NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2019

D. Franke

D. Sibold
K. Teichel
PTB

July 01, 2018

Network Time Security for the Network Time Protocol
draft-ietf-ntp-using-nts-for-ntp-12

Abstract

This memo specifies Network Time Security (NTS), a mechanism for using Transport Layer Security (TLS) and Authenticated Encryption with Associated Data (AEAD) to provide cryptographic security for the Network Time Protocol.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This memo specifies Network Time Security (NTS), a cryptographic security mechanism for network time synchronization. A complete specification is provided for application of NTS to the client-server mode of the Network Time Protocol (NTP) [RFC5905].

1.1. Objectives

The objectives of NTS are as follows:

- o Identity: Through the use of the X.509 PKI, implementations may cryptographically establish the identity of the parties they are communicating with.

- o Authentication: Implementations may cryptographically verify that any time synchronization packets are authentic, i.e., that they were produced by an identified party and have not been modified in transit.

- o Confidentiality: Although basic time synchronization data is considered non-confidential and sent in the clear, NTS includes support for encrypting NTP extension fields.

- o Replay prevention: Client implementations may detect when a received time synchronization packet is a replay of a previous packet.

- o Request-response consistency: Client implementations may verify that a time synchronization packet received from a server was sent in response to a particular request from the client.

- o Unlinkability: For mobile clients, NTS will not leak any information additional to NTP which would permit a passive adversary to determine that two packets sent over different networks came from the same client.

- o Non-amplification: Implementations (especially server implementations) may avoid acting as DDoS amplifiers by never responding to a request with a packet larger than the request packet.

- o Scalability: Server implementations may serve large numbers of clients without having to retain any client-specific state.

1.2. Protocol overview

The Network Time Protocol includes many different operating modes to support various network topologies. In addition to its best-known and most-widely-used client-server mode, it also includes modes for synchronization between symmetric peers, a control mode for server monitoring and administration and a broadcast mode. These various modes have differing and partly contradictory requirements for security and performance. Symmetric and control modes demand mutual authentication and mutual replay protection, and for certain message types control mode may require confidentiality as well as authentication. Client-server mode places more stringent requirements on resource utilization than other modes, because servers may have vast number of clients and be unable to afford to maintain per-client state. However, client-server mode also has more relaxed security needs, because only the client requires replay protection: it is harmless for stateless servers to process replayed packets. The security demands of symmetric and control modes, on the other hand, are in conflict with the resource-utilization demands of client-server mode: any scheme which provides replay protection inherently involves maintaining some state to keep track of what messages have already been seen.

This memo specifies NTS exclusively for the client-server mode of NTP. To this end, NTS is structured as a suite of two protocols:

The "NTS Extension Fields for NTPv4" are a collection of NTP extension fields for cryptographically securing NTPv4 using previously-established key material. They are suitable for securing client-server mode because the server can implement them without retaining per-client state, but on the other hand are suitable *only* for client-server mode because only the client, and not the server, is protected from replay.

The "NTS Key Establishment" protocol (NTS-KE) is a mechanism for establishing key material for use with the NTS Extension Fields for NTPv4. It uses TLS to exchange keys and negotiate some additional protocol options, but then quickly closes the TLS channel and permits the server to discard all associated state.

The typical protocol flow is as follows. The client connects to the server on the NTS TCP port and the two parties perform a TLS handshake. Via the TLS channel, the parties negotiate some additional protocol parameters and the server sends the client a supply of cookies. The parties use TLS key export [RFC5705] to extract key material which will be used in the next phase of the protocol. This negotiation takes only a single round trip, after which the server closes the connection and discards all associated state. At this point the NTS-KE phase of the protocol is complete.

Time synchronization proceeds over the NTP UDP port. The client sends the server an NTP client packet which includes several extension fields. Included among these fields are a cookie (previously provided by the server), and an authentication tag, computed using key material extracted from the NTS-KE handshake. The server uses the cookie to recover this key material (previously discarded to avoid maintaining state) and sends back an authenticated response. The response includes a fresh, encrypted cookie which the client then sends back in the clear with its next request. (This constant refreshing of cookies is necessary in order to achieve NTS's unlinkability goal.)

2. 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].

3. TLS profile for Network Time Security

Network Time Security makes use of TLS for NTS key establishment.

Since securing time protocols is (as of 2017) a novel application of TLS, no backward-compatibility concerns exist to justify using obsolete, insecure, or otherwise broken TLS features or versions. We therefore put forward the following requirements and guidelines, roughly representing 2017's best practices.

Implementations MUST NOT negotiate TLS versions earlier than 1.2.

Implementations willing to negotiate more than one possible version of TLS SHOULD NOT respond to handshake failures by retrying with a downgraded protocol version. If they do, they MUST implement [RFC7507].

TLS clients MUST NOT offer, and TLS servers MUST NOT select, RC4 cipher suites. [RFC7465]

TLS 1.2 clients SHOULD offer, and TLS servers SHOULD accept, the TLS Renegotiation Indication Extension [RFC5746]. Regardless, they MUST NOT initiate or permit insecure renegotiation.

TLS 1.2 clients SHOULD offer, and TLS 1.2 servers SHOULD accept, the TLS Session Hash and Extended Master Secret Extension [RFC7627].

Use of the Application-Layer Protocol Negotiation Extension [RFC7301] is integral to NTS and support for it is REQUIRED for interoperability.

