier each side needs to agree on the key identifier, algorithm and the cookie or secret used to produce the digest along with the digest lengths. Note that the sender may send more bytes than are required by the digest algorithm. This would be done to make it more difficult for a casual observer to identify the algorithm being used based on the length of the data. The digest data begins immediately after the key ID, and any padding octets SHOULD be random.

4. Acknowledgements

MAC-EF: The authors gratefully acknowledge Dave Mills for his insightful comments.

5. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types:

0x0000 CRYPTO-NAK

0x1003 MAC-EF, MAC INCLUDED

0x3003 MAC-EF, MAC OPTIONAL, MAC INCLUDED

0x0008 LAST-EF

0x2008 LAST-EF, MAC OPTIONAL

6. Security Considerations

The security considerations of time protocols in general are discussed in RFC7384 [RFC7384], and the security considerations of NTP are discussed in [RFC5905].

Digests MD5, DES and SHA-1 are considered compromised and should not be used [COMP].

If possible each MAC length should be at least 68 octets long to allow for 4 octets of key ID and at least 64 octets of digest and random padding. This means that for SHA-256 digests there are 4 octets of key ID, 32 bytes digest and 32 random octets of padding. Using larger minimum MAC lengths makes it difficult for an attacker to know which digest algorithms are used.

7. Normative References

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, DOI 10.17487/RFC1305, March 1992, <http://www.rfc-editor.org/info/rfc1305>.

[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>.

[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>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <http://www.rfc-editor.org/info/rfc7384>.

Authors' Addresses

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

Danny Mayer
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: mayer@ntp.org

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 5, 2019

H. Stenn
D. Mayer
Network Time Foundation

October 2, 2018

Network Time Protocol MAC/Last Extension Fields
draft-stenn-ntp-mac-last-ef-03

Abstract

NTPv4 is defined by RFC 5905 [RFC5905], and it and earlier versions of the NTP Protocol have supported symmetric private key Message Authentication Code (MAC) authentication. MACs were first described in Appendix C of RFC 1305 [RFC1305] and are further described in RFC 5905 [RFC5905]. As the number of Extension Fields grows there is an increasing chance of a parsing ambiguity when deciding if the "next" set of data is an Extension Field or a legacy MAC. This proposal defines two new Extension Fields to avoid this ambiguity. One, LAST-EF, is used to signify that it is the last Extension Field in the packet. If the LAST-EF is present, any subsequent data MUST be considered to be a legacy MAC. The other, MAC-EF, allows one or more MACs to be encapsulated in an Extension Field. If all parties in an association support MAC-EF, the use of a legacy MAC may be avoided.

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 https://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 April 5, 2019.

Copyright Notice

Table of Contents

1. Introduction

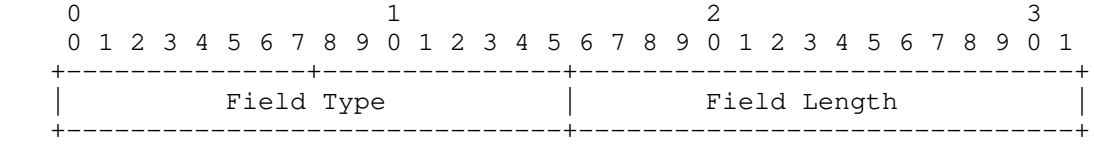
NTPv4 is defined by RFC 5905 [RFC5905], and it and earlier versions of the NTP Protocol have supported symmetric private key Message Authentication Code (MAC) authentication. MACs were first described in Appendix C of RFC 1305 [RFC1305] and are further described in RFC 5905 [RFC5905]. As the number of Extension Fields grows there is an increasing chance of a parsing ambiguity when deciding if the "next" set of data is an Extension Field or a legacy MAC. This proposal defines two new Extension Fields to avoid this ambiguity. One, LAST-EF, is used to signify that it is the last Extension Field in the packet. If the LAST-EF is present, any subsequent data MUST be considered to be a legacy MAC, or if you prefer, any subsequent data MUST NOT be considered to be an EF. The other, MAC-EF, allows one or more MACs to be encapsulated in an Extension Field. If all parties in an association support MAC-EF, the use of a legacy MAC may be avoided.

