

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

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

September 22, 2016

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

## Abstract

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

## Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

## Status of This Memo

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 March 26, 2017.

Internet-Draft

NTS

September 2016

## Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">3</a>
<a href="#">2.</a>	Terminology . . . . .	<a href="#">4</a>
<a href="#">2.1.</a>	Terms and Abbreviations . . . . .	<a href="#">4</a>
<a href="#">2.2.</a>	Common Terminology for PTP and NTP . . . . .	<a href="#">4</a>
<a href="#">3.</a>	Security Threats . . . . .	<a href="#">5</a>
<a href="#">4.</a>	Objectives . . . . .	<a href="#">5</a>
<a href="#">5.</a>	NTS Overview . . . . .	<a href="#">6</a>
<a href="#">6.</a>	Protocol Messages . . . . .	<a href="#">7</a>
<a href="#">6.1.</a>	Unicast Time Synchronisation Messages . . . . .	<a href="#">7</a>
6.1.1.	Preconditions for the Unicast Time Synchronization Exchange . . . . .	<a href="#">7</a>
6.1.2.	Goals of the Unicast Time Synchronization Exchange . . . . .	<a href="#">8</a>
<a href="#">6.1.3.</a>	Message Type: "time_request" . . . . .	<a href="#">8</a>
<a href="#">6.1.4.</a>	Message Type: "time_response" . . . . .	<a href="#">8</a>
6.1.5.	Procedure Overview of the Unicast Time Synchronization Exchange . . . . .	<a href="#">9</a>
<a href="#">6.2.</a>	Broadcast Time Synchronization Exchange . . . . .	<a href="#">10</a>
6.2.1.	Preconditions for the Broadcast Time Synchronization Exchange . . . . .	<a href="#">10</a>
6.2.2.	Goals of the Broadcast Time Synchronization Exchange . . . . .	<a href="#">11</a>
<a href="#">6.2.3.</a>	Message Type: "server_broad" . . . . .	<a href="#">11</a>
6.2.4.	Procedure Overview of Broadcast Time Synchronization Exchange . . . . .	<a href="#">12</a>
<a href="#">6.3.</a>	Broadcast Keycheck . . . . .	<a href="#">13</a>
<a href="#">6.3.1.</a>	Preconditions for the Broadcast Keycheck Exchange . . . . .	<a href="#">13</a>
<a href="#">6.3.2.</a>	Goals of the Broadcast Keycheck Exchange . . . . .	<a href="#">14</a>

6.3.3.	Message Type: "client_keycheck" . . . . .	14
6.3.4.	Message Type: "server_keycheck" . . . . .	14
6.3.5.	Procedure Overview of the Broadcast Keycheck Exchange	15
7.	Server Seed, MAC Algorithms and Generating MACs . . . . .	16
7.1.	Server Seed . . . . .	16

7.2.	MAC Algorithms . . . . .	16
8.	IANA Considerations . . . . .	17
9.	Security Considerations . . . . .	17
9.1.	Privacy . . . . .	17
9.2.	Initial Verification of the Server Certificates . . . . .	18
9.3.	Revocation of Server Certificates . . . . .	18
9.4.	Mitigating Denial-of-Service for broadcast packets . . . . .	18
9.5.	Delay Attack . . . . .	18
9.6.	Random Number Generation . . . . .	20
10.	Acknowledgements . . . . .	20
11.	References . . . . .	20
11.1.	Normative References . . . . .	20
11.2.	Informative References . . . . .	21
Appendix A.	(informative) TICTOC Security Requirements . . . . .	22
Appendix B.	(normative) Inherent Association Protocol Messages . . . . .	23
B.1.	Overview of NTS with Inherent Association Protocol . . . . .	23
B.2.	Access Message Exchange . . . . .	24
B.2.1.	Goals of the Access Message Exchange . . . . .	24
B.2.2.	Message Type: "client_access" . . . . .	24
B.2.3.	Message Type: "server_access" . . . . .	24
B.2.4.	Procedure Overview of the Access Exchange . . . . .	24
B.3.	Association Message Exchange . . . . .	25
B.3.1.	Goals of the Association Exchange . . . . .	25
B.3.2.	Message Type: "client_assoc" . . . . .	25
B.3.3.	Message Type: "server_assoc" . . . . .	26
B.3.4.	Procedure Overview of the Association Exchange . . . . .	26
B.4.	Cookie Message Exchange . . . . .	27
B.4.1.	Goals of the Cookie Exchange . . . . .	28
B.4.2.	Message Type: "client_cook" . . . . .	28
B.4.3.	Message Type: "server_cook" . . . . .	28
B.4.4.	Procedure Overview of the Cookie Exchange . . . . .	29
B.4.5.	Broadcast Parameter Messages . . . . .	30
Appendix C.	(normative) Using TESLA for Broadcast-Type Messages . . . . .	32
C.1.	Server Preparation . . . . .	32
C.2.	Client Preparation . . . . .	34
C.3.	Sending Authenticated Broadcast Packets . . . . .	35

