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

Using the Network Time Security Specification to Secure the Network Time Protocol

draft-ietf-ntp-using-nts-for-ntp-06

Abstract

This document describes how to use the measures described in the Network Time Security (NTS) specification to secure time synchronization with servers using the Network Time Protocol (NTP).

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

Copyright Notice

Table of Contents

1. Introduction

One of the most popular time synchronization protocols, the Network Time Protocol (NTP) [RFC5905], currently does not provide adequate intrinsic security precautions. The Network Time Security draft [I-D.ietf-ntp-network-time-security] specifies security measures which can be used to enable time synchronization protocols to verify authenticity of the time server and integrity of the time synchronization protocol packets.

This document provides detail on how to specifically use those measures to secure time synchronization between NTP clients and servers.

2. Objectives

The objectives of the Network Time Security (NTS) specification are as follows:

- o Authenticity: NTS enables an NTP client to authenticate its time server(s).

- o Integrity: NTS protects the integrity of NTP 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 optionally enables the server to verify the client's authorization.

- 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 Modes of operation: Both the unicast and the broadcast mode of NTP are supported.

- o Hybrid mode: Both secure and insecure communication modes are possible for both NTP servers and clients.

o Compatibility:

* NTP associations which are not secured by NTS are not affected by NTS-secured communication.

* An NTP server that does not support NTS is not affected by NTS-secured authentication requests.

3. Terms and Abbreviations

CMS Cryptographic Message Syntax [RFC5652]

MAC Message Authentication Code

MITM Man In The Middle

NTP Network Time Protocol [RFC5905]

NTS Network Time Security

TESLA Timed Efficient Stream Loss-Tolerant Authentication [RFC4082]

4. Overview of NTS-Secured NTP

4.1. Symmetric and Client/Server Mode

The server does not keep a state of the client. NTS initially verifies the authenticity of the time server and exchanges a symmetric key, the so-called cookie and a key input value (KIV). The "access", "association", and "cookie" message exchanges described in [I-D.ietf-ntp-network-time-security], Appendix B., can be utilized for the exchange of the cookie and KIV. An implementation MUST support the use of these exchanges. It MAY additionally support the use of any alternative secure communication for this purpose, as long as it fulfills the preconditions given in [I-D.ietf-ntp-network-time-security], Section 6.1.1.

After the cookie and KIV are exchanged, the participants then use them to protect the authenticity and the integrity of subsequent unicast-type time synchronization packets. In order to do this, the server attaches a Message Authentication Code (MAC) 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. Therefore, the client can perform a validity check for this MAC on reception of a time synchronization packet.

4.2. Broadcast Mode

After the client has accomplished the necessary initial time synchronization via client-server mode, the necessary broadcast parameters are communicated from the server to the client. The "broadcast parameter" message exchange described in [I-D.ietf-ntp-network-time-security], Appendix B., can be utilized for this communication. An implementation MUST support the use of this exchange. It MAY additionally support the use of any alternative secure communication for this purpose, as long as it fulfills the necessary security goals (given in [I-D.ietf-ntp-network-time-security], Section 6.2.1.).

After the client has received the necessary broadcast parameters, "broadcast time synchronization" message exchanges are utilized in combination with optional "broadcast keycheck" exchanges to protect authenticity and integrity of NTP broadcast time synchronization packets. As in the case of unicast time synchronization messages, this is also achieved by MACs.

5. Protocol Sequence

Throughout this section, the access key, server seed, the nonces, cookies and MACs mentioned have bit lengths of B_accesskey, B_seed, B_nonce, B_cookie and B_mac, respectively. These bit lengths are defined in Appendix B (Appendix B). If a message requires a MAC to cover its contents, this MAC MUST be calculated according to:

    mac = MSB_<B_mac> (HMAC(key, content)),

where the application of the function MSB_<B_mac> returns only the B_mac most significant bits, where content is composed of the NTP header and all extension fields prior to the MAC-carrying extension field (see Section 6), and where key is the cookie for the given association.

Note for clarification that different message exchanges use different nonces. A nonce is always generated by the client for a request message, and then used by the server in its response. After this, it is not to be used again.

5.1. The Client

5.1.1. The Client in Unicast Mode

For a unicast run, the client performs the following steps:

NOTE: Steps 1 through 6 MAY alternatively be replaced by an alternative secure mechanism for access, association and cookie exchange.

Step 1: It sends a client_access message to the server.

Step 2: It waits for a reply in the form of a server_access message.