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. The Last Extension Field Extension Field - LAST-EF

Now that multiple extension fields are a possibility, additional packet data could be either an Extension Field or a legacy MAC. Having a means to indicate that there are no more Extension Fields in an NTP packet and any subsequent data MUST be something else, almost certainly a legacy MAC, is a valuable facility.



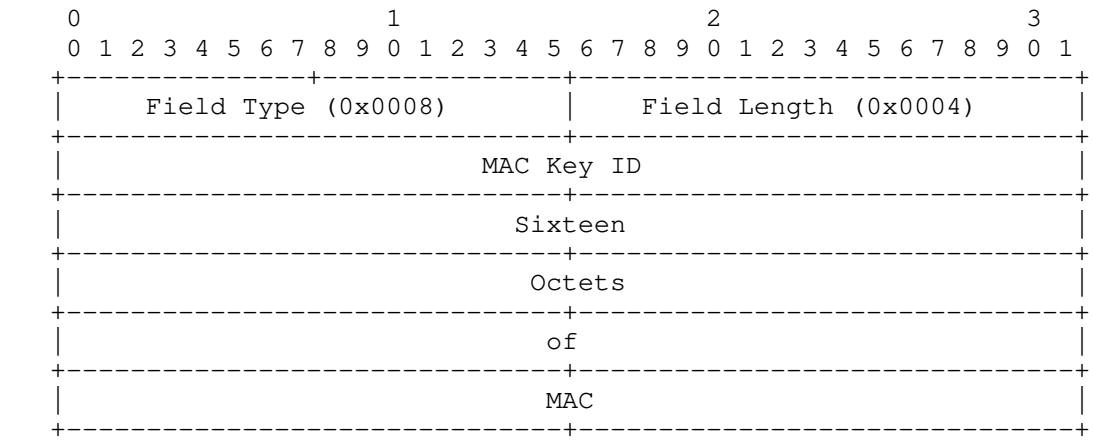
NTP Extension Field: Last Extension Field - LAST-EF

Field Type: TBD (Recommendation for IANA: 0x0008 (Last Extension Field))

Field Length: 4

Payload: None.

Example:

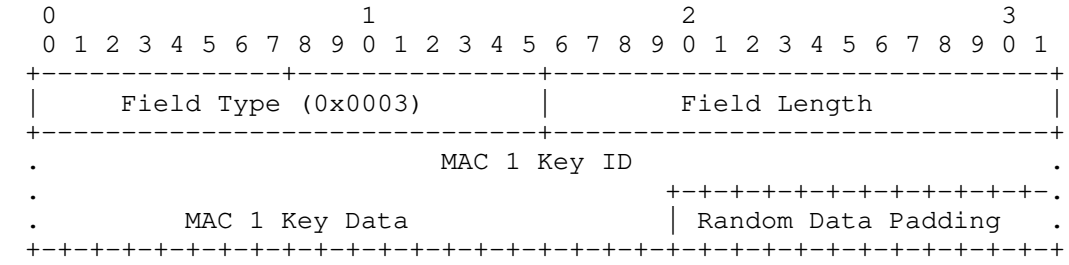


Example: NTP Extension Field: Last Extension Field, followed by a Legacy MAC

3. MAC Extension Field

Now that multiple extension fields are a possibility, there is a chance that additional packet data could be either an Extension Field or a legacy MAC. There is benefit to encapsulating the MAC in an extension field. By encapsulating the MAC in an EF, we also have the option to include multiple MACs in a packet, which may be of use in broadcast scenarios, for example.

There are two forms of this extension field. The first supports a single MAC, requiring 4 octets' overhead for the EF header. The second form supports one or more MACs in the EF payload, and requires at least 8 octets.

