

Internet Working Group

Internet Draft

Intended status: Informational

Expires: April 2016

L. Han

China Mobile

Y. Jiang

J. Xu

X. Liu

Huawei

October 20, 2015

Problem Statements of Scalable Synchronization Networks
draft-hjxl-ssn-ps-00.txt

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on April 20, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

With the wide deployment of 4G and beyond mobile networks, a great number of cells need high precision frequency and/or time synchronization for their normal operation. It is crucial to manage the synchronization network in a scalable way and simplify the monitoring and operation for synchronization networks. This document analyzes the use cases and requirements in synchronization networks, and provides a problem statement for scalable synchronization networks.

Table of Contents

1.	Introduction	2
1.1.	Conventions used in this document	4
1.2.	Terminology	4
2.	Use cases for scalable synchronization network	4
2.1.	Synchronization configuration	4
2.2.	Synchronization OAM	5
2.3.	Synchronization network protection and recovery	6
2.4.	Multi-layer/Multi-domain synchronization network	7
3.	Synchronization Requirements	7
4.	Security Considerations	8
5.	IANA Considerations	8
6.	References	8
6.1.	Normative References	8
6.2.	Informative References	8
7.	Acknowledgments	9

1. Introduction

In modern communication networks, most telecommunication services require that the frequency or phase difference between the whole network equipments should be kept within the reasonable range. Especially for mobile networks, there is a requirement for high precision network clock synchronization, including frequency synchronization and phase synchronization.

For packet switching networks, SyncE and IEEE 1588v2 protocols are widely deployed for frequency and time synchronization respectively in mobile network. Synchronization path planning and provisioning are very complex as so many parameters (e.g., quality level, priority,

synchronization enable/disable, hop limit, holdover timeout, and etc) need to be configured. Furthermore, configuration of SyncE must not introduce any loops in the synchronization paths. Hence, deployment of synchronization network requires professional skills in synchronization protocols and also the engineering capability in analyzing and planning the network topology.

With the deployment of 4G network, the density of cells is explosively growing, as a result, the size of mobile networks and its backhaul network has greatly increased (it may consist of tens of thousands of network equipments in a single metro city). This scalability requirement will pose a great challenge to realize synchronization, and the management and monitoring of the synchronization network becomes dramatically more complex for service providers.

In the past, management and monitoring of synchronization networks are mainly resorted to manual configuration and manual diagnosis, which are complex, error-prone and very time-consuming. Thus it is hard to avoid synchronization loops, erroneous configuration and other mistakes. Therefore, it is important to provide some tools to improve the efficiency of fault monitoring and detection in synchronization networks.

As the synchronization is critical for the mobile services, it will be beneficial to provide path protection for synchronization networks, so that single point of synchronization failure can be avoided (or even provide multipoint protection as much as possible, i.e., even when the working path and a protection synchronization path are both lost, the network can figure out a new synchronization path so that frequency source is still available. This may require that a third synchronization port be configured as a recovery port).

Furthermore, as the mobile network size increases dramatically, the synchronization performance is hard to be satisfied, e.g., care must be taken to guarantee that a certain hop limit (e.g. 20 hops) of time-distribution from the timing source to a cell site is not exceeded.

This document provides some use cases and requirements on configuration and management of a large synchronization network and provides problem statements for scalable synchronization networks.

1.1. Conventions used in this document

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 [RFC2119].

1.2. Terminology

OAM: Operation Administration and Maintenance

BMCA: best master clock algorithm

T-GM: Telecom Grandmaster, a device consisting of a Boundary Clock as defined in [IEEE-1588], with additional performance characteristics defined in [G.8273.2].

2. Use cases for scalable synchronization network

Following are some use cases of scalable synchronization networks from a management and operation viewpoint.

2.1. Synchronization configuration

In a huge mobile backhaul network with more than 10,000 nodes, manual planning and provisioning of synchronization network are very onerous. For example, manual planning and configuration for a simple network may need more than several weeks; furthermore, it is error-prone. And the planning can't eliminate the risk of introducing loops to a synchronization network.

To facilitate synchronization configuration, a central controller may be introduced. The controller shall automatically compute, plan and provision the synchronization paths based on the overall physical network topology, thus it can eliminate the risks associated with manual planning.

A typical controller for synchronization network can compute and provision a synchronization network with tens of thousands of nodes in just a few minutes, and it is guaranteed that no synchronization loop will be introduced if the algorithm is correctly implemented. Synchronization configuration via a centralized controller requires that the controller be highly efficient, agile and reliable.

To accommodate for different types of equipment implementations, a common interface is needed for synchronization network configuration and management, it can further provide the ability to retrieve the network's synchronization configuration and states of a protocol engine in a device. For example, whether the device is locked or not, what is the port state of PTP port (i.e., master, slave or passive), the current port ID associated with a frequency source in syncE, and etc... This capability is essential for the management and maintenance of synchronization networks.

2.2. Synchronization OAM

In the maintenance of a huge synchronization network, an operator may encounter various synchronization problems. The traditional manual trouble shooting hop by hop is very onerous. Even if the malfunction equipments are located in a single operator network, the fault detection procedure is very tedious, let alone in the case of network interworking with a third party.

Traditionally, synchronization fault detection is done by checking synchronization devices on a path one by one manually. I.e., an operator must login to the device (i.e. the device is adjacent to the fault base station or the device nearest to the base station among the devices with the clock alarm), read the configuration information, status and clock alarms information. After analyzing all the information, if the operator still can't locate the source for the fault, the operator must find the upstream device according to the synchronization status information (i.e. the port state of 1588v2 and the current tracing clock port ID of syncE). The operator must login to each upstream device and check the synchronization information one by one, until the source device of the synchronization fault is found.

If the operator cannot locate the fault by the current limited information from the equipments, the operator may have to test the synchronization performance manually by instrument.

This procedure requires that the operator must have a deep understanding of the synchronization protocols and principle of synchronization engineering. And it also is very time-consuming, and sometimes, detecting a single clock fault may even cost up to ten days.

Sometimes the clock synchronization performance of base station degraded but no clock alarm is raised. Through synchronization fault detection an operator cannot locate the true reason of service

disruption. In that case, synchronization performance monitoring may solve the problem by dynamically monitoring the synchronization performance of all devices in the clock synchronization path for a base station in problem.

Therefore, the functions of synchronization OAM shall include synchronization fault detection and synchronization performance monitoring, both are vital in the diagnosis of a synchronization network.

2.3. Synchronization network protection and recovery

If a synchronization path is broken or degraded, it will seriously influence the clock performance of the synchronization network, and further affect the other services of the mobile network. Thus protection and recovery of the synchronization network are very necessary.

In general, if allowed by the network topology, the equipment should be provisioned with a working and a protection synchronization path for SyncE in a mobile network. Thus, the equipments in the mobile network can realize synchronization protection with both the working and backup clock ports.

Even when neither the clock signal on the working port nor on the backup port is available (i.e. loss of signal or degrade of SNR (Signal to Noise Ratio)), the equipment shall not lose the timing source if there is connectivity to it. Ideally, the equipment should select a third port with normal clock signal as a recovery port. And the clock signal of the recovery port mustn't be from the equipment itself (otherwise, a loop will be formed). When the clock signal of the working port or backup port returns to normal, the device may restore to the working or backup port.

In the time synchronization with the IEEE 1588v2, multiple time synchronization ports of the device should be enabled. Through the BMCA automatically selecting the time source can realize the protection and recovery of the time source.

Central controller can also be a solution choice for this use case, for example, provisioning and configuration of the recovery port in advance or dynamic computation and configuration of the recovery port on the fly.

2.4. Multi-layer/Multi-domain synchronization network

In general, to guarantee the time synchronization accuracy, the suggested hop restriction value from the frequency source to the end equipment is 20 in the synchronization network. And the suggested hop restriction value from the time source to the end equipment is 30. The values may be defined differently for different operators.

As tens of thousands of equipments needs to be supported in the same synchronization network, the planning, maintenance and performance of synchronization network face new challenges, for example, the end equipments may hardly satisfy the hop restriction in synchronization. Hierarchical division of a huge synchronization network into multi-layers and/or multi-domains may improve the scalability. For example, the whole synchronization network can be divided into several domains according to their locations.

The operators may also face new challenges after introducing the multi-layer/multi-domain synchronization network, for example, the synchronization OAM for the inter-domain synchronization network is more complex. In the deployment of syncE, the clock fault or performance degradation of edge devices in one domain may even influence the devices of other adjacent domains.

3. Synchronization Requirements

In order to facilitate the provision and management of a large synchronization network, the following requirements need to be addressed:

- a) The synchronization network should support a generic, vendor-independent and protocol-neutral information model for synchronization to support heterogeneous networks;
- b) The synchronization network should support automatic configuration of frequency and time synchronization parameters based on the generic information model, which may requires a generic configuration interface;
- c) The synchronization network should provide high reliability and resiliency, which requires that each synchronization device should maintain at least two useable timing source and switch to an alternate timing source automatically when faults occur in the network; furthermore, a device should restore to the working path when the working path is recovered.
- d) The synchronization network should provide high scalability, which may require a network supports to be divided into multiple logical domains defining the scope of synchronization distribution, or require a

synchronization protocol to maintain high precision timing signal along a long synchronization path. From the management viewpoint, the network is required to support provision and management by a central controller (even for multi-layer/multi-domain case), or each synchronization device should adjust its timing source automatically when the network adds or removes devices;

- e) The synchronization network should provide distributed signaling and centralized signaling to support the traditional network architecture and the innovative SDN architecture;
- f) The synchronization network should provide flexible OAM (Operation Administration and Maintenance) functions for synchronization, such as troubleshooting and synchronization performance monitoring, which can be called on demand if the requested timing performance is not met.

4. Security Considerations

It will be considered in a future revision.

5. IANA Considerations

There are no IANA actions required by this document.

6. References

6.1. Normative References

[IEEE-1588] IEEE 1588, Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2008

6.2. Informative References

[G.8261] ITU-T, Timing and synchronization aspects in packet networks, August, 2013

[G.8275] ITU-T, Architecture and requirements for packet-based time and phase distribution, November, 2013

[ptp-mib] Shankarkumar, V., Montini, L., Frost, T., and Dowd, G.,
Precision Time Protocol Version 2 (PTPv2) Management
Information Base, draft-ietf-tictoc-ntp-mib-06, work in
progress

7. Acknowledgments

TBD

Authors' Addresses

Liuyan Han
China Mobile
Xuanwumenxi Ave, Xuanwu District
Beijing 100053, China
Email: hanliuyan@chinamobile.com

Yuanlong Jiang
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
Email: jiangyuanlong@huawei.com

Jinchun Xu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
Email: xujinchun@huawei.com

Xian Liu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
Email: lene.liuxian@huawei.com

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 28, 2016

D. Sibold
K. Teichel
PTB
S. Roettger
Google Inc.
R. Housley
Vigil Security
February 25, 2016

Protecting Network Time Security Messages with the Cryptographic Message
Syntax (CMS)
draft-ietf-ntp-cms-for-nts-message-06

Abstract

This document describes a convention for using the Cryptographic Message Syntax (CMS) to protect the messages in the Network Time Security (NTS) protocol. NTS provides authentication of time servers as well as integrity protection of time synchronization messages using Network Time Protocol (NTP) or Precision Time Protocol (PTP).

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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 August 28, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. CMS Conventions for NTS Message Protection	3
2.1. Fields of the employed CMS Content Types	5
2.1.1. ContentInfo	5
2.1.2. SignedData	6
2.1.3. EnvelopedData	8
3. Implementation Notes: ASN.1 Structures and Use of the CMS . .	9
3.1. Preliminaries	9
3.2. Unicast Messages	9
3.2.1. Access Messages	9
3.2.2. Association Messages	10
3.2.3. Cookie Messages	11
3.2.4. Time Synchronization Messages	12
3.3. Broadcast Messages	13
3.3.1. Broadcast Parameter Messages	13
3.3.2. Broadcast Time Synchronization Message	14
3.3.3. Broadcast Keycheck	14
4. Certificate Conventions	15
5. IANA Considerations	16
5.1. SMI Security for S/MIME Module Identifier Registry . . .	16
5.2. SMI Security for S/MIME CMS Content Type Registry . . .	16
5.3. SMI Security for PKIX Extended Key Purpose Registry . . .	17
6. Security Considerations	17
7. Acknowledgements	17
8. References	17
8.1. Normative References	17
8.2. Informative References	18
Appendix A. ASN.1 Module	18
Authors' Addresses	18

1. Introduction

This document provides details on how to construct NTS messages in practice. NTS provides secure time synchronization with time servers using Network Time Protocol (NTP) [RFC5905] or Precision Time Protocol (PTP) [IEEE1588]. Among other things, this document describes a convention for using the Cryptographic Message Syntax (CMS) [RFC5652] to protect messages in the Network Time Security (NTS) protocol. Encryption is used to provide confidentiality of secrets, and digital signatures are used to provide authentication and integrity of content.

Sometimes CMS is used in an exclusively ASN.1 [ASN1] environment. In this case, the NTS message may use any syntax that facilitates easy implementation.

2. CMS Conventions for NTS Message Protection

Regarding the usage of CMS, we differentiate between three archetypes according to which the NTS message types can be structured. They are presented below. Note that the NTS Message Object that is at the core of each structure does not necessarily contain all the data needed for the particular message type, but may contain only that data which needs to be secured directly with cryptographic operations using the CMS. Specific information about what is included can be found in Section 3.

NTS-Plain: This archetype is used for actual time synchronization messages (explicitly, the following message types: `time_request`, `time_response`, `server_broad`, see [I-D.ietf-ntp-network-time-security], Section 6) as well as for client-side messages of all unicast and broadcast bootstrapping exchanges (explicitly `client_assoc`, `client_cook` and `client_bpar`) as well as the broadcast keycheck exchange (`client_keycheck` and `server_keycheck`). This archetype does not make use of any CMS structures at all. Figure 1 illustrates this structure.



NTS-Encrypted-and-Signed: This archetype is used for secure transmission of the cookie (only for the `server_cook` message type,

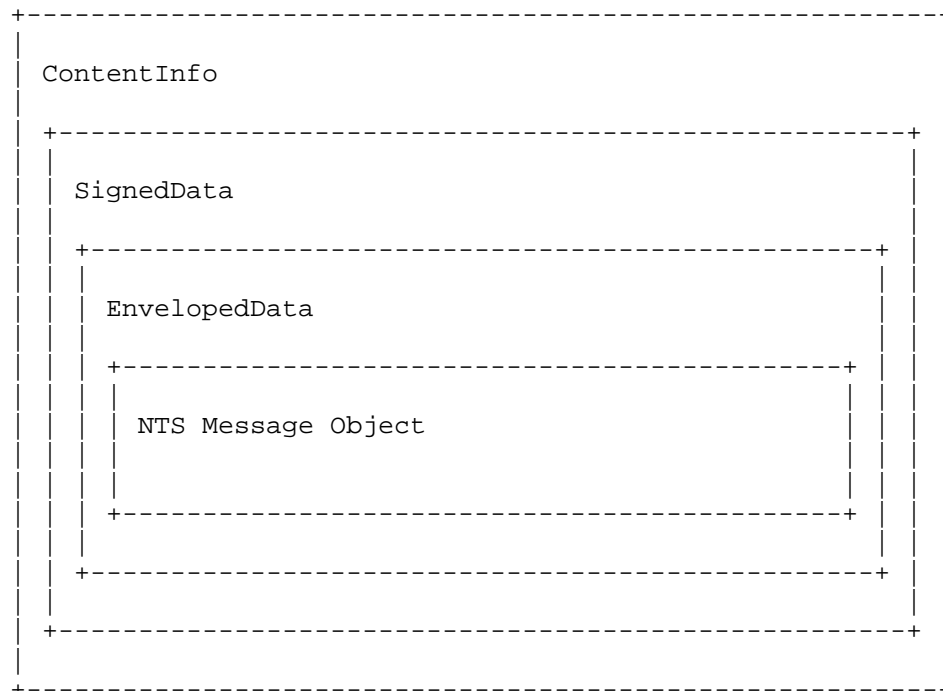
see [I-D.ietf-ntp-network-time-security], Section 6). For this, the following CMS structure is used:

First, the NTS message MUST be encrypted using the EnvelopedData content type. EnvelopedData supports nearly any form of key management. In the NTS protocol the client provides a certificate in an unprotected message, and the public key from this certificate, if it is valid, will be used to establish a pairwise symmetric key for the encryption of the protected NTS message.

Second, the EnvelopedData content MUST be digitally signed using the SignedData content type. SignedData supports nearly any form of digital signature, and in the NTS protocol the server will include its certificate within the SignedData content type.

Third, the SignedData content type MUST be encapsulated in a ContentInfo content type.

Figure 2 illustrates this structure.

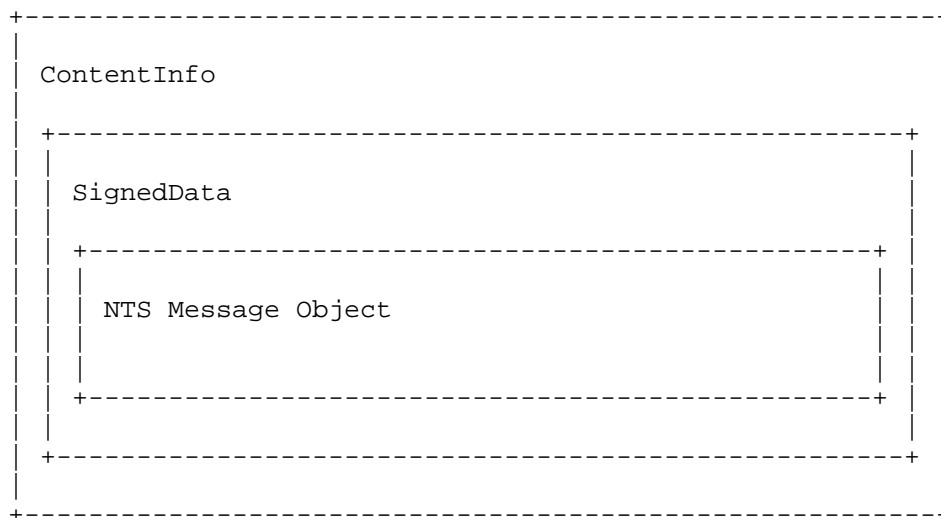


NTS-Signed: This archetype is used for `server_assoc` and `server_bpar` message types. It uses the following CMS structure:

First, the NTS message object MUST be wrapped in a `SignedData` content type. The messages MUST be digitally signed, and certificates included. `SignedData` supports nearly any form of digital signature, and in the NTS protocol the server will include its certificate within the `SignedData` content type.

Second, the `SignedData` content type MUST be encapsulated in a `ContentInfo` content type.

Figure 3 illustrates this structure.



2.1. Fields of the employed CMS Content Types

Overall, three CMS content types are used for NTS messages by the archetypes above. Explicitly, those content types are `ContentInfo`, `SignedData` and `EnvelopedData`. The following is a description of how the fields of those content types are used in detail.

2.1.1. ContentInfo

The `ContentInfo` content type is used in all archetypes except `NTS-Plain`. The fields of the `ContentInfo` content type are used as follows:

`contentType` -- indicates the type of the associated content. For all archetypes which use `ContentInfo` (these are `NTS-Signed` and

NTS-Encrypted-and-Signed), it MUST contain the object identifier for the SignedData content type:

```
id-signedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }
```

content -- is the associated content. For all archetypes using ContentInfo, it MUST contain the DER encoded SignedData content type.

2.1.2. SignedData

The SignedData content type is used in the NTS-Signed and NTS-Encrypted-and-Signed archetypes, but not in the NTS-Plain archetype. The fields of the SignedData content type are used as follows:

version -- the appropriate value depends on the optional items that are included. In the NTS protocol, the signer certificate MUST be included and other items MAY be included. The instructions in [RFC5652] Section 5.1 MUST be followed to set the correct value.

digestAlgorithms -- is a collection of message digest algorithm identifiers. In the NTS protocol, there MUST be exactly one algorithm identifier present. The instructions in Section 5.4 of [RFC5652] MUST be followed.

encapContentInfo -- this structure is always present. In the NTS protocol, it MUST follow these conventions:

eContentType -- is an object identifier. In the NTS protocol, for the NTS-Signed archetype, it MUST identify the type of the NTS message that was encapsulated. For the NTS-Encrypted-and-Signed archetype, it MUST contain the object identifier for the EnvelopedData content type:

```
id-envelopedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3 }.
```

eContent is the content itself, carried as an octet string. For the NTS-Signed archetype, it MUST contain the DER encoded encapsulated NTS message object. The instructions in Section 6.3 of [RFC5652] MUST be followed. For the NTS-Encrypted-and-Signed archetype, it MUST contain the DER encoded EnvelopedData content type.

certificates -- is a collection of certificates. In the NTS protocol, it MUST contain the DER encoded certificate [RFC5280] of

the sender. It is intended that the collection of certificates be sufficient for the recipient to construct a certification path from a recognized "root" or "top-level certification authority" to the certificate used by the sender.

crls -- is a collection of revocation status information. In the NTS protocol, it MAY contain one or more DER encoded CRLs [RFC5280]. It is intended that the collection contain information sufficient to determine whether the certificates in the certificates field are valid.

signerInfos -- is a collection of per-signer information. In the NTS protocol, for the NTS-Signed and the NTS-Encrypted-and-Signed archetypes, there MUST be exactly one SignerInfo structure present. The details of the SignerInfo type are discussed in Section 5.3 of [RFC5652]. In the NTS protocol, it MUST follow these conventions:

version -- is the syntax version number. In the NTS protocol, the SignerIdentifier is subjectKeyIdentifier, therefore the version MUST be 3.

sid -- identifies the signer's certificate. In the NTS protocol, the "sid" field contains the subjectKeyIdentifier from the signer's certificate.

digestAlgorithm -- identifies the message digest algorithm and any associated parameters used by the signer. In the NTS protocol, the identifier MUST match the single algorithm identifier present in the digestAlgorithms.

signedAttrs -- is a collection of attributes that are signed. In the NTS protocol, it MUST be present, and it MUST contain the following attributes:

Content Type -- see Section 11.1 of [RFC5652].

