Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 10, 2017

D. Franke
Akamai

October 7, 2016

Network Time Security
draft-dfranke-nts-00

Abstract

This memo specifies Network Time Security (NTS), a mechanism for using Datagram TLS to provide cryptographic security for the Network Time Protocol or other network time synchronization protocols.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 10, 2017.

Table of Contents

1. Introduction

[[SEE https://github.com/dfoxfranke/nts FOR AN UP-TO-MINUTE DRAFT OF THIS MEMO, AND https://github.com/dfoxfranke/nts/issues FOR A LIST OF OUTSTANDING ISSUES.]]

This memo specifies Network Time Security (NTS), a mechanism for using Datagram Transport Layer Security [RFC6347] (DTLS) to provide cryptographic security for network time synchronization. A complete specification is provided for applying NTS to the Network Time Protocol [RFC5905]. Certain sections, however, are not inherently NTP-specific and include guidance on how future work may apply them

to other time synchronization protocols such as the Precision Time Protocol [IEC.61588_2009].

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 broadcast mode, and a control mode for server monitoring and administration. These various modes have differing and 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 servers to process replayed packets.

The security demands of symmetric and control modes 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. Since therefore no single approach can simultaneously satisfy the needs of all modes, Network Time Security consists of not one protocol but a suite of them:

The "NTS-encapsulated NTPv4" protocol is little more than "NTP over DTLS": the two endpoints perform a DTLS handshake and then exchange NTP packets encapsulated as DTLS Application Data. It is suitable for symmetric and control modes, and is also secure for client/server mode but relatively wasteful of server resources.

The "NTS Key Establishment" protocol (NTS-KE) uses DTLS to establish key material and negotiate some additional protocol options, but then quickly closes the DTLS channel and does not use it for the exchange of time packets. NTS-KE is designed to be extensible, and might be extended to support key establishment for other protocols such as PTP.

The "NTS extensions for NTPv4" are a collection of NTP extension fields for cryptographically securing NTPv4 using key material previously negotiated using NTS-KE. 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.

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. DTLS profile for Network Time Security

Since securing time protocols is (as of 2016) a novel application of DTLS, no backward-compatibility concerns exist to justify using obsolete, insecure, or otherwise broken DTLS features or versions. We therefore put forward the following requirements and guidelines, roughly representing 2016's best practices.

Implementations MUST NOT negotiate DTLS versions earlier than 1.2.

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

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

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

DTLS clients SHOULD offer, and DTLS servers SHOULD accept, the TLS Session Hash and Extended Master Secret Extension [RFC7627]. (*)

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

(*): Note that DTLS 1.3 or beyond may render the indicated recommendations inapplicable.

4. Transport mechanisms for DTLS records

This section specifies two mechanisms, one REQUIRED and one OPTIONAL, for exchanging NTS-related DTLS records. It is intended that the choice of transport mechanism be orthogonal to any concerns at the application layer: DTLS records SHOULD receive identical disposition regardless of which mechanism they arrive by.

4.1. Transport via NTS port

In this transport mechanism, DTLS records, formatted according to RFC 6347 [RFC6347] or a subsequent revision thereof, are exchanged directly on UDP port [[TBD]], with one DTLS record per UDP packet and no additional layer of encapsulation between the UDP header and the DTLS record. Servers which implement NTS MUST support this mechanism.

4.2. Transport via NTP extension field

In this transport mechanism, DTLS records are exchanged within extension fields of specially-formed NTP packets, which are themselves exchanged via the usual NTP service port (123/udp). NTP packets conveying DTLS records SHALL be formatted as in Figure 1. They MUST NOT contain any other extensions or a legacy MAC field.

