Network Time Protocols
Internet-Draft
Intended status: Informational
Expires: 5 September 2024

W. Ladd
Akamai Technologies
M. Dansarie
4 March 2024

Roughtime
draft-ietf-ntp-roughtime-09

Abstract

This document specifies Roughtime - a protocol that aims to achieve rough time synchronization even for clients without any idea of what time it is.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-ntp-roughtime/.

Source for this draft and an issue tracker can be found at https://github.com/wbl/roughtime-draft.

Status of This Memo

Copyright Notice

Table of Contents

```
   1.  Introduction  . . . . . . . . . . . . . . . . . . . . . . . .   3
   2.  Conventions and Definitions . . . . . . . . . . . . . . . . .   3
   3.  Protocol Overview . . . . . . . . . . . . . . . . . . . . . .   3
   4.  The Guarantee . . . . . . . . . . . . . . . . . . . . . . . .   4
   5.  Message Format  . . . . . . . . . . . . . . . . . . . . . . .   4
     5.1.  Data types  . . . . . . . . . . . . . . . . . . . . . . .   5
       5.1.1.  int32 . . . . . . . . . . . . . . . . . . . . . . . .   5
       5.1.2.  uint32  . . . . . . . . . . . . . . . . . . . . . . .   5
       5.1.3.  uint64  . . . . . . . . . . . . . . . . . . . . . . .   5
       5.1.4.  Tag . . . . . . . . . . . . . . . . . . . . . . . . .   6
       5.1.5.  Timestamp . . . . . . . . . . . . . . . . . . . . . .   6
     5.2.  Header  . . . . . . . . . . . . . . . . . . . . . . . . .   6
   6.  Protocol Details  . . . . . . . . . . . . . . . . . . . . . .   6
     6.1.  Requests  . . . . . . . . . . . . . . . . . . . . . . . .   7
       6.1.1.  VER . . . . . . . . . . . . . . . . . . . . . . . . .   8
       6.1.2.  NONC  . . . . . . . . . . . . . . . . . . . . . . . .   8
     6.2.  Responses . . . . . . . . . . . . . . . . . . . . . . . .   8
       6.2.1.  SIG . . . . . . . . . . . . . . . . . . . . . . . . .   8
       6.2.2.  VER . . . . . . . . . . . . . . . . . . . . . . . . .   8
       6.2.3.  NONC  . . . . . . . . . . . . . . . . . . . . . . . .   8
       6.2.4.  PATH  . . . . . . . . . . . . . . . . . . . . . . . .   8
       6.2.5.  SERP  . . . . . . . . . . . . . . . . . . . . . . . .   9
       6.2.6.  CERT  . . . . . . . . . . . . . . . . . . . . . . . .   9
       6.2.7.  INDX  . . . . . . . . . . . . . . . . . . . . . . . .   9
     6.3.  The Merkel Tree (#tree) . . . . . . . . . . . . . . . . .   9
       6.3.1.  Root Value Validity Check Algorithm . . . . . . . . .  10
     6.4.  Validity of Response  . . . . . . . . . . . . . . . . . .  10
   7.  Integration into NTP  . . . . . . . . . . . . . . . . . . . .  11
   8.  Grease  . . . . . . . . . . . . . . . . . . . . . . . . . . .  11
   9.  Roughtime Clients . . . . . . . . . . . . . . . . . . . . . .  11
     9.1.  Necessary configuration . . . . . . . . . . . . . . . . .  11
     9.2.  Measurement sequence  . . . . . . . . . . . . . . . . . .  11
     9.3.  Malfeasence reporting . . . . . . . . . . . . . . . . . .  12
   10. Security Considerations . . . . . . . . . . . . . . . . . . .  12
   11. IANA Considerations . . . . . . . . . . . . . . . . . . . . .  12
     11.1.  Service Name and Transport Protocol Port Number
            Registry . . . . . . . . . . . . . . . . . . . . . . . .  12
```

11.2. Roughtime Version Registry 13
11.3. Roughtime Tag Registry 14
12. Privacy Considerations 15
13. References 15
13.1. Normative References 15
13.2. Informative References 16
Acknowledgments 17
Authors' Addresses 17

1. Introduction

Time synchronization is essential to Internet security as many security protocols and other applications require synchronization [RFC738]. Unfortunately widely deployed protocols such as the Network Time Protocol (NTP) [RFC5905] lack essential security features, and even newer protocols like Network Time Security (NTS) [RFC8915] lack mechanisms to ensure that the servers behave correctly. Furthermore clients may lack even a basic idea of the time, creating bootstrapping problems. Roughtime uses a list of keys and servers to resolve this issue.

