Stream: Internet Engineering Task Force (IETF)
RFC: 8915
Category: Standards Track
Published: September 2020
ISSN: 2070-1721

Authors:
D. Franke, *Akamai*
D. Sibold, *PTB*
K. Teichel, *PTB*
M. Dansarie
R. Sundblad, *Netnod*

RFC 8915
Network Time Security for the Network Time Protocol

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 client-server mode of the Network Time Protocol (NTP).

NTS is structured as a suite of two loosely coupled sub-protocols. The first (NTS Key Establishment (NTS-KE)) handles initial authentication and key establishment over TLS. The second (NTS Extension Fields for NTPv4) handles encryption and authentication during NTP time synchronization via extension fields in the NTP packets, and holds all required state only on the client via opaque cookies.

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:

- Identity: Through the use of a X.509 public key infrastructure, implementations can cryptographically establish the identity of the parties they are communicating with.
- Authentication: Implementations can 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.
- Confidentiality: Although basic time synchronization data is considered nonconfidential and sent in the clear, NTS includes support for encrypting NTP extension fields.
- Replay prevention: Client implementations can detect when a received time synchronization packet is a replay of a previous packet.
- Request-response consistency: Client implementations can verify that a time synchronization packet received from a server was sent in response to a particular request from the client.
- 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.
- Non-amplification: Implementations (especially server implementations) can avoid acting as distributed denial-of-service (DDoS) amplifiers by never responding to a request with a packet larger than the request packet.
- Scalability: Server implementations can serve large numbers of clients without having to retain any client-specific state.
- Performance: NTS must not significantly degrade the quality of the time transfer. The encryption and authentication used when actually transferring time should be lightweight (see Section 5.7 of RFC 7384 [RFC7384]).

1.2. Terms and Abbreviations

AEAD	Authenticated Encryption with Associated Data [RFC5116]
ALPN	Application-Layer Protocol Negotiation [RFC7301]
C2S	Client-to-server
DoS	Denial-of-Service
DDoS	Distributed Denial-of-Service

EF	Extension Field [RFC5905]
HKDF	Hashed Message Authentication Code-based Key Derivation Function [RFC5869]
KoD	Kiss-o'-Death [RFC5905]
NTP	Network Time Protocol [RFC5905]
NTS	Network Time Security
NTS NAK	NTS negative-acknowledgment
NTS-KE	Network Time Security Key Establishment
S2C	Server-to-client
TLS	Transport Layer Security [RFC8446]

1.3. Protocol Overview

The Network Time Protocol includes many different operating modes to support various network topologies (see Section 3 of RFC 5905 [RFC5905]). 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. Additionally, for certain message types, the 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 a 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 that provides replay protection inherently involves maintaining some state to keep track of which 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" define 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. All state is kept by the client and provided to the server in the form of an encrypted cookie supplied with each request. On the other hand, the NTS Extension Fields 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 establish keys, to provide the

client with an initial supply of cookies, and to negotiate some additional protocol options. After this, the TLS channel is closed with no per-client state remaining on the server side.

The typical protocol flow is as follows: The client connects to an NTS-KE 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 along with an address and port of an NTP server for which the cookies are valid. 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. Ideally, the client never needs to connect to the NTS-KE server again.

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

Figure 1 provides an overview of the high-level interaction between the client, the NTS-KE server, and the NTP server. Note that the cookies' data format and the exchange of secrets between NTS-KE and NTP servers are not part of this specification and are implementation dependent. However, a suggested format for NTS cookies is provided in Section 6.

```
                                                     +--------------+
                                                     |              |
                                                 +-> | NTP Server 1 |
                                                 |   |              |
                           Shared cookie         |   +--------------+
+---------------+      encryption parameters     |   +--------------+
|               |    (Implementation dependent)  |   |              |
| NTS-KE Server | <------------------------------+-> | NTP Server 2 |
|               |                                |   |              |
+---------------+                                |   +--------------+
       ^                                         |          .
       |                                         |          .
       | 1. Negotiate parameters,                |          .
       |    receive initial cookie               |   +--------------+
       |    supply, generate AEAD keys,          |   |              |
       |    and receive NTP server IP            +-> | NTP Server N |
       |    addresses using "NTS Key                 |              |
       |    Establishment" protocol.                 +--------------+
       |                                                    ^
       |                                                    |
       |             +----------+                           |
       |             |          |                           |
       +-----------> |  Client  | <-------------------------+
                     |          |  2. Perform authenticated
                     +----------+     time synchronization
                                      and generate new
                                      cookies using "NTS
                                      Extension Fields for
                                      NTPv4".
```