<a href="#">C.4. Authentication of Received Packets</a>	<a href="#">35</a>
<a href="#">Appendix D. (informative) Dependencies</a>	<a href="#">37</a>
Authors' Addresses	<a href="#">39</a>

## [1.](#) Introduction

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

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

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

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

## [2.](#) Terminology

### [2.1.](#) Terms and Abbreviations

MITM Man In The Middle

NTS Network Time Security

TESLA Timed Efficient Stream Loss-tolerant Authentication

MAC Message Authentication Code

## [2.2.](#) Common Terminology for PTP and NTP

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

Sibold, et al.

Expires March 26, 2017

[Page 4]

---

Internet-Draft

NTS

September 2016

## [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:

- o **Authenticity:** NTS enables the client to authenticate its time server(s).
- o **Integrity:** NTS protects the integrity of time synchronization protocol packets via a message authentication code (MAC).
- o **Confidentiality:** NTS does not provide confidentiality protection of the time synchronization packets.
- o **Authorization:** NTS enables the client to verify its time server's authorization. NTS optionally enables the server to verify the client's authorization as well.

- o 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.
- o Applicability to Protocols: NTS can be used to secure different time synchronization protocols, specifically at least NTP and PTP.
- o Integration with Protocols: A client or server running an NTS-secured version of a time protocol does not negatively affect other participants who are running unsecured versions of that protocol.
- o Server-Side Statelessness: All security measures of NTS work without creating the necessity for a server to keep state of a connection.
- o Prevention of Amplification Attacks: All communication introduced by NTS offers protection against abuse for amplification denial-of-service attacks.

## [5.](#) NTS Overview

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

The cookie is calculated according to:

cookie = MSB\_<b> (MAC(server seed, KIV)),

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

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

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

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

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

## [6.](#) Protocol Messages

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

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

### [6.1.](#) Unicast Time Synchronisation Messages

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

#### [6.1.1.](#) Preconditions for the Unicast Time Synchronization Exchange

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

- o They MUST negotiate the algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.
- o The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.
- o Client and server MUST exchange the cookie (which depends on the KIV as described in section [Section 5](#)). Authenticity, confidentiality and integrity of the communication MUST be ensured.

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

#### [6.1.2.](#) Goals of the Unicast Time Synchronization Exchange



The unicast time synchronization exchange:

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

#### [6.1.3.](#) Message Type: "time\_request"

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

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

#### [6.1.4.](#) Message Type: "time\_response"

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

- o the NTS message ID "time\_response",
- o the version number as transmitted in time\_request,
- o the server's time synchronization response data,
- o the nonce transmitted in time\_request,

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

#### [6.1.5.](#) Procedure Overview of the Unicast Time Synchronization Exchange

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

1. The client sends a `time_request` message to the server. The client **MUST** save the included nonce and the `transmit_timestamp` (from the time synchronization data) as a correlated pair for later verification steps. Optionally, the client protects the request message with an appended MAC.
2. Upon receipt of a `time_request` message, the server performs the following steps:
  - \* It re-calculates the cookie.
  - \* If the request message contains a MAC the server re-calculates the MAC and compares this value with the MAC in the received data.
    - + If the re-calculated MAC does not match the MAC in the received data the server **MUST** stop the processing of the request.
    - + If the re-calculated MAC matches the MAC in the received data the server continues to process the request.
  - \* The server computes the necessary time synchronization data and constructs a `time_response` message as given in [Section 6.1.4](#).
3. The client awaits a reply in the form of a `time_response` message. Upon receipt, it checks:
  - \* that the transmitted version number matches the one negotiated previously,
  - \* that the transmitted nonce belongs to a previous `time_request` message,
  - \* that the `transmit_timestamp` in that `time_request` message matches the corresponding time stamp from the synchronization data received in the `time_response`, and



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

- o The client **MUST** receive all the information necessary to process broadcast time synchronization messages from the server. This includes
  - \* the one-way functions used for building the key chain,

- \* the last key of the key chain,
  - \* time interval duration,
  - \* the disclosure delay (number of intervals between use and disclosure of a key),
  - \* the time at which the next time interval will start, and
  - \* the next interval's associated index.
- o The communication of the data listed above **MUST** guarantee authenticity of the server, as well as integrity and freshness of the broadcast parameters to the client.

### [6.2.2.](#) Goals of the Broadcast Time Synchronization Exchange

The broadcast time synchronization exchange:

- o transmits (broadcast) time synchronization data from the server to the client as specified by the appropriate time synchronization protocol,
- o 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,
- o additionally guarantees authenticity of a certain broadcast synchronization message in the client's storage.

