NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2015

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

March 5, 2015

Network Time Security
draft-ietf-ntp-network-time-security-08.txt

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

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 6, 2015.

Table of Contents

1. Introduction 3
2. Terminology 4
2.1. Terms and Abbreviations 4
2.2. Common Terminology for PTP and NTP 4
3. Security Threats 4
4. Objectives 5
5. NTS Overview 5
6. Protocol Messages 6
6.1. Association Message Exchange 7
6.1.1. Goals of the Association Exchange 7
6.1.2. Message Type: "client_assoc" 7
6.1.3. Message Type: "server_assoc" 8
6.1.4. Procedure Overview of the Association Exchange . . . 8
6.2. Cookie Messages 9
6.2.1. Goals of the Cookie Exchange 10
6.2.2. Message Type: "client_cook" 10
6.2.3. Message Type: "server_cook" 10
6.2.4. Procedure Overview of the Cookie Exchange 11
6.3. Unicast Time Synchronisation Messages 12
6.3.1. Goals of the Unicast Time Synchronization Exchange . 12
6.3.2. Message Type: "time_request" 12
6.3.3. Message Type: "time_response" 13
6.3.4. Procedure Overview of the Unicast Time Synchronization Exchange 13
6.4. Broadcast Parameter Messages 14
6.4.1. Goals of the Broadcast Parameter Exchange 15
6.4.2. Message Type: "client_bpar" 15
6.4.3. Message Type: "server_bpar" 15
6.4.4. Procedure Overview of the Broadcast Parameter Exchange 16
6.5. Broadcast Time Synchronization Exchange 17

6.5.1. Goals of the Broadcast Time Synchronization Exchange 17
6.5.2. Message Type: "server_broad" 17
6.5.3. Procedure Overview of Broadcast Time Synchronization Exchange 18
6.6. Broadcast Keycheck 19
6.6.1. Goals of the Broadcast Keycheck Exchange 19
6.6.2. Message Type: "client_keycheck" 20
6.6.3. Message Type: "server_keycheck" 20
6.6.4. Procedure Overview of the Broadcast Keycheck Exchange 20
7. Server Seed Considerations 21
8. Hash Algorithms and MAC Generation 22
8.1. Hash Algorithms 22
8.2. MAC Calculation 22
9. IANA Considerations 22
10. Security Considerations 22
10.1. Privacy 22
10.2. Initial Verification of the Server Certificates 23
10.3. Revocation of Server Certificates 23
10.4. Mitigating Denial-of-Service for broadcast packets . . . 23
10.5. Delay Attack 24
10.6. Random Number Generation 25
11. Acknowledgements 25
12. References 25
12.1. Normative References 25
12.2. Informative References 26
Appendix A. (informative) TICTOC Security Requirements 27
Appendix B. (normative) Using TESLA for Broadcast-Type Messages 28
B.1. Server Preparation 28
B.2. Client Preparation 30
B.3. Sending Authenticated Broadcast Packets 31
B.4. Authentication of Received Packets 31
Appendix C. (informative) Dependencies 32
Authors' Addresses 35

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 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

HMAC Keyed-Hash 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 a 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 optionally enables the server to verify the client's authorization.
- 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.
- Integration with protocols: NTS can be used to secure different time synchronization protocols, specifically at least NTP and PTP. 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.

5. NTS Overview

NTS applies X.509 certificates to verify the authenticity of the time server and to exchange a symmetric key, the so-called cookie. 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.

After the cookie is exchanged, the client then uses it 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> (HMAC(server seed, H(certificate of client))),

with the server seed as the key, where H is a hash function, and where the function MSB_<b> cuts off the b most significant bits of the result of the HMAC function. The client's certificate contains the client's public key and enables the server to identify the client, if client authorization is desired. The server seed is a random value of bit length b that the server possesses, which has to remain secret. The cookie never changes as long as the server seed stays the same, but 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 the hash value of its certificate to each request (see Section 6.3).

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, starting with the penultimate. 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 B.

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 as well as CMS (Cryptographic Message Syntax, see [RFC5652]) conventions that can be used to get the cryptographic aspects right.

6.1. Association Message Exchange

In this message exchange, the participants negotiate the hash 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.

6.1.1. 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 of the negotiation result to the client,
- guarantees to the client that the negotiation result is based on the client's original, unaltered request.