Figure 1: Overview of High-Level Interactions in NTS

2. Requirements Language

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. TLS Profile for Network Time Security

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

Since the NTS protocol is new as of this publication, no backward-compatibility concerns exist to justify using obsolete, insecure, or otherwise broken TLS features or versions. Implementations **MUST** conform with RFC 7525 [RFC7525] or with a later revision of BCP 195.

Implementations **MUST NOT** negotiate TLS versions earlier than 1.3 [RFC8446] and **MAY** refuse to negotiate any TLS version that has been superseded by a later supported version.

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

Implementations **MUST** follow the rules in RFC 5280 [RFC5280] and RFC 6125 [RFC6125] for the representation and verification of the application's service identity. When NTS-KE service discovery (out of scope for this document) produces one or more host names, use of the DNS-ID identifier type [RFC6125] is **RECOMMENDED**; specifications for service discovery mechanisms can provide additional guidance for certificate validation based on the results of discovery. Section 8.5 of this memo discusses particular considerations for certificate verification in the context of NTS.

4. The NTS Key Establishment Protocol

The NTS key establishment protocol is conducted via TCP port 4460. The two endpoints carry out a TLS handshake in conformance with Section 3, with the client offering (via an ALPN extension [RFC7301]), 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. After sending their respective request and response, the client and server **SHALL** send TLS "close_notify" alerts in accordance with Section 6.1 of RFC 8446 [RFC8446].

The client's request and the server's response each **SHALL** consist of a sequence of records formatted according to Figure 2. The request and a non-error response each **SHALL** include exactly one NTS Next Protocol Negotiation record. The sequence **SHALL** be terminated by a "End of Message" record. The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting.

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 2: NTS-KE Record Format

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 (see Section 4.1.3).

Record Type Number: A 15-bit integer in network byte order. The semantics of Record Types 0-7 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 the C programming language, 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]'.

Note that, although the Type-Length-Body format of an NTS-KE record is similar to that of an NTP extension field, the semantics of the length field differ. While the length subfield of an NTP extension field gives the length of the entire extension field including the type and length subfields, the length field of an NTS-KE record gives just the length of the body.