2. Conventions and Definitions

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. Protocol Overview

Roughtime is a protocol for rough time synchronization that enables clients to provide cryptographic proof of server malfeasance. It does so by having responses from servers include a signature over a value derived from a nonce in the client request. This provides cryptographic proof that the timestamp was issued after the server received the client's request. The derived value included in the server's response is the root of a Merkle tree which includes the hash of the client's nonce as the value of one of its leaf nodes. This enables the server to amortize the relatively costly signing operation over a number of client requests. Single server mode: At its most basic level, Roughtime is a one round protocol in which a completely fresh client requests the current time and the server sends a signed response. The response includes a timestamp and a radius used to indicate the server's certainty about the reported time. For example, a radius of 1,000,000 microseconds means the server is absolutely confident that the true time is within one second of the reported time. The server proves freshness of its response as follows. The client's request contains a nonce which the server incorporates into its signed response. The client can verify the server's signatures and - provided that the nonce has sufficient entropy - this proves that the signed response could only have been generated after the nonce.

4. The Guarantee

A Roughtime server guarantees that a response to a query sent at t1, received at t2, and with timestamp t3 has been created between the transmission of the query and its reception. If t3 is not within that interval, a server inconsistency may be detected and used to impeach the server. The propagation of such a guarantee and its use of type synchronization is discussed in Section 7. No delay attacker may affect this: they may only expand the interval between t1 and t2, or of course stop the measurement in the first place.

5. Message Format

Roughtime messages are maps consisting of one or more (tag, value) pairs. They start with a header, which contains the number of pairs, the tags, and value offsets. The header is followed by a message values section which contains the values associated with the tags in the header. Messages MUST be formatted according to Figure 1 as described in the following sections.