6.1.2. Message Type: "client_assoc"

The protocol sequence starts with the client sending an association message, called client_assoc. This message contains

- the NTS message ID "client_assoc",
- a nonce,
- the version number of NTS that the client wants to use (this SHOULD be the highest version number that it supports),
- the hostname of the client,
- a selection of accepted hash algorithms, and
- a selection of accepted encryption algorithms.

6.1.3. 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 hash 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 hostname of the server,
- the server's choice of algorithm for encryption and for cryptographic hashing, 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.

6.1.4. 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 constructs and sends a reply in the form of a server_assoc message as described in Section 6.1.3. 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:

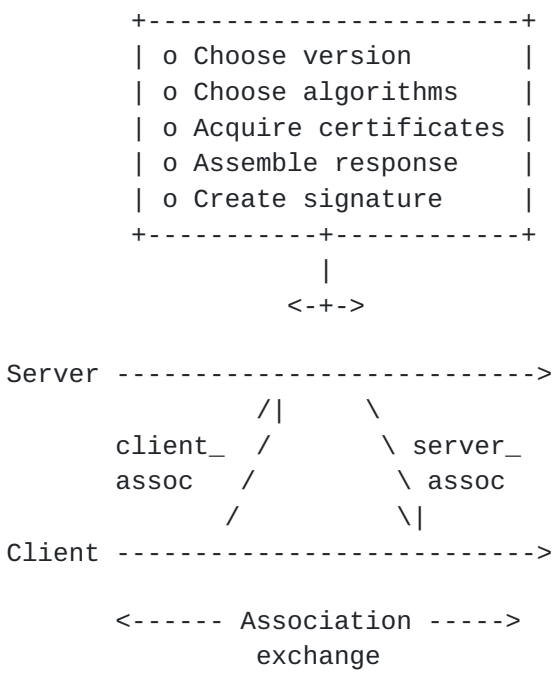
   * The client checks that the message contains a conforming version number.

   * 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 hash 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.

6.2. Cookie Messages

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.

6.2.1. 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 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.

6.2.2. 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 hash algorithm H,
- the client's certificate.

6.2.3. Message Type: "server_cook"

This message is sent by the server upon receipt of a client_cook message. The server generates the hash 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.

6.2.4. Procedure Overview of the Cookie Exchange

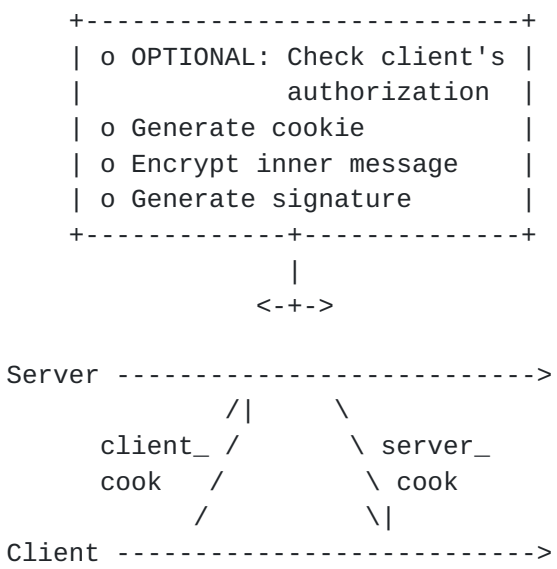
For a cookie exchange, the following steps are performed:

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 Section 6.2.3.

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 128 bit nonce and a 128 bit cookie.
   - 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.

6.3. 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 can be repeatedly exchanged as often as the client desires and as long as the integrity of the server's time responses is verified successfully.

6.3.1. Goals of the Unicast Time Synchronization Exchange

The unicast time synchronization exchange:

- exchanges (unicast) time synchronization data as specified by the appropriate time synchronization protocol,

- guarantees to the client that the response originates from the server and is based on the client's original, unaltered request.

6.3.2. 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 hash algorithm H,
- the hash of the client's certificate under H.

6.3.3. 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 hash of the client's certificate and the transmitted hash 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,
- a MAC (generated with the cookie as key) for verification of all of the above data.