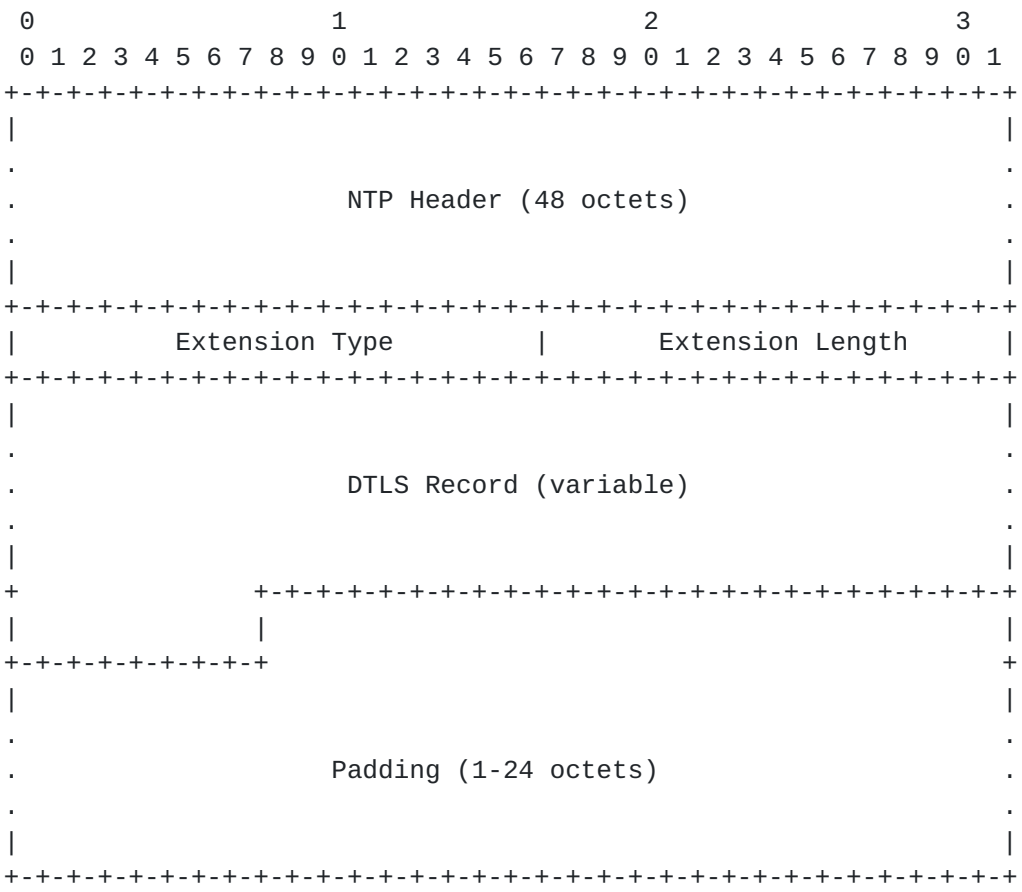

Figure 1: Format of NTP packets conveying DTLS records

Within the NTP header,

The Leap Indicator field SHALL be set to 3 (unsynchronized).

The Version Number field SHALL be set to 4.

DTLS clients SHALL set the Mode field to 3, and DTLS servers SHALL set the Mode field to 4, even if the DTLS record is being used (in the full-encapsulation protocol) to protect some NTP mode other than client/server.

The Stratum field SHALL be set to 0 (unspecified or invalid).

The Reference ID field (conveying a kiss code) SHALL be set to "DTLS"

DTLS servers SHALL set the origin timestamp field from the transmit timestamp field of the packet most recently received from the client.

All other header fields MUST be ignored by the receiver, and MAY contain arbitrary or bogus values.

The Extension Type field SHALL be set to [[TBD]]. The Extension Length field SHALL be computed and set as per RFC 7822 [RFC7822].

The DTLS Record field SHALL contain a DTLS Record formatted as per RFC 6347 [RFC6347] or a subsequent revision thereof.

