                          Network Time Security
                draft-ietf-ntp-network-time-security-05.txt

Abstract

   This document describes the Network Time Security (NTS) protocol that
   enables secure time synchronization with time servers using Network
   Time Protocol (NTP) or 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

Table of Contents

1. Introduction

   Time synchronization protocols are increasingly utilized to
   synchronize clocks in networked infrastructures. The reliable
   performance of such infrastructures can be degraded seriously by
   successful attacks against the time synchronization protocol.
   Therefore, time synchronization protocols have to be secured if they
   are applied in environments that are prone to malicious attacks.
   This can be accomplished by utilization of external security
   protocols like IPsec or by intrinsic security measures of the time
   synchronization protocol.

   The two most popular time synchronization protocols, the Network Time
   Protocol (NTP) [RFC5905] and the Precision Time Protocol (PTP)
   [IEEE1588], currently do not provide adequate intrinsic security
   precautions. This document specifies security measures for NTP and
   PTP which enable these protocols to verify authenticity of the time
   server and integrity of the time synchronization protocol packets.

The protocol is specified with the prerequisite in mind that precise timekeeping can only be accomplished with stateless time synchronization communication, which excludes the utilization of standard security protocols like IPsec or TLS for time synchronization messages. This prerequisite corresponds with the requirement that a security mechanism for timekeeping must be designed in such a way that it does not degrade the quality of the time transfer [RFC7384].

Note:

   The intent is to formulate the protocol to be applicable to NTP and also PTP. In the current state the specification focuses on the application to NTP.

2. Security Threats

A profound analysis of security threats and requirements for NTP and PTP can be found in the "Security Requirements of Time Protocols in Packet Switched Networks" [RFC7384].

3. Objectives

The objectives of the NTS specification are as follows:

o Authenticity: NTS enables the client to authenticate its time servers.

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 Modes of operation: All operational modes of NTP are supported.

o Operational modes of PTP should be supported as far as possible.

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

o Compatibility:

   * Unsecured NTP associations shall not be affected.

   * An NTP server that does not support NTS shall not be affected by NTS authentication requests.

4. Terms and Abbreviations

   MITM   Man In The Middle

   NTP   Network Time Protocol [RFC5905]

   NTS   Network Time Security

   PTP   Precision Time Protocol [IEEE1588]

   TESLA   Timed Efficient Stream Loss-Tolerant Authentication

5. NTS Overview

5.1. Symmetric and Client/Server Mode

   NTS applies X.509 certificates to verify the authenticity of the time
   server and to exchange a symmetric key, the so-called cookie. This
   cookie is then used to protect authenticity and integrity of the
   subsequent time synchronization packets by means of a Message
   Authentication Code (MAC), which 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_128 (HMAC(server seed, H(certificate of client))),

   with the server seed as key, where H is a hash function, and where
   the function MSB_128 cuts off the 128 most significant bits of the
   result of the HMAC function. The server seed is a 128 bit random
   value of the server, which has to be kept secret. The cookie never
   changes as long as the server seed stays the same, but the server
   seed has to be refreshed periodically in order to provide key
   freshness as required in [RFC7384]. See Section 8 for details on the
   seed refresh and Section 7.1.1 for the client's reaction to it.

   The server does not keep a state of the client. Therefore it has to
   recalculate the cookie each time it receives a request from the
   client. To this end, the client has to attach the hash value of its
   certificate to each request (see Section 6.3).

5.2. Broadcast Mode

   Just as in the case of the client server mode and symmetric mode,
   authenticity and integrity of the NTP packets are ensured by a MAC,
   which is attached to the NTP packet by the sender. Verification of
   the 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 last element of the chain is shared securely with all
clients. The server splits time into intervals of uniform duration
and assigns each key to an interval in reverse order, starting with
the penultimate. At each time interval, the server sends an NTP
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 the
disclosure of the key in its associated disclosure interval. In
order to be able to verify the validity of the key, the client has to
be loosely time synchronized to the server. This has to be
accomplished during the initial client server exchange between
broadcast client and server. In addition, NTS uses another, more
rigorous check to what is 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, for use with existing time synchronization protocols, see
[I-D.ietf-ntp-cms-for-nts-messages], a companion document for NTS.
Said document describes ASN.1 encodings for those message parts that
have to be added to a time synchronization protocol for security
reasons as well as CMS (Cryptographic Message Syntax, see [RFC5652])
conventions that can be used to get the cryptographic aspects right.