Figure 3 provides a schematic overview of the key establishment. 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.       |
                |   - Include Record Types:             |
                |       o NTS Next Protocol Negotiation |
                |       o AEAD Algorithm Negotiation    |
                |       o <NTPv4 Server Negotiation>    |
                |       o <NTPv4 Port Negotiation>      |
                |       o New Cookie for NTPv4          |
                |       o <New Cookie for NTPv4>        |
                |       o 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 Algorithm Negotiation    |   +------------------------+
|  o <NTPv4 Server Negotiation>    |
|  o <NTPv4 Port Negotiation>      |
|  o End of Message                |
+----------------------------------+
```

Figure 3: NTS Key Establishment 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 a 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 number 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 (Section 7.7). The Critical Bit **MUST** be set.

The Protocol IDs listed in the client's NTS Next Protocol Negotiation record denote those protocols that the client wishes to speak using the key material established through this NTS-KE session. Protocol IDs listed in the NTS-KE server's response **MUST** comprise a subset of those listed in the request and denote those protocols that the NTP 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 that includes an Error record, they **MUST** discard any key material negotiated during the initial TLS exchange and **MUST NOT** proceed to the Next Protocol. Requirements for retry intervals are described in Section 4.2.

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 that the server did not understand and that 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 the request is not complete and syntactically well-formed, or, upon the expiration of an implementation-defined timeout, it has not yet received such a request. The client **SHOULD NOT** retry its request without modification.
>
> Error code 2 means "Internal Server Error". The server **MUST** respond with this error if it is unable to respond properly due to an internal condition. The client **MAY** 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 that 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 Algorithms" registry [IANA-AEAD]. 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. It 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 that 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 6 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 the Next Protocol Negotiation response record contains Protocol ID 0 (NTPv4) and the AEAD Algorithm Negotiation response record is not empty. The Critical Bit **SHOULD NOT** be set.

4.1.7. NTPv4 Server Negotiation

The NTPv4 Server Negotiation record has a Record Type number of 6. Its body consists of an ASCII-encoded [RFC0020] string. The contents of the string **SHALL** be either an IPv4 address, an IPv6 address, or a fully qualified domain name (FQDN). IPv4 addresses **MUST** be in dotted decimal notation. IPv6 addresses **MUST** conform to the "Text Representation of Addresses" as specified in RFC 4291 [RFC4291] and **MUST NOT** include zone identifiers [RFC6874]. If a label contains at least one non-ASCII character, it is an internationalized domain name, and an A-LABEL **MUST** be used as defined in Section 2.3.2.1 of RFC 5890 [RFC5890]. If the record contains a domain name, the recipient **MUST** treat it as a FQDN, e.g., by making sure it ends with a dot.

When NTPv4 is negotiated as a Next Protocol and this record is sent by the server, the body specifies the hostname or IP address of the NTPv4 server with which the client should associate and that will accept the supplied cookies. If no record of this type is sent, the client **SHALL** interpret this as a directive to associate with an NTPv4 server at the same IP address as the NTS-KE server. Servers **MUST NOT** send more than one record of this type.

When this record is sent by the client, it indicates that the client wishes to associate with the specified NTP server. The NTS-KE server **MAY** incorporate this request when deciding which NTPv4 Server Negotiation records to respond with, but honoring the client's preference is **OPTIONAL**. The client **MUST NOT** send more than one record of this type.

If the client has sent a record of this type, the NTS-KE server **SHOULD** reply with the same record if it is valid and the server is able to supply cookies for it. If the client has not sent any record of this type, the NTS-KE server **SHOULD** respond with either an NTP server address in the same family as the NTS-KE session or a FQDN that can be resolved to an address in that family, if such alternatives are available.

Servers **MAY** set the Critical Bit on records of this type; clients **SHOULD NOT**.

4.1.8. NTPv4 Port Negotiation

The NTPv4 Port Negotiation record has a Record Type number of 7. Its body consists of a 16-bit unsigned integer in network byte order, denoting a UDP port number.

When NTPv4 is negotiated as a Next Protocol, and this record is sent by the server, the body specifies the port number of the NTPv4 server with which the client should associate and that will accept the supplied cookies. If no record of this type is sent, the client **SHALL** assume a default of 123 (the registered port number for NTP).

When this record is sent by the client in conjunction with a NTPv4 Server Negotiation record, it indicates that the client wishes to associate with the NTP server at the specified port. The NTS-KE server **MAY** incorporate this request when deciding what NTPv4 Server Negotiation and NTPv4 Port Negotiation records to respond with, but honoring the client's preference is **OPTIONAL**.

Servers **MAY** set the Critical Bit on records of this type; clients **SHOULD NOT**.

4.2. Retry Intervals

A mechanism for not unnecessarily overloading the NTS-KE server is **REQUIRED** when retrying the key establishment process due to protocol, communication, or other errors. The exact workings of this will be dependent on the application and operational experience gathered over time. Until such experience is available, this memo provides the following suggestion.

Clients **SHOULD** use exponential backoff, with an initial and minimum retry interval of 10 seconds, a maximum retry interval of 5 days, and a base of 1.5. Thus, the minimum interval in seconds, 't', for the nth retry is calculated with the following:

$$t = \min(10 * 1.5^{n-1}, 432000).$$

Clients **MUST NOT** reset the retry interval until they have performed a successful key establishment with the NTS-KE server, followed by a successful use of the negotiated Next Protocol with the keys and data established during that transaction.

4.3. Key Extraction (Generally)

Following a successful run of the NTS-KE protocol, key material **SHALL** be extracted using the HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [RFC5869] in accordance with Section 7.5 of RFC 8446 [RFC8446]. Inputs to the exporter function are to be constructed in a manner specific to the negotiated Next Protocol. However, all protocols that utilize NTS-KE **MUST** conform to the following two rules:

> The disambiguating label string [RFC5705] **MUST** be "EXPORTER-network-time-security".
>
> The per-association context value [RFC5705] **MUST** be provided and **MUST** begin with the two-octet Protocol ID that 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 with the HKDF defined in Section 7.5 of RFC 8446 [RFC8446] using the following inputs:

> The disambiguating label string [RFC5705] **SHALL** be "EXPORTER-network-time-security".
>