### [6.2.3.](#) Message Type: "server\_broad"

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

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

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

#### [6.2.4](#). Procedure Overview of Broadcast Time Synchronization Exchange

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

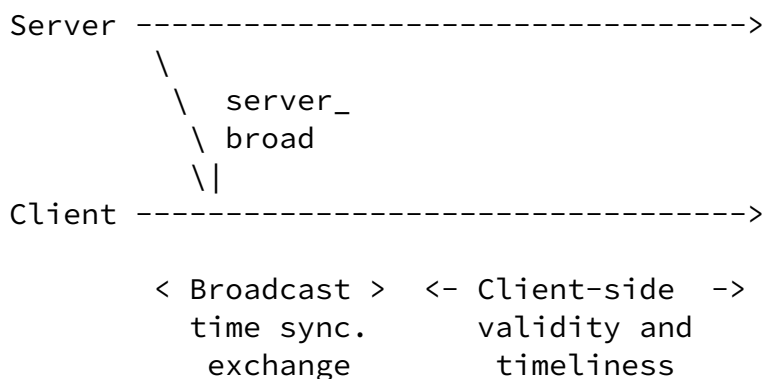
1. The server follows the TESLA protocol by regularly sending server\_broad messages as described in [Section 6.2.3](#), adhering to its own disclosure schedule.
2. The client awaits time synchronization data in the form of a server\_broadcast message. Upon receipt, it performs the following checks:
  - \* Proof that the MAC is based on a key that is not yet disclosed (packet timeliness). This is achieved via a combination of checks. First, the disclosure schedule is used, which

requires loose time synchronization. If this is successful, the client obtains a stronger guarantee via a key check exchange (see below). If its timeliness is verified, the packet will be buffered for later authentication. Otherwise, the client MUST discard it. Note that the time information included in the packet will not be used for synchronization until its authenticity could also be verified.

- \* The client checks that it does not already know the disclosed key. Otherwise, the client SHOULD discard the packet to avoid a buffer overrun. If this check is successful, the client ensures that the disclosed key belongs to the one-way key chain by applying the one-way function until equality with a previous disclosed key is shown. If it is falsified, the client MUST discard the packet.
- \* If the disclosed key is legitimate, then the client verifies the authenticity of any packet that it has received during the corresponding time interval. If authenticity of a packet is verified, then it is released from the buffer and its time information can be utilized. If the verification fails, then authenticity is not given. In this case, the client MUST request authentic time from the server by means other than

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

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



## checks

Procedure for broadcast time synchronization exchange.

### [6.3.](#) Broadcast Keycheck

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

#### [6.3.1.](#) Preconditions for the Broadcast Keycheck Exchange

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

- o They MUST negotiate the algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.
- o The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.
- o Client and server MUST exchange the cookie (which depends on the KIV as described in section [Section 5](#)). Authenticity, confidentiality and integrity of the communication MUST be ensured.

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

#### [6.3.2.](#) Goals of the Broadcast Keycheck Exchange

The keycheck exchange:

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

### 6.3.3. Message Type: "client\_keycheck"

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

- o the NTS message ID "client\_keycheck",
- o the NTS version number negotiated during association,
- o a nonce,
- o an interval number from the TESLA disclosure schedule,
- o the MAC algorithm negotiated during association,
- o the client's key input value KIV, and
- o optional: a MAC (generated with the cookie as key) for verification of all of the above data.

### 6.3.4. Message Type: "server\_keycheck"

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

- o the NTS message ID "server\_keycheck"
- o the version number as transmitted in "client\_keycheck",
- o the nonce transmitted in the client\_keycheck message,

- o the interval number transmitted in the client\_keycheck message, and
- o a MAC (generated with the cookie as key) for verification of all