Step 3: It sends a client_assoc message to the server. It MUST include the access key from the previously received server_access message. It MUST keep the transmitted nonce as well as the values for the version number and algorithms available for later checks.

Step 4: It 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.

Discussion: Note that by performing the above message exchange and checks, the client validates the authenticity of its immediate NTP server only. It does not recursively validate the authenticity of each NTP server on the time synchronization chain. Recursive authentication (and authorization) as formulated in RFC 7384 [RFC7384] depends on the chosen trust anchor.

Step 5: Next it sends a client_cook message to the server. The client MUST save the included nonce until the reply has been processed.

Step 6: It 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 B_nonce bit nonce and a B_cookie 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.

Step 7: The client sends a time_request message to the server. The client MUST append a MAC to the time_request message. The client MUST save the included nonce and the transmit_timestamp (from the time synchronization data) as a correlated pair for later verification steps.

Step 8: 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 repeating the exchange of time_request and time_response messages.

The client's behavior in unicast mode is also expressed in Figure 1.

5.1.2. The Client in Broadcast Mode

To establish a secure broadcast association with a broadcast server, the client MUST initially authenticate the broadcast server and securely synchronize its time with it up to an upper bound for its time offset in unicast mode. After that, the client performs the following steps:

NOTE: Steps 1 and 2 MAY be replaced by an alternative security mechanism for the broadcast parameter exchange.

Step 1: It sends a client_bpar message to the server. It MUST remember the transmitted values for the nonce, the version number and the signature algorithm.

Step 2: It waits for a reply in the form of a server_bpar message after which it performs the following checks:

* The message must contain all the necessary information for the TESLA protocol, as specified for a server_bpar message.

* 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.

Step 3: The client awaits time synchronization data in the form of a server_broadcast message. Upon receipt, it performs the following checks:

1. 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: it sends a client_keycheck message and waits for the appropriate response. Note that it needs to memorize the nonce and the time interval number that it sends as a correlated pair. For more detail on both of the mentioned timeliness checks, see [I-D.ietf-ntp-network-time-security]. 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.

2. 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 verified, 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.

3. 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 it is released from the buffer and the packet's time information can be utilized. If the verification fails, then authenticity is no longer given. In this case, the client MUST request authentic time from the server by means of a unicast time request message. 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.

The client MUST restart the broadcast sequence with a client_bpar message ([I-D.ietf-ntp-network-time-security]) if the one-way key chain expires.

The client's behavior in broadcast mode can also be seen in Figure 2.

5.2. The Server

5.2.1. The Server in Unicast Mode

To support unicast mode, the server MUST be ready to perform the following actions:

- Upon receipt of a client_access message, the server constructs and sends a reply in the form of a server_access message as described in Appendix B of[I-D.ietf-ntp-network-time-security]. The server MUST construct the access key according to:

  access_key = MSB _<B_accesskey> (MAC(server seed; Client's IP address)).

- o Upon receipt of a client_assoc message, the server checks the included access key. To this end it reconstructs the access key and compares it against the received one. If they match, the server constructs and sends a reply in the form of a server_assoc message as described in [I-D.ietf-ntp-network-time-security]. In the case where the validity of the included access key can not be verified, the server MUST NOT reply to the received request.

- o 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 [I-D.ietf-ntp-network-time-security]. With this, it MUST construct a server_cook message as described in [I-D.ietf-ntp-network-time-security].

- o Upon receipt of a time_request message, the server re-calculates the cookie and the MAC for that time_request packet 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 computes the necessary time synchronization data and constructs a time_response message as given in [I-D.ietf-ntp-network-time-security].

If the time_request message was received in the context of an NTP peer association, the server MUST look up whether it has information about the authentication and authorization status for the given hash value of the client's certificate. If it does not, it MUST NOT use the NTP message contents for adjusting its own clock.

In addition to items above, the server MAY be ready to perform the following action:

- o If an external mechanism for association and key exchange is used, the server has to react accordingly.

5.2.2. The Server in Broadcast Mode

A broadcast server MUST also support unicast mode in order to provide the initial time synchronization, which is a precondition for any broadcast association. To support NTS broadcast, the server MUST additionally be ready to perform the following actions:

- o Upon receipt of a client_bpar message, the server constructs and sends a server_bpar message as described in [I-D.ietf-ntp-network-time-security].

- o Upon receipt of a client_keycheck message, the server re-calculates the cookie and the MAC for that client_keycheck packet 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 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 [I-D.ietf-ntp-network-time-security].

- o The server follows the TESLA protocol in all other aspects, by regularly sending server_broad messages as described in [I-D.ietf-ntp-network-time-security], adhering to its own disclosure schedule.