6.3.4. Procedure Overview of the Unicast Time Synchronization Exchange

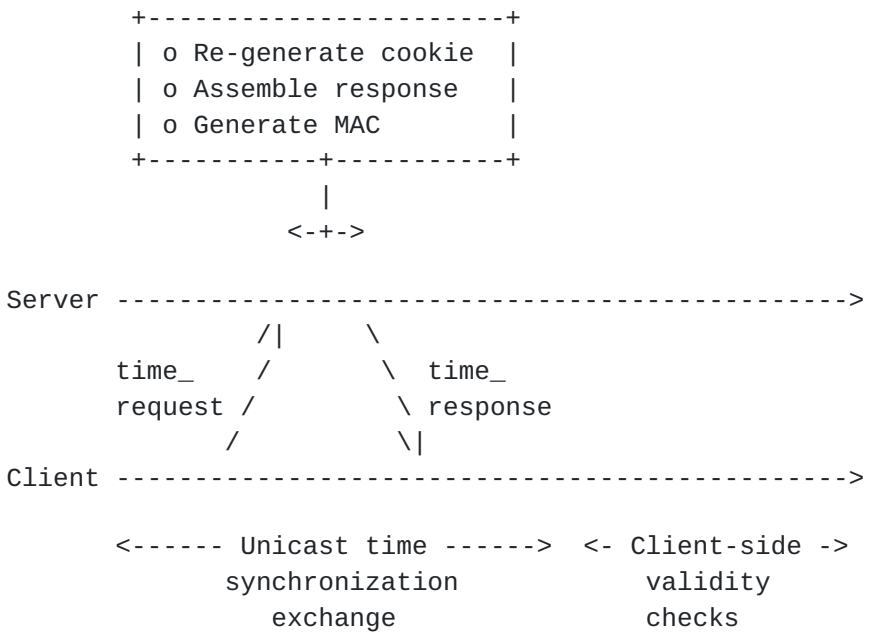
For a unicast time synchronization exchange, the following steps are performed:

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.
2. Upon receipt of a time_request message, the server re-calculates the cookie, then computes the necessary time synchronization data and constructs a time_response message as given in Section 6.3.3.
3. It 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 go back to step 5 (because the cookie might have changed due to a server seed refresh). If both checks are successful, the client SHOULD continue time synchronization by going back to step 7.



Procedure for unicast time synchronization exchange.

6.4. 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 B for more details on TESLA.

6.4.1. 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.

6.4.2. 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,
- the client's hostname, and
- the signature algorithm negotiated during association.

6.4.3. 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.

6.4.4. 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 Section 6.4.3.

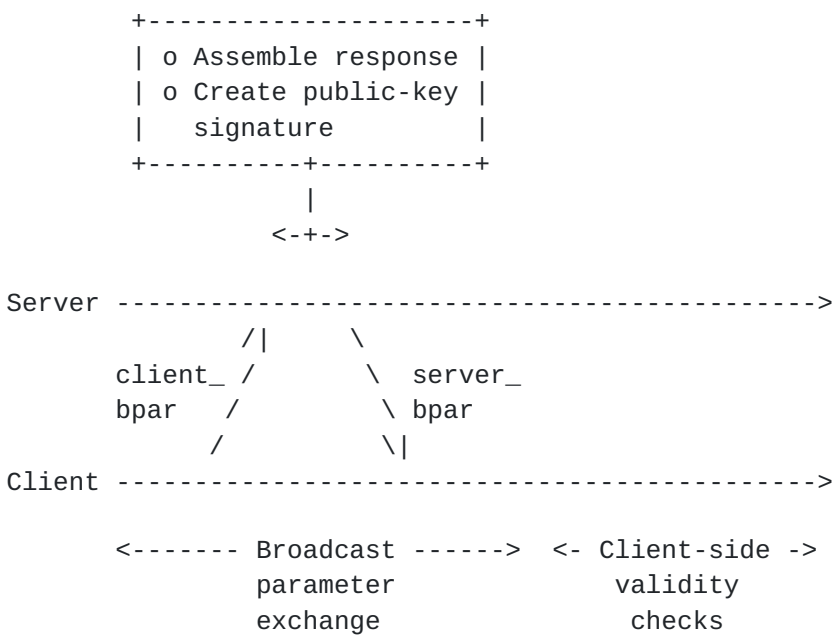
3. The client waits for a reply in the form of a server_bpar message, on which it performs the following checks:

   * The message must contain all the necessary information for the TESLA protocol, as listed in Section 6.4.3.

   * 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.