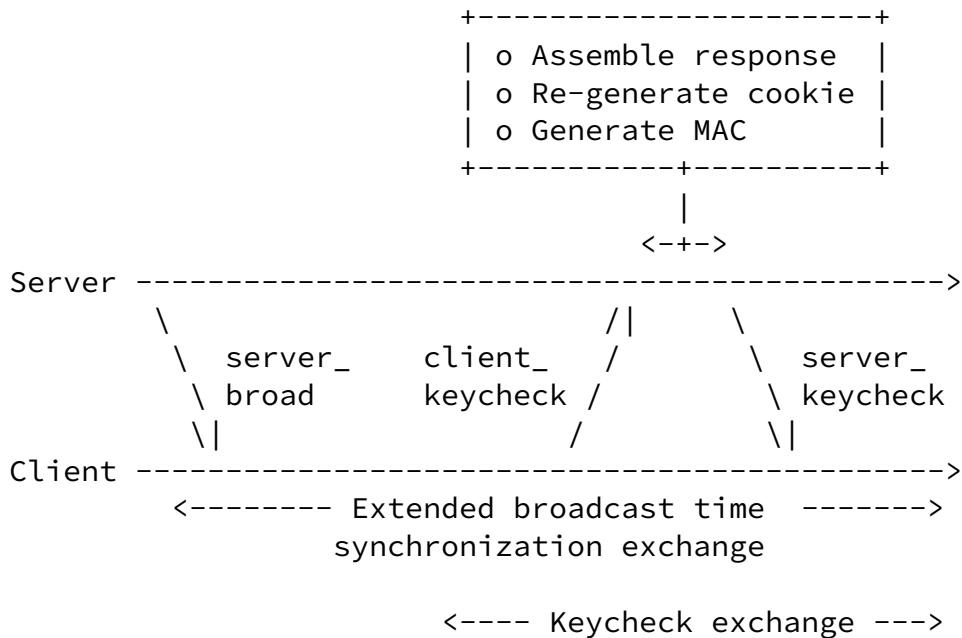


of the above data.

#### [6.3.5.](#) Procedure Overview of the Broadcast Keycheck Exchange

A broadcast keycheck message exchange consists of the following steps:

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



Procedure for extended broadcast time synchronization exchange.

## 7. Server Seed, MAC Algorithms and Generating MACs

### 7.1. Server Seed

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

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

### 7.2. MAC Algorithms

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

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

Internet-Draft

NTS

September 2016

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

## 8. IANA Considerations

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

## 9. Security Considerations

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

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

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

### 9.1. Privacy

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

enhance the privacy of the time synchronization partners, the usage of tunnel protocols such as IPsec and MACsec, where applicable, is therefore more suited than confidentiality protection of the payload.

## [9.2.](#) Initial Verification of the Server Certificates

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

## [9.3.](#) Revocation of Server Certificates

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

## [9.4.](#) Mitigating Denial-of-Service for broadcast packets

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

Discussion: Note that an alternative approach to enhance TESLA's

resistance against DoS attacks involves the addition of a group MAC to each packet. This requires the exchange of an additional shared key common to the whole group. This adds additional complexity to the protocol and hence is currently not considered in this document.

### 9.5. Delay Attack

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

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

1. Usage of multiple time servers: this enables the client to detect the attack, provided that the adversary is unable to delay the synchronization packets between the majority of servers. This approach is commonly used in NTP to exclude incorrect time servers [[RFC5905](#)].
2. Multiple communication paths: The client and server utilize different paths for packet exchange as described in the I-D [[I-D.ietf-tictoc-multi-path-synchronization](#)]. The client can detect the attack, provided that the adversary is unable to manipulate the majority of the available paths [[Shpiner](#)]. Note that this approach is not yet available, neither for NTP nor for PTP.
3. Usage of an encrypted connection: the client exchanges all packets with the time server over an encrypted connection (e.g. IPsec). This measure does not mitigate the delay attack, but it makes it more difficult for the adversary to identify the time synchronization packets.
4. For unicast-type messages: Introduction of a threshold value for the delay time of the synchronization packets. The client can discard a time server if the packet delay time of this time

server is larger than the threshold value.

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

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

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

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

## [9.6.](#) Random Number Generation

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

## [10.](#) Acknowledgements

The authors would like to thank Tal Mizrahi, Russ Housley, Steven Bellovin, David Mills, Kurt Roeckx, Rainer Bermbach, Martin Langer and Florian Weimer for discussions and comments on the design of NTS.

Also, thanks go to Harlan Stenn and Richard Welty for their technical review and specific text contributions to this document.

## 11. References

### 11.1. Normative References

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), DOI 10.17487/RFC2104, February 1997, <<http://www.rfc-editor.org/info/rfc2104>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", [RFC 4082](#), DOI 10.17487/RFC4082, June 2005, <<http://www.rfc-editor.org/info/rfc4082>>.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", [RFC 7384](#), DOI 10.17487/RFC7384, October 2014, <<http://www.rfc-editor.org/info/rfc7384>>.

### 11.2. Informative References

- [I-D.ietf-ntp-cms-for-nts-message]  
Sibold, D., Teichel, K., Roettger, S., and R. Housley, "Protecting Network Time Security Messages with the Cryptographic Message Syntax (CMS)", [draft-ietf-ntp-cms-for-nts-message-04](#) (work in progress), July 2015.
- [I-D.ietf-tictoc-multi-path-synchronization]  
Shpiner, A., Tse, R., Schelp, C., and T. Mizrahi, "Multi-Path Time Synchronization", [draft-ietf-tictoc-multi-path-synchronization-02](#) (work in progress), April 2015.

[IEEE1588]

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

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

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

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

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

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

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



## [Appendix A.](#) (informative) TICTOC Security Requirements

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

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

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

\*) See discussion in [Section 9.5](#).