4. The NTS Key Establishment protocol

The NTS key establishment protocol is conducted via TCP port [[TBD1]]. The two endpoints carry out a TLS handshake in conformance with Section 3, with the client offering (via an ALPN [RFC7301] extension), and the server accepting, an application-layer protocol of "ntske/1". Immediately following a successful handshake, the client SHALL send a single request (as Application Data encapsulated in the TLS-protected channel), then the server SHALL send a single response followed by a TLS "Close notify" alert and then discard the channel state.

The client's request and the server's response each SHALL consist of a sequence of records formatted according to Figure 1. The sequence SHALL be terminated by a "End of Message" record, which has a Record Type of zero and a zero-length body. Furthermore, requests and non-error responses each SHALL include exactly one NTS Next Protocol Negotiation record.

Clients and servers MAY enforce length limits on requests and responses, however servers MUST accept requests of at least 1024 octets, and clients SHOULD accept responses of at least 65536 octets.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|C|         Record Type         |          Body Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                           Record Body                         .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1

The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting.

The fields of an NTS-KE record are defined as follows:

C (Critical Bit): Determines the disposition of unrecognized Record Types. Implementations which receive a record with an unrecognized Record Type MUST ignore the record if the Critical Bit is 0, and MUST treat it as an error if the Critical Bit is 1.

Record Type: A 15-bit integer in network byte order. The semantics of record types 0-5 are specified in this memo; additional type numbers SHALL be tracked through the IANA Network Time Security Key Establishment Record Types registry.

Body Length: The length of the Record Body field, in octets, as a 16-bit integer in network byte order. Record bodies MAY have any representable length and need not be aligned to a word boundary.

Record Body: The syntax and semantics of this field SHALL be determined by the Record Type.

For clarity regarding bit-endianness: the Critical Bit is the most-significant bit of the first octet. In C, given a network buffer \`unsigned char b[]\` containing an NTS-KE record, the critical bit is \`b[0] >> 7\` while the record type is \`((b[0] & 0x7f) << 8) + b[1]\`.

Figure 2 provides a schematic overview of the key exchange. It displays the protocol steps to be performed by the NTS client and server and record types to be exchanged.

```
                   +---------------------------------------+
                   | - verify client request message       |
                   | - extract TLS key material            |
                   | - generate KE response message        |
                   |   - included Record Types:            |
                   |       - NTS Next Protocol Negotiation |
                   |       - AEAD Alg. Negotiation         |
                   |       - New Cookie for NTPv4          |
                   |       - <New Cookie for NTPv4>        |
                   |       - End of Message                |
                   +-----------------+---------------------+
                                     |
                                     |
   Server -----------+---------------+-----+----------------------->
                     ^                      \
                    /                        \
                   /    TLS application       \
                  /     data                   \
                 /                              \
                /                                V
   Client -----+---------------------------------+---------------->
               |                                 |
               |                                 |
               |                                 |
   +-----------+----------------------+   +------+-----------------+
   |- generate KE request message     |   |- verify server response|
   | - include Record Types:          |   |  message               |
   |  o NTS Next Protocol Negotiation |   |- extract cookie(s)     |
   |  o AEAD Alg. Negotiation         |   |                        |
   |  o End of Message                |   |                        |
   +----------------------------------+   +------------------------+
```

Figure 2: NTS Key Exchange messages

4.1. NTS-KE Record Types

The following NTS-KE Record Types are defined.

4.1.1. End of Message

The End of Message record has a Record Type number of 0 and an zero-length body. It MUST occur exactly once as the final record of every NTS-KE request and response. The Critical Bit MUST be set.

4.1.2. NTS Next Protocol Negotiation

The NTS Next Protocol Negotiation record has a Record Type of 1. It MUST occur exactly once in every NTS-KE request and response. Its body consists of a sequence of 16-bit unsigned integers in network byte order. Each integer represents a Protocol ID from the IANA Network Time Security Next Protocols registry. The Critical Bit MUST be set.

The Protocol IDs listed in the client's NTS Next Protocol Negotiation record denote those protocols which the client wishes to speak using the key material established through this NTS-KE session. The Protocol IDs listed in the server's response MUST comprise a subset of those listed in the request, and denote those protocols which the server is willing and able to speak using the key material established through this NTS-KE session. The client MAY proceed with one or more of them. The request MUST list at least one protocol, but the response MAY be empty.

4.1.3. Error

The Error record has a Record Type number of 2. Its body is exactly two octets long, consisting of an unsigned 16-bit integer in network byte order, denoting an error code. The Critical Bit MUST be set.

Clients MUST NOT include Error records in their request. If clients receive a server response which includes an Error record, they MUST discard any negotiated key material and MUST NOT proceed to the Next Protocol.

The following error codes are defined.

Error code 0 means "Unrecognized Critical Record". The server MUST respond with this error code if the request included a record which the server did not understand and which had its Critical Bit set. The client SHOULD NOT retry its request without modification.

Error code 1 means "Bad Request". The server MUST respond with this error if, upon the expiration of an implementation-defined timeout, it has not yet received a complete and syntactically well-formed request from the client. This error is likely to be the result of a dropped packet, so the client SHOULD start over with a new TLS handshake and retry its request.

4.1.4. Warning

The Warning record has a Record Type number of 3. Its body is exactly two octets long, consisting of an unsigned 16-bit integer in network byte order, denoting a warning code. The Critical Bit MUST be set.