Messages MAY be recursive, i.e. the value of a tag can itself be a Roughtime message.

```
   0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                   Number of pairs (uint32)                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   .                                                               .
   .                     N-1 offsets (uint32)                      .
   .                                                               .
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   .                                                               .
   .                        N tags (uint32)                        .
   .                                                               .
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   .                                                               .
   .                            Values                             .
   .                                                               .
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: Roughtime Message

5.1. Data types

5.1.1. int32

An int32 is a 32 bit signed integer. It is serialized least significant byte first in sign-magnitude representation with the sign bit in the most significant bit. The negative zero value (0x80000000) MUST NOT be used and any message with it is syntactically invalid and MUST be ignored.

5.1.2. uint32

A uint32 is a 32 bit unsigned integer. It is serialized with the least significant byte first.

5.1.3. uint64

A uint64 is a 64 bit unsigned integer. It is serialized with the least significant byte first.

5.1.4. Tag

Tags are used to identify values in Roughtime messages. A tag is a uint32 but may also be listed in this document as a sequence of up to four ASCII characters [RFC20]. ASCII strings shorter than four characters can be unambiguously converted to tags by padding them with zero bytes. For example, the ASCII string "NONC" would correspond to the tag 0x434e4f4e and "PAD" would correspond to 0x00444150. Note that when encoded into a message the ASCII values will be in the natural bytewise order.

5.1.5. Timestamp

A timestamp is a uint64 count of seconds since the Unix epoch in UTC.

5.2. Header

All Roughtime messages start with a header. The first four bytes of the header is the uint32 number of tags N, and hence of (tag, value) pairs. The following 4*(N-1) bytes are offsets, each a uint32. The last 4*N bytes in the header are tags. Offsets refer to the positions of the values in the message values section. All offsets MUST be multiples of four and placed in increasing order. The first post-header byte is at offset 0. The offset array is considered to have a not explicitly encoded value of 0 as its zeroth entry. The value associated with the ith tag begins at offset[i] and ends at offset[i+1]-1, with the exception of the last value which ends at the end of the message. Values may have zero length. Tags MUST be listed in the same order as the offsets of their values and MUST also be sorted in ascending order by numeric value. A tag MUST NOT appear more than once in a header.

6. Protocol Details

As described in Section 3, clients initiate time synchronization by sending requests containing a nonce to servers who send signed time responses in return. Roughtime packets can be sent between clients and servers either as UDP datagrams or via TCP streams. Servers SHOULD support the UDP transport mode, while TCP transport is OPTIONAL. A Roughtime packet MUST be formatted according to Figure 2 and as described here. The first field is a uint64 with the value 0x4d49544847554f52 ("ROUGHTIM" in ASCII). The second field is a uint32 and contains the length of the third field. The third and last field contains a Roughtime message as specified in Section 5.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  0x4d49544847554f52 (uint64)                  |
   |                        ("ROUGHTIM")                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Message length (uint32)                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   .                                                               .
   .                      Roughtime message                        .
   .                                                               .
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Roughtime packet

Roughtime request and response packets MUST be transmitted in a single datagram when the UDP transport mode is used. Setting the packet’s don’t fragment bit [RFC791] is OPTIONAL in IPv4 networks. Multiple requests and responses can be exchanged over an established TCP connection. Clients MAY send multiple requests at once and servers MAY send responses out of order. The connection SHOULD be closed by the client when it has no more requests to send and has received all expected responses. Either side SHOULD close the connection in response to synchronization, format, implementation-defined timeouts, or other errors. All requests and responses MUST contain the VER tag. It contains a list of one or more uint32 version numbers. The version of Roughtime specified by this memo has version number 1. NOTE TO RFC EDITOR: remove this paragraph before publication. For testing drafts of this memo, a version number of 0x80000000 plus the draft number is used.

6.1. Requests

A request MUST contain the tags VER and NONC. Tags other than NONC and VER SHOULD be ignored by the server. A future version of this protocol may mandate additional tags in the message and asign them semantic meaning. The size of the request message SHOULD be at least 1024 bytes when the UDP transport mode is used. To attain this size the ZZZZ tag SHOULD be added to the message. Its value SHOULD be all zeros. Responding to requests shorter than 1024 bytes is OPTIONAL and servers MUST NOT send responses larger than the requests they are replying to.

6.1.1. VER

In a request, the VER tag contains a list of versions. The VER tag MUST include at least one Roughtime version supported by the client. The client MUST ensure that the version numbers and tags included in the request are not incompatible with each other or the packet contents.

6.1.2. NONC

The value of the NONC tag is a 32 byte nonce. It SHOULD be generated in a manner indistinguishable from random. BCP 106 contains specific guidelines regarding this [RFC4086].

6.2. Responses

A response MUST contain the tags SIG, VER, NONC, PATH, SREP, CERT, and INDX.

6.2.1. SIG