The Padding field SHALL contain between 1 and 24 octets of padding, with every octet set to the number of padding octets included, e.g., "01", "02 02", or "03 03 03". The number of padding bytes should be chosen in order to comply with the RFC 7822 [RFC7822] requirement that (in the absence of a legacy MAC) extensions have a total length in octets (including the four octets for the type and length fields) which is at least 28 and divisible by 4. Furthermore, since future revisions of DTLS may employ record formats that are not self-delimiting, at least one octet of padding MUST be included so that receivers can unambiguously determine where the DTLS record ends and the padding begins. If the length of the DTLS record is already at least 24 and a multiple of 4, then the correct amount of padding to include is 4 octets.

The NTP header values specified above are selected such that NTP implementations which do not understand NTS will interpret the packet as an innocuous no-op and not attempt to use it for time synchronization. To NTS-aware implementations, however, these packets are best understood as not being NTP packets at all, but simply a means of "smuggling" arbitrary DTLS records across port 123/udp. Indeed, these records need not be pertinent to NTP at all -- for example, they could be NTS-KE messages eventually intended for securing PTP traffic.

This transport mechanism is intended for use as a fallback in situations where firewalls or other middleboxes are preventing communication on the NTS port. Support for it is OPTIONAL.

5. The NTS-encapsulated NTPv4 protocol

The NTS-encapsulated NTPv4 protocol proceeds in two parts. First, DTLS handshake records are exchanged using one of the two transport mechanisms specified in Section 4. The two endpoints carry out a DTLS handshake in conformance with Section 3, with the client offering (via an ALPN [RFC7301] extension), and the server accepting, an application-layer protocol of "ntp/4". Second, once the handshake is successfully completed, the two endpoints use the established channel to exchange arbitrary NTPv4 packets as DTLS-protected Application Data.

In addition to the requirements specified in Section 3, implementations MUST enforce the anti-replay mechanism specified in Section 4.1.2.6 of RFC 6347 [RFC6347] (or an equivalent mechanism specified in a subsequent revision of DTLS). Servers wishing to enforce access control SHOULD either demand a client certificate or use a PSK-based handshake in order to establish the client's identity.

The NTS-encapsulated NTPv4 protocol is the RECOMMENDED mechanism for cryptographically securing mode 1 (symmetric active), 2 (symmetric passive), and 6 (control) NTPv4 traffic. It is equally safe for mode 3/4 (client/server) traffic, but is NOT RECOMMENDED for this purpose because it scales poorly compared to using NTS Extensions for NTPv4 (Section 7).

6. The NTS Key Establishment protocol

The NTS Key Establishment (NTS-KE) protocol is carried out by exchanging DTLS records using one of the two transport mechanisms specified in Section 4. The two endpoints carry out a DTLS 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 DTLS-protected channel), then the server SHALL send a single response followed by a "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 2. 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.

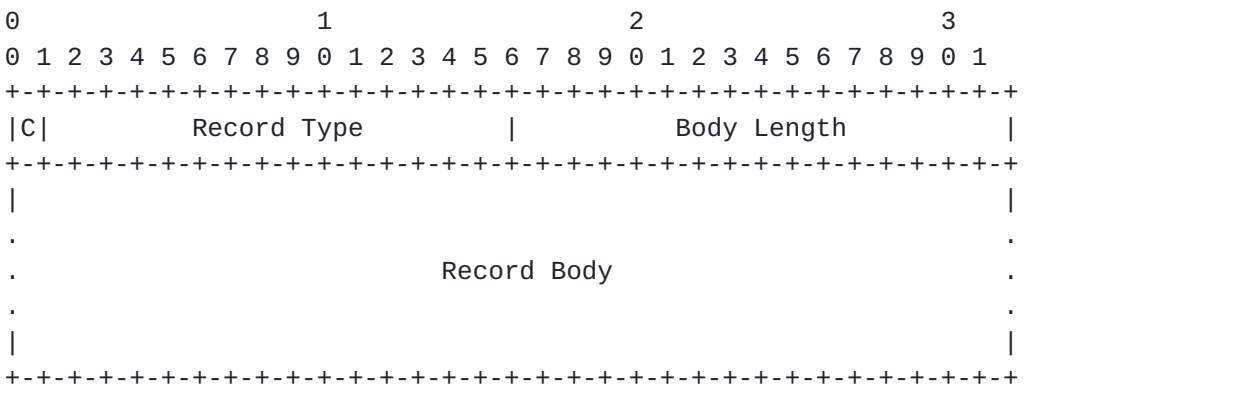

Figure 2

[[Ed. Note: this ad-hoc binary format should be fine as long as we continue to keep things very simple. However, if we think there's any reasonable probability of wanting to include more complex data structures, we should consider using some semi-structured data format such as JSON, Protocol Buffers, or (ugh) ASN.1]]