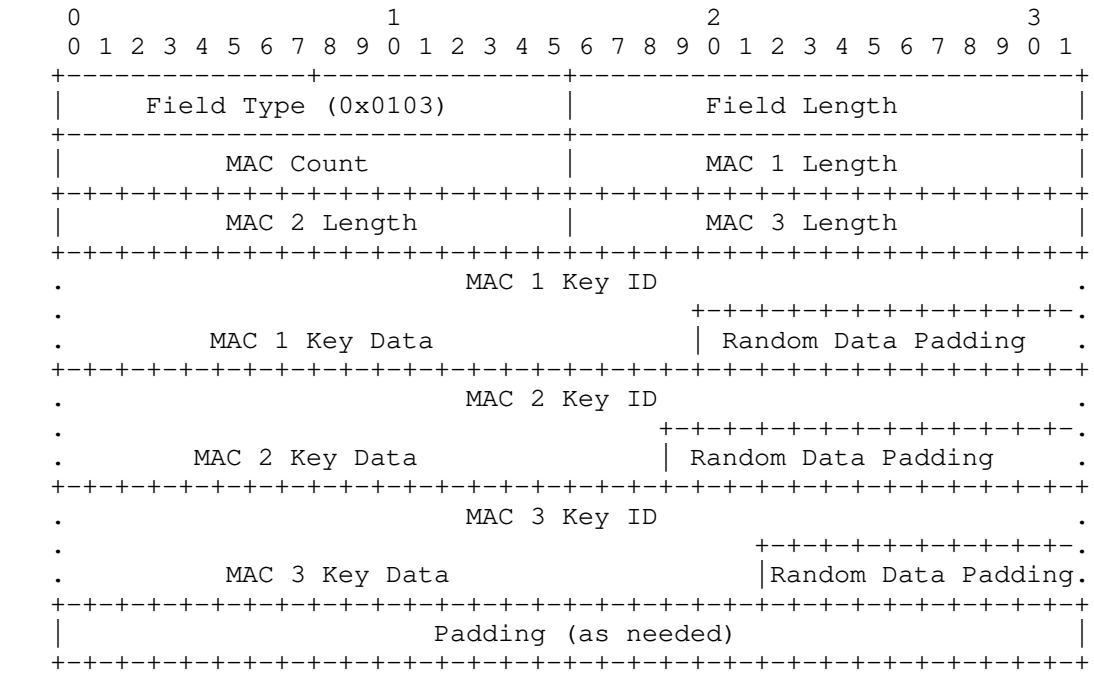


NTP Extension Field: MAC EF Format (Single MAC)

Field Type: TBD (Recommendation for IANA: 0x0003 (MAC-EF: Single MAC))

Field Length: As needed.

Payload: As described.



NTP Extension Field: MAC EF Format (1 or more MACs)

Field Type: TBD (Recommendation for IANA: 0x0103 (MAC-EF: 1 or more MACs))

Field Length: As needed.

Payload: As described.

A Field Type value of TBD (0x0003 is suggested) identifies this extension field as a MAC Extension field for a single MAC.

A Field Type value of TBD (0x0103 is suggested) identifies this extension field as a MAC extension field for one or more MACs. In this case, the MAC Count is an unsigned 16-bit field, as is each MAC length field. If there are an even number of MACs specified there is an unused 16-bit field which SHOULD be 0x0000 at the end of the set of MAC length values so that the subsequent MAC data is longword (4-octet) aligned. Each MAC SHALL be padded so that any subsequent MAC starts on a 4-octet boundary.

A MAC consisting of 4 octets of zeros means the MAC is a crypto-NAK, as defined by RFC5905 [RFC5905].

Additional MACs SHOULD NOT be present if there is a crypto-NAK present in the packet.

Each MAC within the extension field consists of a 32-bit key identifier which SHOULD be unique to the set of key identifiers in this MAC extension field followed by ((MAC Length) - 4) octets of data, optionally followed by random octets to pad the key data to the length specified earlier in the extension field. That key identifier is a shared secret which defines the algorithm to be used and a cookie or secret to be used in generating the digest. The MAC digest is produced by hashing the data from the beginning of the NTP packet up to but not including the start of the MAC extension field. The calculation of the digest SHOULD be a hash of this data concatenated with the 32-bit keyid (in network-order), and the key. When sending or receiving a key identifier each side needs to agree on the key identifier, algorithm and the cookie or secret used to produce the digest along with the digest lengths. Note that the sender may send more bytes than are required by the digest algorithm. This would be done to make it more difficult for a casual observer to identify the algorithm being used based on the length of the data. The digest data begins immediately after the key ID, and any padding octets SHOULD be random.