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

## [Appendix B](#). (normative) Inherent Association Protocol Messages

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

### [B.1](#). Overview of NTS with Inherent Association Protocol

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

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

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

Internet-Draft

NTS

September 2016

cookie = MSB\_<b> (MAC(server seed, H(certificate of client))),

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

## [B.2.](#) Access Message Exchange

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

### [B.2.1.](#) Goals of the Access Message Exchange

The access message exchange:

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

### [B.2.2.](#) Message Type: "client\_access"

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

- o the NTS message ID "client\_access".

### [B.2.3.](#) Message Type: "server\_access"

This message is sent by the server on receipt of a client\_access message. It contains:

- o the NTS message ID "server\_access",
- o an access key.

### [B.2.4.](#) Procedure Overview of the Access Exchange

For an access exchange, the following steps are performed:

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

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

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

### [B.3.](#) Association Message Exchange

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

#### [B.3.1.](#) Goals of the Association Exchange

The association exchange:

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

#### [B.3.2.](#) Message Type: "client\_assoc"

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

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

- o a selection of accepted encryption algorithms.

#### [B.3.3.](#) Message Type: "server\_assoc"

This message is sent by the server upon receipt of client\_assoc. It contains

- o the NTS message ID "server\_assoc",
- o the nonce transmitted in client\_assoc,
- o the client's proposal for the version number, selection of accepted MAC algorithms and selection of accepted encryption algorithms, as transmitted in client\_assoc,
- o 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),
- o the server's choice of algorithm for encryption and for MAC creation, all of which MUST be chosen from the client's proposals,
- o 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
- o a chain of certificates, which starts at the server and goes up to

a trusted authority; each certificate MUST be certified by the one directly following it.

#### [B.3.4.](#) Procedure Overview of the Association Exchange

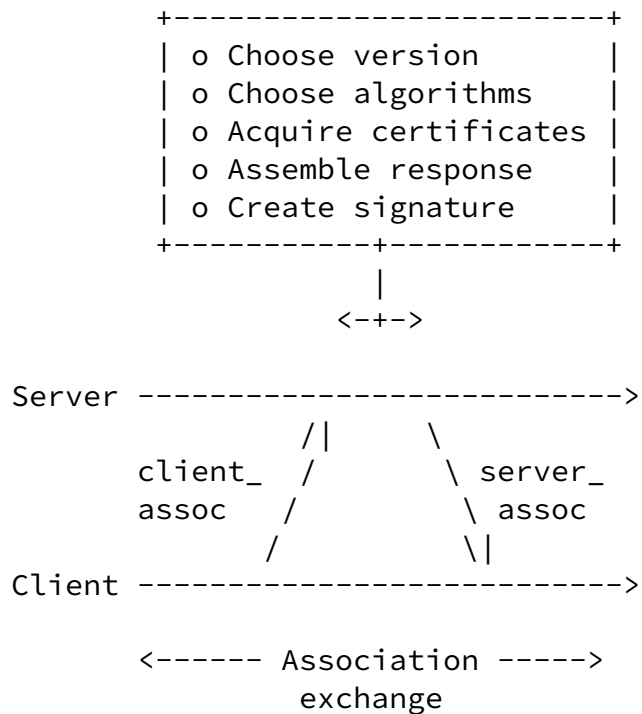
For an association exchange, the following steps are performed:

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

3. The client waits for a reply in the form of a `server_assoc` message. After receipt of the message it performs the following checks:
  - \* The client checks that the message contains a conforming version number.
  - \* It checks that the nonce sent back by the server matches the one transmitted in `client_assoc`,
  - \* It also verifies that the server has chosen the encryption and MAC algorithms from its proposal sent in the `client_assoc` message and that this proposal was not altered.
  - \* Furthermore, it performs authenticity checks on the certificate chain and the signature.

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

```
+-----+
| o Check access key    |
```



Procedure for association and cookie exchange.

#### [B.4.](#) Cookie Message Exchange

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

##### [B.4.1.](#) Goals of the Cookie Exchange

The cookie exchange:

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

- o guarantees that the received cookie is unknown to anyone but the server and the client.

#### [B.4.2.](#) Message Type: "client\_cook"

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

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

#### [B.4.3.](#) Message Type: "server\_cook"

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

- o the NTS message ID "server\_cook"
- o the version number as transmitted in client\_cook,

- o 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.