In general, a SIG tag value is a 64 byte Ed25519 signature [RFC8032] over a concatenation of a signature context ASCII string and the entire value of a tag. All context strings MUST include a terminating zero byte. The SIG tag in the root of a response MUST be a signature over the SREP value using the public key contained in CERT. The context string MUST be "RoughTime v1 response signature".

6.2.2. VER

In a response, the VER tag MUST contain a single version number. It SHOULD be one of the version numbers supplied by the client in its request. The server MUST ensure that the version number corresponds with the rest of the packet contents.

6.2.3. NONC

The NONC tag MUST contain the nonce of the message being responded to.

6.2.4. PATH

The PATH tag value MUST be a multiple of 32 bytes long and represent a path of 32 byte hash values in the Merkle tree used to generate the ROOT value as described in a later section In the case where a response is prepared for a single request and the Merkle tree contains only the root node, the size of PATH MUST be zero.

6.2.5. SERP

The SREP tag contains a time response. Its value MUST be a Roughtime message with the tags ROOT, MIDP, and RADI. The server MAY include any of the tags DUT1, DTAI, and LEAP in the contents of the SREP tag. The ROOT tag MUST contain a 32 byte value of a Merkle tree root as described in Section 6.3. The MIDP tag value MUST be timestamp of the moment of processing. The RADI tag value MUST be a uint32 representing the server's estimate of the accuracy of MIDP in seconds. Servers MUST ensure that the true time is within (MIDP-RADI, MIDP+RADI) at the time they transmit the response message.

6.2.6. CERT

The CERT tag contains a public-key certificate signed with the server's long-term key. Its value is a Roughtime message with the tags DELE and SIG, where SIG is a signature over the DELE value. The context string used to generate SIG MUST be "RoughTime v1 delegation signature--". The DELE tag contains a delegated public-key certificate used by the server to sign the SREP tag. Its value is a Roughtime message with the tags MINT, MAXT, and PUBK. The purpose of the DELE tag is to enable separation of a long-term public key from keys on devices exposed to the public Internet. The MINT tag is the minimum timestamp for which the key in PUBK is trusted to sign responses. MIDP MUST be more than or equal to MINT for a response to be considered valid. The MAXT tag is the maximum timestamp for which the key in PUBK is trusted to sign responses. MIDP MUST be less than or equal to MAXT for a response to be considered valid. The PUBK tag contains a temporary 32 byte Ed25519 public key which is used to sign the SREP tag.

6.2.7. INDX

The INDX tag value is a uint32 determining the position of NONC in the Merkle tree used to generate the ROOT value as described in later section TODO.

6.3. The Merkel Tree (#tree)

A Merkle tree is a binary tree where the value of each non-leaf node is a hash value derived from its two children. The root of the tree is thus dependent on all leaf nodes. In Roughtime, each leaf node in the Merkle tree represents the nonce in one request. Leaf nodes are indexed left to right, beginning with zero. The values of all nodes are calculated from the leaf nodes and up towards the root node using the first 32 bytes of the output of the SHA-512 hash algorithm [RFC6234]. For leaf nodes, the byte 0x00 is prepended to the nonce before applying the hash function. For all other nodes, the byte

0x01 is concatenated with first the left and then the right child node value before applying the hash function. The value of the Merkle tree's root node is included in the ROOT tag of the response. The index of a request's nonce node is included in the INDX tag of the response. The values of all sibling nodes in the path between a request's nonce node and the root node is stored in the PATH tag so that the client can reconstruct and validate the value in the ROOT tag using its nonce. These values are each 32 bytes and are stored one after the other with no additional padding or structure. The order in which they are stored is described in the next section.

6.3.1. Root Value Validity Check Algorithm

We describe how to compute the root hash of the Merkel tree from the values in the tags PATH, INDX, and NONC. Our algorithm maintains a current hash value. The bits of INDX are ordered from least to most significant in this algorithm. At initialization hash is set to H(0x00 || nonce). If no more entries remain in PATH the current hash is the hash of the Merkel tree. All remaining bits of INDX must be zero. Otherwise let node be the next 32 bytes in PATH. If the current bit in INDX is 0 then hash = H(0x01 || node || hash), else hash = H(0x01 || hash || node).

6.4. Validity of Response

A client MUST check the following properties when it receives a response. We assume the long-term server public key is known to the client through other means.

The signature in CERT was made with the long-term key of the server.

The DELE timestamps and the MIDP value are consistent.

The INDX and PATH values prove NONC was included in the Merkle tree with value ROOT using the algorithm in Section 6.3.1.

The signature of SREP in SIG validates with the public key in DELE.

A response that passes these checks is said to be valid. Validity of a response does not prove the time is correct, but merely that the server signed it, and thus promises that it began to compute the signature at a time in the interval (MIDP-RADI, MIDP+RADI).

7. Integration into NTP

We assume that there is a bound PHI on the frequency error in the clock on the machine. Given a measurement taken at a local time t, we know the true time is in (t-delta-sigma, t-delta+sigma). After d seconds have elapsed we know the true time is within (t-delta-sigma-d_PHI, t-delta+sigma+d_PHI). A simple and effective way to mix with NTP or PTP discipline of the clock is to trim the observed intervals in NTP to fit entirely within this window or reject measurements that fall to far outside. This assumes time has not been stepped. If the NTP process decides to step the time, it MUST use Roughtime to ensure the new truetime estimate that will be stepped to is consistent with the true time. Should this window become too large, another Roughtime measurement is called for. The definition of "too large" is implementation defined. Implementations MAY use other, more sophisticated means of adjusting the clock respecting Roughtime information. Other applications such as X.509 verification may wish to apply different rules.

8. Grease

