Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: November 28, 2019

A. Malhotra
Boston University
A. Langley
Google
W. Ladd
Cloudflare

May 27, 2019

Roughtime
draft-roughtime-aanchal-02

Abstract

This document specifies Roughtime - a protocol that aims to achieve rough time synchronization while detecting servers that provide inaccurate time and providing cryptographic proof of their malfeasance.

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 November 28, 2019.

Copyright Notice

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Motivation

Time synchronization is essential to Internet security as many security protocols and other applications require synchronization [RFC7384][MCBG]. 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) [I-D.ietf-ntp-using-nts-for-ntp] fail to ensure that the servers behave correctly. Authenticating time servers prevents network adversaries from modifying time packets, but an authenticated time server still has full control over the contents of time packet and may go rogue. The Roughtime protocol provides cryptographic proof of malfeasance, enabling clients to detect and prove to a third party server's attempts to influence the time a client computes.

Protocol	Authenticated Server	Server Malfeasance Evidence
NTP, Chronos	N	N
NTP-MD5	Y*	N
NTP-Autokey	Y**	N
NTS	Y	N
Roughtime	Y	Y

Security Properties of current protocols

Table 1

Y* For security issues with symmetric-key based NTP-MD5 authentication, please refer to Message Authentication Code for the Network Time Protocol draft [I-D.ietf-ntp-mac]

Y** For security issues with Autokey Public Key Authentication, refer to [Autokey]

More specifically,

If a server's timestamps do not fit into the time context of other servers' responses, then a Roughtime client can cryptographically prove this misbehaviour to third parties. This helps detect "bad" servers.

A Roughtime client can roughly detect (with no absolute guarantee) a delay attack [DelayAttacks] but can not cryptographically prove

this to a third party. However, the absence of proof of malfeasance SHOULD not be considered a proof of absence of malfeasance. So Roughtime SHOULD not be used as a witness that a server is overall "good".

Note that the delay attacks cannot be detected/stopped by any protocol. Delay attacks can not, however, undermine the security guarantees provided by Roughtime.

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. 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 with a certificate rooted in long term public/private key pair over a portion of the initial request, thus providing cryptographic proof that the timestamp was issued after previous responses and before future ones.

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 (the number of microseconds since the Unix epoch) and a radius (in microseconds) 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 request contains a random challenge. The server incorporates the challenge into its signed response so that its needed to verify the signature. This proves that the signed response could only have been generated after the challenge was issued if the challenge has sufficient entropy.

Chaining multiple servers: For subsequent requests, the client generates its nonce by hashing the reply from the first server with a random value. This proves that the nonce was created after the reply from the first server. It sends that to the second server and

receives a signature from it covering that nonce and the time from the second server.

Cryptographic proof of misbehavior: If the time from the second server is before the first, then the client has proof of misbehavior; the reply from the second server implicitly shows that it was created later because of the way that the client constructed the nonce. If the time from the second server is after, then the client can contact the first server again and get a signature that was provably created afterwards, but with an earlier timestamp.

With only two servers, the client can end up with proof that something is wrong, but no idea what the correct time is. But with half a dozen or more independent servers, the client will end up with chain of proof of any server's misbehavior, signed by several others, and (presumably) enough accurate replies to establish what the correct time is. Furthermore this proof may be validated by third parties ultimately leading to a revocation of trust in the misbehaving server.

4. The guarantee

A Roughtime response to a query sent at t_1, received at t_2, and with timestamp t_3 is guaranteed to have been created between the transmission of the query and its reception. If t_3 is not within that interval, a server inconsistency may be detected and used to impeach the server. The use of such a guarantee in synchronization is currently beyond the grasp of this document.

5. Message Format

A uint32 is a 32 bit unsigned integer. It is serialized in bytes with the least significant byte first. A uint64 is a 64 bit unsigned integer. It is also seralized with the least significant byte first. 8 byte timestamps are described in Section 7.

A Roughtime packet is a UDP packet whose contents are interpreted as a map from uint32s to strings of bytes. The byte strings must all have lengths a multiple of four. All uint32 are encoded with the least significant byte first. The keys of this map are called tags, and we speak of the value of a tag as the string of bytes it is mapped to.