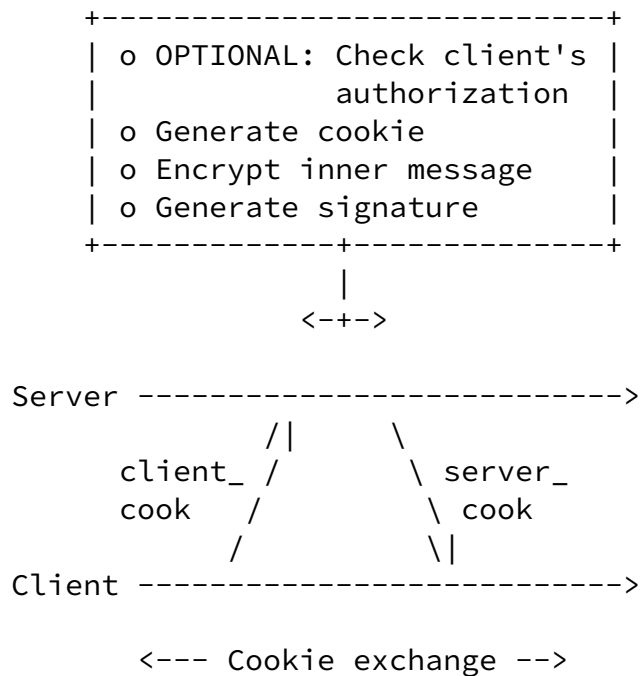
- o a signature, created with the server's private key, calculated over all of the data listed above. This signature MUST be calculated according to the transmitted signature algorithm from the client\_cook message.

#### [B.4.4.](#) Procedure Overview of the Cookie Exchange

For a cookie exchange, the following steps are performed:

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

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



Procedure for association and cookie exchange.

#### [B.4.5.](#) Broadcast Parameter Messages

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

See [Appendix C](#) for more details on TESLA.

##### [B.4.5.1.](#) Goals of the Broadcast Parameter Exchange

The broadcast parameter exchange

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

##### [B.4.5.2.](#) Message Type: "client\_bpar"

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

- o the NTS message ID "client\_bpar",

Internet-Draft

NTS

September 2016

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

#### [B.4.5.3.](#) Message Type: "server\_bpar"

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

- o the NTS message ID "server\_bpar",
- o the version number as transmitted in the client\_bpar message,
- o the nonce transmitted in client\_bpar,
- o the one-way functions used for building the key chain, and
- o 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.
- o The message also contains a signature signed by the server with its private key, verifying all the data listed above.

#### [B.4.5.4.](#) Procedure Overview of the Broadcast Parameter Exchange

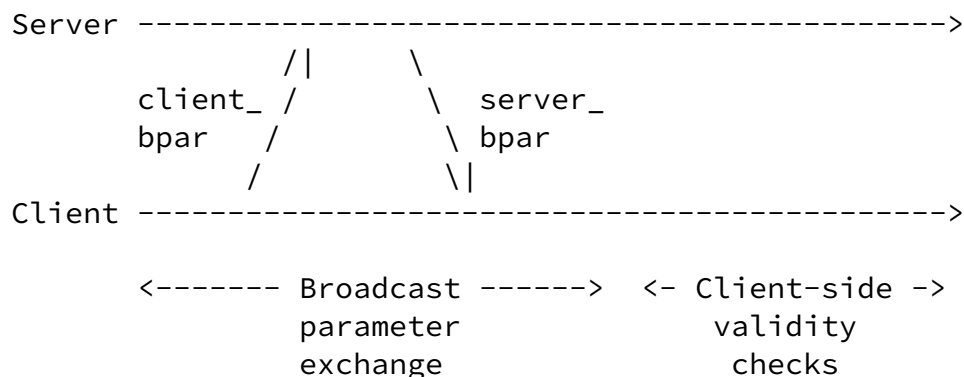
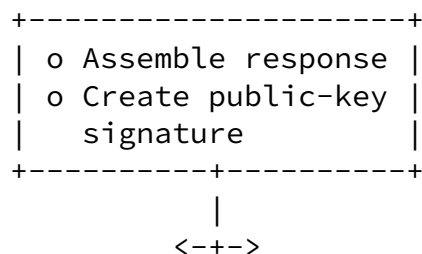
A broadcast parameter exchange consists of the following steps:

1. The client sends a client\_bpar message to the server. It MUST remember the transmitted values for the nonce, the version number and the signature algorithm.

2. Upon receipt of a client\_bpar message, the server constructs and sends a server\_bpar message as described in [Appendix B.4.5.3](#).
3. The client waits for a reply in the form of a server\_bpar message, on which it performs the following checks:

- \* The message must contain all the necessary information for the TESLA protocol, as listed in [Appendix B.4.5.3](#).
- \* The message must contain a nonce belonging to a client\_bpar message that the client has previously sent.
- \* Verification of the message's signature.

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



Procedure for unicast time synchronization exchange.