The server is responsible to watch for the expiration date of the one-way key chain and generate a new key chain accordingly.

In addition to the items above, the server MAY be ready to perform the following action:

- o Upon receipt of external communication for the purpose of broadcast parameter exchange, the server reacts according to the way the external communication is specified.

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

This section presents some hints about the structures of the communication packets for the different message types when one wishes to implement NTS for NTP. See document [I-D.ietf-ntp-cms-for-nts-message] for descriptions of the archetypes for CMS structures as well as for the ASN.1 structures that are referenced here.

The NTP extension field structure is defined in RFC 5905 [RFC5905] and clarified in [I-D.ietf-ntp-extension-field]. It looks as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Field Type           |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
.                                                               .
.                             Value                             .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Padding (as needed)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

All extension fields mentioned in the rest of this section do not require an NTP MAC field. If nothing else is explicitly stated, all of those extension fields MUST have a length of at least 28 octets.

Furthermore, all extension fields mentioned in the rest of this section are notified by one of three Field Type identifiers, signaling content related to NTS:

Field Type	ASN.1 Object of NTS Message
TBD1	ClientAccessData, ServerAccessData
TBD1	ClientAssocData, ServerAssocData
TBD1	ClientCookieData, ServerCookieData
TBD1	BroadcastParameterRequest, BroadcastParameterResponse
TBD2	TimeRequestSecurityData, TimeResponseSecurityData
TBD2	BroadcastTime
TBD2	ClientKeyCheckSecurityData, ServerKeyCheckSecurityData
TBD3	NTSMessageAuthenticationCode

(see IANA considerations (Section 7)).

The outermost structure of the extension field's Value field MUST be an ASN.1 object that is structured as follows:

```
NTSExtensionFieldContent := SEQUENCE {
    oid      OBJECT IDENTIFIER,
    errnum   OCTET STRING (SIZE(2)),
    content  ANY DEFINED BY oid
}
```

The field errnum represents the error code of any message. The client and server MAY ignore this field in any incoming message. The

server MUST set this to zero if the response to the request was generated successfully. If it could not successfully generate a response, the field errnum MUST be set to a non-zero value. The different values of this field is defined in the Appendix C.

Whenever NTS requires a MAC for protection of a message, this MAC MUST be included in an additional extension field. This MAC-carrying extension field MUST be placed after the other NTS-related extension field, and it SHOULD be the last extension field of the message. Any MAC supplied by NTS in a MAC-carrying extension field MUST be generated over the NTP header and all extension fields prior to the MAC-carrying extension field.

Content MAY be added to an NTS-protected NTP message after the MAC provided by NTS. However, it is RECOMMENDED to not make use of this option and to apply the MAC protection of NTS to the whole of an NTP message.