> The per-association context value [RFC5705] **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 [RFC5705] 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 following:

> The usual 48-octet NTP header, which is authenticated but not encrypted.
>
> Some extension fields, which are authenticated but not encrypted.
>
> An extension field that contains AEAD output (i.e., an authentication tag and possible ciphertext). The corresponding plaintext, if non-empty, consists of some extension fields that benefit from both encryption and authentication.

Possibly, some additional extension fields that are neither encrypted nor authenticated. In general, 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, as described in Section 5.7. 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 provides the client with a cryptographically strong means of detecting replayed packets. It has a Field Type of 0x0104. When the extension field is included in a client packet (mode 3), its body **SHALL** consist of a string of octets generated by a cryptographically secure random number generator [RFC4086]. 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. All server packets generated by NTS-implementing servers in response to client packets containing this extension field **MUST** also contain this field with the same content as in the client's request. The field's use in modes other than client-server is not defined.

This extension field **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 this is suboptimal for cryptographic purposes 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 0x0204. Its purpose is to carry information that 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 6 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 0x0304. 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. 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 **SHOULD** be all zeros 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 0x0404. It **SHALL** be formatted according to Figure 4 and include the following fields:

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

Ciphertext Length: Two octets in network byte order, giving the length of the Ciphertext field, excluding any padding, interpreted as an unsigned integer.

Nonce: A nonce as required by the negotiated AEAD algorithm. The end of the field is zero-padded to a word (four octets) boundary.

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. The end of the field is zero-padded to a word (four octets) boundary.

Additional Padding: Clients that use a nonce length shorter than the maximum allowed by the negotiated AEAD algorithm may be required to include additional zero-padding. The necessary length of this field is specified below.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Nonce Length         |      Ciphertext Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.          Nonce, including up to 3 octets padding              .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.        Ciphertext, including up to 3 octets padding           .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                      Additional Padding                       .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: NTS Authenticator and Encrypted Extension Fields Extension Field Format

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 that 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; if multiple extension fields are present, they **SHALL** be joined by concatenation. Each such field **SHALL** be formatted in accordance with RFC 7822 [RFC7822], except that, contrary to the RFC 7822 requirement that fields have a minimum length of 16 or 28 octets, encrypted extension fields **MAY** be arbitrarily short (but still **MUST** be a multiple of 4 octets in length).

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

The purpose of the Additional Padding field is to ensure that servers can always choose a nonce whose length is adequate to ensure its uniqueness, even if the client chooses a shorter one, and still ensure that the overall length of the server's response packet does not exceed the length of the request. For mode 4 (server) packets, no Additional Padding field is ever required. For mode 3 (client) packets, the length of the Additional Padding field **SHALL** be computed as follows. Let 'N_LEN' be the padded length of the Nonce field. Let 'N_MAX' be, as specified by RFC 5116 [RFC5116], the maximum permitted nonce length for the negotiated AEAD algorithm. Let 'N_REQ' be the lesser of 16 and N_MAX, rounded up to the nearest multiple of 4. If N_LEN is greater than or equal to N_REQ, then no Additional Padding field is required. Otherwise, the Additional Padding field **SHALL** be at least N_REQ - N_LEN octets in length. Servers **MUST** enforce this requirement by discarding any packet that does not conform to it.

Senders are always free to include more Additional Padding than mandated by the above paragraph. Theoretically, it could be necessary to do so in order to bring the extension field to the minimum length required by RFC 7822 [RFC7822]. This should never happen in practice because any reasonable AEAD algorithm will have a nonce and an authenticator long enough to bring the extension field to its required length already. Nonetheless, implementers are advised to explicitly handle this case and ensure that the extension field they emit is of legal length.

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).

5.7. Protocol Details

A client sending an NTS-protected request **SHALL** include the following extension fields as displayed in Figure 5:

> Exactly one Unique Identifier extension field that **MUST** be authenticated, **MUST NOT** be encrypted, and whose contents **MUST** be the output of a cryptographically secure random number generator [RFC4086].
>
> Exactly one NTS Cookie extension field that **MUST** be authenticated and **MUST NOT** be encrypted. The cookie **MUST** be one which has been previously provided to the client, either from the key establishment server during the NTS-KE handshake or from the NTP server in response to a previous NTS-protected NTP request.
>
> Exactly one NTS Authenticator and Encrypted Extension Fields extension field, generated using an AEAD algorithm and C2S key established through NTS-KE.

To protect the client's privacy, the client **SHOULD** avoid reusing a cookie. If the client does not have any cookies that it has not already sent, it **SHOULD** initiate a rerun of the NTS-KE protocol. The client **MAY** reuse cookies in order to prioritize resilience over unlinkability. Which of the two that should be prioritized in any particular case is dependent on the application and the user's preference. Section 9.1 describes the privacy considerations of this in further detail.

The client **MAY** include one or more NTS Cookie Placeholder extension fields that **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. The client **SHOULD NOT** include more than seven NTS Cookie Placeholder extension fields in a request. 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.

In rare circumstances, it may be necessary to include fewer NTS Cookie Placeholder extensions than recommended above in order to prevent datagram fragmentation. When cookies adhere to the format recommended in Section 6 and the AEAD in use is the mandatory-to-implement AEAD_AES_SIV_CMAC_256, senders can include a cookie and seven placeholders and still have packet size fall comfortably below 1280 octets if no non-NTS-related extensions are used; 1280 octets is the minimum prescribed MTU for IPv6 and is generally safe for avoiding IPv4 fragmentation. Nonetheless, senders **SHOULD** include fewer cookies and placeholders than otherwise indicated if doing so is necessary to prevent fragmentation.