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

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

### [C.1.](#) Server Preparation

Server setup:

1. The server determines a reasonable upper bound  $B$  on the network delay between itself and an arbitrary client, measured in milliseconds.
2. It determines the number  $n+1$  of keys in the one-way key chain. This yields the number  $n$  of keys that are usable to authenticate broadcast packets. This number  $n$  is therefore also the number of time intervals during which the server can send authenticated broadcast messages before it has to calculate a new key chain.
3. It divides time into  $n$  uniform intervals  $I_1, I_2, \dots, I_n$ . 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  $K_0, K_1, \dots, K_n$  according to
$$K_i = F(K_{i+1}).$$
6. Using another one-way function  $F'$ , it generates a sequence of  $n$  MAC keys  $K'_0, K'_1, \dots, K'_{n-1}$  according to
$$K'_i = F'(K_i).$$
7. Each MAC key  $K'_i$  is assigned to the time interval  $I_i$ .

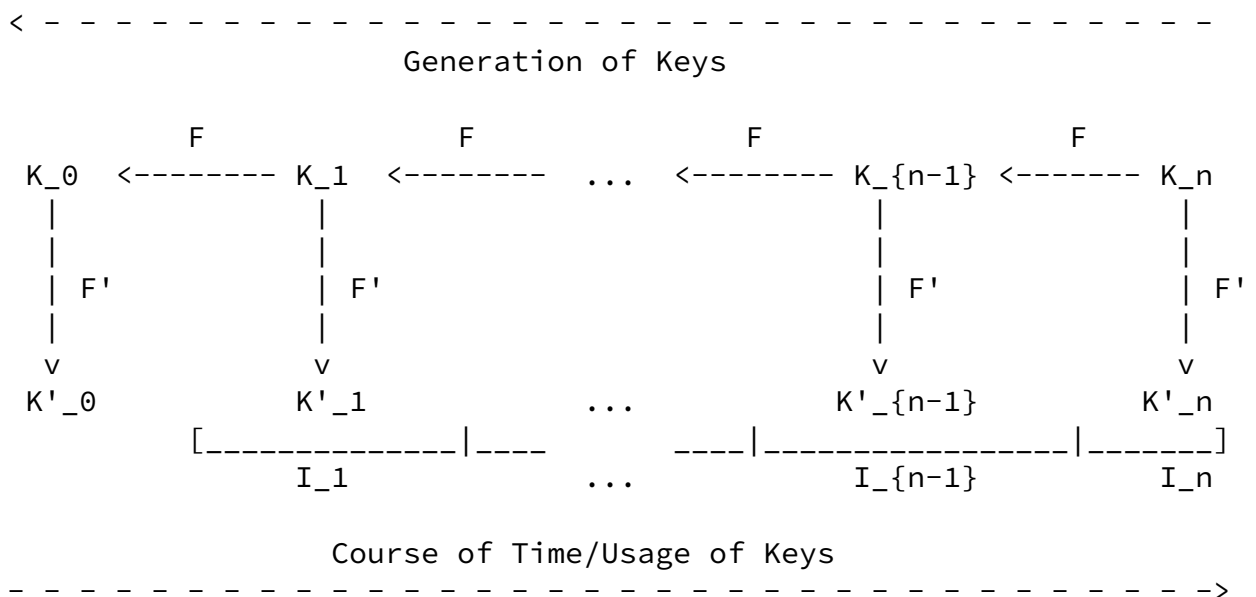
8. The server determines the key disclosure delay  $d$ , which is the number of intervals between using a key and disclosing it. Note that although security is provided for all choices  $d > 0$ , the choice still makes a difference:

- \* If  $d$  is chosen too short, the client might discard packets because it fails to verify that the key used for its MAC has not yet been disclosed.
- \* If  $d$  is chosen too long, the received packets have to be buffered for an unnecessarily long time before they can be verified by the client and be subsequently utilized for time synchronization.

It is RECOMMENDED that the server calculate  $d$  according to

$$d = \text{ceil}( 2*B / L ) + 1,$$

where  $\text{ceil}$  yields the smallest integer greater than or equal to its argument.



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

## C.2. Client Preparation

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

- o One key  $K_i$  from the one-way key chain, which has to be authenticated as belonging to the server. Typically, this will be  $K_0$ .
- o 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  $I_1, I_2, \dots, I_n$ ,
  - \* 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  $d$ .
- o The one-way function  $F$  used to recursively derive the keys in the one-way key chain,
- o The second one-way function  $F'$  used to derive the MAC keys  $K'_0, K'_1, \dots, K'_n$  from the keys in the one-way chain.