The MAC-carrying extension field contains an NTSExtensionFieldContent object, whose content field is structured according to NTS-Plain. The included NTS message object is as follows:

```
NTSMessageAuthenticationCode := SEQUENCE {
    mac      OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier:

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

Note: In the following sections the word MAC is always used as described above. In particular it is not to be confused with NTP's MAC field.

6.1. Unicast Messages

6.1.1. Access Messages

6.1.1.1. Message Type: "client_access"

This message is realized as an NTP packet with an extension field which holds an "NTS-Plain" archetype structure. This structure consists only of an NTS message object of the type "ClientAccessData".

6.1.1.2. Message Type: "server_access"

Like "client_access", this message is realized as an NTP packet with an extension field which holds an "NTS-Plain" archetype structure, i.e. just an NTS message object of the type "ServerAccessData". The latter holds all the data necessary for NTS.

6.1.2. Association Messages

6.1.2.1. Message Type: "client_assoc"

This message is realized as an NTP packet with an extension field which holds an "NTS-Plain" archetype structure. This structure consists only of an NTS message object of the type "ClientAssocData", which holds all the data necessary for the NTS security mechanisms.

6.1.2.2. Message Type: "server_assoc"

Like "client_assoc", this message is realized as an NTP packet with an extension field which holds an "NTS-Plain" archetype structure, i.e. just an NTS message object of the type "ServerAssocData". The latter holds all the data necessary for NTS.

6.1.3. Cookie Messages

6.1.3.1. Message Type: "client_cook"

This message type is realized as an NTP packet with an extension field which holds a CMS structure of archetype "NTS-Plain", containing in its core an NTS message object of the type "ClientCookieData". The latter holds all the data necessary for the NTS security mechanisms.

6.1.3.2. Message Type: "server_cook"

This message type is realized as an NTP packet with an extension field containing a CMS structure of archetype "NTS-Encrypted-and-Signed". The NTS message object in that structure is a "ServerCookieData" object, which holds all data required by NTS for this message type.

6.1.4. Time Synchronization Messages

6.1.4.1. Message Type: "time_request"

This message type is realized as an NTP packet with regular NTP time synchronization data. Furthermore, the packet has an extension field which contains an ASN.1 object of type "TimeRequestSecurityData"

(packed in a CMS structure of archetype "NTS-Plain"). Finally, this message MUST be protected by a MAC.

6.1.4.2. Message Type: "time_response"

This message is also realized as an NTP packet with regular NTP time synchronization data. The packet also has an extension field which contains an ASN.1 object of type "TimeResponseSecurityData". Finally, this message MUST be protected by a MAC.

Note: In these two messages, where two extension fields are present, the respective first extension field (the one not containing the MAC) only needs to have a length of at least 16 octets. The extension fields holding the MACs need to have the usual length of at least 28 octets.

6.2. Broadcast Messages

6.2.1. Broadcast Parameter Messages

6.2.1.1. Message Type: "client_bpar"

This first broadcast message is realized as an NTP packet which is empty except for an extension field which contains an ASN.1 object of type "BroadcastParameterRequest" (packed in a CMS structure of archetype "NTS-Plain"). This is sufficient to transport all data specified by NTS.

6.2.1.2. Message Type: "server_bpar"

This message type is realized as an NTP packet whose extension field carries the necessary CMS structure (archetype: "NTS-Signed"). The NTS message object in this case is an ASN.1 object of type "BroadcastParameterResponse".

6.2.2. Broadcast Time Synchronization Message

6.2.2.1. Message Type: "server_broad"

This message's realization works via an NTP packet which carries regular NTP broadcast time data as well as an extension field, which contains an ASN.1 object of type "BroadcastTime" (packed in a CMS structure with archetype "NTS-Plain"). Finally, this message MUST be protected by a MAC.

Note: In this message, the first extension field (the one not containing the MAC) only needs to have a length of at least 16

octets. The extension field holding the MACs needs to have the usual length of at least 28 octets.

6.2.3. Broadcast Keycheck

6.2.3.1. Message Type: "client_keycheck"

This message is realized as an NTP packet with an extension field, which transports a CMS structure of archetype "NTS-Plain", containing an ASN.1 object of type "ClientKeyCheckSecurityData". Finally, this message MUST be protected by a MAC.

6.2.3.2. Message Type: "server_keycheck"

This message is also realized as an NTP packet with an extension field, which contains an ASN.1 object of type "ServerKeyCheckSecurityData" (packed in a CMS structure of archetype "NTS-Plain"). Finally, this message MUST be protected by a MAC.

Note: In this message, the first extension field (the one not containing the MAC) only needs to have a length of at least 16 octets. The extension field holding the MACs needs to have the usual length of at least 28 octets.

7. IANA Considerations

7.1. Field Type Registry

Within the "NTP Extensions Field Types" registry table, add the field types:

Field Type	Meaning	References
TBD1	NTS-Related Content	[this doc]
TBD2	NTS-Related Content	[this doc]
TBD3	NTS-Related Content	[this doc]

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

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

Decimal	Description	References
TBD4	id-ct-nts-ntsForNtpMessageAuthenticationCode	[this doc]

8. Security Considerations

All security considerations described in [I-D.ietf-ntp-network-time-security] have to be taken into account. The application of NTS to NTP requires the following additional considerations.

8.1. Employing Alternative Means for Access, Association and Cookie Exchange

If an implementation uses alternative means to perform access, association and cookie exchange, it MUST make sure that an adversary cannot abuse the server to obtain a cookie belonging to a chosen KIV.

8.2. Usage of NTP Pools

The certification-based authentication scheme described in [I-D.ietf-ntp-network-time-security] is not applicable to the concept of NTP pools. Therefore, NTS is unable to provide secure usage of NTP pools.

8.3. Server Seed Lifetime

According to Clause 5.6.1 in RFC 7384 [RFC7384] the server MUST provide a means to refresh the value of its server seed from time to time. A generally valid value for the server seed lifetime cannot be given. The value depends on the required security level, the current threat situation, and the chosen MAC mechanisms.

As guidance, a value for the lifetime can be determined by stipulating a maximum number of time requests for which the exchanged cookie remains unchanged. For example, if this value is 1000 and the client sends a time request every 64 seconds, the server seed lifetime should be no longer than 64000 seconds. Corresponding considerations can be made for a minimum number of requests.

8.4. Supported MAC Algorithms

The list of the MAC algorithms supported by the server has to fulfill the following requirements:

- it MUST NOT include HMAC with SHA-1 or weaker algorithms,
- it MUST include HMAC with SHA-256 or stronger algorithms.

8.5. Protection for Initial Messages

Any NTS message providing access, association, or cookie exchange can be encapsulated in NTP an extension field which is piggybacked onto an NTP packet. NTS does not itself provide MAC protection to the NTP header of such a packet, because it only offers MAC protection to the NTP header once the cookie has been successfully exchanged.

9. Acknowledgements

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

10. References

10.1. Normative 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-06 (work in progress), February 2016.

[I-D.ietf-ntp-extension-field]
Mizrahi, T. and D. Mayer, "The Network Time Protocol Version 4 (NTPv4) Extension Fields", draft-ietf-ntp-extension-field-07 (work in progress), February 2016.

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

[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>.

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

[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>.

10.2. Informative References

[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>.

Appendix A. Flow Diagrams of Client Behaviour