Message Digest -- see Section 11.2 of [RFC5652].

In addition, it MAY contain the following attributes:

Signing Time -- see Section 11.3 of [RFC5652].

Binary Signing Time -- see Section 3 of [RFC5652].

signatureAlgorithm -- identifies the signature algorithm and any associated parameters used by the signer to generate the digital signature.

signature is the result of digital signature generation using the message digest and the signer's private key. The instructions in Section 5.5 of [RFC5652] MUST be followed.

unsignedAttrs -- is an optional collection of attributes that are not signed. In the NTS protocol, it MUST be absent.

2.1.3. EnvelopedData

The EnvelopedData content type is used only in the NTS-Encrypted-and-Signed archetype. The fields of the EnvelopedData content type are used as follows:

version -- the appropriate value depends on the type of key management that is used. The instructions in [RFC5652] Section 6.1 MUST be followed to set the correct value.

originatorInfo -- this structure is present only if required by the key management algorithm. In the NTS protocol, it MUST be present when a key agreement algorithm is used, and it MUST be absent when a key transport algorithm is used. The instructions in Section 6.1 of [RFC5652] MUST be followed.

recipientInfos -- this structure is always present. In the NTS protocol, it MUST contain exactly one entry that allows the client to determine the key used to encrypt the NTS message. The instructions in Section 6.2 of [RFC5652] MUST be followed.

encryptedContentInfo -- this structure is always present. In the NTS protocol, it MUST follow these conventions:

contentType -- indicates the type of content. In the NTS protocol, it MUST identify the type of the NTS message that was encrypted.

contentEncryptionAlgorithm -- identifies the content-encryption algorithm and any associated parameters used to encrypt the content.

encryptedContent -- is the encrypted content. In the NTS protocol, it MUST contain the encrypted NTS message. The instructions in Section 6.3 of [RFC5652] MUST be followed.

unprotectedAttrs -- this structure is optional. In the NTS protocol, it MUST be absent.

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

This section presents some hints about the structures of the NTS message objects for the different message types when one wishes to implement the security mechanisms.

3.1. Preliminaries

The following ASN.1 coded data types "NTSAccessKey", "NTSNonce", and "NTSVersion" are needed for other types used below for NTS messages. "NTSAccessKey" specifies an access key, which is required for limitation of client association requests.

```
NTSAccessKey ::= OCTET STRING (SIZE(16))
```

"NTSNonce" specifies a 128 bit nonce as required in several message types.

```
NTSNonce ::= OCTET STRING (SIZE(16))
```

"NTSVersion" specifies a version number, which is required for version negotiation.

```
NTSVersion ::= INTEGER (0..255)
```

The following ASN.1 coded data types are also necessary for other types.

```
KeyEncryptionAlgorithmIdentifiers ::=  
    SET OF KeyEncryptionAlgorithmIdentifier
```

```
ContentEncryptionAlgorithmIdentifiers ::=  
    SET OF ContentEncryptionAlgorithmIdentifier
```

3.2. Unicast Messages

3.2.1. Access Messages

3.2.1.1. Message Type: "client_access"

This message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientAccessData" and structured as follows:

```
ClientAccessData ::= NULL
```

It is identified by the following object identifier:

id-ct-nts-clientAccess OBJECT IDENTIFIER ::= TBD1

3.2.1.2. Message Type: "server_access"

This message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ServerAccessData" and structured as follows:

```
ServerAccessData ::= SEQUENCE {  
    accessKey          NTSAccessKey  
}
```

It is identified by the following object identifier:

id-ct-nts-serverAccess OBJECT IDENTIFIER ::= TBD2

3.2.2. Association Messages

3.2.2.1. Message Type: "client_assoc"

This message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientAssocData" and structured as follows:

```
ClientAssocData ::= SEQUENCE {  
    accessKey          NTSAccessKey,  
    nonce              NTSNonce,  
    minVersion         NTSVersion,  
    hmacHashAlgos      DigestAlgorithmIdentifiers,  
    keyEncAlgos         KeyEncryptionAlgorithmIdentifiers,  
    contentEncAlgos     ContentEncryptionAlgorithmIdentifiers  
}
```

It is identified by the following object identifier:

id-ct-nts-clientAssoc OBJECT IDENTIFIER ::= TBD3

3.2.2.2. Message Type: "server_assoc"

This message is structured according to the NTS-Signed archetype. It requires additional data besides that which is transported in the NTS message object, namely the signature and certificate chain are included in the appropriate fields of the "SignedData" CMS structure that the NTS message object is wrapped in. The NTS message object itself is an ASN.1 object of type "ServerAssocData" and structured as follows:

```
ServerAssocData ::= SEQUENCE {  
    nonce                NTSNonce,  
    proposedVersion      NTSVersion,  
    hmacHashAlgos        DigestAlgorithmIdentifiers,  
    choiceHmacHashAlgo    DigestAlgorithmIdentifier,  
    keyEncAlgos           KeyEncryptionAlgorithmIdentifiers,  
    choiceKeyEncAlgo      KeyEncryptionAlgorithmIdentifier,  
    contentEncAlgos       ContentEncryptionAlgorithmIdentifiers,  
    choiceContentEncAlgo  ContentEncryptionAlgorithmIdentifier  
}
```

It is identified by the following object identifier:

id-ct-nts-serverAssoc OBJECT IDENTIFIER ::= TBD4

3.2.3. Cookie Messages

3.2.3.1. Message Type: "client_cook"

This message is structured according to the NTS-Plain archetype. It requires no additional data besides that which is transported in the NTS message object. The NTS message object itself is an ASN.1 object of type "ClientCookieData" and structured as follows:

```
ClientCookieData ::= SEQUENCE {  
    nonce                NTSNonce,  
    signAlgo             SignatureAlgorithmIdentifier,  
    hmacHashAlgo         DigestAlgorithmIdentifier,  
    encAlgo              ContentEncryptionAlgorithmIdentifier,  
    keyEncAlgo           KeyEncryptionAlgorithmIdentifier,  
    certificates          CertificateSet  
}
```

It is identified by the following object identifier:

id-ct-nts-clientCookie OBJECT IDENTIFIER ::= TBD5

3.2.3.2. Message Type: "server_cook"

This message is structured according to the "NTS-Encrypted-and-Signed" archetype. It requires additional data besides that which is transported in the NTS message object, namely the signature is included in the appropriate field of the "SignedData" CMS structure that the NTS message object is wrapped in. The NTS message object itself is an ASN.1 sequence of type "ServerCookieData" and structured as follows:

```
ServerCookieData ::= SEQUENCE {  
    nonce      NTSNonce,  
    cookie     OCTET STRING (SIZE(16))  
}
```

It is identified by the following object identifier:

```
id-ct-nts-serverCookie OBJECT IDENTIFIER ::= TBD6
```

3.2.4. Time Synchronization Messages

3.2.4.1. Message Type: "time_request"

This message is structured according to the "NTS-Plain" archetype.

This message type requires additional data to that which is included in the NTS message object, namely it requires regular time synchronization data, as an unsecured packet from a client to a server would contain. Optionally, it requires the Message Authentication Code (MAC) to be generated over the whole rest of the packet (including the NTS message object) and transported in some way. The NTS message object itself is an ASN.1 object of type "TimeRequestSecurityData", whose structure is as follows:

```
TimeRequestSecurityData ::=  
SEQUENCE {  
    nonce          NTSNonce,  
    hmacHashAlgo   DigestAlgorithmIdentifier,  
    keyInputValue  OCTET STRING (SIZE(16))  
}
```

It is identified by the following object identifier:

```
id-ct-nts-securityDataReq OBJECT IDENTIFIER ::= TBD7
```

3.2.4.2. Message Type: "time_response"

This message is structured according to the "NTS-Plain" archetype.

It requires two items of data in addition to that which is transported in the NTS message object. Like "time_request", it requires regular time synchronization data. Furthermore, it requires the Message Authentication Code (MAC) to be generated over the whole rest of the packet (including the NTS message object) and transported in some way. The NTS message object itself is an ASN.1 object of type "TimeResponseSecurityData", with the following structure:

```
TimeResponseSecurityData ::=
SEQUENCE {
    nonce      NTSNonce,
}
```

It is identified by the following object identifier:

```
id-ct-nts-securityDataResp OBJECT IDENTIFIER ::= TBD8
```

3.3. Broadcast Messages

3.3.1. Broadcast Parameter Messages

3.3.1.1. Message Type: "client_bpar"

This first broadcast message is structured according to the NTS-Plain archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "BroadcastParameterRequest" and structured as follows:

```
BroadcastParameterRequest ::=
SEQUENCE {
    nonce      NTSNonce,
    clientId   SubjectKeyIdentifier
}
```

It is identified by the following object identifier:

```
id-ct-nts-broadcastParamReq OBJECT IDENTIFIER ::= TBD9
```

3.3.1.2. Message Type: "server_bpar"

This message is structured according to "NTS-Signed". It requires additional data besides that which is transported in the NTS message object, namely the signature is included in the appropriate field of the "SignedData" CMS structure that the NTS message object is wrapped in. The NTS message object itself is an ASN.1 object of type "BroadcastParameterResponse" and structured as follows:

```
BroadcastParameterResponse ::=
SEQUENCE {
    nonce                NTSNonce,
    oneWayAlgo1          DigestAlgorithmIdentifier,
    oneWayAlgo2          DigestAlgorithmIdentifier,
    lastKey              OCTET STRING (SIZE (16)),
    intervalDuration     BIT STRING,
    disclosureDelay      INTEGER,
    nextIntervalTime     BIT STRING,
    nextIntervalIndex    INTEGER
}
```

It is identified by the following object identifier:

id-ct-nts-broadcastParamResp OBJECT IDENTIFIER ::= TBD10

3.3.2. Broadcast Time Synchronization Message

3.3.2.1. Message Type: "server_broad"

This message is structured according to the "NTS-Plain" archetype. It requires regular broadcast time synchronization data in addition to that which is carried in the NTS message object. Like "time_response", this message type also requires a MAC, generated over all other data, to be transported within the packet. The NTS message object itself is an ASN.1 object of type "BroadcastTime". It has the following structure:

```
BroadcastTime ::=
SEQUENCE {
    thisIntervalIndex    INTEGER,
    disclosedKey         OCTET STRING (SIZE (16)),
}
```

It is identified by the following object identifier:

id-ct-nts-broadcastTime OBJECT IDENTIFIER ::= TBD11

3.3.3. Broadcast Keycheck

3.3.3.1. Message Type: "client_keycheck"

This message is structured according to the "NTS-Plain" archetype. There is no data necessary besides that which is transported in the NTS message object, which is an ASN.1 object of type "ClientKeyCheckSecurityData" and structured as follows:


```
ClientKeyCheckSecurityData ::=
SEQUENCE {
    nonce_k            NTSNonce,
    interval_number    INTEGER,
    hmacHashAlgo       DigestAlgorithmIdentifier,
    keyInputValue      OCTET STRING (SIZE(16))
}
```

It is identified by the following object identifier:

id-ct-nts-clientKeyCheck OBJECT IDENTIFIER ::= TBD12

3.3.3.2. Message Type: "server_keycheck"

This message is also structured according to "NTS-Plain". It requires only a MAC, generated over the NTS message object, to be included in the packet in addition to what the NTS message object itself contains. The latter is an ASN.1 object of type "ServerKeyCheckSecurityData", which is structured as follows:

```
ServerKeyCheckSecurityData ::=
SEQUENCE {
    nonce              NTSNonce,
    interval_number    INTEGER
}
```

It is identified by the following object identifier:

id-ct-nts-serverKeyCheck OBJECT IDENTIFIER ::= TBD13

4. Certificate Conventions

The syntax and processing rules for certificates are specified in [RFC5280]. In the NTS protocol, the server certificate MUST contain the following extensions:

Subject Key Identifier -- see Section 4.2.1.2 of [RFC5280].

Key Usage -- see Section 4.2.1.3 of [RFC5280].

Extended Key Usage -- see Section 4.2.1.12 of [RFC5280].

For a certificate issued to a time server, the Extended Key Usage extension MAY include the id-kp-ntsServerAuth object identifier. When a certificate issuer includes this object identifier in the extended key usage extension, it provides an attestation that the certificate subject is a time server that supports the NTS protocol. The extension MAY also include the id-kp-ntsServerAuthz object

identifier. When a certificate issuer includes this object identifier in the extended key usage extension, it provides an attestation that the certificate subject is a time server which the issuer believes to be willing and able to disseminate correct time (for example, this can be used to signal a server's authorization to disseminate legal time).

For a certificate issued to a time client, the Extended Key Usage extension MAY include the `id-kp-ntsClientAuthz` object identifier. When a certificate issuer includes this object identifier in the extended key usage extension, it provides an attestation that the certificate subject is a time client who has authorization from the issuer to access secured time synchronization (for example, this can be used to provide access in the case of a paid service for secure time synchronization).

5. IANA Considerations

5.1. SMI Security for S/MIME Module Identifier Registry

Within the "SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0)" table, add one module identifier:

Decimal	Description	References
-----	-----	-----
TBD0	<code>id-networkTimeSecurity-2015</code>	[this doc]

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

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

Decimal	Description	References
-----	-----	-----
TBD1	<code>id-ct-nts-clientAccess</code>	[this doc]
TBD2	<code>id-ct-nts-serverAccess</code>	[this doc]
TBD3	<code>id-ct-nts-clientAssoc</code>	[this doc]
TBD4	<code>id-ct-nts-serverAssoc</code>	[this doc]
TBD5	<code>id-ct-nts-clientCookie</code>	[this doc]
TBD6	<code>id-ct-nts-serverCookie</code>	[this doc]
TBD7	<code>id-ct-nts-securityDataReq</code>	[this doc]
TBD8	<code>id-ct-nts-securityDataResp</code>	[this doc]
TBD9	<code>id-ct-nts-broadcastParamReq</code>	[this doc]
TBD10	<code>id-ct-nts-broadcastParamResp</code>	[this doc]
TBD11	<code>id-ct-nts-broadcastTime</code>	[this doc]
TBD12	<code>id-ct-nts-clientKeyCheck</code>	[this doc]
TBD13	<code>id-ct-nts-serverKeyCheck</code>	[this doc]

5.3. SMI Security for PKIX Extended Key Purpose Registry

Within the "SMI Security for PKIX Extended Key Purpose Identifiers (1.3.6.1.5.5.7.3)" table, add three key purpose identifiers:

Decimal	Description	References
-----	-----	-----
TBD14	id-kp-ntsServerAuth	[this doc]
TBD15	id-kp-ntsServerAuthz	[this doc]
TBD16	id-kp-ntsClientAuthz	[this doc]

6. Security Considerations

For authentication the server's certificate MAY contain an extended key purpose identifier (id-kp-ntsServerAuth). Additionally the identifiers id-kp-ntsServerAuthz and id-kp-ntsClientAuthz MAY be used to grant the associated roles to the certified entity in the time dissemination infrastructure (see also Appendix D in [I-D.ietf-ntp-network-time-security]).

7. Acknowledgements

The authors would like to thank Harlan Stenn, Richard Welty and Martin Langer for their technical review and specific text contributions to this document.

8. References

8.1. Normative References

- [ASN1] International Telecommunication Union, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, November 2008.
- [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>>.
- [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, <<http://www.rfc-editor.org/info/rfc5280>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<http://www.rfc-editor.org/info/rfc5652>>.

8.2. Informative References

- [I-D.ietf-ntp-network-time-security]
Sibold, D., Roettger, S., and K. Teichel, "Network Time Security", draft-ietf-ntp-network-time-security-13 (work in progress), February 2016.
- [IEEE1588]
IEEE Instrumentation and Measurement Society. TC-9 Sensor Technology, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems", 2008.
- [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>>.

Appendix A. ASN.1 Module

The ASN.1 module contained in this appendix defines the id-kp-NTSserver object identifier.

```
NTSserverKeyPurpose
{ TBD }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

id-kp-NTSserver OBJECT IDENTIFIER ::= { TBD17 }

END
```

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

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

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

Stephen Roettger
Google Inc.

Email: stephen.roettger@googlemail.com

Russ Housley
Vigil Security
918 Spring Knoll Drive
Herndon, VA 20170

Email: housley@vigilsec.com

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

D. Sibold
PTB
S. Roettger
Google Inc.
K. Teichel
PTB
March 21, 2016

Network Time Security
draft-ietf-ntp-network-time-security-14

Abstract

This document describes Network Time Security (NTS), a collection of measures that enable secure time synchronization with time servers using protocols like the Network Time Protocol (NTP) or the Precision Time Protocol (PTP). Its design considers the special requirements of precise timekeeping which are described in Security Requirements of Time Protocols in Packet Switched Networks [RFC7384].

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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 September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
2.1. Terms and Abbreviations	4
2.2. Common Terminology for PTP and NTP	4
3. Security Threats	5
4. Objectives	5
5. NTS Overview	6
6. Protocol Messages	7
6.1. Unicast Time Synchronisation Messages	7
6.1.1. Preconditions for the Unicast Time Synchronization Exchange	7
6.1.2. Goals of the Unicast Time Synchronization Exchange	8
6.1.3. Message Type: "time_request"	8
6.1.4. Message Type: "time_response"	8
6.1.5. Procedure Overview of the Unicast Time Synchronization Exchange	9
6.2. Broadcast Time Synchronization Exchange	10
6.2.1. Preconditions for the Broadcast Time Synchronization Exchange	10
6.2.2. Goals of the Broadcast Time Synchronization Exchange	11
6.2.3. Message Type: "server_broad"	11
6.2.4. Procedure Overview of Broadcast Time Synchronization Exchange	12
6.3. Broadcast Keycheck	13
6.3.1. Preconditions for the Broadcast Keycheck Exchange	13
6.3.2. Goals of the Broadcast Keycheck Exchange	14
6.3.3. Message Type: "client_keycheck"	14
6.3.4. Message Type: "server_keycheck"	14
6.3.5. Procedure Overview of the Broadcast Keycheck Exchange	15
7. Server Seed, MAC Algorithms and Generating MACs	16
7.1. Server Seed	16

7.2. MAC Algorithms	16
8. IANA Considerations	17
9. Security Considerations	17
9.1. Privacy	17
9.2. Initial Verification of the Server Certificates	18
9.3. Revocation of Server Certificates	18
9.4. Mitigating Denial-of-Service for broadcast packets	18
9.5. Delay Attack	18
9.6. Random Number Generation	20
10. Acknowledgements	20
11. References	20
11.1. Normative References	20
11.2. Informative References	21
Appendix A. (informative) TICTOC Security Requirements	22
Appendix B. (normative) Inherent Association Protocol Messages	23
B.1. Overview of NTS with Inherent Association Protocol	23
B.2. Access Message Exchange	24
B.2.1. Goals of the Access Message Exchange	24
B.2.2. Message Type: "client_access"	24
B.2.3. Message Type: "server_access"	24
B.2.4. Procedure Overview of the Access Exchange	24
B.3. Association Message Exchange	25
B.3.1. Goals of the Association Exchange	25
B.3.2. Message Type: "client_assoc"	25
B.3.3. Message Type: "server_assoc"	26
B.3.4. Procedure Overview of the Association Exchange	26
B.4. Cookie Message Exchange	27
B.4.1. Goals of the Cookie Exchange	28
B.4.2. Message Type: "client_cook"	28
B.4.3. Message Type: "server_cook"	28
B.4.4. Procedure Overview of the Cookie Exchange	29
B.4.5. Broadcast Parameter Messages	30
Appendix C. (normative) Using TESLA for Broadcast-Type Messages	32
C.1. Server Preparation	32
C.2. Client Preparation	34
C.3. Sending Authenticated Broadcast Packets	35
C.4. Authentication of Received Packets	35
Appendix D. (informative) Dependencies	37
Authors' Addresses	39

1. Introduction

Time synchronization protocols are increasingly utilized to synchronize clocks in networked infrastructures. Successful attacks against the time synchronization protocol can seriously degrade the reliable performance of such infrastructures. Therefore, time synchronization protocols have to be secured if they are applied in environments that are prone to malicious attacks. This can be

accomplished either by utilization of external security protocols, like IPsec or TLS, or by intrinsic security measures of the time synchronization protocol.

The two most popular time synchronization protocols, the Network Time Protocol (NTP) [RFC5905] and the Precision Time Protocol (PTP) [IEEE1588], currently do not provide adequate intrinsic security precautions. This document specifies generic security measures which enable these and possibly other protocols to verify the authenticity of the time server/master and the integrity of the time synchronization protocol packets. The utilization of these measures for a given specific time synchronization protocol has to be described in a separate document.