Clients MUST NOT include Warning records in their request. If clients receive a server response which includes a Warning record, they MAY discard any negotiated key material and abort without proceeding to the Next Protocol. Unrecognized warning codes MUST be treated as errors.

This memo defines no warning codes.

4.1.5. AEAD Algorithm Negotiation

The AEAD Algorithm Negotiation record has a Record Type number of 4. Its body consists of a sequence of unsigned 16-bit integers in network byte order, denoting Numeric Identifiers from the IANA AEAD registry [RFC5116]. The Critical Bit MAY be set.

If the NTS Next Protocol Negotiation record offers Protocol ID 0 (for NTPv4), then this record MUST be included exactly once. Other protocols MAY require it as well.

When included in a request, this record denotes which AEAD algorithms the client is willing to use to secure the Next Protocol, in decreasing preference order. When included in a response, this record denotes which algorithm the server chooses to use, or is empty if the server supports none of the algorithms offered. In requests, the list MUST include at least one algorithm. In responses, it MUST include at most one. Honoring the client's preference order is OPTIONAL: servers may select among any of the client's offered choices, even if they are able to support some other algorithm which the client prefers more.

Server implementations of NTS extension fields for NTPv4 (Section 5) MUST support AEAD_AES_SIV_CMAC_256 [RFC5297] (Numeric Identifier 15). That is, if the client includes AEAD_AES_SIV_CMAC_256 in its AEAD Algorithm Negotiation record, and the server accepts Protocol ID 0 (NTPv4) in its NTS Next Protocol Negotiation record, then the server's AEAD Algorithm Negotiation record MUST NOT be empty.

4.1.6. New Cookie for NTPv4

The New Cookie for NTPv4 record has a Record Type number of 5. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 7 for a suggested construction.

Clients MUST NOT send records of this type. Servers MUST send at least one record of this type, and SHOULD send eight of them, if they accept Protocol ID 0 (NTPv4) as a Next Protocol. The Critical Bit SHOULD NOT be set.

4.2. Key Extraction (generally)

Following a successful run of the NTS-KE protocol, key material SHALL be extracted according to RFC 5705 [RFC5705]. Inputs to the exporter function are to be constructed in a manner specific to the negotiated Next Protocol. However, all protocols which utilize NTS-KE MUST conform to the following two rules:

The disambiguating label string MUST be "EXPORTER-network-time-security/1".

The per-association context value MUST be provided, and MUST begin with the two-octet Protocol ID which was negotiated as a Next Protocol.

5. NTS Extension Fields for NTPv4

5.1. Key Extraction (for NTPv4)

Following a successful run of the NTS-KE protocol wherein Protocol ID 0 (NTPv4) is selected as a Next Protocol, two AEAD keys SHALL be extracted: a client-to-server (C2S) key and a server-to-client (S2C) key. These keys SHALL be computed according to RFC 5705 [RFC5705], using the following inputs.

The disambiguating label string SHALL be "EXPORTER-network-time-security/1".

The per-association context value SHALL consist of the following five octets:

The first two octets SHALL be zero (the Protocol ID for NTPv4).

The next two octets SHALL be the Numeric Identifier of the negotiated AEAD Algorithm, in network byte order.

The final octet SHALL be 0x00 for the C2S key and 0x01 for the S2C key.

Implementations wishing to derive additional keys for private or experimental use MUST NOT do so by extending the above-specified syntax for per-association context values. Instead, they SHOULD use their own disambiguating label string. Note that RFC 5705 provides that disambiguating label strings beginning with "EXPERIMENTAL" MAY be used without IANA registration.

5.2. Packet structure overview

In general, an NTS-protected NTPv4 packet consists of:

The usual 48-octet NTP header, which is authenticated but not encrypted.

Some extension fields which are authenticated but not encrypted.

An extension field which contains AEAD output (i.e., an authentication tag and possible ciphertext). The corresponding plaintext, if non-empty, consists of some extension fields which benefit from both encryption and authentication.

Possibly, some additional extension fields which are neither encrypted nor authenticated. These are discarded by the receiver.

Always included among the authenticated or authenticated-and-encrypted extension fields are a cookie extension field and a unique-identifier extension field. The purpose of the cookie extension field is to enable the server to offload storage of session state onto the client. The purpose of the unique-identifier extension field is to protect the client from replay attacks.

5.3. The Unique Identifier extension field

The Unique Identifier extension field has a Field Type of [[TBD2]]. When the extension field is included in a client packet (mode 3), its body SHALL consist of a string of octets generated uniformly at random. The string MUST be at least 32 octets long. When the extension field is included in a server packet (mode 4), its body SHALL contain the same octet string as was provided in the client packet to which the server is responding. Its use in modes other than client-server is not defined.

The Unique Identifier extension field provides the client with a cryptographically strong means of detecting replayed packets. It MAY also be used standalone, without NTS, in which case it provides the

client with a means of detecting spoofed packets from off-path attackers. Historically, NTP's origin timestamp field has played both these roles, but for cryptographic purposes this is suboptimal because it is only 64 bits long and, depending on implementation details, most of those bits may be predictable. In contrast, the Unique Identifier extension field enables a degree of unpredictability and collision-resistance more consistent with cryptographic best practice.