6.5. Broadcast Time Synchronization Exchange

Via a stream of messages of the following message type, the server keeps sending broadcast time synchronization messages to all participating clients.

6.5.1. 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.

6.5.2. 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),
- a MAC, calculated with the key for the current time interval, verifying
  - the message ID,
  - the version number, and
  - the time data.

6.5.3. Procedure Overview of Broadcast Time Synchronization Exchange

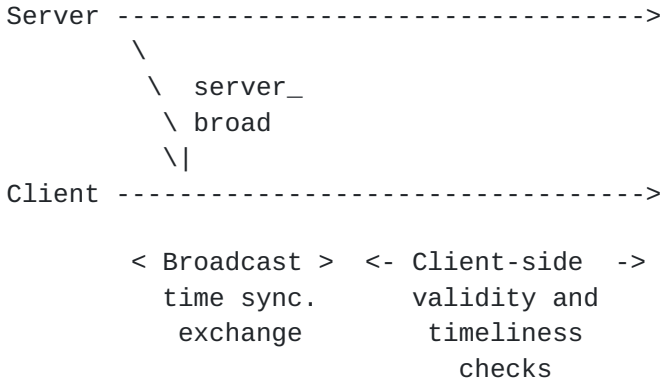
A broadcast time synchronization message exchange consists of the following steps:

1. The server follows the TESLA protocol by regularly sending server_broad messages as described in Section 6.5.2, 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.

6.6. Broadcast Keycheck

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

6.6.1. 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.

6.6.2. 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 hash algorithm H negotiated during association, and
- the hash of the client's certificate under H.

6.6.3. 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 hash of the client's certificate and the transmitted hash 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.

6.6.4. 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 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 6.6.3. For more details, see also Appendix B.

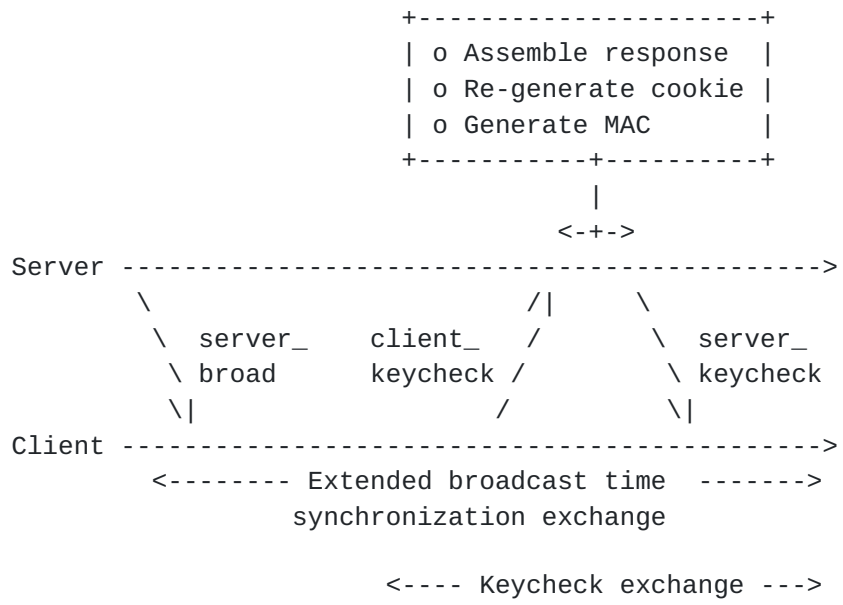
3. The client awaits a reply in the form of a server_keycheck message. On receipt, it performs the following checks:

   * 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 Considerations

The server has to calculate a random seed which has to be kept secret. The server MUST generate a seed for each supported hash algorithm, see Section 8.1.

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 (Section 6.2).

8. Hash Algorithms and MAC Generation

8.1. Hash Algorithms

Hash algorithms are used at different points: calculation of the cookie and the MAC, and hashing of the client's certificate. The client and the server negotiate a hash algorithm H during the association message exchange (Section 6.1) at the beginning. The selected algorithm H is used for all hashing processes in that run.

In the TESLA scheme, hash algorithms are used as pseudo-random functions to construct the one-way key chain. Here, the utilized hash algorithm is communicated by the server and is non-negotiable.

Note:

> Any hash algorithm is prone to be compromised in the future. A successful attack on a hash 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 hash algorithms. This way, knowledge gained from an attack on a hash algorithm H can at least only be used to compromise such clients who use hash algorithm H as well.