[RFC7384] specifies that a security mechanism for timekeeping must be designed in such a way that it does not degrade the quality of the time transfer. This implies that for time keeping the increase in bandwidth and message latency caused by the security measures should be small. Also, NTP as well as PTP work via UDP and connections are stateless on the server/master side. Therefore, all security measures in this document are designed in such a way that they add little demand for bandwidth, that the necessary calculations can be executed in a fast manner, and that the measures do not require a server/master to keep state of a connection.

2. Terminology

2.1. Terms and Abbreviations

MITM Man In The Middle

NTS Network Time Security

TESLA Timed Efficient Stream Loss-tolerant Authentication

MAC Message Authentication Code

2.2. Common Terminology for PTP and NTP

This document refers to different time synchronization protocols, in particular to both the PTP and the NTP. Throughout the document the term "server" applies to both a PTP master and an NTP server. Accordingly, the term "client" applies to both a PTP slave and an NTP client.

3. Security Threats

The document "Security Requirements of Time Protocols in Packet Switched Networks" [RFC7384] contains a profound analysis of security threats and requirements for time synchronization protocols.

4. Objectives

The objectives of the NTS specification are as follows:

- o Authenticity: NTS enables the client to authenticate its time server(s).
- o Integrity: NTS protects the integrity of time synchronization protocol packets via a message authentication code (MAC).
- o Confidentiality: NTS does not provide confidentiality protection of the time synchronization packets.
- o Authorization: NTS enables the client to verify its time server's authorization. NTS optionally enables the server to verify the client's authorization as well.
- o Request-Response-Consistency: NTS enables a client to match an incoming response to a request it has sent. NTS also enables the client to deduce from the response whether its request to the server has arrived without alteration.
- o Applicability to Protocols: NTS can be used to secure different time synchronization protocols, specifically at least NTP and PTP.
- o Integration with Protocols: A client or server running an NTS-secured version of a time protocol does not negatively affect other participants who are running unsecured versions of that protocol.
- o Server-Side Statelessness: All security measures of NTS work without creating the necessity for a server to keep state of a connection.
- o Prevention of Amplification Attacks: All communication introduced by NTS offers protection against abuse for amplification denial-of-service attacks.

5. NTS Overview

NTS initially verifies the authenticity of the time server and exchanges a symmetric key, the so-called cookie, as well as a key input value (KIV). The KIV can be opaque for the client. After the cookie and the KIV are exchanged, the client then uses them to protect the authenticity and the integrity of subsequent unicast-type time synchronization packets. In order to do this, a Message Authentication Code (MAC) is attached to each time synchronization packet. The calculation of the MAC includes the whole time synchronization packet and the cookie which is shared between client and server.

The cookie is calculated according to:

$$\text{cookie} = \text{MSB}_{}(\text{MAC}(\text{server seed}, \text{KIV})),$$

with the server seed as the key, where KIV is the client's key input value, and where the application of the function $\text{MSB}_{}$ returns only the b most significant bits. The server seed is a random value of bit length b that the server possesses, which has to remain secret. The cookie deterministically depends on KIV as long as the server seed stays the same. The server seed has to be refreshed periodically in order to provide key freshness as required in [RFC7384]. See Section 7 for details on seed refreshing.

Since the server does not keep a state of the client, it has to recalculate the cookie each time it receives a unicast time synchronization request from the client. To this end, the client has to attach its KIV to each request (see Section 6.1).

Note: The communication of the KIV and the cookie can be performed between client and server directly, or via a third party key distribution entity.

For broadcast-type messages, authenticity and integrity of the time synchronization packets are also ensured by a MAC, which is attached to the time synchronization packet by the sender. Verification of the broadcast-type packets' authenticity is based on the TESLA protocol, in particular on its "not re-using keys" scheme, see Section 3.7.2 of [RFC4082]. TESLA uses a one-way chain of keys, where each key is the output of a one-way function applied to the previous key in the chain. The server securely shares the last element of the chain with all clients. The server splits time into intervals of uniform duration and assigns each key to an interval in reverse order. At each time interval, the server sends a broadcast packet appended by a MAC, calculated using the corresponding key, and the key of the previous disclosure interval. The client verifies the

MAC by buffering the packet until disclosure of the key in its associated disclosure interval occurs. In order to be able to verify the timeliness of the packets, the client has to be loosely time synchronized with the server. This has to be accomplished before broadcast associations can be used. For checking timeliness of packets, NTS uses another, more rigorous check in addition to just the clock lookup used in the TESLA protocol. For a more detailed description of how NTS employs and customizes TESLA, see Appendix C.

6. Protocol Messages

This section describes the types of messages needed for secure time synchronization with NTS.

For some guidance on how these message types can be realized in practice, and integrated into the communication flow of existing time synchronization protocols, see [I-D.ietf-ntp-cms-for-nts-message], a companion document for NTS. Said document describes ASN.1 encodings for those message parts that have to be added to a time synchronization protocol for security reasons.

6.1. Unicast Time Synchronisation Messages

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

6.1.1. Preconditions for the Unicast Time Synchronization Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- o They MUST negotiate the algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.
- o The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.
- o Client and server MUST exchange the cookie (which depends on the KIV as described in section Section 5). Authenticity, confidentiality and integrity of the communication MUST be ensured.

One way of realizing these requirements is to use the Association and Cookie Message Exchanges described in Appendix B.

6.1.2. Goals of the Unicast Time Synchronization Exchange

The unicast time synchronization exchange:

- o exchanges time synchronization data as specified by the appropriate time synchronization protocol,
- o guarantees authenticity and integrity of the request to the server,
- o guarantees authenticity and integrity of the response to the client,
- o guarantees request-response-consistency to the client.

6.1.3. Message Type: "time_request"

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

- o the NTS message ID "time_request",
- o the negotiated version number,
- o a nonce,
- o the negotiated MAC algorithm,
- o the client's key input value (for which the client knows the associated cookie),
- o optional: a MAC (generated with the cookie as key) for verification of all of the above data.

6.1.4. Message Type: "time_response"

This message is sent by the server after it has received a time_request message. Prior to this the server MUST recalculate the client's cookie by using the received key input value and the transmitted MAC algorithm. The message contains

- o the NTS message ID "time_response",
- o the version number as transmitted in time_request,
- o the server's time synchronization response data,
- o the nonce transmitted in time_request,

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

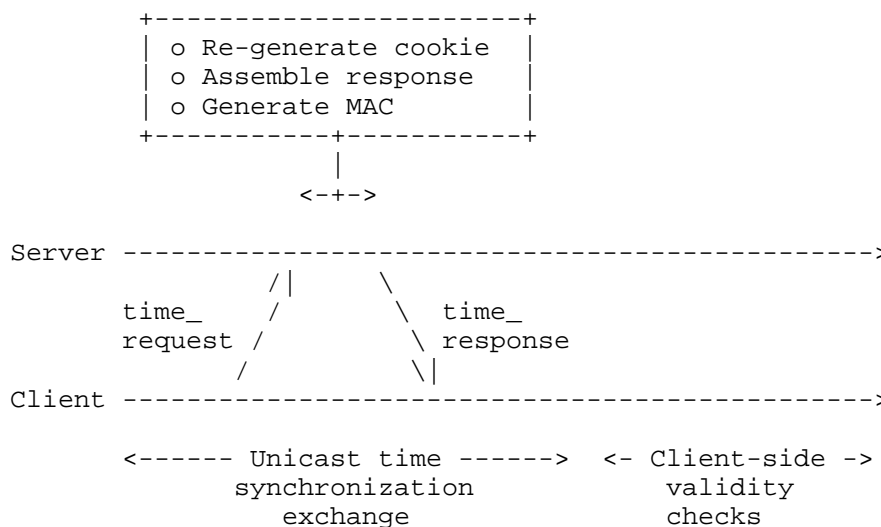
6.1.5. Procedure Overview of the Unicast Time Synchronization Exchange

For a unicast time synchronization exchange, the following steps are performed:

1. The client sends a `time_request` message to the server. The client **MUST** save the included nonce and the `transmit_timestamp` (from the time synchronization data) as a correlated pair for later verification steps. Optionally, the client protects the request message with an appended MAC.
2. Upon receipt of a `time_request` message, the server performs the following steps:
 - * It re-calculates the cookie.
 - * If the request message contains a MAC the server re-calculates the MAC and compares this value with the MAC in the received data.
 - + If the re-calculated MAC does not match the MAC in the received data the server **MUST** stop the processing of the request.
 - + If the re-calculated MAC matches the MAC in the received data the server continues to process the request.
 - * The server computes the necessary time synchronization data and constructs a `time_response` message as given in Section 6.1.4.
3. The client awaits a reply in the form of a `time_response` message. Upon receipt, it checks:
 - * that the transmitted version number matches the one negotiated previously,
 - * that the transmitted nonce belongs to a previous `time_request` message,
 - * that the `transmit_timestamp` in that `time_request` message matches the corresponding time stamp from the synchronization data received in the `time_response`, and

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

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



Procedure for unicast time synchronization exchange.

6.2. Broadcast Time Synchronization Exchange

6.2.1. Preconditions for the Broadcast Time Synchronization Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- o The client MUST receive all the information necessary to process broadcast time synchronization messages from the server. This includes
 - * the one-way functions used for building the key chain,

- * the last key of the key chain,
 - * time interval duration,
 - * the disclosure delay (number of intervals between use and disclosure of a key),
 - * the time at which the next time interval will start, and
 - * the next interval's associated index.
- o The communication of the data listed above MUST guarantee authenticity of the server, as well as integrity and freshness of the broadcast parameters to the client.

6.2.2. Goals of the Broadcast Time Synchronization Exchange

The broadcast time synchronization exchange:

- o transmits (broadcast) time synchronization data from the server to the client as specified by the appropriate time synchronization protocol,
- o guarantees to the client that the received synchronization data has arrived in a timely manner as required by the TESLA protocol and is trustworthy enough to be stored for later checks,
- o additionally guarantees authenticity of a certain broadcast synchronization message in the client's storage.

6.2.3. Message Type: "server_broad"

This message is sent by the server over the course of its broadcast schedule. It is part of any broadcast association. It contains

- o the NTS message ID "server_broad",
- o the version number that the server is working under,
- o time broadcast data,
- o the index that belongs to the current interval (and therefore identifies the current, yet undisclosed, key),
- o the disclosed key of the previous disclosure interval (current time interval minus disclosure delay),

- o a MAC, calculated with the key for the current time interval, verifying
 - * the message ID,
 - * the version number, and
 - * the time data.

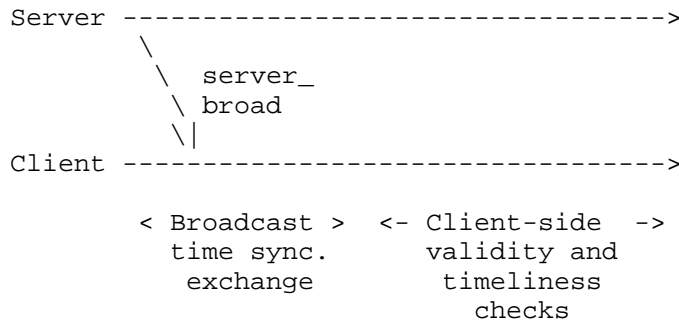
6.2.4. Procedure Overview of Broadcast Time Synchronization Exchange

A broadcast time synchronization message exchange consists of the following steps:

1. The server follows the TESLA protocol by regularly sending server_broad messages as described in Section 6.2.3, adhering to its own disclosure schedule.
2. The client awaits time synchronization data in the form of a server_broadcast message. Upon receipt, it performs the following checks:
 - * Proof that the MAC is based on a key that is not yet disclosed (packet timeliness). This is achieved via a combination of checks. First, the disclosure schedule is used, which requires loose time synchronization. If this is successful, the client obtains a stronger guarantee via a key check exchange (see below). If its timeliness is verified, the packet will be buffered for later authentication. Otherwise, the client MUST discard it. Note that the time information included in the packet will not be used for synchronization until its authenticity could also be verified.
 - * The client checks that it does not already know the disclosed key. Otherwise, the client SHOULD discard the packet to avoid a buffer overrun. If this check is successful, the client ensures that the disclosed key belongs to the one-way key chain by applying the one-way function until equality with a previous disclosed key is shown. If it is falsified, the client MUST discard the packet.
 - * If the disclosed key is legitimate, then the client verifies the authenticity of any packet that it has received during the corresponding time interval. If authenticity of a packet is verified, then it is released from the buffer and its time information can be utilized. If the verification fails, then authenticity is not given. In this case, the client MUST request authentic time from the server by means other than

broadcast messages. Also, the client MUST re-initialize the broadcast sequence with a "client_bpar" message if the one-way key chain expires, which it can check via the disclosure schedule.

See RFC 4082[RFC4082] for a detailed description of the packet verification process.



Procedure for broadcast time synchronization exchange.

6.3. Broadcast Keycheck

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

6.3.1. Preconditions for the Broadcast Keycheck Exchange

Before this message exchange is available, there are some requirements that the client and server need to meet:

- o They MUST negotiate the algorithm for the MAC used in the time synchronization messages. Authenticity and integrity of the communication MUST be ensured.
- o The client MUST know a key input value KIV. Authenticity and integrity of the communication MUST be ensured.
- o Client and server MUST exchange the cookie (which depends on the KIV as described in section Section 5). Authenticity, confidentiality and integrity of the communication MUST be ensured.

These requirements conform to those for the unicast time synchronization exchange. Accordingly, they too can be realized via the Association and Cookie Message Exchanges described in Appendix B (Appendix B).

6.3.2. Goals of the Broadcast Keycheck Exchange

The keycheck exchange:

- o guarantees to the client that the key belonging to the respective TESLA interval communicated in the exchange had not been disclosed before the client_keycheck message was sent.
- o guarantees to the client the timeliness of any broadcast packet secured with this key if it arrived before client_keycheck was sent.

6.3.3. Message Type: "client_keycheck"

A message of this type is sent by the client in order to initiate an additional check of packet timeliness for the TESLA scheme. It contains

- o the NTS message ID "client_keycheck",
- o the NTS version number negotiated during association,
- o a nonce,
- o an interval number from the TESLA disclosure schedule,
- o the MAC algorithm negotiated during association,
- o the client's key input value KIV, and
- o optional: a MAC (generated with the cookie as key) for verification of all of the above data.

6.3.4. Message Type: "server_keycheck"

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

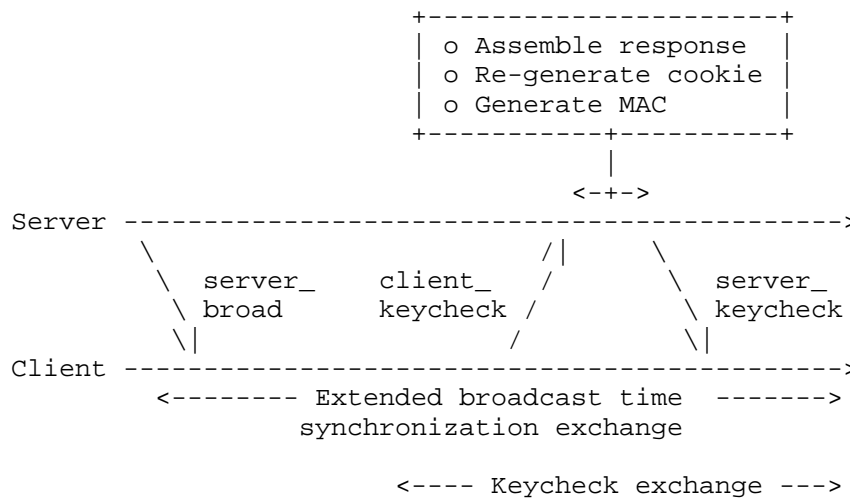
- o the NTS message ID "server_keycheck"
- o the version number as transmitted in "client_keycheck",
- o the nonce transmitted in the client_keycheck message,

- o the interval number transmitted in the client_keycheck message, and
- o a MAC (generated with the cookie as key) for verification of all of the above data.

6.3.5. Procedure Overview of the Broadcast Keycheck Exchange

A broadcast keycheck message exchange consists of the following steps:

1. The client sends a client_keycheck message. It MUST memorize the nonce and the time interval number that it sends as a correlated pair.
2. Upon receipt of a client_keycheck message the server performs as follows: If the client_keycheck message contains a MAC the server re-calculates the MAC and compares this value with the MAC in the received data.
 - * If the re-calculated MAC does not match the MAC in the received data the server MUST stop the processing of the request.
 - * If the re-calculated MAC matches the MAC in the received data the server continues to process the request: It looks up whether it has already disclosed the key associated with the interval number transmitted in that message. If it has not disclosed it, it constructs and sends the appropriate server_keycheck message as described in Section 6.3.4. For more details, see also Appendix C.
3. The client awaits a reply in the form of a server_keycheck message. On receipt, it performs the following checks:
 - * that the transmitted version number matches the one negotiated previously,
 - * that the transmitted nonce belongs to a previous client_keycheck message,
 - * that the TESLA interval number in that client_keycheck message matches the corresponding interval number from the server_keycheck, and
 - * that the appended MAC verifies the received data.



Procedure for extended broadcast time synchronization exchange.

7. Server Seed, MAC Algorithms and Generating MACs

7.1. Server Seed

The server has to calculate a random seed which has to be kept secret. The server **MUST** generate a seed for each supported MAC algorithm, see Section 7.2.

According to the requirements in [RFC7384], the server **MUST** refresh each server seed periodically. Consequently, the cookie memorized by the client becomes obsolete. In this case, the client cannot verify the MAC attached to subsequent time response messages and has to respond accordingly by re-initiating the protocol with a cookie request (Appendix B.4).

7.2. MAC Algorithms

MAC algorithms are used for calculation of the cookie and the actual MAC. The client and the server negotiate a MAC algorithm during the association phase at the beginning. The selected algorithm **MUST** be used for all cookie and MAC creation processes in that run.

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

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

8. IANA Considerations

As mentioned, this document generically specifies security measures whose utilization for any given specific time synchronization protocol requires a separate document. Consequently, this document itself does not have any IANA actions (TO BE REVIEWED).

9. Security Considerations

Aspects of security for time synchronization protocols are treated throughout this document. For a comprehensive discussion of security requirements in time synchronization contexts, refer to [RFC7384]. See Appendix A for a tabular overview of how NTS deals with those requirements.

Additional NTS specific discussion of security issues can be found in the following subsections.

Note: Any separate document describing the utilization of NTS to a specific time synchronization protocol may additionally introduce discussion of its own specific security considerations.

9.1. Privacy

The payload of time synchronization protocol packets of two-way time transfer approaches like NTP and PTP consists basically of time stamps, which are not considered secret [RFC7384]. Therefore, encryption of the time synchronization protocol packet's payload is not considered in this document. However, an attacker can exploit the exchange of time synchronization protocol packets for topology detection and inference attacks as described in [RFC7624]. To make such attacks more difficult, that draft recommends the encryption of the packet payload. Yet, in the case of time synchronization protocols the confidentiality protection of time synchronization packet's payload is of secondary importance since the packet's meta data (IP addresses, port numbers, possibly packet size and regular sending intervals) carry more information than the payload. To enhance the privacy of the time synchronization partners, the usage of tunnel protocols such as IPsec and MACsec, where applicable, is therefore more suited than confidentiality protection of the payload.

9.2. Initial Verification of the Server Certificates

The client may wish to verify the validity of certificates during the initial association phase. Since it generally has no reliable time during this initial communication phase, it is impossible to verify the period of validity of the certificates. To solve this chicken-and-egg problem, the client has to rely on external means.

9.3. Revocation of Server Certificates

According to Section 7, it is the client's responsibility to initiate a new association with the server after the server's certificate expires. To this end, the client reads the expiration date of the certificate during the certificate message exchange (Appendix B.3.3). Furthermore, certificates may also be revoked prior to the normal expiration date. To increase security the client MAY periodically verify the state of the server's certificate via Online Certificate Status Protocol (OCSP) Online Certificate Status Protocol (OCSP) [RFC6960].

9.4. Mitigating Denial-of-Service for broadcast packets

TESLA authentication buffers packets for delayed authentication. This makes the protocol vulnerable to flooding attacks, causing the client to buffer excessive numbers of packets. To add stronger DoS protection to the protocol, the client and the server use the "not re-using keys" scheme of TESLA as pointed out in Section 3.7.2 of RFC 4082 [RFC4082]. In this scheme the server never uses a key for the MAC generation more than once. Therefore, the client can discard any packet that contains a disclosed key it already knows, thus preventing memory flooding attacks.

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

9.5. Delay Attack

In a packet delay attack, an adversary with the ability to act as a MITM delays time synchronization packets between client and server asymmetrically [RFC7384]. This prevents the client from accurately measuring the network delay, and hence its time offset to the server [Mizrahi]. The delay attack does not modify the content of the exchanged synchronization packets. Therefore, cryptographic means do not provide a feasible way to mitigate this attack. However, several

non-cryptographic precautions can be taken in order to detect this attack.