5.4. The NTS Cookie extension field

The NTS Cookie extension field has a Field Type of [[TBD3]]. Its purpose is to carry information which enables the server to recompute keys and other session state without having to store any per-client state. The contents of its body SHALL be implementation-defined and clients MUST NOT attempt to interpret them. See Section 7 for a suggested construction. The NTS Cookie extension field MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

5.5. The NTS Cookie Placeholder extension field

The NTS Cookie Placeholder extension field has a Field Type of [[TBD4]]. When this extension field is included in a client packet (mode 3), it communicates to the server that the client wishes it to send additional cookies in its response. This extension field MUST NOT be included in NTP packets whose mode is other than 3.

Whenever an NTS Cookie Placeholder extension field is present, it MUST be accompanied by an NTS Cookie extension field, and the body length of the NTS Cookie Placeholder extension field MUST be the same as the body length of the NTS Cookie extension field. (This length requirement serves to ensure that the response will not be larger than the request, in order to improve timekeeping precision and prevent DDoS amplification). The contents of the NTS Cookie Placeholder extension field's body are undefined and, aside from checking its length, MUST be ignored by the server.

5.6. The NTS Authenticator and Encrypted Extension Fields extension field

The NTS Authenticator and Encrypted Extension Fields extension field is the central cryptographic element of an NTS-protected NTP packet. Its Field Type is [[TBD5]] and the format of its body SHALL be as follows:

Nonce length: Two octets in network byte order, giving the length of the Nonce field and interpreted as an unsigned integer.

Nonce: A nonce as required by the negotiated AEAD Algorithm.

Ciphertext: The output of the negotiated AEAD Algorithm. The structure of this field is determined by the negotiated algorithm, but it typically contains an authentication tag in addition to the actual ciphertext.

Padding: several octets of padding, with every octet set to the number of padding octets included, e.g., "01", "02 02", or "03 03 03". Constraints on the number of padding octets included are enumerated below.

The Ciphertext field SHALL be formed by providing the following inputs to the negotiated AEAD Algorithm:

K: For packets sent from the client to the server, the C2S key SHALL be used. For packets sent from the server to the client, the S2C key SHALL be used.

A: The associated data SHALL consist of the portion of the NTP packet beginning from the start of the NTP header and ending at the end of the last extension field which precedes the NTS Authenticator and Encrypted Extension Fields extension field.

P: The plaintext SHALL consist of all (if any) NTP extension fields to be encrypted. The format of any such fields SHALL be in accordance with RFC 7822 [RFC7822], and if multiple extension fields are present they SHALL be joined by concatenation.

N: The nonce SHALL be formed however required by the negotiated AEAD Algorithm.

The number of padding octets included SHALL conform to the following constraints:

The number MUST be at least 1, so that the final octet of the extension field always gives the padding length.

The number MUST NOT be greater than 255, since high numbers are unrepresentable in a single octet

The number MUST result in an extension field length which is legal per [RFC7822]. That is, the number of padding octets must be chosen so that the total length of the extension field (including the Field Type and Length subfields) is a multiple of 4 greater than or equal to 16, and greater than or equal to 28 if the extension field is the last one in the packet.

For mode 3 (client) packets only, the number MUST be at least MAX(MIN(N_MAX, 16) - N_len, 0) + 4, where `N_len` represents the actual length of the nonce and N_MAX is, per [RFC5116], the maximum permitted nonce length for the AEAD algorithm in use. This constraint ensures that servers can always use an adequately long nonce without causing the size of their response packet to exceed the size of the request packet. Servers SHOULD enforce this constraint by dropping client packets that do not conform to it. Clients MUST NOT enforce it since it is not binding on mode 4 (server) packets to begin with.

The NTS Authenticator and Encrypted Extension Fields extension field MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

6. **Protocol details**

A client sending an NTS-protected request SHALL include the following extension fields as displayed in Figure 3:

Exactly one Unique Identifier extension field, which MUST be authenticated, MUST NOT be encrypted, and whose contents MUST NOT duplicate those of any previous request.

Exactly one NTS Cookie extension field, which MUST be authenticated and MUST NOT be encrypted. The cookie MUST be one which the server previously provided the client; it may have been provided during the NTS-KE handshake or in response to a previous NTS-protected NTP request. To protect client's privacy, the same cookie SHOULD NOT be included in multiple requests. If the client does not have any cookies that it has not already sent, it SHOULD re-run the NTS-KE protocol before continuing.

Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using an AEAD Algorithm and C2S key established through NTS-KE.

The client MAY include one or more NTS Cookie Placeholder extension field, which MUST be authenticated and MAY be encrypted. The number of NTS Cookie Placeholder extension fields that the client includes SHOULD be such that if the client includes N placeholders and the server sends back N+1 cookies, the number of unused cookies stored by the client will come to eight. When both the client and server adhere to all cookie-management guidance provided in this memo, the number of placeholder extension fields will equal the number of dropped packets since the last successful volley.

The client MAY include additional (non-NTS-related) extension fields, which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension fields (therefore authenticated but not encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). In general, however, the server MUST discard any unauthenticated extension fields and process the packet as though they were not present. Servers MAY implement exceptions to this requirement for particular extension fields if their specification explicitly provides for such.