Note that currently, the companion document describes realizations of
NTS messages only for utilization with NTP, in which the NTS specific
data are enclosed in extension fields on top of NTP packets. A
specification of NTS messages for PTP will have to be developed
accordingly.

The steps described in Section 6.1 - Section 6.3 belong to the
unicast mode, while Section 6.4 and Section 6.5 explain the steps
involved in the broadcast mode of NTS.

6.1. Association Messages

In this message exchange, the hash and encryption algorithms that are
used throughout the protocol are negotiated. Also, 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, authenticity of future NTS messages
from the server is ensured.

6.1.1. Message Type: "client_assoc"

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

o the NTS message ID "client_assoc",

o the version number of NTS that the client wants to use (this
   SHOULD be the highest version number that it supports),

o the hostname of the client,

o a selection of accepted hash algorithms, and

o a selection of accepted encryption algorithms.

6.1.2. 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 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 hostname of the server, and

o the server's choice of algorithm for encryption and for
   cryptographic hashing, 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 which are included (see
   below),

o a chain of certificates, which starts at the server and goes up to
   a trusted authority, and each certificate MUST be certified by the
   one directly following it.

6.2. Cookie Messages

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

6.2.1. 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 the negotiated version number,

   o the negotiated signature algorithm,

   o the negotiated encryption algorithm,

   o a 128-bit nonce,

   o the negotiated hash algorithm H,

   o the client's certificate.

6.2.2. Message Type: "server_cook"

   This message is sent by the server, upon receipt of a client_cook
   message. The server generates the hash of the client's certificate,
   as conveyed during client_cook, in order to calculate the cookie
   according to Section 5.1. 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.

6.3. Unicast Time Synchronisation Messages

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

6.3.1. Message Type: "time_request"

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

o   the NTS message ID "time_request",

o   the negotiated version number,

o   a 128-bit nonce,

o   the negotiated hash algorithm H,

o   the hash of the client's certificate under H.

6.3.2. Message Type: "time_response"

This message is sent by the server, after it received a time_request
message. Prior to this the server MUST recalculate the client's
cookie by using the hash of the client's certificate and the
transmitted hash algorithm. The message contains

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 128-bit nonce transmitted in time_request,

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

6.4. Broadcast Parameter Messages

In this message exchange, the client receives the necessary
information to execute the TESLA protocol in a secured broadcast
association. The client can only initiate a secure broadcast
association after a successful unicast run, see Section 7.1.2.

See Appendix C for more details on TESLA.

6.4.1. 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",

o the version number negotiated during association in unicast mode,

o the client's hostname, and

o the signature algorithm negotiated during unicast.

6.4.2. 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 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.

6.5. Broadcast Messages

Via this message, the server keeps sending broadcast time
synchronization messages to all participating clients.

6.5.1. 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's broadcast mode 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.

6.6. Broadcast Key Check

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

6.6.1. 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 version number chosen for the broadcast,

o a 128-bit nonce,

o an interval number from the TESLA disclosure schedule,

o the hash algorithm H negotiated in unicast mode, and

o the hash of the client's certificate under H.

6.6.2. Message Type: "server_keycheck"

A message of this type is sent by the server upon receipt of a
client_keycheck message during the broadcast loop of the server.
Prior to this the server MUST recalculate the client's cookie by
using the hash of the client's certificate and the transmitted hash
algorithm. It contains

o the NTS message ID "server_keycheck"

o the version number that the server's broadcast mode is working
   under,

o the 128-bit 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.

7. Protocol Sequence

7.1. The Client

7.1.1. The Client in Unicast Mode

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

1. It sends a client_assoc message to the server. It MUST keep the
    transmitted values for version number and algorithms available
    for later checks.

2. 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 conform version
       number.

    * It also verifies that the server has chosen the encryption and
       hash algorithms from its proposal sent in the client_assoc
       message.

   * Furthermore, it performs authenticity checks on the
      certificate chain and the signature for the version number.

   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 [RFC7384] depends on the chosen trust anchor.