A Roughtime packet is serialized as follows: First there is a header, The first four bytes in the header are the uint32 number of tags N, and hence of (tag, value) pairs. 4*(N-1) bytes are offsets, each offset a uint32. The last 4*N bytes are the tags.

Tags are in ascending order, and no tag can be repeated. Offsets are all a multiple of four and MUST be strictly increasing. The offset array is considered to have a not explicitly encoded value of 0 as its zeroeth entry.

Immediately following the header is a concatenation of all the strings. The first post-header byte is at offset 0, and the end of the final byte string is indicated by the end of the packet. The ith byte string ends at offset[i+1]-1, counting of course from 0, and begins at offset[i]. It is the value associated to the ith tag.

This encoding may be recursive: a value may be said to be in Roughtime format and thus have a header, etc. Tags may be listed as four ASCII characters [RFC0020]. In this case the tag when serialized will be those four ASCII characters. For example NONC would be the numeric value 0x434e4f4e. They may also be listed as fewer then four ASCII characters with hex escape codes at the end.

6. Protocol

6.1. Queries

A query is a Roughtime packet with the tag NONC. The contents of NONC are 64 bytes. The request packet MUST be a minimum of 1024 bytes. To attain this size the tag PAD\xff MAY be added at the end of the packet with a conent of all zeros. Other tags MUST be ignored by the server. Future versions may specify additional tags and their semantics, so clients MUST NOT add other tags.

6.2. Responses

A response contains the following tags: SREP, SIG\x00, CERT, INDX, PATH, SREP value is itself in Roughtime format that contains the folowing tags: ROOT, MIDP, RADI. SIG\x00 is an Ed25519 signature [RFC8032] over the SREP value using the public key contained in CERT as explained later.

CERT in Roughtime format and contains the following tags: DELE, SIG\x00. This SIG\x00 is an Ed25519 signature over DELE that can be verified using the long term public key of the server. DELE is itself in Roughtime format containing tags MINT, MAXT, PUBK.

6.2.1. SREP

- ROOT contains the root hash value of a Merkle tree using SHA512 as described when we reach the PATH and INDX blocks

- MIDP contains an 8 byte timestamp of the moment of processing

- RADI is a u32 contains the server's estimate of the accuracy of MIDP in microseconds. Servers MUST ensure the true time is within (MIDP-RADI, MIDP+RADI) at the time they compose the response packet.

6.2.2. DELE

MINT is the minimum 8 byte timestamp at which the key in PUBK is trusted to begin signing time. MIDP > MINT for validity.

MAXT is the maximum 8 byte timestamp at which PUBK may sign. MIDP < MAXT for validity.

PUBK is a temporary Ed25519 public key. The use of this field is to enable seperation of a root public key from keys on devices exposed to the public Internet.

6.2.3. INDX and PATH

INDX is a uint32 determining the position of NONC in a Merkle tree. PATH contains the values to be hashed with the running hash as one ascends the tree. PATH is a multiple of 64 bytes long. The following algorithm verifies inclusion in the Merkle tree:

One starts by computing the hash of the NONC value from the request, with \x00 preappended. Then one walks from the least significant bit of INDX to the most significant bit, and also walks towards the end of PATH.

If PATH ends then the remaining bits of the INDX MUST be all zero. This indicates the termination of the walk, and the current value MUST equal ROOT if the response is valid.

If the current bit is 0, one hashes \x01, the current hash, and the value from PATH.

If the current bit is 1 one hashes \x01, the value from PATH, and the current hash.

6.3. 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

The signature of SREP in SIG\x00 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 more specifically began to compute the signature at a time in between (MIDP-RADI, MIDP+RADI).

7. Time

An 8 byte timestamp contains a 4 byte Modified Julian Date (as in [MJD] followed by a 4 byte count of the number of microseconds since midnight on that day. This is not a unique representation: leap seconds are handled by changing the day number early or late, and hence having the number of microseconds increase even more. Unlike NTP this is not a representation that uses the full number of bits in the fraction part.

8. Cheater detection

A chain of responses is a series of responses where the SHA-512 hash of the preceding response H, is concatenated with a 64 byte blind X, and then SHA-512(H, X) is the NONC used in the subsequent response. These may be represented as an array of objects in JSON where each object may have keys "blind" and "packet". Packet has the base64 encoded bytes of the packet and blind is the blind used for the next nonce. The last packet needs no blind.