Upon receiving an NTS-protected request, the server SHALL (through some implementation-defined mechanism) use the cookie to recover the AEAD Algorithm, C2S key, and S2C key associated with the request, and then use the C2S key to authenticate the packet and decrypt the ciphertext. If the cookie is valid and authentication and decryption succeed, then the server SHALL include the following extension fields in its response:

> Exactly one Unique Identifier extension field, which MUST be authenticated, MUST NOT be encrypted, and whose contents SHALL echo those provided by the client.
>
> Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using the AEAD algorithm and S2C key recovered from the cookie provided by the client.
>
> One or more NTS Cookie extension fields, which MUST be encrypted and authenticated. The number of NTS Cookie extension fields included SHOULD be equal to, and MUST NOT exceed, one plus the number of valid NTS Cookie Placeholder extension fields included in the request.

We emphasize the contrast that NTS Cookie extension fields MUST NOT be encrypted when sent from client to server, but MUST be encrypted from sent from server to client. The former is necessary in order for the server to be able to recover the C2S and S2C keys, while the latter is necessary to satisfy the unlinkability goals discussed in Section 11.1. We emphasize also that " encrypted" means encapsulated within the the NTS Authenticator and Encrypted Extensions extension field. While the body of a NTS Cookie extension field will generally consist of some sort of AEAD output (regardless of whether the recommendations of Section 7 are precisely followed), this is not sufficient to make the extension field "encrypted".

The server MAY include additional (non-NTS-related) extension fields, which MAY appear prior to the NTS Authenticator and Encrypted Extension Fields extension field (therefore authenticated but not

encrypted), within it (therefore encrypted and authenticated), or after it (therefore neither encrypted nor authenticated). In general, however, the client MUST discard any unauthenticated extension fields and process the packet as though they were not present. Clients MAY implement exceptions to this requirement for particular extension fields if their specification explicitly provides for such.

If the server is unable to validate the cookie or authenticate the request, it SHOULD respond with a Kiss-o'-Death packet (see RFC 5905, Section 7.4) [RFC5905]) with kiss code "NTSN" (meaning "NTS NAK"). Such a response MUST include exactly one Unique Identifier extension field whose contents SHALL echo those provided by the client. It MUST NOT include any NTS Cookie or NTS Authenticator and Encrypted Extension Fields extension fields.

Upon receiving an NTS-protected response, the client MUST verify that the Unique Identifier matches that of an outstanding request, and that the packet is authentic under the S2C key associated with that request. If either of these checks fails, the packet MUST be discarded without further processing.

Upon receiving an NTS NAK, the client MUST verify that the Unique Identifier matches that of an outstanding request. If this check fails, the packet MUST be discarded without further processing. If this check passes, the client SHOULD wait until the next poll for a valid NTS-protected response and if none is received, discard all cookies and AEAD keys associated with the server which sent the NAK and initiate a fresh NTS-KE handshake.

```
                   +-----------------+---------------------+
                   | - verify time request message         |
                   | - generate time response message      |
                   |   - included NTPv4 extension fields   |
                   |      o Unique Identifier EF           |
                   |      o NTS Authentication and         |
                   |        Encrypted Extension Fields EF  |
                   |        - NTS Cookie EF                |
                   |        - <NTS Cookie EF>              |
                   | - transmit time request packet        |
                   +-----------------+---------------------+
                                     |
                                     |
   Server -------- --+---------------+-----+----------------------->
                     ^                      \
                    /                        \
     time request  /                          \   time response
     (mode 3)     /                            \  (mode 4)
                 /                              \
                /                                V
   Client -----+---------------------------------+---------------->
               |                                 |
               |                                 |
               |                                 |
   +-----------+----------------------+   +------+-----------------+
   |- generate time request message   |   |- verify time response  |
   | - include NTPv4 Extension fields |   |  message               |
   |    o Unique Identifier EF        |   |- extract cookie(s)     |
   |    o NTS Cookie EF               |   |- time synchronization  |
   |    o <NTS Cookie Placeholder EF> |   |  processing            |
   |                                  |   +------------------------+
   |- generate AEAD tag of NTP message|
   |- add NTS Authentication and      |
   |  Encrypted Extension Fields EF   |
   |- transmit time request packet    |
   +----------------------------------+
```

Figure 3: NTS Time Synchronization Message

7. Suggested format for NTS cookies

This section is non-normative. It gives a suggested way for servers to construct NTS cookies. All normative requirements are stated in Section 4.1.6 and Section 5.4.

The role of cookies in NTS is closely analogous to that of session cookies in TLS. Accordingly, the thematic resemblance of this

section to RFC 5077 [RFC5077] is deliberate, and the reader should likewise take heed of its security considerations.

Servers should select an AEAD algorithm which they will use to encrypt and authenticate cookies. The chosen algorithm should be one such as AEAD_AES_SIV_CMAC_256 [RFC5297] which resists accidental nonce reuse, and it need not be the same as the one that was negotiated with the client. Servers should randomly generate and store a master AEAD key `K`. Servers should additionally choose a non-secret, unique value `I` as key-identifier for `K`.

Servers should periodically (e.g., once daily) generate a new pair (I,K) and immediately switch to using these values for all newly-generated cookies. Immediately following each such key rotation, servers should securely erase any keys generated two or more rotation periods prior. Servers should continue to accept any cookie generated using keys that they have not yet erased, even if those keys are no longer current. Erasing old keys provides for forward secrecy, limiting the scope of what old information can be stolen if a master key is somehow compromised. Holding on to a limited number of old keys allows clients to seamlessly transition from one generation to the next without having to perform a new NTS-KE handshake.