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

4. 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 before.

   * It verifies the signature using the server's public key. The
      signature has to authenticate the encrypted data.

   * It decrypts the encrypted data with its own private key.

   * It checks that the decrypted message is of the expected
      format: the concatenation of a 128 bit nonce and a 128 bit
      cookie.

   * It verifies that the received nonce matches the nonce sent in
      the client_cook message.

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

5. 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.

6. 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
      before,

* that the transmitted nonce belongs to a previous time_request message,

* that the transmit_timestamp in that time_request message matches the corresponding time stamp from the synchronization data received in the time_response, and

* that the appended MAC verifies the received synchronization data, version number and nonce.

If at least one of the first three checks fails (i.e. if the version number does not match, if the client has never used the nonce transmitted in the time_response message or if it has used the nonce with initial time synchronization data different from that in the response), then the client MUST ignore this time_response message. If the MAC is invalid, the client MUST do one of the following: abort the run or go back to step 5 (because the cookie might have changed due to a server seed refresh). If both checks are successful, the client SHOULD continue time synchronization by going back to step 7.

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

7.1.2. 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 to it up to an upper bound for its time offset in unicast mode. After that, the client performs the following steps:

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

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 listed in Section 6.4.2.

    * Verification of the message's signature.

    If any information is missing or 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.

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 the loose time synchronization. If this
        is successful, the client gets 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 Appendix Appendix C.4. 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 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 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.

    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 Section 6.4.1 if the one-way key chain expires.

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

7.2. The Server

7.2.1. The Server in Unicast Mode

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

   o Upon receipt of a client_assoc message, the server constructs and
      sends a reply in the form of a server_assoc message as described
      in Section 6.1.2.

   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
      Section 5.1. With this, it MUST construct a server_cook message
      as described in Section 6.2.2.

   o Upon receipt of a time_request message, the server re-calculates
      the cookie, then computes the necessary time synchronization data
      and constructs a time_response message as given in Section 6.3.2.

   The server MUST refresh its server seed periodically (see
   Section 8.1).

7.2.2. 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 Section 6.4.2.

   o Upon receipt of a client_keycheck message, the server looks up if
      it has already disclosed the key associated with the interval
      number transmitted in that message. If it has not disclosed it,
      it constructs and sends the appropriate server_keycheck message as
      described in Section 6.6.2. For more detail, see also Appendix C.

   o The server follows the TESLA protocol in all other aspects, by
      regularly sending server_broad messages as described in
      Section 6.5.1, adhering to its own disclosure schedule.

   It is also the server's responsibility to watch for the expiration
   date of the one-way key chain and generate a new key chain
   accordingly.

8. Server Seed Considerations

The server has to calculate a random seed which has to be kept
secret. The server MUST generate a seed for each supported hash
algorithm, see Section 9.1.

8.1. Server Seed Refresh

According to the requirements in [RFC7384] the server MUST refresh
each server seed periodically. As a consequence, the cookie
memorized by the client becomes obsolete. In this case the client
cannot verify the MAC attached to subsequent time response messages
and has to respond accordingly by re-initiating the protocol with a
cookie request (Section 6.2).

8.2. Server Seed Algorithm

8.3. Server Seed Lifetime

9. Hash Algorithms and MAC Generation

9.1. Hash Algorithms

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

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

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

o it MUST NOT include SHA-1 or weaker algorithms,

o it MUST include SHA-256 or stronger algorithms.

Note

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

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

9.2. MAC Calculation

For the calculation of the MAC, client and server are using a Keyed-
Hash Message Authentication Code (HMAC) approach [RFC2104]. The HMAC
is generated with the hash algorithm specified by the client (see
Section 9.1).

10. IANA Considerations

11. Security Considerations

11.1. Initial Verification of the Server Certificates

The client has to verify the validity of the certificates during the
certification message exchange (Section 6.1.2). Since it generally
has no reliable time during this initial communication phase, it is
impossible to verify the period of validity of the certificates.
Therefore, the client MUST use one of the following approaches:

o The validity of the certificates is preconditioned. Usually this
   will be the case in corporate networks.

o The client ensures that the certificates are not revoked. To this
   end, the client uses the Online Certificate Status Protocol (OCSP)
   defined in [RFC6277].

o The client requests a different service to get an initial time
   stamp in order to be able to verify the certificates' periods of
   validity. To this end, it can, e.g., use a secure shell
   connection to a reliable host. Another alternative is to request
   a time stamp from a Time Stamping Authority (TSA) by means of the
   Time-Stamp Protocol (TSP) defined in [RFC3161].