Servers MAY send back a fraction of responses that are syntactically invalid or contain invalid signatures as well as incorrect times. Clients MUST properly reject such responses. Servers MUST NOT send back responses with incorrect times and valid signatures. Either signature MAY be invalid for this application.

9. Roughtime Clients

9.1. Necessary configuration

To carry out a roughtime measurement a client must be equiped with a list of servers, a minimum of three of which are operational, not run by the same parties. It must also have a means of reporting to the provider of such a list, such as an OS vendor or software vendor, a failure report as described below.

9.2. Measurement sequence

The client randomly permutes three servers from the list, and sequentially queries them. The first probe uses a NONC that is randomly generated. The second query uses H(resp||rand) where rand is a random 32 byte value and resp is the entire response to the first probe. The third query uses H(resp||rand) for a different 32 byte value. If the times reported are consistent with the causal ordering, and the delay is within a system provided parameter, the measurement succeeds. If they are not consistent, there has been malfeasance and the client SHOULD store a report for evaluation, alert the operator, and make another measurement.

9.3. Malfeasence reporting

A malfeasance report is a JSON object with keys "nonces" containing an array of the rand values as base64 encoded strings and "responses" containing the responses as base64 encoded strings. This report is cryptographic proof that at least one server generated an incorrect response. Malfeasence reports MAY be transported by any means to the relevant vendor or server operator for discussion. A malfeasance report is cryptographic proof that the responses arrived in that order, and can be used to demonstrate that a server sent the wrong time. The venues for sharing such reports and what to do about them are outside the scope of this document.

10. Security Considerations

Since the only supported signature scheme, Ed25519, is not quantum resistant, the Roughtime version described in this memo will not survive the advent of quantum computers. Maintaining a list of trusted servers and adjudicating violations of the rules by servers is not discussed in this document and is essential for security. Roughtime clients MUST regularly update their view of which servers are trustworthy in order to benefit from the detection of misbehavior. Validating timestamps made on different dates requires knowledge of leap seconds in order to calculate time intervals correctly. Servers carry out a significant amount of computation in response to clients, and thus may experience vulnerability to denial of service attacks. This protocol does not provide any confidentiality. Given the nature of timestamps such impact is minor. The compromise of a PUBK’s private key, even past MAXT, is a problem as the private key can be used to sign invalid times that are in the range MINT to MAXT, and thus violate the good behavior guarantee of the server. Servers MUST NOT send response packets larger than the request packets sent by clients, in order to prevent amplification attacks.

11. IANA Considerations

11.1. 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:

Service Name: Roughtime

Transport Protocol: tcp,udp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Roughtime time synchronization

Reference: [[this memo]]

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

11.2. Roughtime Version Registry

IANA is requested to create a new registry entitled "Roughtime Version Registry". Entries shall have the following fields:

Version ID (REQUIRED): a 32-bit unsigned integer

Version name (REQUIRED): A short text string naming the version being identified.

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

The policy for allocation of new entries SHOULD be: IETF Review.

The initial contents of this registry shall be as follows:

Version ID	Version name	Reference
0x0	Reserved	[[this memo]]
0x1	Roughtime version 1	[[this memo]]
0x2-0x7fffffff	Unassigned	
0x80000000-0xffffffff	Reserved for Private	[[this memo]]
	or Experimental use	

Table 1

11.3. Roughtime Tag Registry

IANA is requested to create a new registry entitled "Roughtime Tag Registry". Entries SHALL have the following fields:

Tag (REQUIRED): A 32-bit unsigned integer in hexadecimal format.

ASCII Representation (OPTIONAL): The ASCII representation of the tag in accordance with Section 5.1.4 of this memo, if applicable.

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

The policy for allocation of new entries in this registry SHOULD be: Specification Required.

The initial contents of this registry SHALL be as follows:

Tag	ASCII Representation	Reference
0x7a7a7a7a	ZZZZ	[[this memo]]
0x00474953	SIG	[[this memo]]
0x00524556	VER	[[this memo]]
0x434e4f4e	NONC	[[this memo]]
0x454c4544	DELE	[[this memo]]
0x48544150	PATH	[[this memo]]
0x49444152	RADI	[[this memo]]
0x4b425550	PUBK	[[this memo]]
0x5044494d	MIDP	[[this memo]]
0x50455253	SREP	[[this memo]]
0x544e494d	MINT	[[this memo]]
0x544f4f52	ROOT	[[this memo]]
0x54524543	CERT	[[this memo]]
0x5458414d	MAXT	[[this memo]]
0x58444e49	INDX	[[this memo]]

Table 2

12. Privacy Considerations

This protocol is designed to obscure all client identifiers. Servers necessarily have persistent long-term identities essential to enforcing correct behavior. Generating nonces in a nonrandom manner can cause leaks of private data or enable tracking of clients as they move between networks.

13. References

13.1. Normative References

[RFC20]   Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/rfc/rfc20>.

[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/rfc/rfc2119>.

[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/rfc/rfc4086>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/rfc/rfc6234>.

[RFC791]   Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[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/rfc/rfc8174>.

13.2. Informative References

[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/rfc/rfc5905>.

[RFC738]   Harrenstien, K., "Time server", RFC 738, DOI 10.17487/RFC0738, October 1977, <https://www.rfc-editor.org/rfc/rfc738>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <https://www.rfc-editor.org/rfc/rfc8915>.

Acknowledgments

Thomas Peterson corrected multiple nits. Peter Löthberg, Tal Mizrahi, Ragnar Sundblad, Kristof Teichel, and the other members of the NTP working group contributed comments and suggestions.

Authors' Addresses

Watson Ladd
Akamai Technologies
Email: watsonbladd@gmail.com

Marcus Dansarie
Email: marcus@dansarie.se