```
                +---------------------------------------+
                | - 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 5: NTS-Protected NTP Time Synchronization Messages

The client **MAY** include additional (non-NTS-related) extension fields that **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). The server **MUST** discard any unauthenticated extension fields. Future specifications of extension fields **MAY** provide exceptions to this rule.

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, the server **SHALL** include the following extension fields in its response:

> Exactly one Unique Identifier extension field that **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 that **MUST** be authenticated and encrypted. 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. The cookies returned in those fields **MUST** be valid for use with the NTP server that sent them. They **MAY** be valid for other NTP servers as well, but there is no way for the server to indicate this.

We emphasize the contrast that NTS Cookie extension fields **MUST NOT** be encrypted when sent from client to server but **MUST** be encrypted when 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 9.1. We emphasize also that "encrypted" means encapsulated within the NTS Authenticator and Encrypted Extensions extension field. While the body of an NTS Cookie extension field will generally consist of some sort of AEAD output (regardless of whether the recommendations of Section 6 are precisely followed), this is not sufficient to make the extension field "encrypted".

The server **MAY** include additional (non-NTS-related) extension fields that **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). The client **MUST** discard any unauthenticated extension fields. Future specifications of extension fields **MAY** provide exceptions to this rule.

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. In particular, the client **MUST** discard unprotected responses to NTS-protected requests.

If the server is unable to validate the cookie or authenticate the request, it **SHOULD** respond with a Kiss-o'-Death (KoD) packet (see Section 7.4 of RFC 5905 [RFC5905]) with kiss code "NTSN", meaning "NTS NAK" (NTS negative-acknowledgment). It **MUST NOT** include any NTS Cookie or NTS Authenticator and Encrypted Extension Fields extension fields.

If the NTP server has previously responded with authentic NTS-protected NTP packets, the client **MUST** verify that any KoD packets received from the server contain the Unique Identifier extension field and 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 **MUST** comply with Section 7.4 of RFC 5905 [RFC5905] where required.

A client **MAY** automatically rerun the NTS-KE protocol upon forced disassociation from an NTP server. In that case, it **MUST** avoid quickly looping between the NTS-KE and NTP servers by rate limiting the retries. Requirements for retry intervals in NTS-KE are described in Section 4.2.

Upon reception of the NTS NAK kiss code, the client **SHOULD** wait until the next poll for a valid NTS-protected response, and if none is received, initiate a fresh NTS-KE handshake to try to renegotiate new cookies, AEAD keys, and parameters. If the NTS-KE handshake succeeds, the client **MUST** discard all old cookies and parameters and use the new ones instead. As long as the NTS-KE handshake has not succeeded, the client **SHOULD** continue polling the NTP server using the cookies and parameters it has.