```
                           +---------------+
                           |Access Messages|
                           +-------+-------+
                                   |
                                   v
                        +---------------------+
                        |Association Messages |
                        +----------+----------+
                                   |
   +------------------------------>o
   |                               |
   |                               v
   |                       +---------------+
   |                       |Cookie Messages|
   |                       +-------+-------+
   |                               |
   |                               o<------------------------------+
   |                               |                               |
   |                               v                               |
   |                     +-------------------+                     |
   |                     |Time Sync. Messages|                     |
   |                     +---------+---------+                     |
   |                               |                               |
   |                               v                               |
   |                            +-----+                            |
   |                            |Check|                            |
   |                            +--+--+                            |
   |                               |                               |
   |            /------------------+------------------\            |
   |           v                   v                   v           |
   |     .-----------.       .-------------.        .-------.      |
   |    ( MAC Failure )     ( Nonce Failure )      ( Success )     |
   |     '-----+-----'       '------+------'        '---+---'      |
   |           |                   |                   |           |
   |           v                   v                   v           |
   |    +-------------+     +-------------+     +--------------+   |
   |    |Discard Data |     |Discard Data |     |Sync. Process |   |
   |    +-------------+     +------+------+     +------+-------+   |
   |           |                   |                   |           |
   |           |                   |                   v           |
   +-----------+                   +------------------>o-----------+
```

           Figure 1: The client's behavior in NTS unicast mode.

```
                            +-----------------------------+
                            |Broadcast Parameter Messages |
                            +--------------+--------------+
                                           |
                                           o<--------------------------+
                                           |                           |
                                           v                           |
                            +-----------------------------+            |
                            |Broadcast Time Sync. Message |            |
                            +--------------+--------------+            |
                                           |                           |
   +-------------------------------------->o                           |
   |                                       |                           |
   |                                       v                           |
   |                             +-------------------+                 |
   |                             |Key and Auth. Check|                 |
   |                             +---------+---------+                 |
   |                                       |                           |
   |                      /----------------*----------------\          |
   |                    v                                   v          |
   |               .---------.                         .---------.    |
   |              ( Verified  )                       ( Falsified )   |
   |               '----+----'                         '----+----'    |
   |                    |                                   |          |
   |                    v                                   v          |
   |             +-------------+                        +-------+      |
   |             |Store Message|                        |Discard|      |
   |             +------+------+                        +---+---+      |
   |                    |                                   |          |
   |                    v                                   +---------o
   |            +---------------+                                      |
   |            |Check Previous |                                      |
   |            +-------+-------+                                      |
   |                    |                                              |
   |           /--------*--------\                                     |
   |          v                   v                                    |
   |     .---------.         .---------.                              |
   |    ( Verified  )       ( Falsified )                             |
   |     '----+----'         '----+----'                              |
   |          |                   |                                    |
   |          v                   v                                    |
   |   +-------------+   +-----------------+                          |
   |   |Sync. Process|   |Discard Previous |                          |
   |   +------+------+   +--------+--------+                          |
   |          |                   |                                    |
   +-----------+                   +-----------------------------------+
```

Figure 2: The client's behaviour in NTS broadcast mode.

Appendix B. Bit Lengths for Employed Primitives

Define the following bit lengths for server seed, nonces, cookies and MACs:

B_accesskey = 128,

B_seed = 128,

B_nonce = 128,

B_cookie = 128, and

B_mac = 128.

Appendix C. Error Codes

Bit	Meaning
1	D2

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