11.2. Revocation of Server Certificates

According to Section 8.1, it is the client's responsibility to
initiate a new association with the server after the server's
certificate expires. To this end the client reads the expiration
date of the certificate during the certificate message exchange
(Section 6.1.2). Besides, certificates may also be revoked prior to
the normal expiration date. To increase security the client MAY
verify the state of the server's certificate via OCSP periodically.

11.3. Usage of NTP Pools

   The certification based authentication scheme described in Section 6
   is not applicable to the concept of NTP pools. Therefore, NTS is not
   able to provide secure usage of NTP pools.

11.4. Denial-of-Service in Broadcast Mode

   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, client and 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 knows already, thus
   preventing memory flooding attacks.

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

11.5. 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 to measure the
   network delay, and hence its time offset to the server, accurately
   [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
       synchronizations 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 are utilizing
       different paths for packet exchange as described in the I-D
       [I-D.shpiner-multi-path-synchronization]. The client can detect
       the attack provided that the adversary is unable to manipulate
       the majority of the available paths [Shpiner]. Note that this
       approach is not yet available, neither for NTP nor for PTP.

3. Usage of an encrypted connection: the client exchanges all
    packets with the time server over an encrypted connection (e.g.
    IPsec). This measure does not mitigate the delay attack but it
    makes it more difficult for the adversary to identify the time
    synchronization packets.

4. For the unicast mode: 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 in the
broadcast mode. 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 maintains
time synchronization by utilization of the broadcast mode. To this
end it initially establishes a unicast association with its broadcast
server until time synchronization and calibration of the packet delay
time is achieved. After that it establishes a broadcast association
to 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 section
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.

12. 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 for his technical review and
specific text contributions to this document.

13. References

13.1. Normative References

   [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.

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

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

   [RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato,
              "Internet X.509 Public Key Infrastructure Time-Stamp
              Protocol (TSP)", RFC 3161, August 2001.

   [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B.
              Briscoe, "Timed Efficient Stream Loss-Tolerant
              Authentication (TESLA): Multicast Source Authentication
              Transform Introduction", RFC 4082, June 2005.

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

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

   [RFC6277] Santesson, S. and P. Hallam-Baker, "Online Certificate
              Status Protocol Algorithm Agility", RFC 6277, June 2011.

13.2. Informative References

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

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

   [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness
              Requirements for Security", BCP 106, RFC 4086, June 2005.