To allow for NTP session restart when the NTS-KE server is unavailable and to reduce NTS-KE server load, the client **SHOULD** keep at least one unused but recent cookie, AEAD keys, negotiated AEAD algorithm, and other necessary parameters in persistent storage. This way, the client is able to resume the NTP session without performing renewed NTS-KE negotiation.

6. 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 tickets 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 that 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. It need not be the same as the one that was negotiated with the client. Servers should randomly generate and store a secret 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. Following each such key rotation, servers should securely erase any previously generated keys that should now be expired. 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 between NTS-KE and NTP servers as well as 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 pseudorandom 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.

7. IANA Considerations

7.1. Service Name and Transport Protocol Port Number Registry

IANA has allocated the following entry in the "Service Name and Transport Protocol Port Number Registry" [RFC6335]:

Service Name: ntske

Port Number: 4460

Transport Protocol: tcp

Description: Network Time Security Key Establishment

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Registration Date: 2020-04-07

Reference: RFC 8915

7.2. TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs Registry

IANA has allocated the following entry in the "TLS Application-Layer Protocol Negotiation (ALPN) Protocol IDs" registry [RFC7301]:

Protocol: Network Time Security Key Establishment, version 1

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

Reference: RFC 8915, Section 4

7.3. TLS Exporter Labels Registry

IANA has allocated the following entry in the TLS Exporter Labels registry [RFC5705]:

Value	DTLS-OK	Recommended	Reference	Note
EXPORTER-network-time-security	Y	Y	RFC 8915, Section 4.3	

Table 1

7.4. NTP Kiss-o'-Death Codes Registry

IANA has allocated the following entry in the "NTP Kiss-o'-Death Codes" registry [RFC5905]:

Code	Meaning	Reference
NTSN	Network Time Security (NTS) negative-acknowledgment (NAK)	RFC 8915, Section 5.7

Table 2

7.5. NTP Extension Field Types Registry

IANA has allocated the following entries in the "NTP Extension Field Types" registry [RFC5905]:

Field Type	Meaning	Reference
0x0104	Unique Identifier	RFC 8915, Section 5.3
0x0204	NTS Cookie	RFC 8915, Section 5.4
0x0304	NTS Cookie Placeholder	RFC 8915, Section 5.5

Field Type	Meaning	Reference
0x0404	NTS Authenticator and Encrypted Extension Fields	RFC 8915, Section 5.6

Table 3

7.6. Network Time Security Key Establishment Record Types Registry

IANA has created a new registry entitled "Network Time Security Key Establishment Record Types". Entries have the following fields:

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

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

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

The registration policy varies by Record Type Number, as follows:

0-1023: IETF Review

1024-16383: Specification Required

16384-32767: Private or Experimental Use

The initial contents of this registry are as follows:

Record Type Number	Description	Reference
0	End of Message	RFC 8915, Section 4.1.1
1	NTS Next Protocol Negotiation	RFC 8915, Section 4.1.2
2	Error	RFC 8915, Section 4.1.3
3	Warning	RFC 8915, Section 4.1.4
4	AEAD Algorithm Negotiation	RFC 8915, Section 4.1.5
5	New Cookie for NTPv4	RFC 8915, Section 4.1.6
6	NTPv4 Server Negotiation	RFC 8915, Section 4.1.7
7	NTPv4 Port Negotiation	RFC 8915, Section 4.1.8
8-16383	Unassigned	
16384-32767	Reserved for Private or Experimental Use	RFC 8915

Table 4

7.7. Network Time Security Next Protocols Registry

IANA has created a new registry entitled "Network Time Security Next Protocols". Entries 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 (**REQUIRED**): A reference to a relevant specification document.

The registration policy varies by Protocol ID, as follows:

0-1023: IETF Review

1024-32767: Specification Required

32768-65535: Private or Experimental Use

The initial contents of this registry are as follows:

Protocol ID	Protocol Name	Reference
0	Network Time Protocol version 4 (NTPv4)	RFC 8915
1-32767	Unassigned	
32768-65535	Reserved for Private or Experimental Use	RFC 8915

Table 5

7.8. Network Time Security Error and Warning Codes Registries

IANA has created two new registries entitled "Network Time Security Error Codes" and "Network Time Security Warning Codes". Entries in each 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 registration policy varies by Number, as follows:

0-1023: IETF Review

1024-32767: Specification Required

32768-65535: Private or Experimental Use

The initial contents of the "Network Time Security Error Codes" registry are as follows:

Number	Description	Reference
0	Unrecognized Critical Record	RFC 8915, Section 4.1.3
1	Bad Request	RFC 8915, Section 4.1.3
2	Internal Server Error	RFC 8915, Section 4.1.3
3-32767	Unassigned	
32768-65535	Reserved for Private or Experimental Use	RFC 8915

Table 6

The "Network Time Security Warning Codes" registry is initially empty except for the reserved range, i.e.:

Number	Description	Reference
0-32767	Unassigned	
32768-65535	Reserved for Private or Experimental Use	RFC 8915

Table 7

8. Security Considerations

8.1. Protected Modes

NTP provides many different operating modes in order to support different network topologies and to adapt to various requirements. This memo only specifies NTS for NTP modes 3 (client) and 4 (server) (see Section 1.3). The best current practice for authenticating the other NTP modes is using the symmetric message authentication code feature as described in RFC 5905 [RFC5905] and RFC 8573 [RFC8573].

8.2. Cookie Encryption Key Compromise

If the suggested format for NTS cookies in Section 6 of this document is used, an attacker who has gained access to the secret cookie encryption key 'K' can impersonate the NTP server, including generating new cookies. NTP and NTS-KE server operators **SHOULD** remove compromised keys as soon as the compromise is discovered. This will cause the NTP servers to respond with NTS NAK, thus forcing key renegotiation. Note that this measure does not protect against MITM attacks where the attacker has access to a compromised cookie encryption key. If another cookie scheme is used, there are likely similar considerations for that particular scheme.

8.3. Sensitivity to DDoS Attacks

The introduction of NTS brings with it the introduction of asymmetric cryptography to NTP. Asymmetric cryptography is necessary for initial server authentication and AEAD key extraction. Asymmetric cryptosystems are generally orders of magnitude slower than their symmetric counterparts. This makes it much harder to build systems that can serve requests at a rate corresponding to the full line speed of the network connection. This, in turn, opens up a new possibility for DDoS attacks on NTP services.

The main protection against these attacks in NTS lies in that the use of asymmetric cryptosystems is only necessary in the initial NTS-KE phase of the protocol. Since the protocol design enables separation of the NTS-KE and NTP servers, a successful DDoS attack on an NTS-KE server separated from the NTP service it supports will not affect NTP users that have already performed initial authentication, AEAD key extraction, and cookie exchange.

NTS users should also consider that they are not fully protected against DoS attacks by on-path adversaries. In addition to dropping packets and attacks such as those described in Section 8.6, an on-path attacker can send spoofed Kiss-o'-Death replies, which are not authenticated, in response to NTP requests. This could result in significantly increased load on the NTS-KE server. Implementers have to weigh the user's need for unlinkability against the added resilience that comes with cookie reuse in cases of NTS-KE server unavailability.

8.4. Avoiding DDoS Amplification

Certain nonstandard and/or deprecated features of the Network Time Protocol enable clients to send a request to a server that causes the server to send a response much larger than the request. Servers that enable these features can be abused in order to amplify traffic volume in DDoS attacks by sending them a request with a spoofed source IP address. 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 RFC 7822 [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.

8.5. 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 NTS-KE server is valid and rooted in a trusted certificate authority. RFC 5280 [RFC5280] and RFC 6125 [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. It may be operationally necessary in some cases for a client to accept a certificate that 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. On systems with the ntp_adjtime() system call, a return code other than TIME_ERROR indicates that some trusted software has already set the time 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 that have a battery-backed clock or that 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.
>
> NTP time replies are expected to be consistent with the NTS-KE TLS certificate validity period, i.e. time replies received immediately after an NTS-KE handshake are expected to lie within the certificate validity period. Implementations are recommended to check that this is the case. Performing a new NTS-KE handshake based solely on the fact that the certificate used by the NTS-KE server in a previous handshake has expired is normally not necessary. Clients that still wish to do this must take care not to cause an inadvertent denial-of-service attack on the NTS-KE server, for example by picking a random time in the week preceding certificate expiry to perform the new handshake.
>
> 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 (Section 11.2.1 of RFC 5905 [RFC5905]) will protect the client from accepting bad time from the adversary-controlled servers.

8.6. 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.