1. Usage of multiple time servers: this enables the client to detect the attack, provided that the adversary is unable to delay the synchronization packets between the majority of servers. This approach is commonly used in NTP to exclude incorrect time servers [RFC5905].
2. Multiple communication paths: The client and server utilize different paths for packet exchange as described in the I-D [I-D.ietf-tictoc-multi-path-synchronization]. The client can detect the attack, provided that the adversary is unable to manipulate the majority of the available paths [Shpiner]. Note that this approach is not yet available, neither for NTP nor for PTP.
3. Usage of an encrypted connection: the client exchanges all packets with the time server over an encrypted connection (e.g. IPsec). This measure does not mitigate the delay attack, but it makes it more difficult for the adversary to identify the time synchronization packets.
4. For unicast-type messages: Introduction of a threshold value for the delay time of the synchronization packets. The client can discard a time server if the packet delay time of this time server is larger than the threshold value.

Additional provision against delay attacks has to be taken for broadcast-type messages. This mode relies on the TESLA scheme which is based on the requirement that a client and the broadcast server are loosely time synchronized. Therefore, a broadcast client has to establish time synchronization with its broadcast server before it starts utilizing broadcast messages for time synchronization.

One possible way to achieve this initial synchronization is to establish a unicast association with its broadcast server until time synchronization and calibration of the packet delay time is achieved. After that, the client can establish a broadcast association with the broadcast server and utilizes TESLA to verify integrity and authenticity of any received broadcast packets.

An adversary who is able to delay broadcast packets can cause a time adjustment at the receiving broadcast clients. If the adversary delays broadcast packets continuously, then the time adjustment will accumulate until the loose time synchronization requirement is violated, which breaks the TESLA scheme. To mitigate this vulnerability the security condition in TESLA has to be supplemented

by an additional check in which the client, upon receipt of a broadcast message, verifies the status of the corresponding key via a unicast message exchange with the broadcast server (see Appendix C.4 for a detailed description of this check). Note that a broadcast client should also apply the above-mentioned precautions as far as possible.

9.6. Random Number Generation

At various points of the protocol, the generation of random numbers is required. The employed methods of generation need to be cryptographically secure. See [RFC4086] for guidelines concerning this topic.

10. Acknowledgements

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

11. References

11.1. Normative References

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <<http://www.rfc-editor.org/info/rfc2104>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, DOI 10.17487/RFC4082, June 2005, <<http://www.rfc-editor.org/info/rfc4082>>.
- [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>>.

11.2. Informative References

- [I-D.ietf-ntp-cms-for-nts-message]
Sibold, D., Teichel, K., Roettger, S., and R. Housley,
"Protecting Network Time Security Messages with the
Cryptographic Message Syntax (CMS)", draft-ietf-ntp-cms-
for-nts-message-04 (work in progress), July 2015.
- [I-D.ietf-tictoc-multi-path-synchronization]
Shpiner, A., Tse, R., Schelp, C., and T. Mizrahi, "Multi-
Path Time Synchronization", draft-ietf-tictoc-multi-path-
synchronization-02 (work in progress), April 2015.
- [IEEE1588]
IEEE Instrumentation and Measurement Society. TC-9 Sensor
Technology, "IEEE standard for a precision clock
synchronization protocol for networked measurement and
control systems", 2008.
- [Mizrahi] Mizrahi, T., "A game theoretic analysis of delay attacks
against time synchronization protocols", in Proceedings
of Precision Clock Synchronization for Measurement Control
and Communication, ISPCS 2012, pp. 1-6, September 2012.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker,
"Randomness Requirements for Security", BCP 106, RFC 4086,
DOI 10.17487/RFC4086, June 2005,
<<http://www.rfc-editor.org/info/rfc4086>>.
- [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>>.
- [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A.,
Galperin, S., and C. Adams, "X.509 Internet Public Key
Infrastructure Online Certificate Status Protocol - OCSP",
RFC 6960, DOI 10.17487/RFC6960, June 2013,
<<http://www.rfc-editor.org/info/rfc6960>>.
- [RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T.,
Trammell, B., Huitema, C., and D. Borkmann,
"Confidentiality in the Face of Pervasive Surveillance: A
Threat Model and Problem Statement", RFC 7624,
DOI 10.17487/RFC7624, August 2015,
<<http://www.rfc-editor.org/info/rfc7624>>.

[Shpiner] Shpiner, A., Revah, Y., and T. Mizrahi, "Multi-path Time Protocols", in Proceedings of Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2013, pp. 1-6, September 2013.

Appendix A. (informative) TICTOC Security Requirements

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

Section	Requirement from RFC 7384	Requirement level	NTS
5.1.1	Authentication of Servers	MUST	OK
5.1.1	Authorization of Servers	MUST	OK
5.1.2	Recursive Authentication of Servers (Stratum 1)	MUST	OK
5.1.2	Recursive Authorization of Servers (Stratum 1)	MUST	OK
5.1.3	Authentication and Authorization of Clients	MAY	Optional, Limited
5.2	Integrity protection	MUST	OK
5.3	Spoofing Prevention	MUST	OK
5.4	Protection from DoS attacks against the time protocol	SHOULD	OK
5.5	Replay protection	MUST	OK
5.6	Key freshness	MUST	OK
	Security association	SHOULD	OK
	Unicast and multicast associations	SHOULD	OK
5.7	Performance: no degradation in quality of time transfer	MUST	OK
	Performance: lightweight computation	SHOULD	OK

	Performance: storage	SHOULD	OK
	Performance: bandwidth	SHOULD	OK
5.8	Confidentiality protection	MAY	NO
5.9	Protection against Packet Delay and Interception Attacks	MUST	Limited*)
5.10	Secure mode	MUST	OK
	Hybrid mode	SHOULD	-

*) See discussion in Section 9.5.

Comparison of NTS specification against Security Requirements of Time Protocols in Packet Switched Networks (RFC 7384)

Appendix B. (normative) Inherent Association Protocol Messages

This appendix presents a procedure that performs the association, the cookie, and also the broadcast parameter message exchanges between a client and a server. This procedure is one possible way to achieve the preconditions listed in Sections Section 6.1.1, Section 6.2.1, and Section 6.3.1 while taking into account the objectives given in Section Section 4.

B.1. Overview of NTS with Inherent Association Protocol

This inherent association protocol applies X.509 certificates to verify the authenticity of the time server and to exchange the cookie. This is done in two separate message exchanges, described below. An additional required exchange in advance serves to limit the amplification potential of the association message exchange.

A client needs a public/private key pair for encryption, with the public key enclosed in a certificate. A server needs a public/private key pair for signing, with the public key enclosed in a certificate. If a participant intends to act as both a client and a server, it MUST have two different key pairs for these purposes.

If this protocol is employed, the hash value of the client's certificate is used as the client's key input value, i.e. the cookie is calculated according to:

cookie = MSB_ (MAC(server seed, H(certificate of client))),

Where the hash function H is the one used in the MAC algorithm. The client's certificate contains the client's public key and enables the server to identify the client, if client authorization is desired.

B.2. Access Message Exchange

This message exchange serves only to prevent the next (association) exchange from being abusable for amplification denial-of-service attacks.

B.2.1. Goals of the Access Message Exchange

The access message exchange:

- o transfers a secret value from the server to the client (initiator),
- o the secret value permits the client to initiate an association message exchange.

B.2.2. Message Type: "client_access"

This message is sent by a client who intends to perform an association exchange with the server in the future. It contains:

- o the NTS message ID "client_access".

B.2.3. Message Type: "server_access"

This message is sent by the server on receipt of a client_access message. It contains:

- o the NTS message ID "server_access",
- o an access key.

B.2.4. Procedure Overview of the Access Exchange

For an access exchange, the following steps are performed:

1. The client sends a client_access message to the server.
2. Upon receipt of a client_access, the server calculates the access key. It then sends a reply in the form of a server_access message. The server must either memorize the access key or alternatively apply a means by which it can reconstruct the

access key. Note that in both cases the access key must be correlated with the address of the requester. Note also that if the server memorizes the access key for a requester, it has to keep state for a certain amount of time.

3. The client waits for a response in the form of a `server_access` message. Upon receipt of one, it **MUST** memorize the included access key.

B.3. Association Message Exchange

In this message exchange, the participants negotiate the MAC and encryption algorithms that are used throughout the protocol. In addition, the client receives the certification chain up to a trusted anchor. With the established certification chain the client is able to verify the server's signatures and, hence, the authenticity of future NTS messages from the server is ensured.

B.3.1. Goals of the Association Exchange

The association exchange:

- o enables the client to verify any communication with the server as authentic,
- o lets the participants negotiate NTS version and algorithms,
- o guarantees authenticity and integrity of the negotiation result to the client,
- o guarantees to the client that the negotiation result is based on the client's original, unaltered request.

B.3.2. Message Type: "client_assoc"

This message is sent by the client if it wants to perform association with a server. It contains

- o the NTS message ID "client_assoc",
- o a nonce,
- o the access key obtained earlier via an access message exchange,
- o the version number of NTS that the client wants to use (this **SHOULD** be the highest version number that it supports),
- o a selection of accepted MAC algorithms, and

- o a selection of accepted encryption algorithms.

B.3.3. Message Type: "server_assoc"

This message is sent by the server upon receipt of client_assoc. It contains

- o the NTS message ID "server_assoc",
- o the nonce transmitted in client_assoc,
- o the client's proposal for the version number, selection of accepted MAC algorithms and selection of accepted encryption algorithms, as transmitted in client_assoc,
- o the version number used for the rest of the protocol (which SHOULD be determined as the minimum over the client's suggestion in the client_assoc message and the highest supported by the server),
- o the server's choice of algorithm for encryption and for MAC creation, all of which MUST be chosen from the client's proposals,
- o a signature, calculated over the data listed above, with the server's private key and according to the signature algorithm which is also used for the certificates that are included (see below), and
- o a chain of certificates, which starts at the server and goes up to a trusted authority; each certificate MUST be certified by the one directly following it.

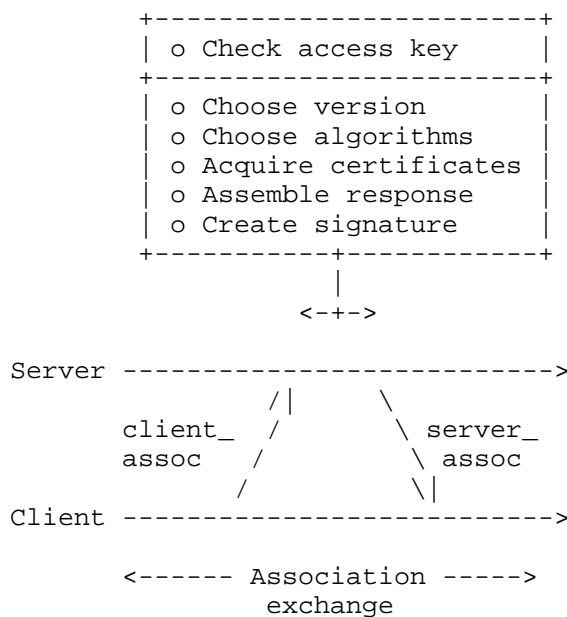
B.3.4. Procedure Overview of the Association Exchange

For an association exchange, the following steps are performed:

1. The client sends a client_assoc message to the server. It MUST keep the transmitted values for the version number and algorithms available for later checks.
2. Upon receipt of a client_assoc message, the server checks the validity of the included access key. If it is not valid, the server MUST abort communication. If it is valid, the server constructs and sends a reply in the form of a server_assoc message as described in Appendix B.3.3. Upon unsuccessful negotiation for version number or algorithms the server_assoc message MUST contain an error code.

3. The client waits for a reply in the form of a `server_assoc` message. After receipt of the message it performs the following checks:
- * The client checks that the message contains a conforming version number.
 - * It checks that the nonce sent back by the server matches the one transmitted in `client_assoc`,
 - * It also verifies that the server has chosen the encryption and MAC algorithms from its proposal sent in the `client_assoc` message and that this proposal was not altered.
 - * Furthermore, it performs authenticity checks on the certificate chain and the signature.

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



Procedure for association and cookie exchange.

B.4. Cookie Message Exchange

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

B.4.1. Goals of the Cookie Exchange

The cookie exchange:

- o enables the server to check the client's authorization via its certificate (optional),
- o supplies the client with the correct cookie and corresponding KIV for its association to the server,
- o guarantees to the client that the cookie originates from the server and that it is based on the client's original, unaltered request.
- o guarantees that the received cookie is unknown to anyone but the server and the client.

B.4.2. Message Type: "client_cook"

This message is sent by the client upon successful authentication of the server. In this message, the client requests a cookie from the server. The message contains

- o the NTS message ID "client_cook",
- o a nonce,
- o the negotiated version number,
- o the negotiated signature algorithm,
- o the negotiated encryption algorithm,
- o the negotiated MAC algorithm,
- o the client's certificate.

B.4.3. Message Type: "server_cook"

This message is sent by the server upon receipt of a client_cook message. The server generates the hash (the used hash function is the one used for the MAC algorithm) of the client's certificate, as conveyed during client_cook, in order to calculate the cookie according to Section 5. This message contains

- o the NTS message ID "server_cook"
- o the version number as transmitted in client_cook,

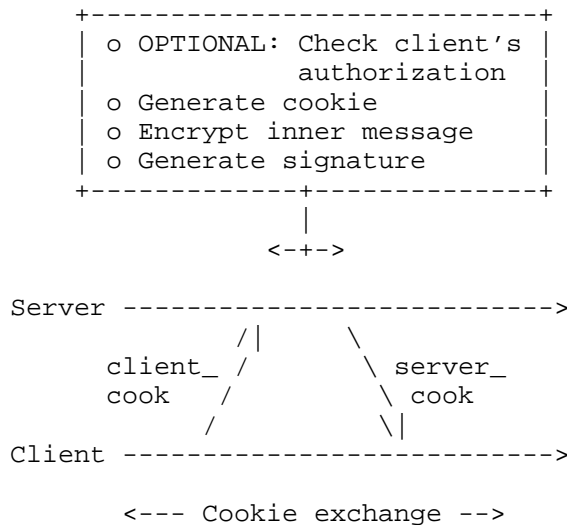
- o a concatenated datum which is encrypted with the client's public key, according to the encryption algorithm transmitted in the client_cook message. The concatenated datum contains
 - * the nonce transmitted in client_cook, and
 - * the cookie.
- o a signature, created with the server's private key, calculated over all of the data listed above. This signature MUST be calculated according to the transmitted signature algorithm from the client_cook message.

B.4.4. Procedure Overview of the Cookie Exchange

For a cookie exchange, the following steps are performed:

1. The client sends a client_cook message to the server. The client MUST save the included nonce until the reply has been processed.
2. Upon receipt of a client_cook message, the server checks whether it supports the given cryptographic algorithms. It then calculates the cookie according to the formula given in Section 5. The server MAY use the client's certificate to check that the client is authorized to use the secure time synchronization service. With this, it MUST construct a server_cook message as described in Appendix B.4.3.
3. The client awaits a reply in the form of a server_cook message; upon receipt it executes the following actions:
 - * It verifies that the received version number matches the one negotiated beforehand.
 - * It verifies the signature using the server's public key. The signature has to authenticate the encrypted data.
 - * It decrypts the encrypted data with its own private key.
 - * It checks that the decrypted message is of the expected format: the concatenation of a nonce and a cookie of the expected bit lengths.
 - * It verifies that the received nonce matches the nonce sent in the client_cook message.

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



Procedure for association and cookie exchange.

B.4.5. Broadcast Parameter Messages

In this message exchange, the client receives the necessary information to execute the TESLA protocol in a secured broadcast association. The client can only initiate a secure broadcast association after successful association and cookie exchanges and only if it has made sure that its clock is roughly synchronized to the server's.

See Appendix C for more details on TESLA.

B.4.5.1. Goals of the Broadcast Parameter Exchange

The broadcast parameter exchange

- o provides the client with all the information necessary to process broadcast time synchronization messages from the server, and
- o guarantees authenticity, integrity and freshness of the broadcast parameters to the client.

B.4.5.2. Message Type: "client_bpar"

This message is sent by the client in order to establish a secured time broadcast association with the server. It contains

- o the NTS message ID "client_bpar",

- o the NTS version number negotiated during association,
- o a nonce, and
- o the signature algorithm negotiated during association.

B.4.5.3. Message Type: "server_bpar"

This message is sent by the server upon receipt of a client_bpar message during the broadcast loop of the server. It contains

- o the NTS message ID "server_bpar",
- o the version number as transmitted in the client_bpar message,
- o the nonce transmitted in client_bpar,
- o the one-way functions used for building the key chain, and
- o the disclosure schedule of the keys. This contains:
 - * the last key of the key chain,
 - * time interval duration,
 - * the disclosure delay (number of intervals between use and disclosure of a key),
 - * the time at which the next time interval will start, and
 - * the next interval's associated index.
- o The message also contains a signature signed by the server with its private key, verifying all the data listed above.

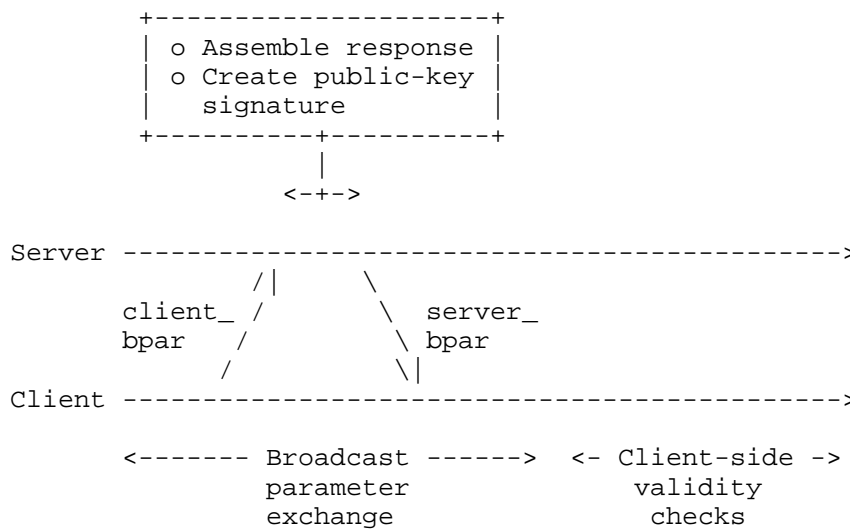
B.4.5.4. Procedure Overview of the Broadcast Parameter Exchange

A broadcast parameter exchange consists of the following steps:

1. The client sends a client_bpar message to the server. It MUST remember the transmitted values for the nonce, the version number and the signature algorithm.
2. Upon receipt of a client_bpar message, the server constructs and sends a server_bpar message as described in Appendix B.4.5.3.
3. The client waits for a reply in the form of a server_bpar message, on which it performs the following checks:

- * The message must contain all the necessary information for the TESLA protocol, as listed in Appendix B.4.5.3.
- * The message must contain a nonce belonging to a client_bpar message that the client has previously sent.
- * Verification of the message's signature.

If any information is missing or if the server's signature cannot be verified, the client MUST abort the broadcast run. If all checks are successful, the client MUST remember all the broadcast parameters received for later checks.



Procedure for unicast time synchronization exchange.

Appendix C. (normative) Using TESLA for Broadcast-Type Messages

For broadcast-type messages, NTS adopts the TESLA protocol with some customizations. This appendix provides details on the generation and usage of the one-way key chain collected and assembled from [RFC4082]. Note that NTS uses the "not re-using keys" scheme of TESLA as described in Section 3.7.2. of [RFC4082].

C.1. Server Preparation

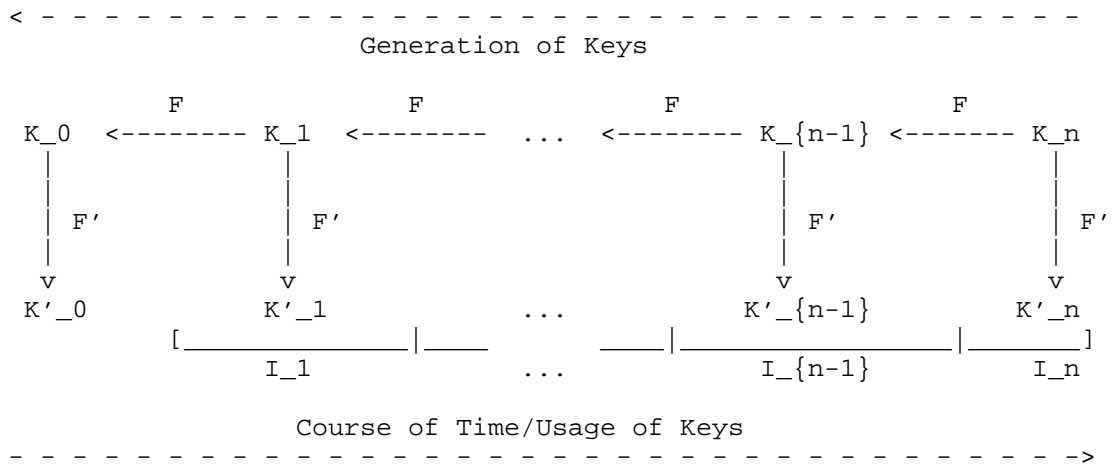
Server setup:

1. The server determines a reasonable upper bound B on the network delay between itself and an arbitrary client, measured in milliseconds.
2. It determines the number $n+1$ of keys in the one-way key chain. This yields the number n of keys that are usable to authenticate broadcast packets. This number n is therefore also the number of time intervals during which the server can send authenticated broadcast messages before it has to calculate a new key chain.
3. It divides time into n uniform intervals I_1, I_2, \dots, I_n . Each of these time intervals has length L , measured in milliseconds. In order to fulfill the requirement 3.7.2. of RFC 4082, the time interval L has to be shorter than the time interval between the broadcast messages.
4. The server generates a random key K_n .
5. Using a one-way function F , the server generates a one-way chain of $n+1$ keys K_0, K_1, \dots, K_n according to
$$K_i = F(K_{i+1}).$$
6. Using another one-way function F' , it generates a sequence of n MAC keys $K'_0, K'_1, \dots, K'_{n-1}$ according to
$$K'_i = F'(K_i).$$
7. Each MAC key K'_i is assigned to the time interval I_i .
8. The server determines the key disclosure delay d , which is the number of intervals between using a key and disclosing it. Note that although security is provided for all choices $d > 0$, the choice still makes a difference:
 - * If d is chosen too short, the client might discard packets because it fails to verify that the key used for its MAC has not yet been disclosed.
 - * If d is chosen too long, the received packets have to be buffered for an unnecessarily long time before they can be verified by the client and be subsequently utilized for time synchronization.