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

   [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. Flow Diagrams of Client Behaviour

```
                       +---------------------+
                       |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----------+
```
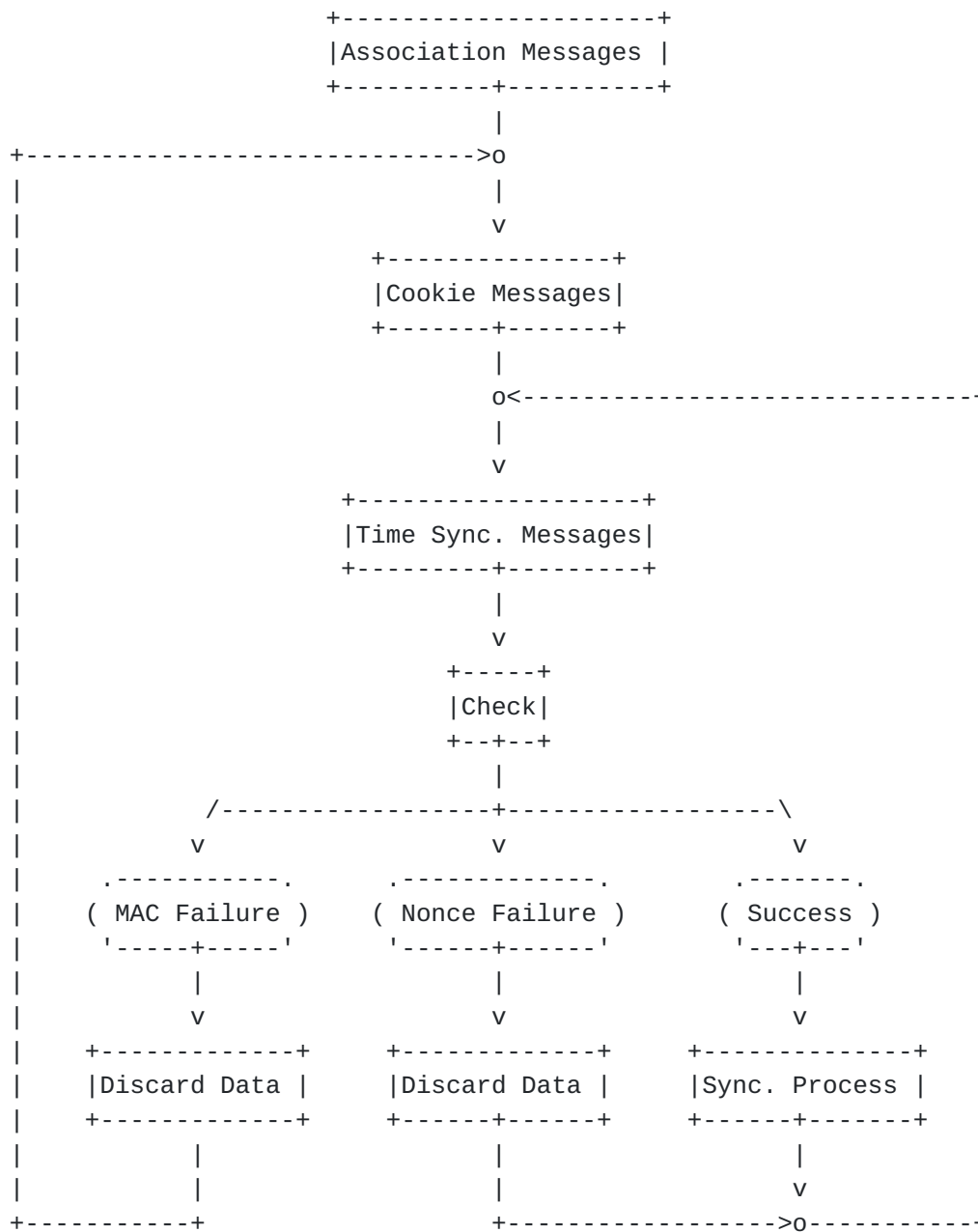
                Figure 1: The client's behavior in NTS unicast mode.

```
                    +------------------------------+
                    |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 |                 |
     | +------+------+  +--------+--------+                 |
     |        |                  |                          |
     +--------+                  +--------------------------+
```
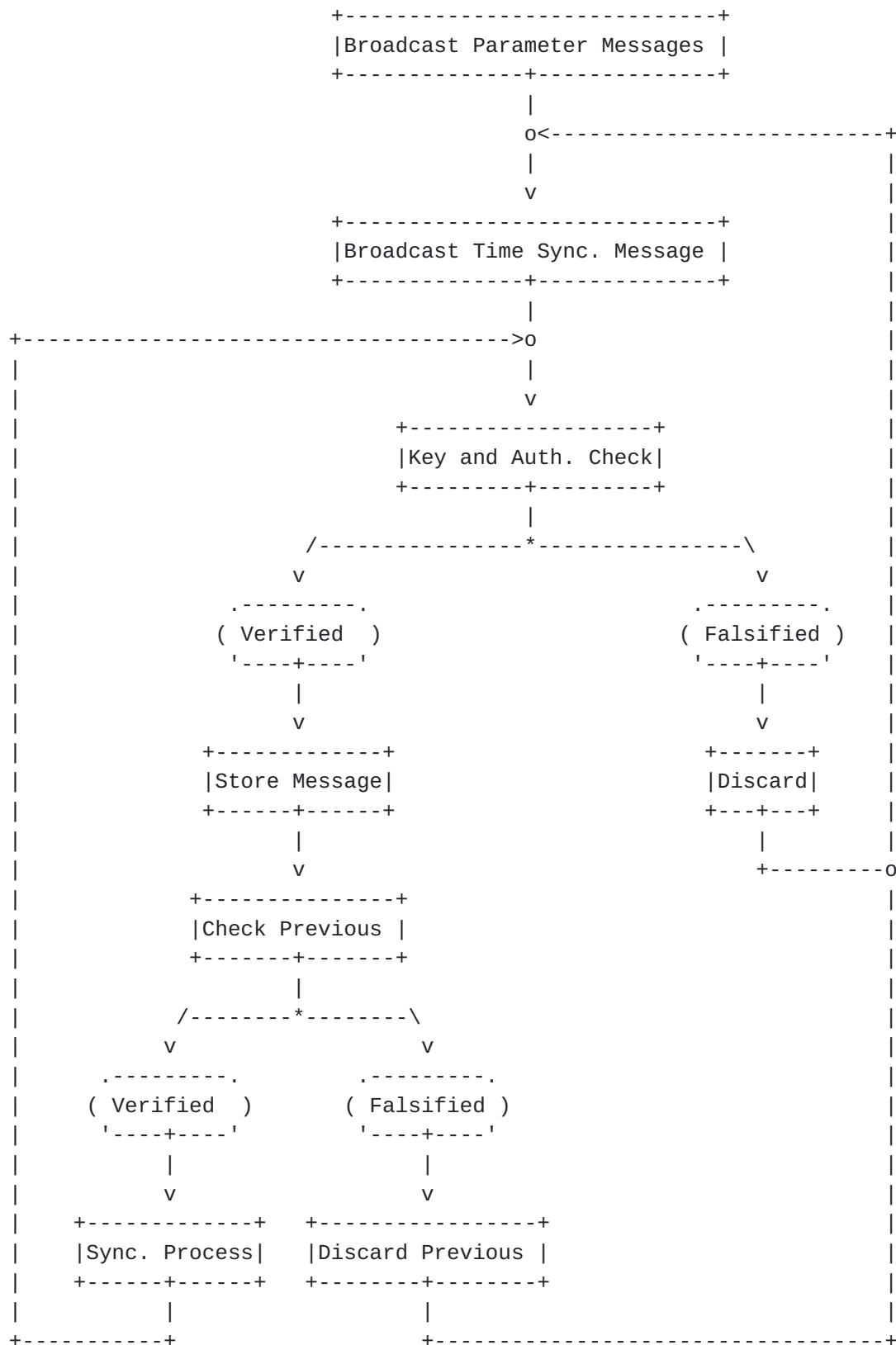
             Figure 2: The client's behaviour in NTS broadcast mode.