8.2. MAC Calculation

For the calculation of the MAC, client and server use a Keyed-Hash Message Authentication Code (HMAC) approach [RFC2104]. The HMAC is generated with the hash algorithm specified by the client (see Section 8.1).

9. IANA Considerations

10. Security Considerations

10.1. 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 [I-D.iab-privsec-confidentiality-threat]. 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 role since the packets 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.

10.2. Initial Verification of the Server Certificates

The client has to verify the validity of the certificates during the certification message exchange (Section 6.1.3). 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 as to rely on external means.

10.3. 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 (Section 6.1.3). 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 OCSP.

10.4. 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.

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.

10.5. 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.shpiner-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 B.4 for a detailed description of this check). Note that a broadcast client should also apply the above-mentioned precautions as far as possible.

10.6. 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.

11. Acknowledgements

The authors would like to thank Tal Mizrahi, Russ Housley, Steven Bellovin, David Mills and Kurt Roeckx for discussions and comments on the design of NTS. Also, thanks go to Harlan Stenn for his technical review and specific text contributions to this document.

12. References

12.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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, June 2005.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, October 2014.

12.2. Informative References

[I-D.iab-privsec-confidentiality-threat]
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", draft-iab-privsec-confidentiality-threat-03 (work in progress), February 2015.

[I-D.ietf-ntp-cms-for-nts-message]
Sibold, D., Roettger, S., Teichel, K., and R. Housley, "Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS)", draft-ietf-ntp-cms-for-nts-message-00 (work in progress), October 2014.

[I-D.shpiner-multi-path-synchronization]
Shpiner, A., Tse, R., Schelp, C., and T. Mizrahi, "Multi-Path Time Synchronization", draft-shpiner-multi-path-synchronization-03 (work in progress), February 2014.

[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, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[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 10.5.

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

Appendix B. (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].

B.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:

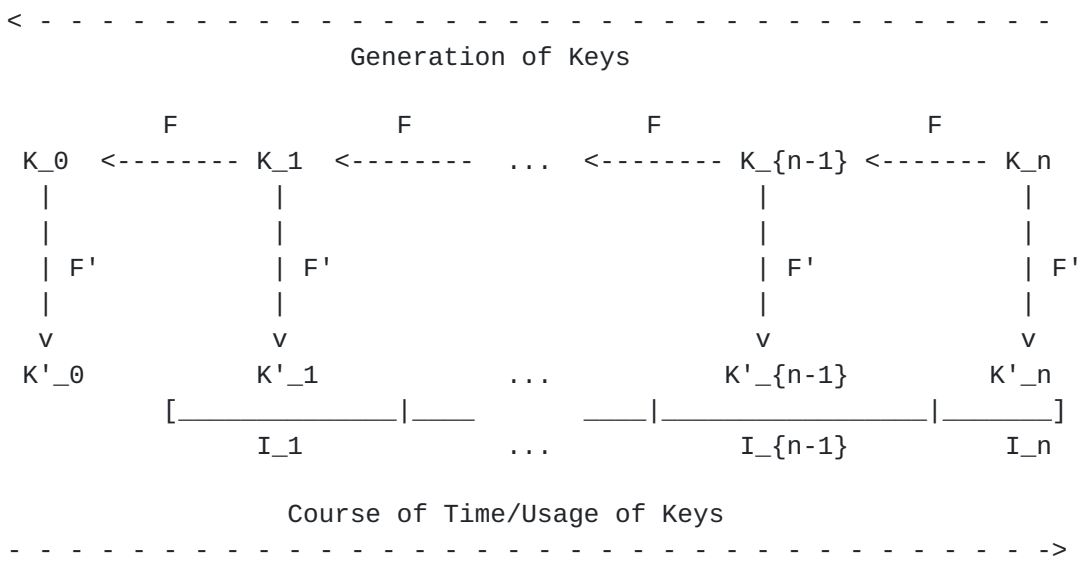
   * If d is chosen too short, the client might discard packets because it fails to verify that the key used for its MAC has not yet been disclosed.

   * 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.

   The server SHOULD 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

B.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.

B.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.