The requirement that all NTS-KE messages be terminated by an End of Message record makes them self-delimiting. One DTLS record MAY, and typcially will, contain multiple NTS-KE records. NTS-KE records MAY be split across DTLS record boundaries. If, likely due to packet loss, an incomplete NTS-KE message is received, implementations MUST treat this an error, which clients SHOULD handle by restarting with a fresh DTLS handshake and trying again.

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 (from most-to-least significant, its bits are record bits 7-1 and then 15-8). 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.

6.1. NTS-KE record types

The following NTS-KE Record Types are defined.

6.1.1. 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.

6.1.2. 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-octet strings. Each 16-octet string represents a Protocol Name from the IANA Network Time Security Next Protocols registry. The Critical Bit MUST be set.

The Protocol Names 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 Names 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.

6.1.3. 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 code 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 DTLS handshake and retry its request.

6.1.4. 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 an 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.

6.1.5. 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 "ntp/4",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 extensions for NTPv4 (Section 7) 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 the "ntp/4" protocol in its NTS Next Protocol Negotiation record, then the server's AEAD Algorithm Negotation record MUST NOT be empty.

6.1.6. 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 [[TODO]] for a RECOMMENDED 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 "ntp/4" as a Next Protocol. The Critical Bit SHOULD NOT be set.

[[Ed. Note: the purpose of sending eight cookies is to allow the client to recover from dropped packets without reusing cookies or starting a new handshake. Discussion of cookie management should probably be broken out into its own section.]]

6.2. 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 16-octet Protocol Name which was negotiated as a Next Protocol.

7. NTS Extensions for NTPv4

7.1. Key Extraction (for NTPv4)

Following a successful run of the NTS-KE protocol wherein "ntp/4" 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 19 octets:

The first 16 octets SHALL be (in hexadecimal):

6E 74 70 2F 34 00 00 00 00 00 00 00 00 00 00 00

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.

7.2. 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 extensions which are authenticated but not encrypted.

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

Possibly, some additional extensions which are neither encrypted nor authenticated. These are discarded by the receiver. [[Ed. Note: right now there's no good reason for the sender to include anything here, but eventually there might be. We've seen Checksum Complement [RFC7821] and LAST-EF as two examples of semantically-void extensions that are included to satsify constraints imposed lower on the protocol stack, and while there's no reason to use either of these on NTS-protected packets, I think we could see similar examples in the future. So, rejecting packets with unauthenticated extensions could cause interoperability problems, while accepting and processing those extensions would of course be

a security risk. Thus, I think "allow and discard" is the correct policy.]]

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

7.3. The Unique Identifier extension

The Unique Identifier extension has a Field Type of [[TBD]]. When the extension is included in a client packet (mode 3), its body SHALL consist of a string of octets generated uniformly at random. The string SHOULD be 32 octets long. When the extension 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 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 enables a degree of unpredictability and collision-resistance more consistent with cryptographic best practice.

[[TODO: consider using separate extension types for request and response, thus allowing for use in symmetric mode. But proper handling in the presence of dropped packets needs to be documented and involves a lot of subtlety.]]

7.4. The NTS Cookie extension

The NTS Cookie extension has a Field Type of [[TBD]]. 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 [[TODO]] for a RECOMMENDED construction. The NTS Cookie extension MUST NOT be included in NTP packets whose mode is other than 3 (client) or 4 (server).