RFC 5905 [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 that 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.

8.7. NTS Stripping

Implementers must be aware of the possibility of "NTS stripping" attacks, where an attacker attempts to trick clients into reverting to plain NTP. Naive client implementations might, for example, revert automatically to plain NTP if the NTS-KE handshake fails. A man-in-the-middle attacker can easily cause this to happen. Even clients that already hold valid cookies can be vulnerable, since an attacker can force a client to repeat the NTS-KE handshake by sending faked NTP mode 4 replies with the NTS NAK kiss code. Forcing a client to repeat the NTS-KE handshake can also be the first step in more advanced attacks.

For the reasons described here, implementations **SHOULD NOT** revert from NTS-protected to unprotected NTP with any server without explicit user action.

9. Privacy Considerations

9.1. 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 document NTP Client Data Minimization [NTP-DATA-MIN].

The unlinkability objective only holds for time synchronization traffic, as opposed to key establishment 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 linkable data, such as the TLS certificate).

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.

Note that the unlinkability objective does not prevent a client device from being tracked by its time servers.

9.2. Confidentiality

NTS does not protect the confidentiality of information in NTP's header fields. When clients implement NTP Client Data Minimization [NTP-DATA-MIN], client packet headers do not contain any information that 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.

10. References

10.1. Normative References

[IANA-AEAD] IANA, "Authenticated Encryption with Associated Data (AEAD) Parameters", <https://www.iana.org/assignments/aead-parameters/>.

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[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>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[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>.

[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>.

[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>.

[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>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[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>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.

[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>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

10.2. Informative References

[Mizrahi]

Mizrahi, T., "A game theoretic analysis of delay attacks against time synchronization protocols", 2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings, pp. 1-6, DOI 10.1109/ISPCS.2012.6336612, September 2012, <https://doi.org/10.1109/ISPCS.2012.6336612>.

[NTP-DATA-MIN] Franke, D. F. and A. Malhotra, "NTP Client Data Minimization", Work in Progress, Internet-Draft, draft-ietf-ntp-data-minimization-04, 25 March 2019, <https://tools.ietf.org/html/draft-ietf-ntp-data-minimization-04>.

[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>.

[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>.

[RFC8573] Malhotra, A. and S. Goldberg, "Message Authentication Code for the Network Time Protocol", RFC 8573, DOI 10.17487/RFC8573, June 2019, <https://www.rfc-editor.org/info/rfc8573>.

[Shpiner] Shpiner, A., Revah, Y., and T. Mizrahi, "Multi-path Time Protocols", 2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings, pp. 1-6, DOI 10.1109/ISPCS.2013.6644754, September 2013, <https://doi.org/10.1109/ISPCS.2013.6644754>.

Acknowledgments

The authors would like to thank Richard Barnes, Steven Bellovin, Scott Fluhrer, Patrik Fältström, Sharon Goldberg, Russ Housley, Benjamin Kaduk, Suresh Krishnan, Mirja Kühlewind, Martin Langer, Barry Leiba, Miroslav Lichvar, Aanchal Malhotra, Danny Mayer, Dave Mills, Sandra Murphy, Hal Murray, Karen O'Donoghue, Eric K. Rescorla, Kurt Roeckx, Stephen Roettger, Dan Romascanu, Kyle Rose, Rich Salz, Brian Sniffen, Susan Sons, Douglas Stebila, Harlan Stenn, Joachim Strömbergsson, Martin Thomson, Éric Vyncke, Richard Welty, Christer Weinigel, and Magnus Westerlund for contributions to this document and comments on the design of NTS.

Authors' Addresses

Daniel Fox Franke
Akamai Technologies
145 Broadway
Cambridge, MA 02142
United States of America
Email: dafranke@akamai.com

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
D-38116 Braunschweig
Germany
Phone: +49-(0)531-592-8462
Email: dieter.sibold@ptb.de

Kristof Teichel
Physikalisch-Technische Bundesanstalt
Bundesallee 100
D-38116 Braunschweig
Germany
Phone: +49-(0)531-592-4471
Email: kristof.teichel@ptb.de

Marcus Dansarie
Sweden
Email: marcus@dansarie.se
URI: https://orcid.org/0000-0001-9246-0263

Ragnar Sundblad
Netnod
Sweden
Email: ragge@netnod.se