B.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

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

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

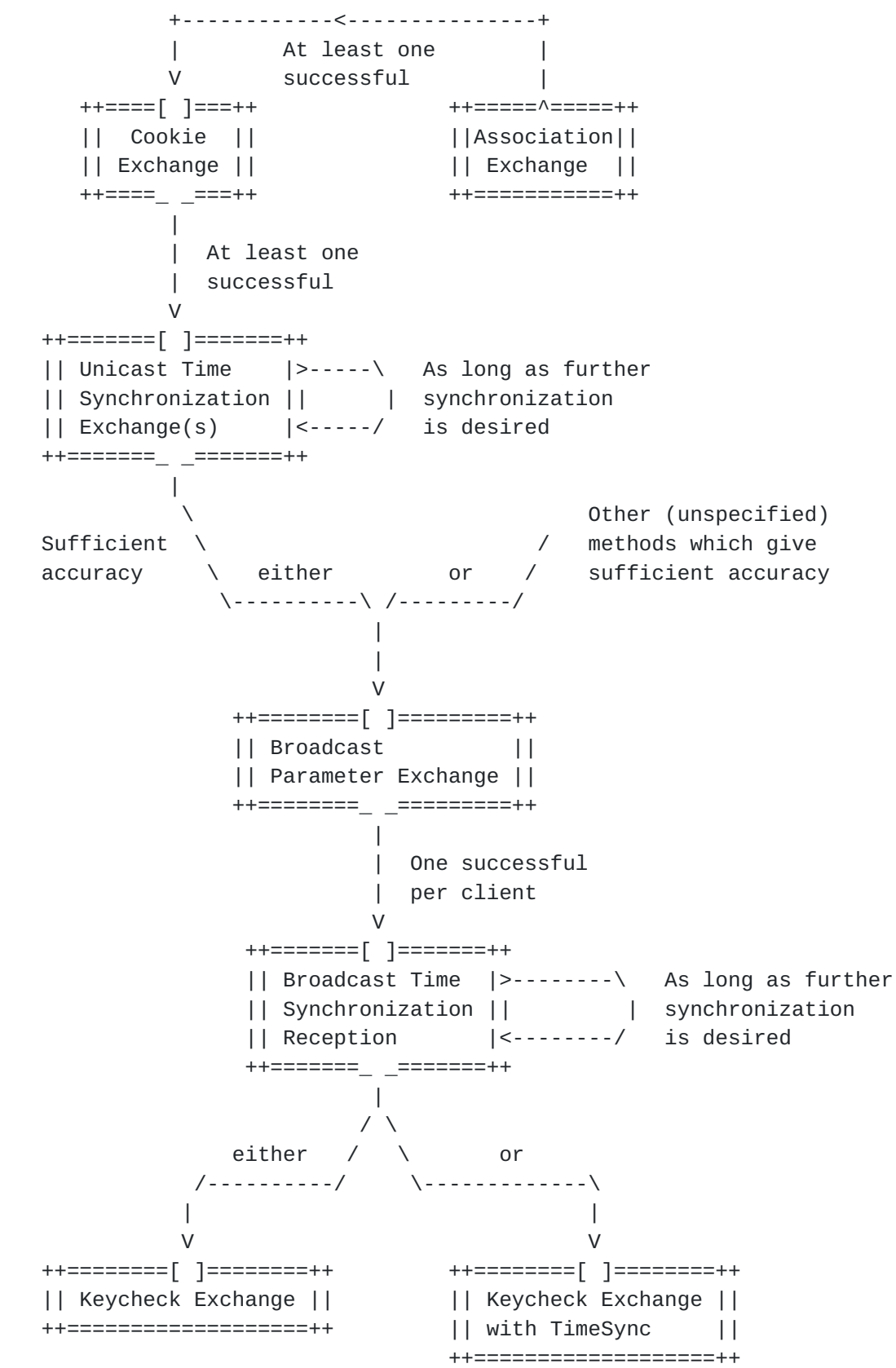
s_i < T_{i+1}.

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 C. (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	


At least one
successful
Cookie Exchange
Association Exchange
At least one
successful
Unicast Time Synchronization Exchange(s)
As long as further
synchronization
is desired
Sufficient accuracy
either
or
Other (unspecified)
methods which give
sufficient accuracy
Broadcast Parameter Exchange
One successful
per client
Broadcast Time Synchronization Reception
As long as further
synchronization
is desired
either
or
Keycheck Exchange
Keycheck Exchange
with TimeSync

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