Appendix B. TICTOC Security Requirements

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

Section	Requirement from I-D tictoc security-requirements-05	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 Slaves	MAY	-
5.2	Integrity protection.	MUST	OK
5.4	Protection against DoS attacks	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, bandwidth	SHOULD	OK
5.7	Confidentiality protection	MAY	NO
5.9	Protection against Packet Delay and Interception Attacks	SHOULD	NA*)

```
| 5.10    | Secure mode                        | MUST        | -    |
+---------+------------------------------------+-------------+------+
|         | Hybrid mode                        | SHOULD      | -    |
+---------+------------------------------------+-------------+------+
```

*) See discussion in section [Section 11.5](#).

           Comparison of NTS spececification against TICTOC security
                             requirements.

[Appendix C](#). Broadcast Mode

For the broadcast mode, 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 is using the "not re-using keys" scheme of
TESLA as described in [section 3.7.2. of [RFC4082]](#).

[C.1](#). Server Preparations

Server setup:

1. The server determines a reasonable upper bound B on the network
    delay between itself and an arbitrary client, measured in
    milliseconds.

2. It determines the number n+1 of keys in the one-way key chain.
    This yields the number n of keys that are usable to authenticate
    broadcast packets. This number n is therefore also the number of
    time intervals during which the server can send authenticated
    broadcast messages before it has to calculate a new key chain.

3. It divides time into n uniform intervals I_1, I_2, ..., I_n.
    Each of these time intervals has length L, measured in
    milliseconds. In order to fulfill the requirement 3.7.2. of [RFC
    4082](#) the time interval L has to be smaller than the time interval
    between the broadcast messages.