It is RECOMMENDED that the server calculate d according to

$$d = \text{ceil}(2*B / L) + 1,$$

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



A schematic explanation of the TESLA protocol's one-way key chain

C.2. Client Preparation

A client needs the following information in order to participate in a TESLA broadcast:

- o One key K_i from the one-way key chain, which has to be authenticated as belonging to the server. Typically, this will be K_0 .
- o The disclosure schedule of the keys. This consists of:
 - * the length n of the one-way key chain,
 - * the length L of the time intervals I_1, I_2, \dots, I_n ,
 - * the starting time T_i of an interval I_i . Typically this is the starting time T_1 of the first interval;
 - * the disclosure delay d .
- o The one-way function F used to recursively derive the keys in the one-way key chain,
- o The second one-way function F' used to derive the MAC keys K'_0, K'_1, \dots, K'_n from the keys in the one-way chain.

- o An upper bound D_t on how far its own clock is "behind" that of the server.

Note that if D_t is greater than $(d - 1) * L$, then some authentic packets might be discarded. If D_t is greater than $d * L$, then all authentic packets will be discarded. In the latter case, the client SHOULD NOT participate in the broadcast, since there will be no benefit in doing so.

C.3. Sending Authenticated Broadcast Packets

During each time interval I_i , the server sends at most one authenticated broadcast packet P_i . Such a packet consists of:

- o a message M_i ,
- o the index i (in case a packet arrives late),
- o a MAC authenticating the message M_i , with K'_i used as key,
- o the key $K_{\{i-d\}}$, which is included for disclosure.

C.4. Authentication of Received Packets

When a client receives a packet P_i as described above, it first checks that it has not already received a packet with the same disclosed key. This is done to avoid replay/flooding attacks. A packet that fails this test is discarded.

Next, the client begins to check the packet's timeliness by ensuring that according to the disclosure schedule and with respect to the upper bound D_t determined above, the server cannot have disclosed the key K_i yet. Specifically, it needs to check that the server's clock cannot read a time that is in time interval $I_{\{i+d\}}$ or later. Since it works under the assumption that the server's clock is not more than D_t "ahead" of the client's clock, the client can calculate an upper bound t_i for the server's clock at the time when P_i arrived. This upper bound t_i is calculated according to

$$t_i = R + D_t,$$

where R is the client's clock at the arrival of P_i . This implies that at the time of arrival of P_i , the server could have been in interval I_x at most, with

$$x = \text{floor}((t_i - T_1) / L) + 1,$$

where floor gives the greatest integer less than or equal to its argument. The client now needs to verify that

$$x < i+d$$

is valid (see also Section 3.5 of [RFC4082]). If it is falsified, it is discarded.

If the check above is successful, the client performs another more rigorous check: it sends a key check request to the server (in the form of a client_keycheck message), asking explicitly if K_i has already been disclosed. It remembers the time stamp t_{check} of the sending time of that request as well as the nonce it used correlated with the interval number i . If it receives an answer from the server stating that K_i has not yet been disclosed and it is able to verify the HMAC on that response, then it deduces that K_i was undisclosed at t_{check} and therefore also at R . In this case, the client accepts P_i as timely.

Next the client verifies that a newly disclosed key $K_{\{i-d\}}$ belongs to the one-way key chain. To this end, it applies the one-way function F to $K_{\{i-d\}}$ until it can verify the identity with an earlier disclosed key (see Clause 3.5 in RFC 4082, item 3).

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

$$T_i \leq s_i, \text{ and}$$

$$s_i < T_{\{i+1\}}.$$

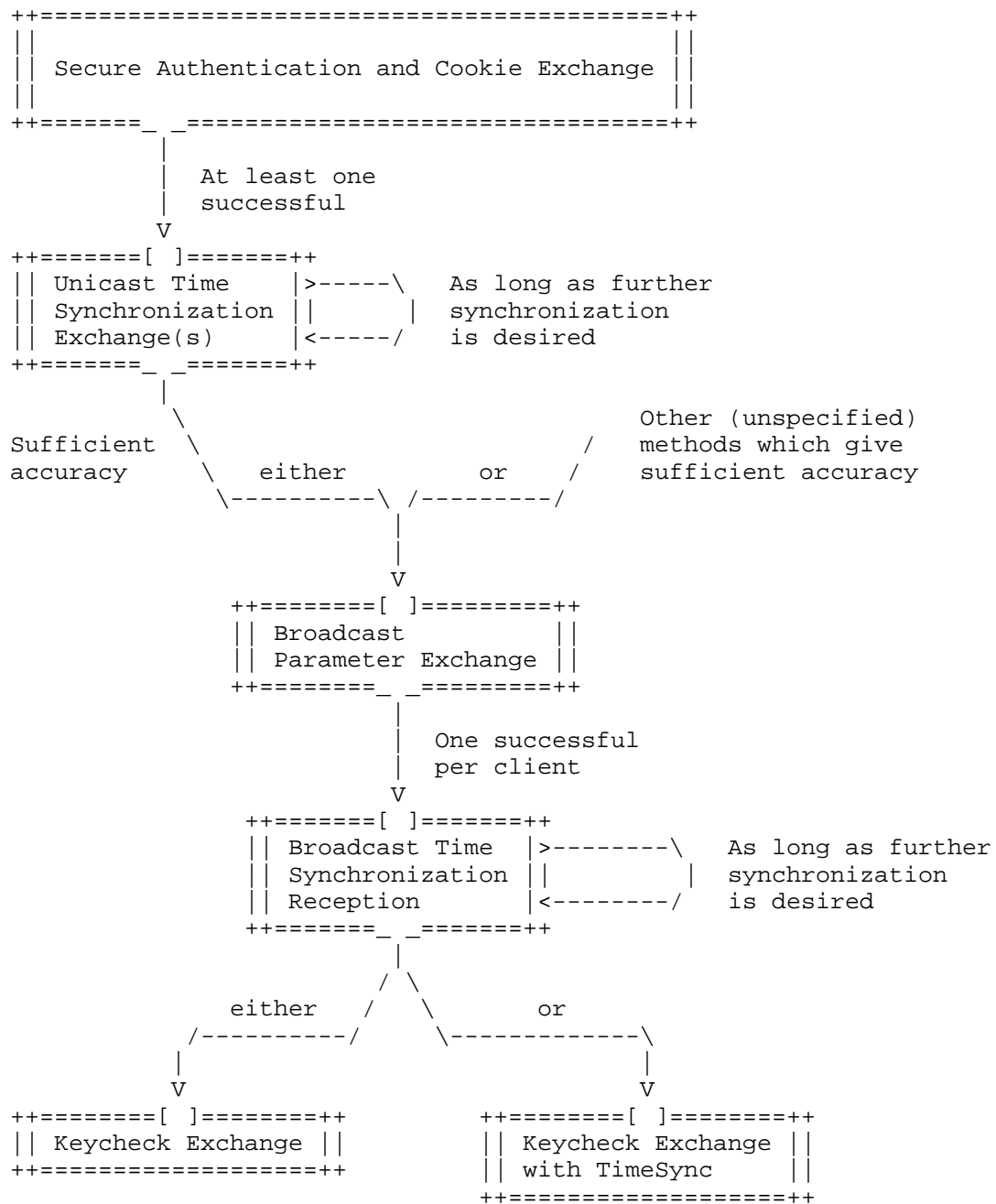
If it is falsified, the packet MUST be discarded and the client MUST reinitialize its broadcast module by performing time synchronization by other means than broadcast messages, and it MUST perform a new broadcast parameter exchange (because a falsification of this check yields that the packet was not generated according to protocol, which suggests an attack).

If a packet P_i passes all the tests listed above, it is stored for later authentication. Also, if at this time there is a package with index $i-d$ already buffered, then the client uses the disclosed key $K_{\{i-d\}}$ to derive $K'_{\{i-d\}}$ and uses that to check the MAC included in package $P_{\{i-d\}}$. Upon success, it regards $M_{\{i-d\}}$ as authenticated.

Appendix D. (informative) Dependencies

Issuer	Type	Owner	Description
Server PKI	private key (signature)	server	Used for server_assoc, server_cook, server_bpar.
	public key (signature)	client	The server uses the private key to sign these messages. The client uses the public key to verify them.
	certificate	server	The certificate is used in server_assoc messages, for verifying authentication and (optionally) authorization.
Client PKI	private key (encryption)	client	The server uses the client's public key to encrypt the content of server_cook
	public key (encryption)	server	messages. The client uses the private key to decrypt them. The certificate is
	certificate	client	sent in client_cook messages, where it is used for trans- portation of the public key as well as (optionally) for verification of client authorization.

This table shows the kind of cryptographic resources that NTS participants of server and client role should have ready before NTS communication starts.



This diagram shows the dependencies between the different message exchanges and procedures which NTS offers.

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Stephen Roettger
Google Inc.

Email: stephen.roettger@googlemail.com

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

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

NTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

D. Sibold
PTB
S. Roettger
Google Inc
K. Teichel
PTB
March 21, 2016

Using the Network Time Security Specification to Secure the Network Time
Protocol
draft-ietf-ntp-using-nts-for-ntp-05

Abstract

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

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

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 September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Objectives	3
3. Terms and Abbreviations	4
4. Overview of NTS-Secured NTP	4
4.1. Symmetric and Client/Server Mode	4
4.2. Broadcast Mode	5
5. Protocol Sequence	5
5.1. The Client	5
5.1.1. The Client in Unicast Mode	5
5.1.2. The Client in Broadcast Mode	8
5.2. The Server	9
5.2.1. The Server in Unicast Mode	9
5.2.2. The Server in Broadcast Mode	10
6. Implementation Notes: ASN.1 Structures and Use of the CMS . .	11
6.1. Unicast Messages	13
6.1.1. Access Messages	13
6.1.2. Association Messages	14
6.1.3. Cookie Messages	14
6.1.4. Time Synchronization Messages	14
6.2. Broadcast Messages	15
6.2.1. Broadcast Parameter Messages	15
6.2.2. Broadcast Time Synchronization Message	15
6.2.3. Broadcast Keycheck	16
7. IANA Considerations	16
7.1. Field Type Registry	16
7.2. SMI Security for S/MIME CMS Content Type Registry	16
8. Security Considerations	17
8.1. Employing Alternative Means for Access, Association and Cookie Exchange	17
8.2. Usage of NTP Pools	17
8.3. Server Seed Lifetime	17
8.4. Supported MAC Algorithms	17
8.5. Protection for Initial Messages	18
9. Acknowledgements	18
10. References	18
10.1. Normative References	18

10.2. Informative References	19
Appendix A. Flow Diagrams of Client Behaviour	19
Appendix B. Bit Lengths for Employed Primitives	22
Appendix C. Error Codes	22
Authors' Addresses	22

1. Introduction

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

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

2. Objectives

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

- o Authenticity: NTS enables an NTP client to authenticate its time server(s).
- o Integrity: NTS protects the integrity of NTP time synchronization protocol packets via a message authentication code (MAC).
- o Confidentiality: NTS does not provide confidentiality protection of the time synchronization packets.
- o Authorization: NTS optionally enables the server to verify the client's authorization.
- o Request-Response-Consistency: NTS enables a client to match an incoming response to a request it has sent. NTS also enables the client to deduce from the response whether its request to the server has arrived without alteration.
- o Modes of operation: Both the unicast and the broadcast mode of NTP are supported.
- o Hybrid mode: Both secure and insecure communication modes are possible for both NTP servers and clients.

- o Compatibility:

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

3. Terms and Abbreviations

CMS Cryptographic Message Syntax [RFC5652]

MAC Message Authentication Code

MITM Man In The Middle

NTP Network Time Protocol [RFC5905]

NTS Network Time Security

TESLA Timed Efficient Stream Loss-Tolerant Authentication [RFC4082]

4. Overview of NTS-Secured NTP

4.1. Symmetric and Client/Server Mode

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

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

4.2. Broadcast Mode

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

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

5. Protocol Sequence

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

$$\text{mac} = \text{MSB}_{<\text{B_mac}>} (\text{HMAC}(\text{key}, \text{content})),$$

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

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

5.1. The Client

5.1.1. The Client in Unicast Mode

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

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

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

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

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

Step 4: It waits for a reply in the form of a server_assoc message. After receipt of the message it performs the following checks:

- * The client checks that the message contains a conforming version number.
- * It checks that the nonce sent back by the server matches the one transmitted in client_assoc,
- * It also verifies that the server has chosen the encryption and MAC algorithms from its proposal sent in the client_assoc message and that this proposal was not altered.
- * Furthermore, it performs authenticity checks on the certificate chain and the signature.

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

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

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

Step 6: It awaits a reply in the form of a server_cook message; upon receipt it executes the following actions:

- * It verifies that the received version number matches the one negotiated beforehand.

- * It verifies the signature using the server's public key. The signature has to authenticate the encrypted data.
- * It decrypts the encrypted data with its own private key.
- * It checks that the decrypted message is of the expected format: the concatenation of a B_nonce bit nonce and a B_cookie bit cookie.
- * It verifies that the received nonce matches the nonce sent in the client_cook message.

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

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

Step 8: It awaits a reply in the form of a time_response message. Upon receipt, it checks:

- * that the transmitted version number matches the one negotiated previously,
- * that the transmitted nonce belongs to a previous time_request message,
- * that the transmit_timestamp in that time_request message matches the corresponding time stamp from the synchronization data received in the time_response, and
- * that the appended MAC verifies the received synchronization data, version number and nonce.

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

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

5.1.2. The Client in Broadcast Mode

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

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

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

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

- * The message must contain all the necessary information for the TESLA protocol, as specified for a server_bpar message.
- * The message must contain a nonce belonging to a client_bpar message that the client has previously sent.
- * Verification of the message's signature.

If any information is missing or if the server's signature cannot be verified, the client **MUST** abort the broadcast run. If all checks are successful, the client **MUST** remember all the broadcast parameters received for later checks.

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

1. Proof that the MAC is based on a key that is not yet disclosed (packet timeliness). This is achieved via a combination of checks. First, the disclosure schedule is used, which requires loose time synchronization. If this is successful, the client obtains a stronger guarantee via a key check exchange: it sends a client_keycheck message and waits for the appropriate response. Note that it needs to memorize the nonce and the time interval number that it sends as a correlated pair. For more detail on both of the mentioned timeliness checks, see [I-D.ietf-ntp-network-time-security]. If its timeliness is verified, the packet will be buffered for

later authentication. Otherwise, the client MUST discard it. Note that the time information included in the packet will not be used for synchronization until its authenticity could also be verified.

2. The client checks that it does not already know the disclosed key. Otherwise, the client SHOULD discard the packet to avoid a buffer overrun. If verified, the client ensures that the disclosed key belongs to the one-way key chain by applying the one-way function until equality with a previous disclosed key is shown. If it is falsified, the client MUST discard the packet.
3. If the disclosed key is legitimate, then the client verifies the authenticity of any packet that it has received during the corresponding time interval. If authenticity of a packet is verified it is released from the buffer and the packet's time information can be utilized. If the verification fails, then authenticity is no longer given. In this case, the client MUST request authentic time from the server by means of a unicast time request message. Also, the client MUST re-initialize the broadcast sequence with a "client_bpar" message if the one-way key chain expires, which it can check via the disclosure schedule.

See RFC 4082 [RFC4082] for a detailed description of the packet verification process.

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

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

5.2. The Server

5.2.1. The Server in Unicast Mode

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

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

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

- o Upon receipt of a client_assoc message, the server checks the included access key. To this end it reconstructs the access key and compares it against the received one. If they match, the server constructs and sends a reply in the form of a server_assoc message as described in [I-D.ietf-ntp-network-time-security]. In the case where the validity of the included access key can not be verified, the server MUST NOT reply to the received request.
- o Upon receipt of a client_cook message, the server checks whether it supports the given cryptographic algorithms. It then calculates the cookie according to the formula given in [I-D.ietf-ntp-network-time-security]. With this, it MUST construct a server_cook message as described in [I-D.ietf-ntp-network-time-security].
- o Upon receipt of a time_request message, the server re-calculates the cookie and the MAC for that time_request packet and compares this value with the MAC in the received data.
 - * If the re-calculated MAC does not match the MAC in the received data the server MUST stop the processing of the request.
 - * If the re-calculated MAC matches the MAC in the received data the server computes the necessary time synchronization data and constructs a time_response message as given in [I-D.ietf-ntp-network-time-security].

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

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

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

5.2.2. The Server in Broadcast Mode

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

- o Upon receipt of a client_bpar message, the server constructs and sends a server_bpar message as described in [I-D.ietf-ntp-network-time-security].
- o Upon receipt of a client_keycheck message, the server re-calculates the cookie and the MAC for that client_keycheck packet and compares this value with the MAC in the received data.
 - * If the re-calculated MAC does not match the MAC in the received data the server MUST stop the processing of the request.
 - * If the re-calculated MAC matches the MAC in the received data the server looks up whether it has already disclosed the key associated with the interval number transmitted in that message. If it has not disclosed it, it constructs and sends the appropriate server_keycheck message as described in [I-D.ietf-ntp-network-time-security].
- o The server follows the TESLA protocol in all other aspects, by regularly sending server_broad messages as described in [I-D.ietf-ntp-network-time-security], adhering to its own disclosure schedule.

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

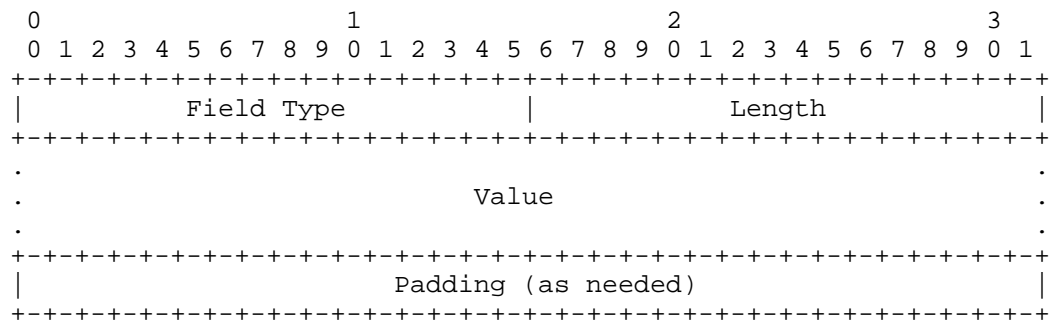
In addition to the items above, the server MAY be ready to perform the following action:

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

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

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

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



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

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

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

(see IANA considerations (Section 7)).

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

```

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

```

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

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

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

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

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

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

It is identified by the following object identifier:

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

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

6.1. Unicast Messages

6.1.1. Access Messages

6.1.1.1. Message Type: "client_access"

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

6.1.1.2. Message Type: "server_access"

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

6.1.2. Association Messages

6.1.2.1. Message Type: "client_assoc"

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

6.1.2.2. Message Type: "server_assoc"

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

6.1.3. Cookie Messages

6.1.3.1. Message Type: "client_cook"

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

6.1.3.2. Message Type: "server_cook"

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

6.1.4. Time Synchronization Messages

6.1.4.1. Message Type: "time_request"

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

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

6.1.4.2. Message Type: "time_response"

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

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

6.2. Broadcast Messages

6.2.1. Broadcast Parameter Messages

6.2.1.1. Message Type: "client_bpar"

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

6.2.1.2. Message Type: "server_bpar"

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

6.2.2. Broadcast Time Synchronization Message

6.2.2.1. Message Type: "server_broad"

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

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

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

6.2.3. Broadcast Keycheck

6.2.3.1. Message Type: "client_keycheck"

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

6.2.3.2. Message Type: "server_keycheck"

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

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

7. IANA Considerations

7.1. Field Type Registry

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

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

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

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

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

8. Security Considerations

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

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

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

8.2. Usage of NTP Pools

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

8.3. Server Seed Lifetime

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

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

8.4. Supported MAC Algorithms

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

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

8.5. Protection for Initial Messages

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

9. Acknowledgements

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

10. References

10.1. Normative References

- [I-D.ietf-ntp-cms-for-nts-message]
Sibold, D., Teichel, K., Roettger, S., and R. Housley,
"Protecting Network Time Security Messages with the
Cryptographic Message Syntax (CMS)", draft-ietf-ntp-cms-
for-nts-message-06 (work in progress), February 2016.
- [I-D.ietf-ntp-extension-field]
Mizrahi, T. and D. Mayer, "The Network Time Protocol
Version 4 (NTPv4) Extension Fields", draft-ietf-ntp-
extension-field-07 (work in progress), February 2016.
- [I-D.ietf-ntp-network-time-security]
Sibold, D., Roettger, S., and K. Teichel, "Network Time
Security", draft-ietf-ntp-network-time-security-13 (work
in progress), February 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B.
Briscoe, "Timed Efficient Stream Loss-Tolerant
Authentication (TESLA): Multicast Source Authentication
Transform Introduction", RFC 4082, DOI 10.17487/RFC4082,
June 2005, <<http://www.rfc-editor.org/info/rfc4082>>.

- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<http://www.rfc-editor.org/info/rfc5652>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<http://www.rfc-editor.org/info/rfc5905>>.

10.2. Informative References

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

Appendix A. Flow Diagrams of Client Behaviour

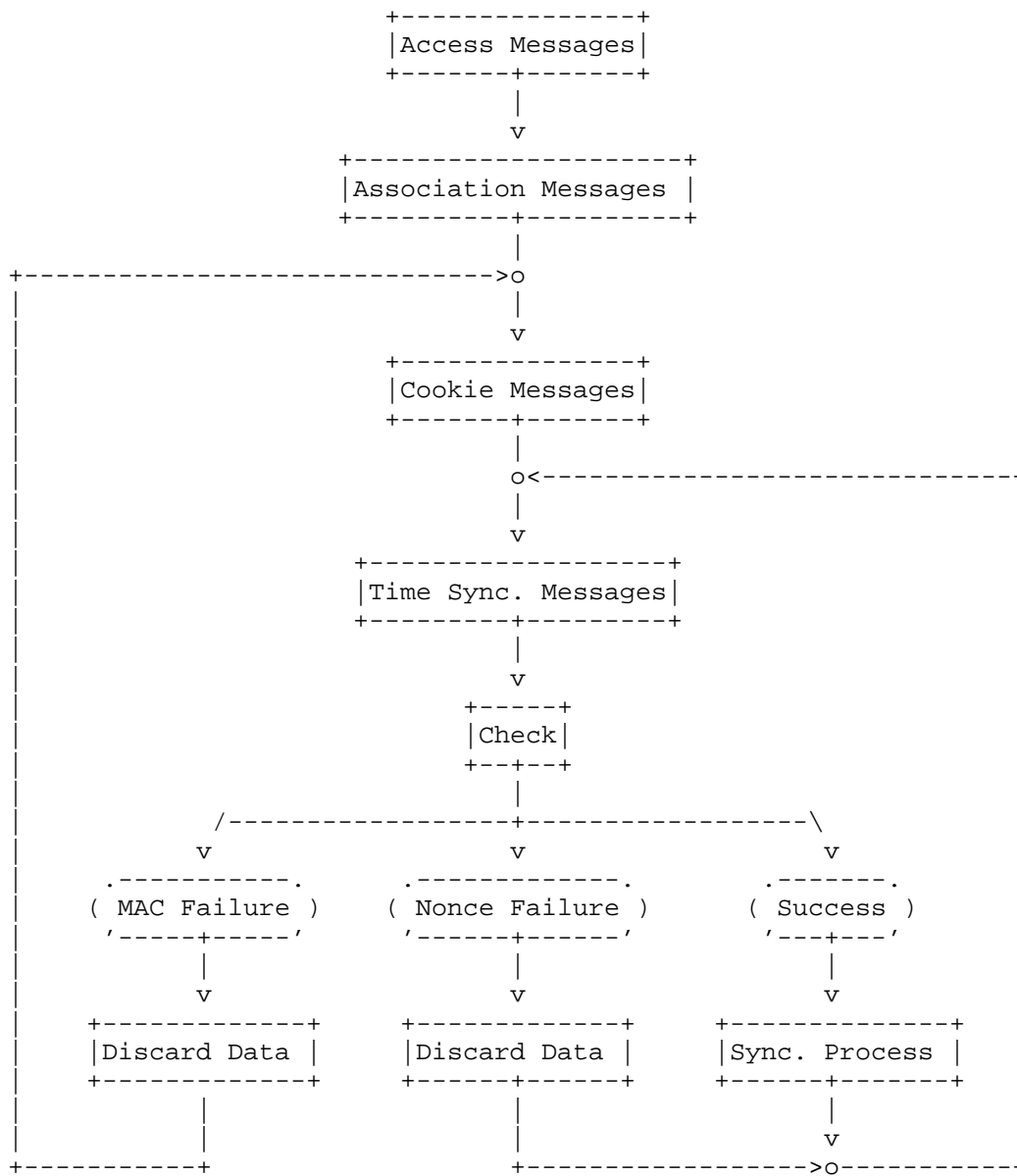


Figure 1: The client's behavior in NTS unicast mode.

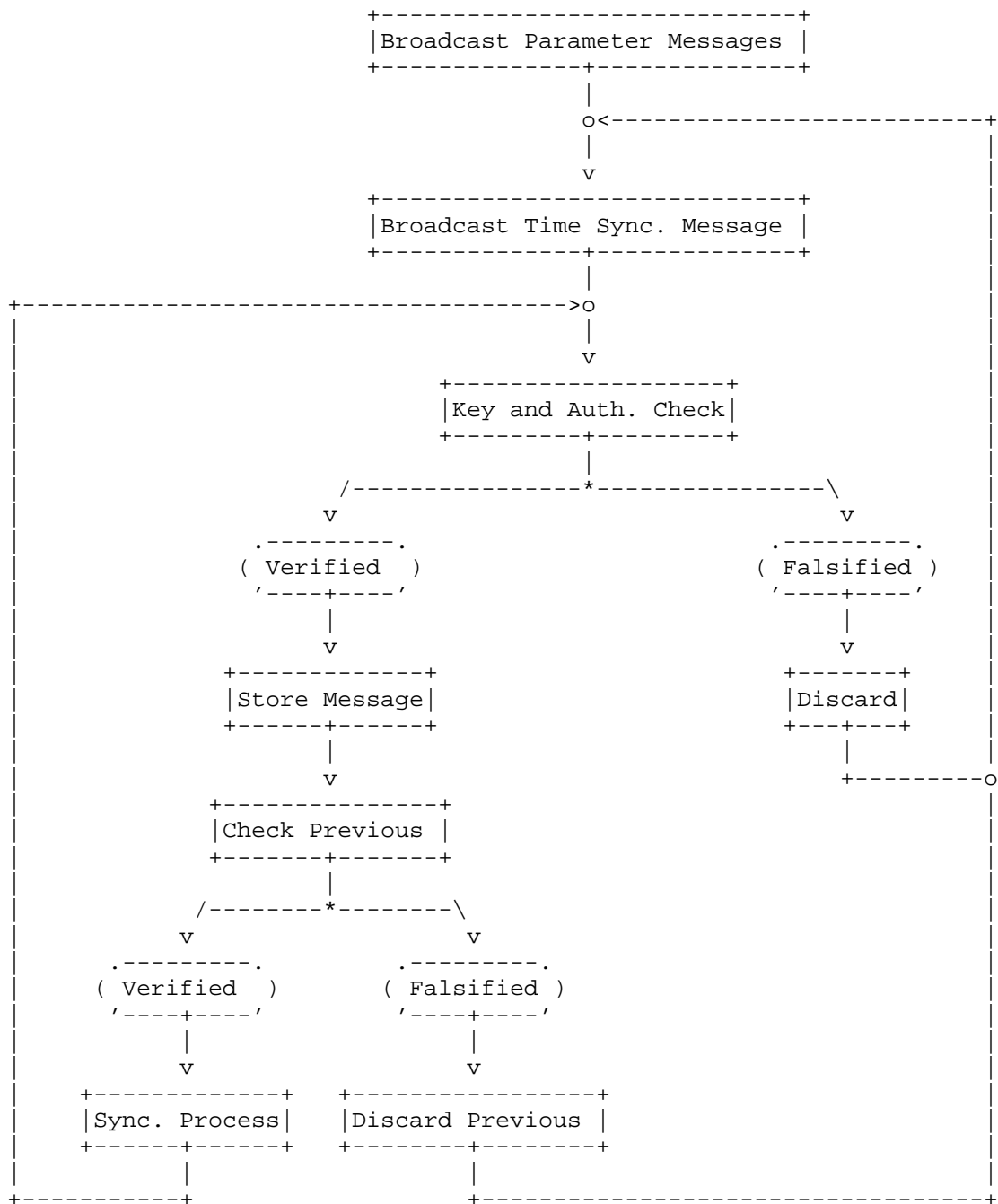


Figure 2: The client's behaviour in NTS broadcast mode.

Appendix B. Bit Lengths for Employed Primitives

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

B_accesskey = 128,
B_seed = 128,
B_nonce = 128,
B_cookie = 128, and
B_mac = 128.

Appendix C. Error Codes

+-----+-----+	
Bit	Meaning
+-----+-----+	
1	D2
+-----+-----+	

Authors' Addresses

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Stephen Roettger
Google Inc

Email: stephen.roettger@googlemail.com

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

Phone: +49-(0)531-592-8421
Email: kristof.teichel@ptb.de

Internet Working Group

Internet Draft

Intended status: Standards Track

Expires: April 2015

Y. Jiang

X. Liu

J. Xu

Huawei

R. Cummings

National Instruments

October 16, 2015

YANG Data Model for IEEE 1588v2
draft-jlx-tictoc-1588v2-yang-02.txt

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on April 16, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

This document defines a YANG data model for the configuration of IEEE 1588v2 devices and clocks, and also retrieval of the configuration information, data set and running states of IEEE 1588v2 clocks.

Table of Contents

1.	Introduction	2
2.	Conventions used in this document	3
3.	Terminology	3
4.	IEEE 1588V2 YANG Model hierarchy	4
5.	IEEE 1588v2 YANG Module	7
6.	Security Considerations	19
7.	IANA Considerations	19
8.	References	19
8.1.	Normative References	19
8.2.	Informative References	19
9.	Acknowledgments	20

1. Introduction

As a synchronization protocol, IEEE 1588v2 [IEEE1588] is widely supported in the carrier networks. It can provide high precision time synchronization as high as nano-seconds. The protocol depends on a Precision Time Protocol (PTP) engine to automatically decide its state, and a PTP transportation layer to carry the PTP timing and various quality messages. The configuration parameters and state data sets of IEEE 1588v2 are numerous.

Some work on IEEE 1588v2 MIB [PTP-MIB] is in progress in the IETF TICTOC WG. But the work is only scoped with retrieval of the state data of IEEE 1588v2 by Simple Network Management Protocol (SNMP) and configuration is not considered, thus its use is limited.

Some service providers require the management of the IEEE 1588v2 synchronization network can be more flexible and more Internet-based (typically overlaid on their transport networks). Software Defined Network (SDN) is another driving factor which demands a greater control over synchronization networks.

YANG [RFC6020] is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF) [RFC6241]. A small set of built-in data types are defined in [RFC6020], and a collection of common data types are further defined in [RFC6991]. Advantages of YANG include Internet based configuration capability, validation, roll-back and etc., all these characteristics make it attractive to become a modeling language for IEEE 1588v2.

This document defines a YANG [RFC6020] data model for the configuration of IEEE 1588v2 devices and clocks, and also retrieval of the state data of IEEE 1588v2 clocks.

In order to fulfill the need of a lightweight implementation, the core module is designed to be generic and minimal, but be extensible with capability negotiation. That is, if a node is verified with a capability of more functions, then more modules can be loaded on demand, otherwise, only a basic module is loaded on the node.

This document defines PTP system information, PTP data sets and running states following the structure and definitions in IEEE 1588v2, and compatible with [PTP-MIB]. The router specific 1588v2 information is out of scope of this document.

2. Conventions used in this document

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 [RFC2119].

3. Terminology

Terminologies used in this document are extracted from [IEEE1588] and [PTP-MIB].

ARB	Arbitrary Timescale
BC	Boundary Clock
DS	Data Set
E2E	End-to-End
EUI	Extended Unique Identifier.
GPS	Global Positioning System

IANA	Internet Assigned Numbers Authority
IP	Internet Protocol
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OC	Ordinary Clock
P2P	Peer-to-Peer
PTP	Precision Time Protocol
TAI	International Atomic Time
TC	Transparent Clock
UDP	User Datagram Protocol
UTC	Coordinated Universal Time

4. IEEE 1588V2 YANG Model hierarchy

This section describes the hierarchy of IEEE 1588v2 YANG module. Query and retrieval of device wide or port specific configuration information and clock data set is described for this version.

Query and retrieval of clock information include:

- Clock data set attributes in a clock node, including: current-DS, clock-parent-DS, default-DS, time-properties-DS, and transparentClock-default-DS.
- Port specific data set attributes, including: port-DS and transparentClock-port-DS.

```

module: ietf-yang-ptp-dataset
+--rw ptp-datasets* [domain-number]
+--rw domain-number      uint8
+--rw default-DS
|   +--rw two-step-flag?    boolean
|   +--rw clock-identity?   binary
|   +--rw number-ports?     uint16
|   +--rw clock-quality
|   |   +--rw clock-class?           uint8
|   |   +--rw clock-accuracy?        uint8
|   |   +--rw offset-scaled-log-variance? uint16
|   +--rw priority1?         uint8
|   +--rw priority2?         uint8
|   +--rw slave-only?        boolean
+--rw current-DS
|   +--rw steps-removed?      uint16
|   +--rw offset-from-master? binary
|   +--rw mean-path-delay?    binary
+--rw parent-DS
|   +--rw parent-port-identity
|   |   +--rw clock-identity?   binary
|   |   +--rw port-number?      uint32
|   +--rw parent-stats?        boolean
|   |   +--rw observed-parent-offset-scaled-log-variance? uint16
|   |   +--rw observed-parent-clock-phase-change-rate?   int32
|   |   +--rw grandmaster-identity?                       binary
|   |   +--rw grandmaster-clock-quality
|   |   |   +--rw grandmaster-clock-class?           uint8
|   |   |   +--rw grandmaster-clock-accuracy?        uint8
|   |   |   +--rw grandmaster-offset-scaled-log-variance? uint16
|   |   +--rw grandmaster-priority1?                   uint8
|   |   +--rw grandmaster-priority2?                   uint8
+--rw time-properties-DS
|   +--rw current-UTC-offset-valid?    boolean
|   +--rw current-UTC-offset?          uint16
|   +--rw leap59?                      boolean
|   +--rw leap61?                      boolean
|   +--rw time-traceable?              boolean
|   +--rw frequency-traceable?         boolean
|   +--rw PTP-timescale?               boolean
|   +--rw time-source?                 uint8
+--rw port-DS-list* [port-number]
|   +--rw port-number                  uint32
|   +--rw port-identity
|   |   +--rw clock-identity?   binary
|   |   +--rw port-number?      uint32
|   +--rw port-state?            uint8

```



```
|   +--rw log-min-delay-req-interval?    int8
|   +--rw peer-mean-path-delay?         int64
|   +--rw log-announce-interval?       int8
|   +--rw announce-receipt-timeout?    uint8
|   +--rw log-sync-interval?           int8
|   +--rw delay-mechanism?             enumeration
|   +--rw log-min-Pdelay-req-interval?  int8
|   +--rw version-number?              uint8
+--rw transparent-clock-default-DS
|   +--rw clock-identity?              binary
|   +--rw number-ports?                uint16
|   +--rw delay-mechanism?             enumeration
|   +--rw primary-domain?              uint32
+--rw transparent-clock-port-DS-list* [port-number]
|   +--rw port-number                  uint32
|   +--rw port-identity
|   |   +--rw clock-identity?          binary
|   |   +--rw port-number?             uint32
|   +--rw log-min-Pdelay-req-interval? int8
|   +--rw faulty-flag?                 boolean
|   +--rw peer-mean-path-delay?        int64
```