4. Acknowledgements

MAC-EF: The authors gratefully acknowledge Dave Mills for his insightful comments. Hal Murray asked if there was a way for the MAC-EF to require only 4 octets of overhead if there was only a single MAC in the payload.

5. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types:

0x0003 MAC-EF (Single MAC)

0x0103 MAC-EF (1 or more MACs)

0x0008 LAST-EF

6. Security Considerations

The security considerations of time protocols in general are discussed in RFC7384 [RFC7384], and the security considerations of NTP are discussed in RFC5905 [RFC5905].

Digests MD5, DES and SHA-1 are considered compromised and should not be used [COMP].

[DISCUSS] Each MAC length should be at least 20 octets long to allow for 4 octets of key ID and at least 16 octets of digest and random padding. For a 128-bit digest, there would be 4 octets of key ID, 16 octets of digest, plus any desired octets of random padding. For SHA-256 digests there are 4 octets of key ID, 32 octets digest, plus any desired octets of random padding. Using MAC lengths that include random padding may make it more difficult for an attacker to know which digest algorithms are used.

7. Normative References

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, DOI 10.17487/RFC1305, March 1992, <https://www.rfc-editor.org/info/rfc1305>.

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

[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, <https://www.rfc-editor.org/info/rfc5905>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

Authors' Addresses

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org

Danny Mayer
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: mayer@ntp.org

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 6, 2019

H. Stenn
Network Time Foundation

October 3, 2018

Network Time Protocol Suggest REFID Extension Field
draft-stenn-ntp-suggest-refid-04

Abstract

NTP has been widely used through several revisions, with the latest being RFC 5905 [RFC5905]. A core component of the protocol and the algorithms is the Reference ID, or REFID, which is used to identify the source of time used for synchronization. Traditionally, when the source of time was another system the REFID was the IPv4 address of that other system. The core purpose of the REFID is to prevent a one-degree timing loop, where if A has several timing sources that include B, if B decides to get its time from A we don't want A then deciding to get its time from B. The REFID is considered to be "public data" and is a vital core-component of the base NTP packet. If a system's REFID is the IPv4 address of its system peer, an attacker can try to use that information to send spoofed time packets to either or both the target or the target's server, attempting to cause a disruption in time service. This proposal is a backward-compatible way for a time source to tell its peers or clients "If you use me as your system peer, use this nonce as your REFID." This nonce SHOULD be untraceable to the original system, and if it is used as the REFID this type of attack is prevented.

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 https://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 April 6, 2019.

Table of Contents

1. Introduction

NTP has been widely used through several revisions, with the latest being RFC 5905 [RFC5905]. A core component of the protocol and the algorithms is the Reference ID, or REFID, which is used to identify the source of time used for synchronization. Traditionally, when the source of time was another system, the REFID was the IPv4 address of that other system. If the remote system was using IPv6 for its connection, a 4 octet digest value of the IPv6 address was used. The purpose of the REFID is to prevent a one-degree timing loop, where if A has several timing sources that include B, if B decides to get its time from A we don't want A then deciding to get its time from B. The REFID is considered to be "public data" and is a vital core-component of the base NTP packet. If a system's REFID is the IPv4 address of its system peer, an attacker can try to use that information to send spoofed time packets to either or both the target or the target's server, attempting to cause a disruption in time service. This proposal is a backward-compatible way for a time source to tell its peers or clients "If you use me as your system

peer, use this nonce as your REFID." This nonce, a Suggested REFID, SHOULD be untraceable to the sending system. If the receiving system uses this Suggested REFID nonce instead of the IPv4 address as its REFID, this type of attack and information disclosure is prevented.

The NTP protocol was designed with a mechanism that allowed for a depth-1 loop detection to avoid a simple "time loop". Recently, this mechanism was discovered to be a potential vulnerability exploit. The best way to mitigate this vulnerability is to decouple the IPv4 address of the server from its REFID. But there is no current way for a potential time source to tell the other party any other alternative to use as the REFID. This proposal creates an extension field to accomplish this.