4. The server generates a random key K_n.

5. Using a one-way function F, the server generates a one-way chain
    of n+1 keys K_0, K_1, ..., K_{n} according to

        K_i = F(K_{i+1}).

6. Using another one-way function F', it generates a sequence of n+1
    MAC keys K'_0, K'_1, ..., K'_{n-1} according to
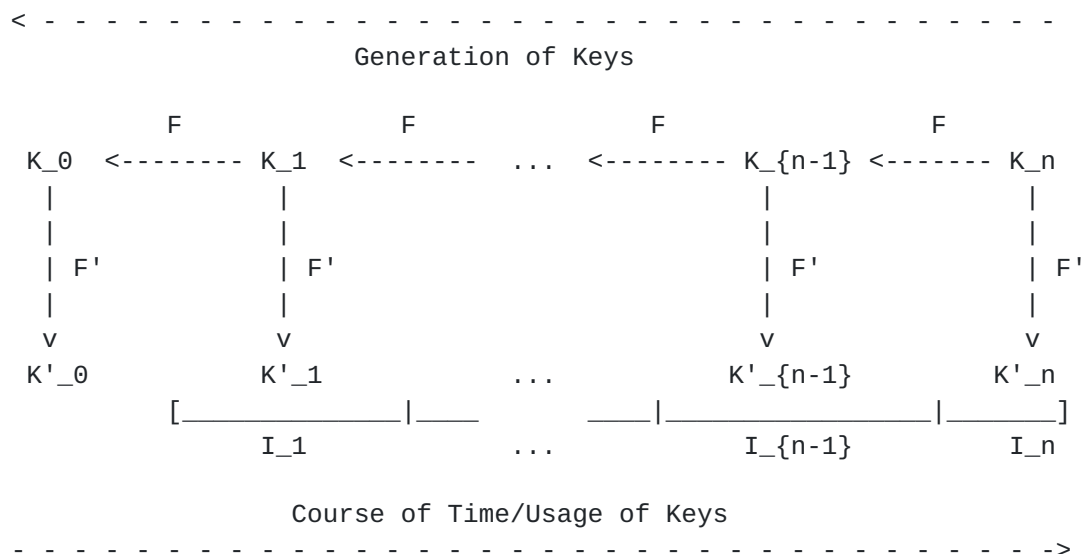
```
      K'_i = F'(K_i).
```

7. Each MAC key K'_i is assigned to the time interval I_i.

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

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

    * If d is chosen too long, the received packets have to be
       buffered for a unnecessarily long time before they can be
       verified by the client and subsequently be utilized for time
       synchronization.

    The server SHOULD calculate d according to

       d = ceil(2*B / L) + 1,

    where ceil gives the smallest integer greater than or equal to
    its argument.

```
< - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                      Generation of Keys


          F                F                F                F
  K_0   <-------- K_1  <--------  ...  <-------- K_{n-1} <------- K_n
   |              |                             |                |
   |              |                             |                |
   | F'           | F'                          | F'             | F'
   |              |                             |                |
   v              v                             v                v
  K'_0           K'_1            ...          K'_{n-1}          K'_n
         [_____|____      .    ____|_____|_____]
               I_1                ...          I_{n-1}          I_n

                   Course of Time/Usage of Keys
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ->
```

    A Schematic explanation on 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, ..., 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, ... , 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 one authenticated
   broadcast packet P_i. This 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 received a packet with the same disclosed key
   before. 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 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 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 timestamp t_check of the
   sending time of that request as well as the nonce it used correlated
   with the interval number i. If it receives an answer from the server
   stating that K_i has not yet been disclosed and it is able to verify
   the HMAC on that response, then it deduces that K_i was undisclosed
   at t_check and therefore also at R. In this case, the clients
   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 identity with an earlier
   disclosed key (see Clause 3.5 in RFC 4082, item 3).

Next the client verifies that the transmitted time value s_i belongs
to the time interval I_i, by checking

$T_i =< s_i$, and

$s_i < T_{i+1}$.

If falsified, the packet MUST be discarded and the client MUST
reinitialize the broadcast mode with a unicast association (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 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}. On success, it regards M_{i-d} as authenticated.

Appendix D. 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.

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