5. IEEE 1588v2 YANG Module

```
module ietf-yang-ptp-dataset {
  namespace "urn:ietf:params:xml:ns:yang:1588v2";
  prefix "ptp-dataset";
  organization "IETF TICTOC WG";
  contact "jiangyuanlong@huawei.com";
  description
    "This YANG module defines a data model for the configuration
    of IEEE 1588v2 devices and clocks, and also retrieval of the
    state data of IEEE 1588v2 clocks.";
  revision "2015-10-15" {
    description "Initial revision.";
    reference "draft-jxl-tictoc-1588v2-yang";
  }

  grouping default-DS-entry {
    description
      "This group bundles together all information about the
      PTP clock Default Datasets for a single device.";

    leaf two-step-flag {
      description
        "This object specifies whether the Two Step process is
        used.";
      type boolean;
    }
    leaf clock-identity {
      description
        "This object specifies the clockIdentity of the local
        clock";
      config true;
      type binary {
        length "8";
      }
    }
  }

  leaf number-ports {
    description
      "This object specifies the number of PTP ports on the
      device.";
    type uint16;
  }

  container clock-quality {
    description
```

```
        "This object specifies the default clockQuality of the
        device, which contains clockClass, clockAccuracy and
        offsetScaledLogVariance.";

    leaf clock-class {
        description
            "This object specifies the Quality Class in the
            defaultDS.";
        type uint8;
    }
    leaf clock-accuracy {
        description
            "This object specifies the Quality Accuracy in the
            defaultDS.";
        type uint8;
    }
    leaf offset-scaled-log-variance {
        description
            "This object specifies the Quality offset in the
            defaultDS.";
        type uint16;
    }
}

leaf priority1 {
    description
        "This object specifies the clock Priority1 in the
        defaultDS ";
    type uint8;
}
leaf priority2{
    description
        "This object specifies the clock Priority2 in the
        defaultDS ";
    type uint8;
}

leaf slave-only {
    description
        "This object indicates whether the SlaveOnly flag is
        set";
    type boolean;
}
}

grouping current-DS-entry {
```

```
description
    "This group bundles together all information about the
    PTP clock Current Datasets for a single device.";

leaf steps-removed {
    description
        "This object specifies the distance measured by the
        number of Boundary clocks between the local clock and
        the Foreign master as indicated in the stepsRemoved
        field of Announce messages.";
    type uint16;
    default 0;
}
leaf offset-from-master {
    description
        "This object specifies the current clock dataset
        ClockOffset value. The value of the computation of the
        offset in time between a slave and a master clock.";
    type binary {
        length "1..255";
    }
}
leaf mean-path-delay {
    description
        "The mean path delay between a pair of ports as measured
        by the delay request-response mechanism.";
    type binary {
        length "1..255";
    }
}
}

grouping parent-DS-entry {
    description
        "This group bundles together all information about the
        PTP clock Parent Datasets for a single device.";

    leaf parent-port-identity {
        description
            "This object specifies the value of portIdentity of the
            port on the master that issues the Sync messages used in
            synchronizing this clock.";

        leaf clock-identity {
            description
                "This object identifies the clockIdentity of the
                master clock.";
        }
    }
}
```

```
        type binary {
            length "8";
        }
    }

    leaf port-number {
        description
            "This object identifies the PortNumber for the port
            on the specific master.";
        type uint32{
            range "0..65535";
        }
    }
}

leaf parent-stats {
    description
        "This object indicates whether the values of
        parentDS.observedParentOffsetScaledLogVariance and
        parentDS.observedParentClockPhaseChangeRate have been
        measured and are valid.";
    type boolean;
    default false;
}

leaf observed-parent-offset-scaled-log-variance {
    description
        "This object specifies an estimate of the parent clock's
        phase change rate as measured by the slave clock.";

    type uint16;
    default 0xFFFF;
}

leaf observed-parent-clock-phase-change-rate {
    description
        "This object specifies the clock's parent dataset
        ParentClockPhaseChangeRate value. This value is an
        estimate of the parent clock's phase change rate as
        measured by the slave clock.";
    type int32;
}

leaf grandmaster-identity {
    description
        "This object specifies the clockIdentity of the
        Grandmaster clock.";
    type binary{
        length "8";
    }
}
```

```
container grandmaster-clock-quality {
  description
    "This object specifies the clockQuality of the
    grandmaster clock.";

  leaf grandmaster-clock-class {
    description
      "This object specifies the Quality Class of
      grandmaster clock.";
    type uint8;
  }
  leaf grandmaster-clock-accuracy {
    description
      "This object specifies the Quality Accuracy of the
      grandmaster clock.";
    type uint8;
  }
  leaf grandmaster-offset-scaled-log-variance {
    description
      "This object specifies the Quality offset of the
      grandmaster clock.";
    type uint16;
  }
}

leaf grandmaster-priority1 {
  description
    "This object specifies the priority1 attribute of the
    grandmaster clock.";
  type uint8;
}

leaf grandmaster-priority2 {
  description
    "This object specifies the priority2 attribute of the
    grandmaster clock.";
  type uint8;
}

}

grouping time-properties-DS-entry {
  description
    "This group bundles together all information about the
    PTP clock time properties datasets for a single device.";

  leaf current-UTC-offset-valid {
```

```
        description
            "This object indicates whether current UTC offset is
            valid.";
        type boolean;
    }
    leaf current-UTC-offset {
        description
            "This object specifies the offset between TAI and UTC
            when the epoch of the PTP system is the PTP epoch,
            otherwise the value has no meaning. The value shall be
            in units of seconds.";
        type uint16;
    }
    leaf leap59 {
        description
            "This object indicates whether the last minute of the
            current UTC day contains 59 seconds.";
        type boolean;
    }
    leaf leap61 {
        description
            "This object indicates whether the last minute of the
            current UTC day contains 61 seconds.";
        type boolean;
    }
    leaf time-traceable {
        description
            "This object indicates whether the timescale and the
            currentUtcOffset are traceable to a primary reference.";
        type boolean;
    }
    leaf frequency-traceable {
        description
            "This object indicates whether the frequency determining
            the timescale is traceable to a primary reference.";
        type boolean;
    }
    leaf PTP-timescale {
        description
            "This object indicates whether the clock timescale of
            the grandmaster clock is PTP.";
        type boolean;
    }
    leaf time-source {
        description
            "This object specifies the source of time used by the
            grandmaster clock.";
```

```
    type uint8;
  }
}

grouping port-DS-entry {
  description
    "This group bundles together all information about the
    clock ports dataset for a single clock port.";

  container port-identity {
    description
      "This object specifies the PTP clock port Identity,
      composed of clock-identity and portNumber.";
    leaf clock-identity {
      description
        "This object identify a specific PTP node.";
      config true;
      type binary {
        length "8";
      }
    }

    leaf port-number {
      description
        "This object specifies the PTP Portnumber for this
        port.";
      type uint32 {
        range "0..65535";
      }
    }
  }
}

leaf port-state {
  description
    "This object specifies the current state of the protocol
    engine associated with the port.";
  type uint8;
  default 1;
}

leaf log-min-delay-req-interval {
  description
    "This object specifies the Delay_Req message
    transmission interval.";
  type int8;
}
```



```
leaf peer-mean-path-delay {
  description
    "This object specifies an estimate of the current one-
    way propagation delay on the link when the
    delayMechanism is P2P, otherwise it shall be zero.";
  type int64;
  default 0;
}

leaf log-announce-interval {
  description
    "This object specifies the Announce message transmission
    interval associated with this clock port.";
  type int8;
}

leaf announce-receipt-timeout {
  description
    "This object specifies the number of announceInterval
    that have to pass without receipt of an Announce message
    before the occurrence of the event
    ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES.";
  type uint8;
}

leaf log-sync-interval {
  description
    "This object specifies the mean time interval between
    successive Sync messages.";
  type int8;
}

leaf delay-mechanism {
  description
    "This object specifies the delay measuring mechanism
    used by the port. If the clock is an end-to-end clock,
    the value is e2e, else if the clock is a peer-to-peer
    clock, the value shall be p2p.";
  type enumeration {
    enum E2E {
      value 01;
      description
        "The port is configured to use the delay request-
        response mechanism.";
    }
    enum P2P {
```

```
        value 02;
        description
            "The port is configured to use the peer delay
            mechanism.";
    }
    enum DISABLED {
        value 254;
        description
            "The port does not implement the delay mechanism.";
    }
}

leaf log-min-Pdelay-req-interval {
    description
        "This object specifies the minimum permitted mean time
        interval between successive Pdelay_Req messages.";
    type int8;
}

leaf version-number {
    description
        "This object specifies the version of this standard
        implemented on the port.";
    type uint8;
}
}

grouping transparent-clock-default-DS-entry {
    description
        "This group bundles together the default data sets of a
        transparent clock.";
    leaf clock-identity {
        description
            "This object specifies the value of the clockIdentity
            attribute of the local clock.";
        type binary {
            length "8";
        }
    }
    leaf number-ports {
        description
            "This object specifies the number of PTP ports of the
            device.";
        type uint16;
    }
    leaf delay-mechanism {
```

```
description
    "This object specifies the delayMechanism of the
    transparent clock.";
type enumeration {
    enum E2E {
        value 1;
        description
            "The port is configured to use the delay request-
            response mechanism.";
    }
    enum P2P {
        value 2;
        description
            "The port is configured to use the peer delay
            mechanism.";
    }
    enum DISABLED {
        value 254;
        description
            "The port does not implement the delay mechanism.";
    }
}
}
leaf primary-domain {
    description
        "This object specifies the value of the primary
        syntonization domain.";
    type uint32 {
        range "0..255";
    }
    default 0;
}
}

grouping transparent-clock-port-DS-entry {
    description
        "This group bundles together the port data sets of a
        transparent clock.";

    container port-identity {
        description
            "This object specifies the portIdentity of the local
            port.";
        leaf clock-identity {
            config true;
            type binary {
                length "8";
            }
        }
    }
}
```

```
    }
  }

  leaf port-number {
    type uint32 {
      range "0..65535";
    }
  }
}
leaf log-min-Pdelay-req-interval {
  description
    "This object specifies the minimum permitted mean time
    interval between successive Pdelay_Req messages.";
  type int8;
}
leaf faulty-flag {
  description
    "This object indicates whether the port is faulty.";
  type boolean;
  default false;
}
leaf peer-mean-path-delay {
  description
    "This object specifies an estimate of the current one-
    way propagation delay on the link when the
    delayMechanism is P2P, otherwise it shall be zero.";

  type int64;
  default 0;
}
}

list ptp-datasets {

  key "domain-number";
  min-elements "1";

  description
    "List of one or more PTP datasets in the device, one for
    each domain-number (see IEEE 1588-2008 subclause 6.3)";

  leaf domain-number {
    type uint8;
  }

  container default-DS {
    uses default-DS-entry;
  }
}
```

```
    }

    container current-DS {
        uses current-DS-entry;
    }

    container parent-DS {
        uses parent-DS-entry;
    }

    container time-properties-DS {
        uses time-properties-DS-entry;
    }

    list port-DS-list {
        key "port-number";
        min-elements "1";
        leaf port-number {
            type uint32 {
                range "0..65535";
            }
        }
        uses port-DS-entry;
    }

    container transparent-clock-default-DS {
        uses transparent-clock-default-DS-entry;
    }

    list transparent-clock-port-DS-list {
        key "port-number";
        min-elements "1";
        leaf port-number {
            type uint32 {
                range "0..65535";
            }
        }
        uses transparent-clock-port-DS-entry;
    }
}
```

6. Security Considerations

YANG modules are designed to be accessed via the NETCONF protocol [RFC6241], thus security considerations in [RFC6241] apply here. Security measures such as using the NETCONF over SSH [RFC6242] and restricting its use with access control [RFC6536] can further improve its security, avoid injection attacks and misuse of the protocol.

Some data nodes defined in this YANG module are writable, and any changes to them may adversely impact a synchronization network.

7. IANA Considerations

This document registers a URI in the IETF XML registry, and the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:1588v2

This document registers a YANG module in the YANG Module Names:

name: 1588v2 namespace: urn:ietf:params:xml:ns:yang:1588v2

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) ", RFC 6020, October 2010
- [RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013
- [IEEE1588] IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE Std 1588-2008, July 2008

8.2. Informative References

- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011

- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012
- [PTP-MIB] Shankarkumar, V., Montini, L., Frost, T., and Dowd, G., "Precision Time Protocol Version 2 (PTPv2) Management Information Base", draft-ietf-tictoc-ntp-mib-07, Work in progress

9. Acknowledgments

TBD

Authors' Addresses

Yuanlong Jiang
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China
Email: jiangyuanlong@huawei.com

Xian Liu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China

Jinchun Xu
Huawei Technologies Co., Ltd.
Bantian, Longgang district
Shenzhen 518129, China

Rodney Cummings
National Instruments
Email: Rodney.Cummings@ni.com

NTP Working Group
Internet Draft
Intended status: Standards Track
Updates: 5905
Expires: September 2016

D. Mayer
Network Time Foundation
H. Stenn
Network Time Foundation
March 14, 2016

The Network Time Protocol Version 4 (NTPv4) MAC Extension Field
draft-mayer-ntp-mac-extension-field-00.txt

Abstract

The Network Time Protocol Version 4 (NTPv4) defines in RFC5905 the optional usage of Message Authentication Code (MAC). The MAC is an optional component of the NTP packet at the end of the packet. There can only be one MAC segment in the packet but there is no way of knowing if the last data segment at the end of an NTP packet is a MAC or an extension field, which is also defined in RFC5905. This draft defines a MAC extension field which will allow the existing MAC segment to be moved into an extension field and have a known length and deprecates the existing MAC.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on September 14, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction.....	2
2. Conventions Used in this Document.....	3
2.1. Terminology.....	3
2.2. Terms & Abbreviations.....	3
3. MAC Extension Field.....	3
4. Security Considerations.....	5
5. IANA Considerations.....	5
6. Acknowledgments.....	6
7. References.....	6
7.1. Normative References.....	6
7.2. Informative References.....	6

1. Introduction

The NTP packet format consists of a set of fixed fields that may be followed by some optional fields. Two types of optional fields are currently defined, a Message Authentication Code (MAC), and extension fields, as defined in Section 7.5 of [RFC5905].

If a MAC is used it resides at the end of the packet. This field has a length which depends on the digest algorithm being used. While extension fields have a known length specified in the extension field header, there is no simple way to unequivocally know if the final extra data segment in an NTP packet is a MAC or if it is an extension field. There is also no currently implemented way to pad the length of a MAC to make it difficult to determine the digest algorithm being used.

This document creates a MAC extension field to remove this ambiguity, clearly defining a MAC in an extension field with known size, and

allows us the possibility of deprecating the MAC as described in [RFC5905]. The content of the MAC extension field is almost identical to the existing MAC field but with a size specified in the extension field and the ability to have multiple MAC's within the extension field for different digest algorithms. We note that the only current potential use for multiple MAC algorithms would be for certain broadcast scenarios. By deprecating the original MAC field all parts of the NTP packet will have well-specified lengths.

2. Conventions Used in this Document

2.1. Terminology

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 [KEYWORDS].

2.2. Terms & Abbreviations

NTPv4	Network Time Protocol Version 4 [RFC5905]
MAC	Message Authentication Code
Legacy MAC	MAC as defined in RFC5095

3. MAC Extension Field

The MAC extension field is designed to allow one or more MAC digests to be present within the MAC extension field. The MAC extension field contains the unsigned number of MACs present followed by the unsigned size of each MAC. The number of MACs listed in the MAC COUNT in this extension field MUST be greater than zero. The MAC extension field SHOULD be the last extension field in the packet and a legacy MAC at the end of the packet is OPTIONAL. The extension field MAC supplants the use of a legacy MAC. All new extension fields that require a MAC SHOULD use this MAC extension field, if the recipient implements the MAC extension field. The MACs present in the extension field should perform the digest on all parts of the packet up to but not including the MAC extension field. A legacy MAC MAY be present at the end of the extension fields provided it covers all extension fields including the MAC extension field and is present only for reasons of interoperability with servers that do not understand the new MAC extension field but require a MAC for authentication of the packet. The layout of the data in a MAC extension field is as follows:

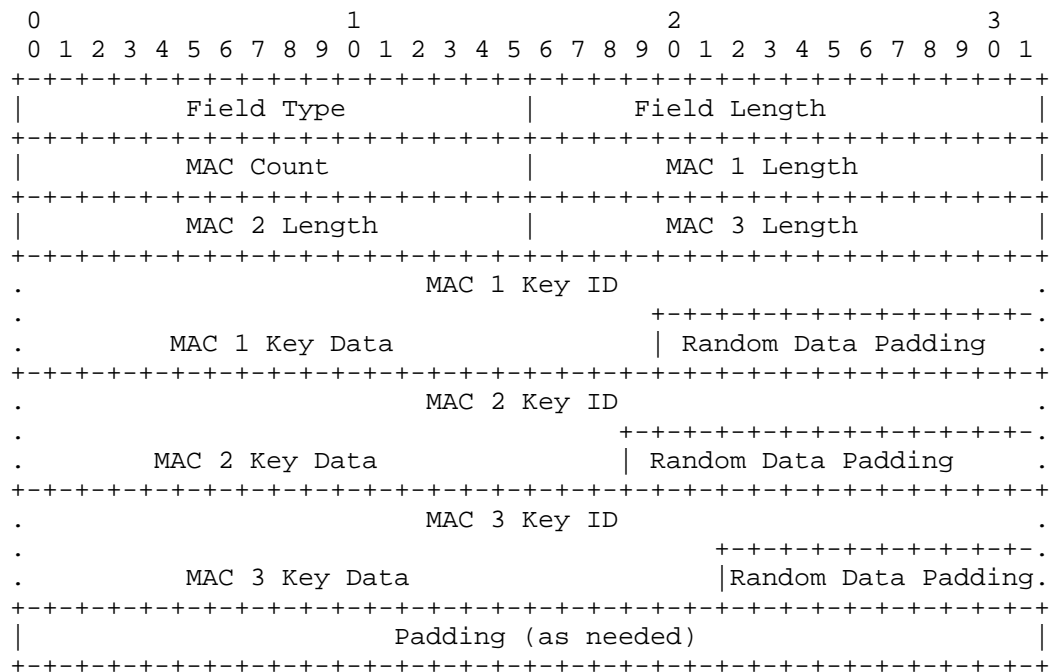


Figure 1: MAC Extension Field Format

A Field Type of 0 and a Length of 0 means this extension field is a CRYPTO-NAK, as defined by RFC5905. Otherwise, a Field Type value of TBD identifies this extension field as a MAC Extension field. The MAC Count is an unsigned 16-bit field, as is each MAC length field. If there are an even number of MACs specified there is an unused 16-bit field which SHOULD be 0x0000 at the end of the set of MAC length values so that the subsequent MAC data is longword (4-octet) aligned. Each MAC SHALL be padded so that any subsequent MAC starts on a 4-octet boundary.

A MAC SHOULD not be present if there is a crypto-NAK present in the packet.

Each MAC within the extension field consists of a 32-bit key identifier which SHOULD be unique to the set of key identifiers in this MAC extension field followed by ((MAC Length) - 4) octets of data, optionally followed by random octets to pad the key data to the length specified earlier in the extension field. That key identifier

is a shared secret which defines the algorithm to be used and a cookie or secret to be used in generating the digest. The MAC digest is produced by hashing the data from the beginning of the NTP packet up to but not including the start of the MAC extension field. The calculation of the digest SHOULD be a hash of this data concatenated with the 32-bit keyid (in network-order), and the key. When sending or receiving a key identifier each side needs to agree on the key identifier, algorithm and cookie to be used to produce the digest along with the digest lengths. Note that the sender may send more bytes than are required by the digest algorithm. This would be done to make it more difficult for a casual observer to identify the algorithm being used based on the length of the data. The digest data begins immediately after the key ID, and any padding octets SHOULD be random.

MAC values should be processed until either one of the MACs is validated, in which case the entire packet up to the beginning of the MAC extension field is considered to be validated, or no more MAC values are left to be validated, in which case the NTP packet is considered to have failed MAC validation.

4. Security Considerations

The security considerations of time protocols in general are discussed in [RFC7384], and the security considerations of NTP are discussed in [RFC5905].

Digests MD5, DES and SHA-1 are considered compromised and should not be used [COMP].

If possible each MAC length should be at least 68 octets long to allow for 4 octets of key ID and at least 64 octets of digest and random padding. This means that for SHA-256 digests there are 4 octets of key ID, 32 bytes digest and 32 random octets of padding. Using larger minimum MAC lengths makes it difficult for an attacker to know which digest algorithms are used.

5. IANA Considerations

IANA is requested to allocate the NTP extension Field Type value of 0x0000 for CRYPTO-NAK.

IANA is requested to allocate an NTP extension Field Type value for the MAC extension. We recommend 0x3003.

6. Acknowledgments

The authors gratefully acknowledge Dave Mills for his insightful comments.

This document was prepared using 2-Word-v2.0.template.dot.

7. References

7.1. Normative References

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

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

7.2. Informative References

[RFC5906] Haberman, B., Mills, D., "Network Time Protocol Version 4: Autokey Specification", RFC 5906, June 2010.

[COMP] TBF

Authors' Addresses

Danny Mayer
Network Time Foundation
PO Box 918
Talent OR 97540

Email: mayer@ntp.org

Harlan Stenn
Network Time Foundation
PO Box 918
Talent OR 97540

Email: stenn@nwttime.org

Internet Engineering Task Force
Internet-Draft
Intended status: Best Current Practice
Expires: September 10, 2016

D. Reilly
Spectracom Corporation
H. Stenn
Network Time Foundation
D. Sibold
PTB
March 9, 2016

Network Time Protocol Best Current Practices
draft-reilly-ntp-bcp-01

Abstract

NTP Version 4 (NTPv4) has been widely used since its publication as RFC 5905 [RFC5905]. This documentation is a collection of Best Practices from across the NTP community.

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 September 10, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
2. Keeping NTP up to date	3
3. General Network Security Best Practices	3
3.1. BCP 38	4
4. NTP Configuration Best Practices	4
4.1. Use enough time sources	4
4.2. Use a diversity of Reference Clocks	5
4.3. Mode 6 and 7	5
4.4. Monitoring	6
4.5. Security	6
4.5.1. Pre-Shared Key Approach	7
4.5.2. Autokey	8
4.5.3. External Security Means	8
4.6. Using Pool Servers	8
4.7. Starting, Cold-Starting, and Re-Starting NTP	9
4.8. Leap Second Handling	9
4.8.1. Leap Smearing	9
5. NTP in Embedded Devices	10
5.1. Updating Embedded Devices	10
5.2. KISS Packets	10
5.3. Server configuration	11
5.3.1. Get a vendor subdomain for pool.ntp.org	11
6. NTP Deployment Examples	11
6.1. Client-Only configuration	11
6.2. Server-Only Configuration	11
6.3. Anycast	11
7. Acknowledgements	12
8. IANA Considerations	12
9. Security Considerations	13
10. References	13
10.1. Normative References	13
10.2. URIs	13
Authors' Addresses	14

1. Introduction

NTP Version 4 (NTPv4) has been widely used since its publication as RFC 5905 [RFC5905]. This documentation is a collection of Best Practices from across the NTP community.

1.1. 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].

2. Keeping NTP up to date

No software (not even NTP) is perfect. Bugs can be present in any software. Even if software is thoroughly tested and "all" the bugs are discovered and fixed, users continuously find new ways to use software that their authors did not conceive of, which can uncover more bugs. Thousands of individual bugs have been found and fixed in the NTP Project's reference implementation since the first NTPv4 release in 1997.

There are always new ideas about security on the Internet, and an application which is secure today could be insecure tomorrow once an unknown bug (or a known behavior) is exploited in the right way. Many security mechanisms rely on time, either directly or indirectly, as part of their operation. If an attacker can spoof the time, they may be able to bypass or neutralize other security elements. For example, incorrect time can disrupt the ability to reconcile logfile entries on the affected system with events on other systems.

In general, the best way to protect yourself and your networks against these bugs and security threats is to make sure that you keep your NTP implementation up-to-date. There are multiple versions of the NTP protocol in use and multiple implementations and versions of NTP software also in use, on many different platforms. It is recommended that NTP users actively monitor wherever they get their software to find out if their versions are vulnerable to any known attacks, and deploy updates containing security fixes as soon as practical.

The reference implementation of NTP Version 4 from Network Time Foundation (NTF) continues to be actively maintained and developed by NTF's NTP Project, with help from volunteers and NTF's supporters. The NTP software can be downloaded from ntp.org [1] and also from NTF's github page [2].

3. General Network Security Best Practices

NTP deployments are only as secure as the networks they are running over.

3.1. BCP 38

Many network attacks rely on modifying the IP source address of a packet to point to a different IP address than the computer which originated it. This modification/abuse vector has been known for quite some time, and BCP 38 [RFC2827] was approved in 2000 to address this. BCP 38 [RFC2827] calls for filtering outgoing and incoming traffic to make sure that the source and destination IP addresses are consistent with the expected flow of traffic on each network interface. It is recommended that all networks (and ISP's of any size) implement this. If a machine on your network is sending out packets claiming to be from an address that is not on your network, this could be the first indication that you have a machine that has been cracked, and is being used abusively. If packets are arriving on an external interface with a source address that should only be seen on an internal network, that's a strong indication that an attacker is trying to inject spoofed packets into your network. More information is available at <http://www.bcp38.info>.

4. NTP Configuration Best Practices

NTP can be made more secure by making a few simple changes to the `ntp.conf` file.

4.1. Use enough time sources

NTP takes the available sources of time and submits their timing data to intersection and clustering algorithms, looking for the best idea of the correct time. If there is only 1 source of time, the answer is obvious. It may not be a good source of time, but it's the only one. If there are 2 sources of time and they agree well enough, that's good. But if they don't, then `ntpd` has no way to know which source to believe. This gets easier if there are 3 sources of time. But if one of those 3 sources becomes unreachable or unusable, we're back to only having 2 time sources. 4 sources of time is another interesting choice, assuming things go well. If one of these sources develops a problem there are still 3 others. Seems good. Until the leap second we had in June of 2015, where several operators implemented leap smearing while others did not. See Section 4.8.1 for more information.

Starting with `ntp-4.2.6`, the 'pool' directive will spin up "enough" associations to provide robust time service, and will disconnect poor servers and add in new servers as-needed.

Monitor your `ntpd` instances. If your times sources do not generally agree, find out why and either correct the problems or stop using defective servers. See Section 4.4 for more information.

4.2. Use a diversity of Reference Clocks

If you are using reference clocks, it is recommended that you use several different types. Having a diversity of sources means that any one issue is less likely to cause a service interruption.