1.1. Requirements Language

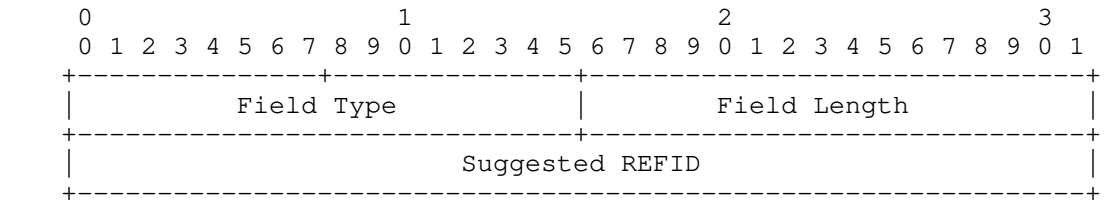
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. The REFID

The core reason for the REFID in the NTP Protocol is to prevent a timing loop of degree 1. Put another way, if servers A and B are exchanging time with each other and server B decides to follow A as its system peer, the REFID that B will use must be able to identify server A. The interpretation of a REFID is based on the stratum, as documented in RFC 5905 [RFC5905], section 7.3, "Packet Header Variables". At Stratum 2+, which will be the case if servers A and B are exchanging packets over IPv4, if server B follows A, then B will have A's IPv4 address as its REFID. When A asks B for its time, A will see that B is synchronized to A because B will tell A that its REFID is A's IPv4 address, so when A sees its IP address as B's REFID, A knows that if it were to follow B for its time then there would be a timing loop. In this case, A will not select B as a potential source of time.

3. The Suggested REFID Extension Field

Since there is no way in the base NTP packet for "this" instance of an NTP server to tell the "other" instance what REFID it should use if the "other" instance decides to use "this" instance as its system peer, the best available way to convey this information is via an extension field.



NTP Extension Field: REFID Suggestion

Field Type: TBD (Recommendation for IANA: 0x0006 (Suggested REFID))

Field Length: 0x0008

Suggested REFID: The 4 octets of the suggested REFID. This value SHOULD be 0xFDxxxxxx, where the bottom 3 octets SHOULD be random values.

Examples: When decoded as an IPv4 address, suggested REFIDs would decode as 253.0.0.0 thru 253.255.255.255.

4. Generating and Sending the Suggested REFID Extension Field

A system that decides to send a Suggested REFID extension field SHOULD generate a new Suggested REFID for each new association. It MAY generate a new Suggested REFID for any association in any response. In addition to remembering the IP-based REFID, the sender MUST also remember its most-recent Suggested REFID.

Since the core NTPv4 and earlier protocols do not contain any way to tell the recipient what to use as a REFID and RFC 5905 [RFC5905] uses the IPv4 address of the sender as the REFID if the association is effected over an IPv4 connection, this means that an attacker can simply send an NTP client request to a server knowing that server's system peer will be returned as the REFID in the response packet. At this point, an attacker can, if that REFID is an IPv4 address, begin to launch attacks at the target forging the putative IP of the target's time source, or the attacker can start forging packets to the putative time server claiming to be from the target, in an attempt to cause the time server to limit or deny time service to the target.

Using a nonce for the REFID that is only recognized by the sending machine effectively prevents this type of attack.

If servers S1, S2, and S3 are all exchanging time with each other and are all using the Suggested REFID mechanism, there is a 3 in 16,777,216 (2^24) chance that two different servers in the same group

will happen to choose the same nonce, and that would produce a false-positive timing loop detection. If the Suggested REFID is never changed, this false-positive condition will occur for potentially a long time. This small risk can be reduced by periodically generating a new Suggested REFID.

5. Receiving a Suggested REFID Extension Field

An NTP server keeps track of the IP address it uses to talk to a client. If an NTP server chooses to send a Suggested REFID to an association, it MUST remember this value. When checking for a timing loop, the Suggested REFID must also be included in the list of tested REFID values.

6. Acknowledgements

The author wishes to acknowledge the contributions of Martin Burnicki and Sam Weiler.

7. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Type 0x0006 (Suggested REFID) for this proposal.

8. Security Considerations

Additional information TBD

9. Normative References

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

[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, <https://www.rfc-editor.org/info/rfc5905>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwtime.org