7.5. The NTS Cookie Placeholder extension

The NTS Cookie Placeholder extension has a Field Type of [[TBD]]. When this extension 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 MUST NOT be included in NTP packets whose mode is other than 3.

Whenever an NTS Cookie Placeholder extension is present, it MUST be accompanied by an NTS Cookie extension, and the body length of the NTS Cookie Placeholder extension MUST be the same as the body length of the NTS Cookie Extension. (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's body are undefined and, aside from checking its length, MUST be ignored by the server.

7.6. The NTS Authenticator and Encrypted Extensions extension

The NTS Authenticator and Encrypted Extensions extension is the central cryptographic element of an NTS-protected NTP packet. Its Field Type is [[TBD]] 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.

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: between 1 and 24 octets of padding, with every octet set to the number of padding octets included, e.g., "01", "02 02", or "03 03 03". The number of padding bytes should be chosen in order to comply with the RFC 7822 [RFC7822] requirement that (in the absence of a legacy MAC) extensions have a total length in octets (including the four octets for the type and length fields) which is at least 28 and divisible by 4. At least one octet of padding MUST be included, so that implementations can unambiguously delimit the end of the ciphertext from the start of the padding.

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 which precedes the NTS Authenticator and Encrypted Extensions extension.

P: The plaintext SHALL consist of all (if any) extensions to be encrypted.

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

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

7.7. Protocol details

A client sending an NTS-protected request SHALL include the following extensions:

Exactly one Unique Identifier extension, which MUST be authenticated and MUST NOT be encrypted [[Ed. Note: so that if the server can't decrypt the request, it can still echo back the Unique Identifier in the NTS NAK it sends]]. MUST NOT duplicate those of any previous request.

Exactly one NTS Cookie extension, 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 Extensions extension, generated using an AEAD Algorithm and C2S key established through NTS-KE.

The client MAY include one or more NTS Cookie Placeholder extensions, which MUST be authenticated and MAY be encrypted. The number of NTS Cookie Placeholder extensions 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 extensions will equal the number of dropped packets since the last successful volley.

The client MAY include additional (non-NTS-related) extensions, which MAY appear prior to the NTS Authenticator and Encrypted Extensions extension (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 extensions and process the packet as though they were not present. Servers MAY implement exceptions to this requirement for particular extensions 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 extensions in its response:

- Exactly one Unique Identifier extension, which MUST be authenticated, MUST NOT be encrypted, and whose contents SHALL echo those provided by the client.

- Exactly one NTS Authenticator and Encrypted Extensions extension, generated using the AEAD algorithm and S2C key recovered from the cookie provided by the client.

- One or more NTS Cookie extensions, which MUST be authenticated and encrypted. The number of NTS Cookie extensions included SHOULD be equal to, and MUST NOT exceed, one plus the number of valid NTS Cookie Placeholder extensions included in the request.

The server MAY include additional (non-NTS-related) extensions, which MAY appear prior to the NTS Authenticator and Encrypted Extensions extension (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 extensions and process the packet as though they were not present. Clients MAY implement exceptions to this requirement for particular extensions 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 whose contents SHALL echo those provided by the client. It MUST NOT include any NTS Cookie or NTS Authenticator and Encrypted Extensions extension. [[Ed. Note: RFC 5905 already provides the kiss code "CRYP" meaning "Cryptographic authentication or identification failed" but I think this is meant to be Autokey-specific.]]

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 discard all cookies and AEAD keys associated with the server which sent the NAK and initiate a fresh NTS-KE handshake.

8. Recommended format for NTS cookies

This section provides a RECOMMENDED way for servers to construct NTS cookies. Clients MUST NOT examine the cookie under the assumption that it is constructed according to this section.

The role of cookies in NTS is closely analagous 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 accidential 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.

[[TODO: discuss key management considerations for load-balanced servers]]

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 the 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)`.

[[TODO: explicitly specify how to verify and decrypt a cookie, not just how to form one]]

9. Security Considerations

[[TODO. Outline follows.]]

Cite RFC 7384 [RFC7384] for general considerations.