The need to keep keys synchronized across load-balanced clusters can make automatic key rotation challenging. However, the task can be accomplished without the need for central key-management infrastructure by using a ratchet, i.e., making each new key a deterministic, cryptographically pseudo-random function of its predecessor. A recommended concrete implementation of this approach is to use HKDF [RFC5869] to derive new keys, using the key's predecessor as Input Keying Material and its key identifier as a salt.

To form a cookie, servers should first form a plaintext `P` consisting of the following fields:

The AEAD algorithm negotiated during NTS-KE

The S2C key

The C2S key

Servers should then generate a nonce `N` uniformly at random, and form AEAD output `C` by encrypting `P` under key `K` with nonce `N` and no associated data.

The cookie should consist of the tuple `(I,N,C)`.

To verify and decrypt a cookie provided by the client, first parse it into its components `I`, `N`, and `C`. Use `I` to look up its decryption key `K`. If the key whose identifier is `I` has been erased or never existed, decryption fails; reply with an NTS NAK. Otherwise, attempt to decrypt and verify ciphertext `C` using key `K` and nonce `N` with no associated data. If decryption or verification fails, reply with an NTS NAK. Otherwise, parse out the contents of the resulting plaintext `P` to obtain the negotiated AEAD algorithm, S2C key, and C2S key.

8. IANA Considerations

IANA is requested to allocate two entries, identical except for the Transport Protocol, in the Service Name and Transport Protocol Port Number Registry as follows:

Service Name: nts

Transport Protocol: tcp, udp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Network Time Security

Reference: [[this memo]]

Port Number: [[TBD1]], selected by IANA from the user port range

IANA is requested to allocate the following entry in the Application-Layer Protocol Negotation (ALPN) Protocol IDs registry:

Protocol: Network Time Security Key Establishment, version 1

Identification Sequence:
0x6E 0x74 0x73 0x6B 0x65 0x2F 0x31 ("ntske/1")

Reference: [[this memo]]

IANA is requested to allocate the following entry in the TLS Exporter Label Registry:

Value	DTLS-OK	Reference	Note
EXPORTER-network-time-security/1	Y	[[this memo]]	

IANA is requested to allocate the following entry in the registry of NTP Kiss-o'-Death codes:

Code	Meaning
NTSN	NTS NAK

IANA is requested to allocate the following entries in the NTP Extensions Field Types registry:

Field Type	Meaning	Reference
[[TBD2]]	Unique Identifier	[[this memo]]
[[TBD3]]	NTS Cookie	[[this memo]]
[[TBD4]]	NTS Cookie Placeholder	[[this memo]]
[[TBD5]]	NTS Authenticator and Encrypted Extension Fields	[[this memo]]

IANA is requested to create a new registry entitled "Network Time Security Key Establishment Record Types". Entries SHALL have the following fields:

Type Number (REQUIRED): An integer in the range 0-32767 inclusive.

Description (REQUIRED): A short text description of the purpose of the field.

Set Critical Bit (REQUIRED): One of "MUST", "SHOULD", "MAY", "SHOULD NOT", or "MUST NOT".

Reference (REQUIRED): A reference to a document specifying the semantics of the record.

The policy for allocation of new entries in this registry SHALL vary by the Type Number, as follows:

0-1023: IETF Review

1024-16383: Specification Required

16384-32767: Private and Experimental Use

Applications for new entries SHALL specify the contents of the Description, Set Critical Bit and Reference fields and which of the above ranges the Type Number should be allocated from. Applicants MAY request a specific Type Number, and such requests MAY be granted at the registrar's discretion.

The initial contents of this registry SHALL be as follows:

Field Number	Description	Critical	Reference
0	End of message	MUST	[[this memo]]
1	NTS next protocol negotiation	MUST	[[this memo]]
2	Error	MUST	[[this memo]]
3	Warning	MUST	[[this memo]]
4	AEAD algorithm negotiation	MAY	[[this memo]]
5	New cookie for NTPv4	SHOULD NOT	[[this memo]]
16384-32767	Reserved for Private & Experimental Use	MAY	[[this memo]]

IANA is requested to create a new registry entitled "Network Time Security Next Protocols". Entries SHALL have the following fields:

Protocol ID (REQUIRED): An integer in the range 0-65535 inclusive, functioning as an identifier.

Protocol Name (REQUIRED): A short text string naming the protocol being identified.

Reference (RECOMMENDED): A reference to a relevant specification document. If no relevant document exists, a point-of-contact for questions regarding the entry SHOULD be listed here in lieu.

Applications for new entries in this registry SHALL specify all desired fields, and SHALL be granted upon approval by a Designated Expert. Protocol IDs 32768-65535 SHALL be reserved for Private or Experimental Use, and SHALL NOT be registered.

The initial contents of this registry SHALL be as follows:

Protocol ID	Human-Readable Name	Reference
0	Network Time Protocol version 4 (NTPv4)	[[this memo]]
32768-65535	Reserved for Private or Experimental Use	Reserved by [[this memo]]

IANA is requested to create two new registries entitled "Network Time Security Error Codes" and "Network Time Security Warning Codes". Entries in each SHALL have the following fields:

Number (REQUIRED): An integer in the range 0-65535 inclusive.

Description (REQUIRED): A short text description of the condition.