A pair of responses (r_1, r_2) is invalid if MIDP_1-RADI_1 > MIDP_2+RADI_2. A chain of longer length is invalid if for any i, j such that i < j, (r_i, r_j) is an invalid pair.

Invalidity of a chain is proof that causality has been violated if all servers were reporting correct time. An invalid chain where all individual responses are valid is cryptographic proof of malfeasance of at least one server: if all servers had the correct time in the chain, causality would imply that MIDP_1-RADI_1 < MIDP_2+RADI_2.

In conducting the comparison of timestamps one must know the length of a day and hence have historical leap second data for the days in question. However if violations are greater then a second the loss of leap second data doesn't impede their detection.

9. 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.

10. Roughtime Servers

The below list contains a list of servers with their public keys in Base64 format. These servers implement an older version of this specification.

roughtime.int08h.com:2002;
AW5uAoTSTDfG5NfY1bTh08GUnOqlRb+HVhbJ3ODJvsE=

roughtime.cloudflare.com:2002; gD63hSj3ScS+wuOeGrubXlq35N1c5Lby/
S+T7MNTjxo=

roughtime.sandbox.google.com:2002;
etPaaIxcBMY1oUeGpwvPMCJMwlRVNxv51KK/tktoJTQ=

11. Trust anchors and policies

A trust anchor is any distributor of a list of trusted servers. It is RECOMMENDED that trust anchors subscribe to a common public forum where evidence of malfeasance may be shared and discussed. Trust anchors SHOULD subscribe to a zero-tolerance policy: any generation of incorrect timestamps will result in removal. To enable this trust anchors SHOULD list a wide variety of servers so the removal of a server does not result in operational issues for clients. Clients SHOULD attempt to detect malfeasance and have a way to report it to trust anchors.

Because only a single roughtime server is required for successful synchronization, Roughtime does not have the incentive problems that have prevented effective enforcement of discipline on the web PKI. We expect that some clients will aggressively monitor server behavior.

12. Acknowledgements

Thomas Peterson corrected multiple nits. Marcus Dansarie, Kristof Teichel, Tal Mizrahi, and the other members of the NTP working group contributed comments and suggestions.

13. IANA Considerations

We request IANA assign a UDP port and create a new registry for Roughtime tags.

14. Security Considerations

This protocol will not survive the advent of quantum computers. Currently only one signature scheme is supported. Maintaining a list of trusted servers and adjudicating violations of the rules by servers are not discussed in this document and are essential for security. Arithmetic on the adjusted timescale is interesting with intervals, and this may impact the interpretation of the MAXT and MINT fields. 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, and 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.

Roughtime clients MUST update their view of which servers are trustworthy in order to benefit from the detection of misbehavior.

Packets sent by the client MUST be at least 1024 bytes in length in order to mitigate amplification attacks, and servers MUST ignore request packets that are smaller than this length.

15. Privacy Considerations

This protocol is designed to obscure all client identifiers. Servers necessarily have persistent long term identities essential to enforcing correct behavior.

16. References

16.1. Normative References

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

[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/info/rfc8032>.

16.2. Informative References

[Autokey] Rottger, S., "Analysis of the NTP Autokey Procedures", 2012, <https://zero-entropy.de/autokey_analysis.pdf>.

[DelayAttacks] Mizrahi, T., "A Game Theoretic Analysis of Delay Attacks Against Time Synchronization Protocols", 2012, <https://ieeexplore.ieee.org/document/6336612>.

[I-D.ietf-ntp-mac] Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", draft-ietf-ntp-mac-06 (work in progress), January 2019.

[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-19 (work in progress), April 2019.

[MCBG] Malhotra, A., Cohen, I., Brakke, E., and S. Goldberg, "Attacking the Network Time Protocol", 2015, <https://eprint.iacr.org/2015/1020>.

[MJD] Moyer, G., "The Origin of the Julian Day System", 1981.

[resolution] International Earth Rotation and Reference Systems Service, "Resolution B1", 2000.

[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>.

[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>.

[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>.

[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>.

Authors' Addresses

Aanchal Malhotra
Boston University
111 Cummington Mall
Boston 02215
USA

Email: aanchal4@bu.edu

Adam Langley
Google

Watson Ladd
Cloudflare
101 Townsend St
San Francisco
USA

Email: watson@cloudflare.com