State security goals (authentication (defined in terms of agreement), client privacy) and threat model (active network adversary).

Incorporate content from "What Makes NTP Cryptographically Exceptional?" of NTS design essay.

Address strategies for management of AEAD nonces and stress importance of avoiding repetition.

Give recommendations for validating X.509 certificates during the DTLS handshake. Discuss what to expect for the CN/SANs, and how to deal with verifying the validity period if correct time is not yet known.

Caution that NTS will not prevent an adversary from skewing time by up to MAXDIST/2 and discuss why this limitation is fundamental.

Possibly include informal security proofs.

10. IANA Considerations

IANA is requested to allocate an entry in the Service Name and Transport Protocol Port Number Registry as follows:

Service Name: nts

Transport Protocol: udp

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: Network Time Security

Reference: [[this memo]]

Port Number: selected by IANA from the user port range

IANA is requested to allocate the following two entries 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]]

Protocol: Network Time Protocol, version 4

Identification Sequence:
0x6E 0x74 0x70 0x2F 0x34 ("ntp/4")

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 entries in the registry of NTP Kiss-o'-Death codes:

Code	Meaning
DTLS	Packet conveys a DTLS record
NTSN	NTS NAK

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

Field Type	Meaning	Reference
[[TBD]]	DTLS Record	[[this memo]]
[[TBD]]	Unique Identifier	[[this memo]]
[[TBD]]	NTS Cookie	[[this memo]]
[[TBD]]	NTS Cookie Placeholder	[[this memo]]
[[TBD]]	NTS Authenticator and Encrypted Extensions	[[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): 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: Standards Action

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 negotation	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 Name (REQUIRED): a sequence of 16 octets. Shorter sequences SHALL implicitly be right-padded with null octets (0x00).

Human-Readable Name (OPTIONAL): if the sequence of octets making up the protocol name intentionally represent a valid UTF-8 [RFC3629] string, this field SHALL consist of that string.

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 on a First Come, First Serve basis. Protocol Names beginning with 0x78 0x2D ("x-") SHALL be reserved for Private or Experimental Use, and SHALL NOT be registered. The reserved entry "ptp/2" may be updated or released by a future Standards Action.

The initial contents of this registry SHALL be as follows:

Protocol Name	Human-Readable Name	Reference
0x6E 0x74 0x70 0x2F 0x34	ntp/4	[[this memo]]
0x70 0x74 0x70 0x2F 0x32	ptp/2	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): a 16-bit unsigned integer

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: Standards Action

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.

11. References

11.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5905>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC7465] Popov, A., "Prohibiting RC4 Cipher Suites", RFC 7465, DOI 10.17487/RFC7465, February 2015, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7627>.

[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, <http://www.rfc-editor.org/info/rfc7301>.

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

11.2. Informative References

[IEC.61588_2009]
IEEE/IEC, "Precision clock synchronization protocol for networked measurement and control systems", IEEE 1588-2008(E), IEC 61588:2009(E), DOI 10.1109/IEEESTD.2009.4839002, February 2009, <http://ieeexplore.ieee.org/servlet/opac?punumber=4839000>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc7384>.

[RFC7821] Mizrahi, T., "UDP Checksum Complement in the Network Time Protocol (NTP)", RFC 7821, DOI 10.17487/RFC7821, March 2016, <http://www.rfc-editor.org/info/rfc7821>.

Appendix A. Acknowledgements

The author gratefully acknowledges the following contributors: Richard Barnes, Prof. Sharon Goldberg, Miroslav Lichvar, Aanchal Malhotra, Danny Mayer, Karen O'Donoghue, Eric K. Rescorla, Stephen Roettger, Kyle Rose, Rich Salz, Dieter Sibold, Brian Sniffen, Susan Sons, Douglas Stebila, Harlan Stenn, Kristof Teichel, and Martin Thomson.

Author's Address

Daniel Fox Franke
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA 02142
United States

Email: dafranke@akamai.com
URI: https://www.dfranke.us