Reference (REQUIRED): A reference to a relevant specification document.

The policy for allocation of new entries in these registries SHALL vary by their Number, as follows:

0-1023: IETF Review

1024-32767: Specification Required

32768-65535: Private and Experimental Use

The initial contents of the Network Time Security Error Codes Registry SHALL be as follows:

Number	Description	Reference
0	Unrecognized Critical Extension	[[this memo]]
1	Bad Request	[[this memo]]

The Network Time Security Warning Codes Registry SHALL initially be empty.

9. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. Implementation PoC 1

Organization: Ostfalia University of Applied Science

Implementor: Martin Langer

Maturity: Proof-of-Concept Prototype

This implementation was used to verify consistency and to ensure completeness of this specification. It also demonstrate interoperability with NTP's client-server mode messages.

9.1.1. Coverage

This implementation covers the complete specification.

9.1.2. Licensing

The code is released under a Apache License 2.0 license.

The source code is available at: https://gitlab.com/MLanger/nts/

9.1.3. Contact Information

Contact Martin Langer: mart.langer@ostfalia.de

9.1.4. Last Update

The implementation was updated 3rd May 2018.

9.2. Implementation PoC 2

Organization: tbd

Implementor: Daniel Fox Franke

Maturity: Proof-of-Concept Prototype

This implementation was used to verify consistency and to ensure completeness of this specification.

9.2.1. Coverage

This implementation provides the client and the server for the initial TLS handshake and NTS key exchange. It provides the the client part of the NTS protected NTP messages.

9.2.2. Licensing

Public domain.

The source code is available at: https://github.com/dfoxfranke/nts-hackathon

9.2.3. Contact Information

Contact Daniel Fox Franke: dfoxfranke@gmail.com

9.2.4. Last Update

The implementation was updated 16th March 2018.

9.3. Interoperability

The Interoperability tests distinguished between NTS key exchange and NTS time exchange messages. For the NTS key exchange, interoperability between the two implementations has been verified successfully. Interoperability of NTS time exchange messages has been verified successfully for the case that PoC 1 represents the server and PoC 2 the client.

These tests successfully demonstrate that there are at least two running implementations of this draft which are able to interoperate.

10. Security considerations

10.1. Avoiding DDoS amplification

Certain non-standard and/or deprecated features of the Network Time Protocol enable clients to send a request to a server which causes the server to send a response much larger than the request. Servers which enable these features can be abused in order to amplify traffic volume in distributed denial-of-service (DDoS) attacks by sending them a request with a spoofed source IP. In recent years, attacks of this nature have become an endemic nuisance.

NTS is designed to avoid contributing any further to this problem by ensuring that NTS-related extension fields included in server responses will be the same size as the NTS-related extension fields sent by the client. In particular, this is why the client is required to send a separate and appropriately padded-out NTS Cookie Placeholder extension field for every cookie it wants to get back, rather than being permitted simply to specify a desired quantity.

Due to the [RFC7822] requirement that extensions be padded and aligned to four-octet boundaries, response size may still in some cases exceed request size by up to three octets. This is sufficiently inconsequential that we have declined to address it.

10.2. Initial verification of server certificates

NTS's security goals are undermined if the client fails to verify that the X.509 certificate chain presented by the server is valid and rooted in a trusted certificate authority. [RFC5280] and [RFC6125] specify how such verification is to be performed in general. However, the expectation that the client does not yet have a correctly-set system clock at the time of certificate verification presents difficulties with verifying that the certificate is within its validity period, i.e., that the current time lies between the times specified in the certificate's notBefore and notAfter fields, and it may be operationally necessary in some cases for a client to accept a certificate which appears to be expired or not yet valid. While there is no perfect solution to this problem, there are several mitigations the client can implement to make it more difficult for an adversary to successfully present an expired certificate:

Check whether the system time is in fact unreliable. If the system clock has previously been synchronized since last boot, then on operating systems which implement a kernel-based phase-

locked-loop API, a call to ntp_gettime() should show a maximum error less than NTP_PHASE_MAX. In this case, the clock SHOULD be considered reliable and certificates can be strictly validated.

Allow the system administrator to specify that certificates should *always* be strictly validated. Such a configuration is appropriate on systems which have a battery-backed clock and which can reasonably prompt the user to manually set an approximately-correct time if it appears to be needed.

Once the clock has been synchronized, periodically write the current system time to persistent storage. Do not accept any certificate whose notAfter field is earlier than the last recorded time.

Do not process time packets from servers if the time computed from them falls outside the validity period of the server's certificate.

Use multiple time sources. The ability to pass off an expired certificate is only useful to an adversary who has compromised the corresponding private key. If the adversary has compromised only a minority of servers, NTP's selection algorithm ([RFC5905] section 11.2.1) will protect the client from accepting bad time from the adversary-controlled servers.

10.3. Usage of NTP pools

Additional standardization work and infrastructure development is necessary before NTS can be used with public NTP server pools. First, a scheme will need to be specified for determining what constitutes an acceptable certificate for a pool server, such as establishing a value required to be contained in its Extended Key Usage attribute, and how to determine, given the DNS name of a pool, what Subject Alternative Name to expect in the certificates of its members. Implementing any such specification will necessitate infrastructure work: pool organizers will need to act as certificate authorities, regularly monitor the behavior of servers to which certificates have been issued, and promptly revoke the certificate of any server found to be serving incorrect time.