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

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

## C.3. Sending Authenticated Broadcast Packets

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

- o a message  $M_i$ ,

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

#### C.4. Authentication of Received Packets

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

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

$$t_i = R + D_t,$$

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

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

where  $\text{floor}$  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  $t_{\text{check}}$  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  $t_{\text{check}}$  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

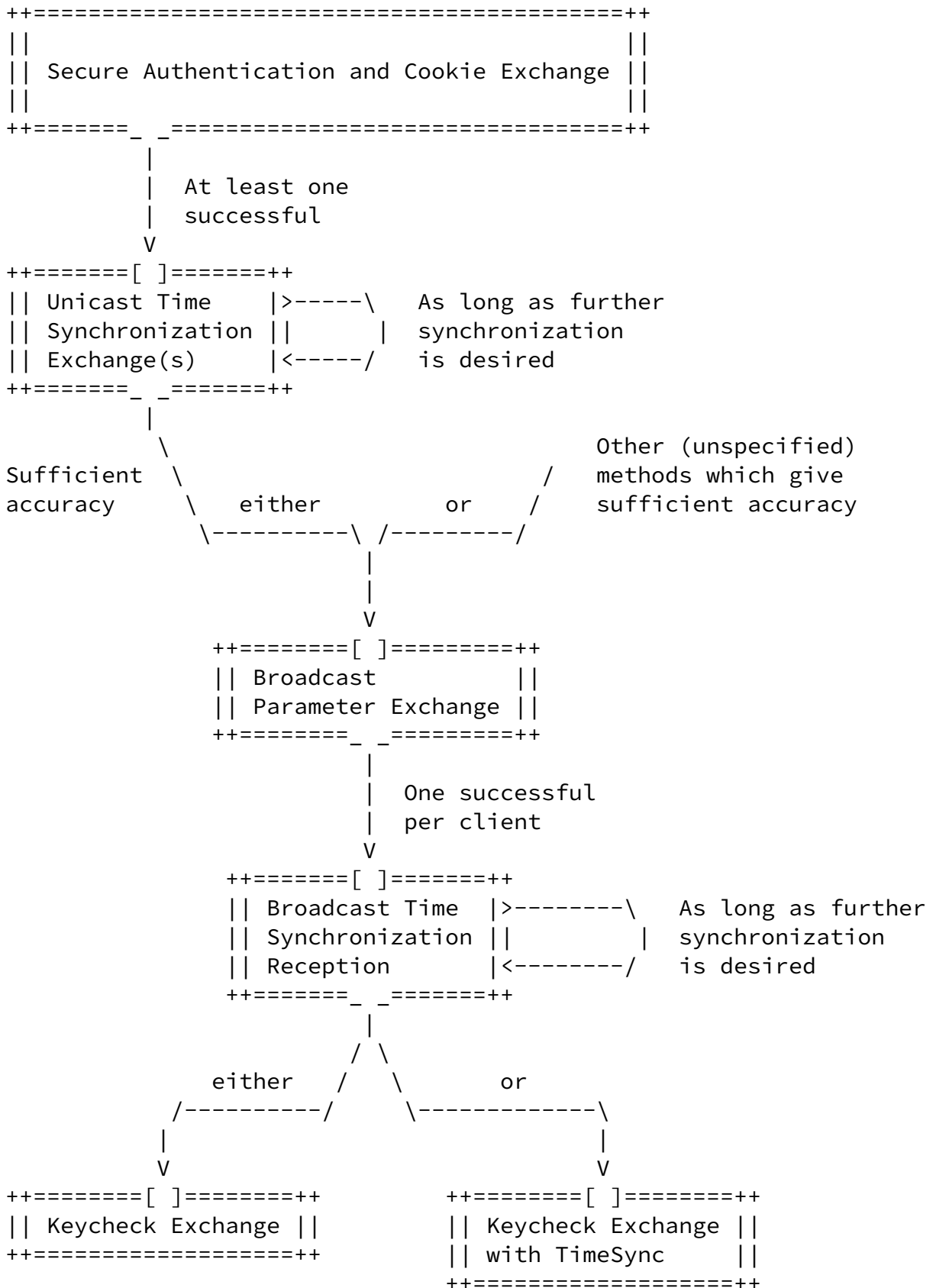
$$T_i \leq s_i, \text{ and}$$
$$s_i < T_{\{i+1\}}.$$

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.

Issuer	Type	Owner	Description
Server PKI	private key (signature)	server	Used for server_assoc, server_cook, server_bpar.
	public key (signature)	client	The server uses the private key to sign these messages. The client uses the public key to verify them.
	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
	public key (encryption)	server	messages. The client uses the private key to decrypt them. The certificate is
	certificate	client	sent in client_cook messages, where it is used for trans- portation of the public key as well as (optionally) for verification of client authorization.

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



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

Sibold, et al.

Expires March 26, 2017

[Page 38]

---

Internet-Draft

NTS

September 2016

#### 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](mailto:dieter.sibold@ptb.de)

Stephen Roettger  
Google Inc.

Email: [stephen.roettger@googlemail.com](mailto: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](mailto:kristof.teichel@ptb.de)