10.4. Delay attacks

In a packet delay attack, an adversary with the ability to act as a man-in-the-middle delays time synchronization packets between client and server asymmetrically [RFC7384]. Since NTP's formula for computing time offset relies on the assumption that network latency is roughly symmetrical, this leads to the client to compute an

inaccurate value [Mizrahi]. The delay attack does not reorder or modify the content of the exchanged synchronization packets. Therefore, cryptographic means do not provide a feasible way to mitigate this attack. However, the maximum error that an adversary can introduce is bounded by half of the round trip delay.

[RFC5905] specifies a parameter called MAXDIST which denotes the maximum round-trip latency (including not only the immediate round trip between client and server but the whole distance back to the reference clock as reported in the Root Delay field) that a client will tolerate before concluding that the server is unsuitable for synchronization. The standard value for MAXDIST is one second, although some implementations use larger values. Whatever value a client chooses, the maximum error which can be introduced by a delay attack is MAXDIST/2.

Usage of multiple time sources, or multiple network paths to a given time source [Shpiner], may also serve to mitigate delay attacks if the adversary is in control of only some of the paths.

10.5. Random number generation

At various points in NTS, the generation of cryptographically secure random numbers is required. Whenever this draft specifies the use of random numbers, then cryptographically secure random number generation MUST be used. See [RFC4086] for guidelines concerning this topic.

11. Privacy Considerations

11.1. Unlinkability

Unlinkability prevents a device from being tracked when it changes network addresses (e.g. because said device moved between different networks). In other words, unlinkability thwarts an attacker that seeks to link a new network address used by a device with a network address that it was formerly using, because of recognizable data that the device persistently sends as part of an NTS-secured NTP association. This is the justification for continually supplying the client with fresh cookies, so that a cookie never represents recognizable data in the sense outlined above.

NTS's unlinkability objective is merely to not leak any additional data that could be used to link a device's network address. NTS does not rectify legacy linkability issues that are already present in NTP. Thus, a client that requires unlinkability must also minimize information transmitted in a client query (mode 3) packet as described in the draft [I-D.ietf-ntp-data-minimization].

The unlinkability objective only holds for time synchronization traffic, as opposed to key exchange traffic. This implies that it cannot be guaranteed for devices that function not only as time clients, but also as time servers (because the latter can be externally triggered to send authentication data).

It should also be noted that it could be possible to link devices that operate as time servers from their time synchronization traffic, using information exposed in (mode 4) server response packets (e.g. reference ID, reference time, stratum, poll). Also, devices that respond to NTP control queries could be linked using the information revealed by control queries.

11.2. Confidentiality

NTS does not protect the confidentiality of information in NTP's header fields. When clients implement [I-D.ietf-ntp-data-minimization], client packet headers do not contain any information which the client could conceivably wish to keep secret: one field is random, and all others are fixed. Information in server packet headers is likewise public: the origin timestamp is copied from the client's (random) transmit timestamp, and all other fields are set the same regardless of the identity of the client making the request.

Future extension fields could hypothetically contain sensitive information, in which case NTS provides a mechanism for encrypting them.

12. Acknowledgements

The authors would like to thank Richard Barnes, Steven Bellovin, Scott Fluhrer, Sharon Goldberg, Russ Housley, Martin Langer, Miroslav Lichvar, Aanchal Malhotra, Dave Mills, Danny Mayer, Karen O'Donoghue, Eric K. Rescorla, Stephen Roettger, Kurt Roeckx, Kyle Rose, Rich Salz, Brian Sniffen, Susan Sons, Douglas Stebila, Harlan Stenn, Martin Thomson, and Richard Welty for contributions to this document and comments on the design of NTS.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, DOI 10.17487/RFC5297, October 2008, <https://www.rfc-editor.org/info/rfc5297>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <https://www.rfc-editor.org/info/rfc5746>.

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

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[RFC7465] Popov, A., "Prohibiting RC4 Cipher Suites", RFC 7465, DOI 10.17487/RFC7465, February 2015, <https://www.rfc-editor.org/info/rfc7465>.

[RFC7507] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", RFC 7507, DOI 10.17487/RFC7507, April 2015, <https://www.rfc-editor.org/info/rfc7507>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <https://www.rfc-editor.org/info/rfc7627>.

[RFC7822] Mizrahi, T. and D. Mayer, "Network Time Protocol Version 4 (NTPv4) Extension Fields", RFC 7822, DOI 10.17487/RFC7822, March 2016, <https://www.rfc-editor.org/info/rfc7822>.

13.2. Informative References

[I-D.ietf-ntp-data-minimization] Franke, D. and A. Malhotra, "NTP Client Data Minimization", draft-ietf-ntp-data-minimization-00 (work in progress), May 2017.

[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, <https://www.rfc-editor.org/info/rfc4086>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/info/rfc5077>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[Shpiner] "Multi-path Time Protocols", in Proceedings of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS), September 2013.

Appendix A. Terms and Abbreviations

AEAD	Authenticated Encryption with Associated Data [RFC5116]
DDoS	Distributed Denial of Service
NTP	Network Time Protocol [RFC5905]
NTS	Network Time Security
TLS	Transport Layer Security

Authors' Addresses

Daniel Fox Franke

Email: dfoxfranke@gmail.com
URI: https://www.dfranke.us

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

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-4471
Email: kristof.teichel@ptb.de