Are all your clocks from the same vendor? Are they using the same base chipset, regardless of whether or not the finished products are from different vendors? Are they all running the same version of firmware? A systemic problem with time from any satellite navigation service is possible and has happened. Sunspot activity can render satellite or radio-based time source unusable. A chipset problem can happen. There may be a bug in the vendor's firmware.

4.3. Mode 6 and 7

NTP Mode 6 (ntpq) and Mode 7 (ntpd) packets are designed to permit monitoring and optional authenticated control of ntpd and its configuration. Used properly, these facilities provide vital debugging and performance information and control. Used improperly, these facilities can be an abuse vector.

Mode 7 queries have been disabled by default since 4.2.7p230, released on 2011/11/01. Unless you have a good reason for using ntpdc, do not enable Mode 7.

The ability to use Mode 6 beyond its basic monitoring capabilities can be limited to authenticated sessions that provide a 'controlkey', and similarly, if Mode 7 has been explicitly enabled its use for more than basic monitoring can be limited to authenticated sessions that provide a 'requestkey'.

Older versions of the reference implementation of NTP could be abused to participate in high-bandwidth DDoS attacks. Starting with ntp-4.2.7p26, released in April of 2010, ntpd requires the use of a nonce before replying with potentially large response packets.

As mentioned above, there are two general ways to use Mode 6 and Mode 7 requests. One way is to query ntpd for information, and this mode can be disabled with:

```
restrict ... noquery
```

The second way to use Mode 6 and Mode 7 requests is to modify ntpd's behavior. Modification of ntpd ordinarily requires an authenticated session. By default, if no authentication keys have been specified no modifications can be made. For additional protection, the ability to perform these modifications can be controlled with:

```
restrict ... nomodify
```

Users can prevent their NTP servers from participating by adding the following to their ntp.conf file:

```
restrict default -4 nomodify notrap nopeer noquery
```

```
restrict default -6 nomodify notrap nopeer noquery
```

```
restrict source nomodify notrap noquery # nopeer is OK if you don't  
use the 'pool' directive
```

4.4. Monitoring

The reference implementation of NTP allows remote monitoring. The access to this service is controlled by the restrict statement in NTP's configuration file (ntp.conf). The syntax reads:

```
restrict address mask address_mask nomodify
```

Monitor your ntpd instances so machines that are "out of sync" can be quickly identified. Monitor your system logs for messages from ntpd so abuse attempts can be quickly identified.

If your system starts getting unexpected time replies from its time servers, that can be an indication that the IP address of your server is being forged in requests to that time server, and these abusers are trying to convince your time servers to stop serving time to you.

If your system is a broadcast client and your syslog shows that you are receiving "early" time messages from your server, that is an indication that somebody may be forging packets from a broadcast server.

If your syslog shows messages that indicate you are receiving timestamps that are earlier than the current system time, then either your system clock is unusually fast or somebody is trying to launch a replay attack against your server.

If you are using broadcast mode and have ntp-4.2.8p6 or later, use the 4th field of the ntp.keys file to identify the IPs of machines that are allowed to serve time to the group.

4.5. Security

In the standard configuration NTP packets are exchanged unprotected between client and server. An adversary that is able to become a Man-In-The-Middle is therefore able to drop, replay or modify the

content of the NTP packet, which leads to degradation of the time synchronization or the transmission of false time information. A profound threat analysis for time synchronization protocols are given in RFC 7384 [RFC7384]. NTP provides two security measures to protect authenticity and integrity of the NTP packets. Both measures protect the NTP packet by means of a Message Authentication Code (MAC). Neither of them encrypts the NTP's payload, because it is not considered to be confidential.

4.5.1. Pre-Shared Key Approach

This approach applies a symmetric key for the calculation of the MAC, which protects authenticity and integrity of the exchanged packets for a association. NTP does not provide a mechanism for the exchange of the keys between the associated nodes. Therefore, for each association, keys have to be exchanged securely by external means. It is recommended that each association is protected by its own unique key. NTP does not provide a mechanism to automatically refresh the applied keys. It is therefore recommended that the participants periodically agree on a fresh key. The calculation of the MAC may always be based on an MD5 hash. If the NTP daemon is built against an OpenSSL library, NTP can also base the calculation of the MAC upon the SHA-1 or any other digest algorithm supported by each side's OpenSSL library.

To use this approach the communication partners have to exchange the key, which consists of a keyid with a value between 1 and 65534, inclusive, and a label which indicates the chosen digest algorithm. Each communication partner adds this information to their key file in the form:

```
keyid label key
```

The key file contains the key in clear text. Therefore it should only be readable by the NTP process. Different keys are added line by line to the key file.

A NTP client establishes a protected association by appending the option "key keyid" to the server statement in the NTP configuration file:

```
server address key keyid
```

Note that the NTP process has to trust the applied key. An NTP process explicitly has to add each key it want to trust to a list of trusted keys by the "trustedkey" statement in the NTP configuration file.

trustedkey keyid_1 keyid_2 ... keyid_n

4.5.2. Autokey

Autokey was designed in 2003 to provide a means for clients to authenticate servers. By 2011, security researchers had identified computational areas in the Autokey protocol that, while secure at the time of its original design, were no longer secure. Work was begun on an enhanced replacement for Autokey, which was called Network Time Security (NTS) [3]. NTS was published in the summer of 2013. As of February 2016, this effort was at draft #13, and about to begin 'final call'. The first unicast implementation of NTS was started in the summer of 2015 and is expected to be released in the summer of 2016.

We recommend that Autokey NOT BE USED. Know that as of the fall of 2011, a common(?) laptop computer could crack the security cookie used in the Autokey protocol in 30 minutes' time. If you must use Autokey, know that your session keys should be set to expire in under 30 minutes' time. If you have reason to believe your autokey-protected associations will be attacked, you should read <https://lists.ntp.org/pipermail/ntpwg/2011-August/001714.html> and decide what resources your attackers might be using, and adjust the session key expiration time accordingly.

4.5.3. External Security Means

TBD

4.6. Using Pool Servers

It only takes a small amount of bandwidth and system resources to synchronize one NTP client, but NTP servers that can service tens of thousands of clients take more resources to run. Users who want to synchronize their computers should only synchronize to servers that they have permission to use.

The NTP pool project is a collection of volunteers who have donated their computing and bandwidth resources to provide time on the Internet for free. The time is generally of good quality, but comes with no guarantee whatsoever. If you are interested in using the pool, please review their instructions at <http://www.pool.ntp.org/en/use.html>.

If you want to synchronize many computers using the pool, consider running your own NTP servers, synchronizing them to the pool, and synchronizing your clients to your in-house NTP servers. This reduces the load on the pool.

Set up or sponsor one or more pool servers.

4.7. Starting, Cold-Starting, and Re-Starting NTP

Only use -g on cold-start. Other things TBD.

Editor's Note: I think I'd like to expand this a bit to cover how to deal with NTP stopping, when to restart it, and under what circumstances to not restart it!

4.8. Leap Second Handling

The UTC timescale is kept in sync with the rotation of the earth through the use of leap seconds. NTP time is based on the UTC timescale, and the protocol has the capability to broadcast leap second information. Some GNSS systems (like GPS) broadcast leap second information, so if you have a Stratum-1 server synced to GNSS (or you are synced to a lower stratum server that is ultimately synced to GNSS), you will get advance notification of impending leap seconds automatically.

The International Earth Rotation and Reference Systems Service (IERS) is responsible to announce the introduction of a leap second. It maintains a leap second list at <https://hpiers.obspm.fr/iers/bul/bulc/ntp/leap-seconds.list> for NTP users who are not receiving leap second information through an automatic source. After fetching the leap seconds file onto the server, add this line to ntpd.conf to apply the file:

```
leapfile "/path/to your/leap-file"
```

You will need to restart to apply the changes.

4.8.1. Leap Smearing

Some NTP installations may instead make use of a technique called "Leap Smearing". With this method, instead of introducing an extra second (or eliminating a second), NTP time will be slewed in small increments over a comparably large window of time around the leap second event. The amount of the slew should be small enough that clients will follow the smeared time without objecting. During the adjustment window, the NTP server's time may be offset from UTC by as much as .5 seconds. This is done to enable software that doesn't deal with minutes that have more or less than 60 seconds to function correctly, at the expense of fidelity to UTC during the smear window.

Leap Smearing was introduced in ntpd versions 4.2.8.p3 and 4.3.47. Support is not configured by default and must be added at compile

time. In addition, no leap smearing will occur unless a leap smear interval is specified in `ntpd.conf`. For more information, refer to <http://bkl.ntp.org/ntp-stable/README.leapsmear?PAGE=anno>.

Leap Smearing is not recommended for public-facing NTP servers, as they will disagree with non-smearing servers during the leap smear interval. However, some public-facing servers may be configured this way anyway. Users are advised to be aware of impending leap seconds and how the servers (inside and outside their organization) they are using deal with them.

5. NTP in Embedded Devices

Readers of this BCP already understand how important accurate time is for network computing. And as computing becomes more ubiquitous, there will be many small "Internet of Things" devices that require accurate time. These embedded devices may not have a traditional user interface, but if they connect to the Internet they will be subject to the same security threats as traditional deployments.

5.1. Updating Embedded Devices

Vendors of embedded devices have a special responsibility to pay attention to the current state of NTP bugs and security issues, because their customers usually don't have the ability to update their NTP implementation on their own. Those devices may have a single firmware upgrade, provided by the manufacturer, that updates all capabilities at once. This means that the vendor assumes the responsibility of making sure their devices have the latest NTP updates applied.

This should also include the ability to update the NTP server address.

(Note: do we find specific historical instances of devices behaving badly and cite them here?)

5.2. KISS Packets

The "Kiss-o'-Death" (KoD) packet is a rate limiting mechanism where a server can tell a misbehaving client to "back off" its query rate. It is important for all NTP devices to respect these packets and back off when asked to do so by a server. It is even more important for an embedded device, which may not have exposed a control interface for NTP.

The KoD mechanism relies on clients behaving properly in order to be effective. Some clients ignore the KoD packet entirely, and other

poorly-implemented clients might unintentionally increase their poll rate and simulate a denial of service attack. Server administrators should be prepared for this and take measures outside of the NTP protocol to drop packets from misbehaving clients.

5.3. Server configuration

Vendors of embedded devices that need time synchronization should also carefully consider where they get their time from. There are several public-facing NTP servers available, but they may not be prepared to service requests from thousands of new devices on the Internet.

Vendors are encouraged to invest resources into providing their own time servers for their devices.

5.3.1. Get a vendor subdomain for pool.ntp.org

The NTP Pool Project offers a program where vendors can obtain their own subdomain that is part of the NTP Pool. This offers vendors the ability to safely make use of the time distributed by the Pool for their devices. Vendors are encouraged to support the pool if they participate. For more information, visit <http://www.pool.ntp.org/en/vendors.html>.

6. NTP Deployment Examples

A few examples of interesting NTP Deployments

6.1. Client-Only configuration

TBD

6.2. Server-Only Configuration

TBD

6.3. Anycast

Anycast is described in BCP 126 [RFC4786]. (Also see RFC 7094 [RFC7094]). With anycast, a single IP address is assigned to multiple interfaces, and routers direct packets to the closest active interface.

Anycast is often used for Internet services at known IP addresses, such as DNS. Anycast can also be used in large organizations to simplify configuration of a large number of NTP clients. Each client can be configured with the same NTP server IP address, and a pool of

anycast servers can be deployed to service those requests. New servers can be added to or taken from the pool, and other than a temporary loss of service while a server is taken down, these additions can be transparent to the clients.

If clients are connected to an NTP server via anycast, the client does not know which particular server they are connected to. As anycast servers may arbitrarily enter and leave the network, the server a particular client is connected to may change. This may cause a small shift in time from the perspective of the client when the server it is connected to changes. It is recommended that anycast be deployed in environments where these small shifts can be tolerated.

Configuration of an anycast interface is independent of NTP. Clients will always connect to the closest server, even if that server is having NTP issues. It is recommended that anycast NTP implementations have an independent method of monitoring the performance of NTP on a server. In the event the server is not performing to specification, it should remove itself from the Anycast network. It is also recommended that each Anycast NTP server have at least one Unicast interface, so its performance can be checked independently of the anycast routing scheme.

One useful application in large networks is to use a hybrid unicast/anycast approach. Stratum 1 NTP servers can be deployed with unicast interfaces at several sites. Each site may have several Stratum 2 servers with a unicast interface and an anycast interface (with a shared IP address per location). The unicast interfaces can be used to obtain time from the Stratum 1 servers globally (and perhaps peer with the other Stratum 2 servers at their site). Clients at each site can be configured to use the shared anycast address for their site, simplifying their configuration. Keeping the anycast routing restricted on a per-site basis will minimize the disruption at the client if its closest anycast server changes.

7. Acknowledgements

The author wishes to acknowledge the contributions of Sue Graves, Samuel Weiler, Lisa Perdue, and Karen O'Donoghue.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

TBD

10. References

10.1. 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>>.
- [RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827, May 2000, <<http://www.rfc-editor.org/info/rfc2827>>.
- [RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <<http://www.rfc-editor.org/info/rfc4786>>.
- [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>>.
- [RFC7094] McPherson, D., Oran, D., Thaler, D., and E. Osterweil, "Architectural Considerations of IP Anycast", RFC 7094, DOI 10.17487/RFC7094, January 2014, <<http://www.rfc-editor.org/info/rfc7094>>.
- [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>>.

10.2. URIs

- [1] <http://www.ntp.org/downloads.html>
- [2] <https://github.com/ntp-project/ntp>
- [3] <https://tools.ietf.org/html/draft-ietf-ntp-network-time-security-00>

Authors' Addresses

Denis Reilly
Spectracom Corporation
1565 Jefferson Road, Suite 460
Rochester, NY 14623
US

Email: denis.reilly@spectracom.orolia.com

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwttime.org

Dieter Sibold
Physikalisch-Technische Bundesanstalt
Bundesallee 100
Braunschweig D-38116
Germany

Phone: +49-(0)531-592-8420
Fax: +49-531-592-698420
Email: dieter.sibold@ptb.de

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 26, 2019

H. Stenn
Network Time Foundation
March 25, 2019

Network Time Protocol I-Do Extension Field
draft-stenn-ntp-i-do-06

Abstract

This proposal defines and describes a mechanism by which cooperating NTP instances may communicate any optional features they are willing to admit they support.

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH BEFORE PUBLISHING:

The source code and issues list for this draft can be found in <https://github.com/hstenn/ietf-ntp-i-do>

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 <https://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 September 26, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	2
2. The I-Do Extension Field	2
2.1. Overview	2
2.2. I-DO Packet Format	4
2.3. Behavior	5
3. Acknowledgements	6
4. IANA Considerations	6
5. Security Considerations	6
6. References	6
6.1. Normative References	6
6.2. Informative References	7
Author's Address	7

1. Introduction

The first implementation of NTPv4 was released in 2003, and was defined by RFC 5905 [RFC5905]. It contains an optional and now obsolete public-key security protocol, Autokey, which is defined by RFC 5906 [RFC5906]. Until very recently, Autokey has been the only implemented use of NTP packet Extension Fields. New proposals for extension fields are being written and there is currently no convenient way to learn if a remote instance of NTP supports any extension fields or not. This proposal contains a method to tell a remote instance of NTP what we (are willing to admit we) support, and ask what they (are willing to admit they) support.

1.1. 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].

2. The I-Do Extension Field

2.1. Overview

The purpose of the I-DO EF is to provide information to the remote side about our capabilities.

If an incoming packet contains an unrecognized extension field, one of several things will happen. While that unrecognized extension

field SHOULD be ignored, an implementation MAY choose to drop the entire packet.

If any extension field is present there ordinarily SHOULD be a MAC following the extension field. However, an older conforming NTP implementation will require that any EF MUST be followed by a MAC.

Some extension fields are unable to be "signed" by a MAC, regardless of whether or not that MAC is a traditional MAC or an extension field MAC.

In the previous two cases, a conforming legacy system that receives these types of packets will interpret the unrecognized EF as a missing or legacy MAC, and return a crypto-NAK.

If the remote system replies with a crypto-NAK, that is a good indication that it is running older software that does not recognize EFs and thinks we have sent an invalid MAC. In this case, we SHOULD NOT send that system newer EFs.

If the remote system replies without including an I-DO-RESPONSE EF, we at least know they can handle EFs, but they either don't understand I-DO or are not willing to tell us anything. In this case, we SHOULD NOT send any newer EFs.

If the remote system replies with a packet that includes an I-DO-RESPONSE EF, then we SHOULD remember what they told us, and use that information appropriately. In other words, we can exchange packets containing any new EFs that we agree on, and we should not exchange packets containing any new EFs that we have not agreed on.

In client/server mode, it makes sense for the client to send an I-DO to the server, and notice how the server responds. While the server SHOULD respond with an I-DO-RESPONSE EF, it likely does not make sense for the server to send an I-DO EF in response to a client request.

In symmetric mode, either side may initiate sending an I-DO EF, and the receiving side SHOULD reply with an I-DO-RESPONSE EF.

In broadcast mode, the broadcast server MAY send broadcast packets that include an I-DO EF, but note that if, counter to recommended practice, these packets are unauthenticated they MAY cause client machines to misinterpret the packet as having invalid authentication. In this situation, the broadcast server SHOULD alternate sending broadcast server packets with and without an I-DO EF, to insure that all clients receive time packets they will accept. Note that if, as recommended, broadcast packets are authenticated, a conforming client

SHOULD have no difficulty in receiving a broadcast (mode 5) packet from a server that includes an I-DO EF.

2.2. I-DO Packet Format

The content of the I-DO extension field is an ordinary four octet Extension Field header followed by a payload consisting of an appropriate number of two octet I-DO values that use nonzero values to indicate a supported feature. An I-DO value of zero is ignored. The payload section must end on a four-octet boundary.

There are two types of nonzero I-DO values that may be used. They are both defined in the IANA NTP Extension Field Table (Section 4). These values are either Extension Field Types, where only the low-order values (0x01 thru 0xFE) are used, or I-DO Types, where all 16 bits are used and the bottom octet is currently always 0xFF.

The examples below are built using information from the following Standards and proposals:

RFC 5906 [RFC5906]

NTP-EXTENSION-FIELDS [NTP-EXTENSION-FIELD]

MAC-LAST-EF [DRAFT-MAC-LAST-EF]

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Field Type										Field Length																													
I-Do 1										...																													
I-Do N										Padding																													

NTP Extension Field: I-DO - Overview

Field Type: TBD (Recommendation for IANA: 0x0007 (I-Do), 0x8007 (I-Do Response))

Field Length: as needed

Payload: An enumeration of the supported base Field Types, followed by any zero padding (0x0000) needed to fill the payload to the desired 32-bit boundary.

Example: A system that wants to advertise support for Autokey and I-DO, sending to a system that responds with support for I-DO, NTS, MAC-EF, and LAST-EF.

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								
+-----+																																							

NTP Extension Field: I-Do - Example

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								
+-----+																																							

NTP Extension Field: I-Do Response - Example

2.3. Behavior

The sender of any I-Do extension field MUST send an extension field with a Field Type of 0x0007 (I-Do) and SHOULD include a payload with any 0x0000 padding values after enumerating the supported base Extension Field Types. If the responding system recognizes the I-Do extension field, its response MUST include an extension field with a Field Type of 0x8007 (I-Do Response), and SHOULD include a payload with any 0x0000 padding values after enumerating the supported base Extension Field Types.

Any system that receives an I-Do extension field as either an "offer" or a "response" SHOULD scan the entire payload looking for nonzero values that specify the capabilities of the remote association.

Any system that receives an I-Do "offer", 0x0007, SHOULD reply with an I-Do "response", 0x8007.

Any system that sends an I-Do "offer" or "response" may send as few or as many of its supported Field Types as it chooses. At any subsequent time, either side may re-negotiate the list of supported

field types it is prepared to accept from the other system by sending a new I-Do extension field.

The most-recently received I-Do list replaces any previous I-Do list.

3. Acknowledgements

The author wishes to acknowledge the contributions of Sam Weiler.

4. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types:

0x0007 (I-DO)

0x8007 (I-DO Response)

and NTP Extension Field I-DO types:

0x00FF through

0xFDFE Reserved for future I-DO types

0xFEFF (I-DO Leap Smear REFIDs)

0xFFFF (I-DO IPv6 REFID hash)

for this proposal.

5. Security Considerations

No additional or unusual security considerations are expected if this proposal is adopted.

No feedback has been received suggesting this proposal creates any new security considerations.

6. References

6.1. Normative References

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

- [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>>.

6.2. Informative References

- [DRAFT-MAC-LAST-EF] Stenn, H., "draft-stenn-ntp-mac-last-ef", 2018.
- [NTP-EXTENSION-FIELD] Stenn, H., "draft-stenn-ntp-extension-fields", 2018.
- [RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, DOI 10.17487/RFC5906, June 2010, <<https://www.rfc-editor.org/info/rfc5906>>.

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwttime.org

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2016

H. Stenn
Network Time Foundation
March 14, 2016

Network Time Protocol IPv6 REFID Hash
draft-stenn-ntp-ipv6-refid-hash-00

Abstract

RFC 5905 [RFC5905], section 7.3, "Packet Header Variables", defines the value to be used as the REFID for network associations. For IPv4 associations the IPv4 address is used, and for IPv6 associations four octets of the MD5 hash of the IPv6 are used. Often, the REFID is simplistically and incorrectly used to identify upstream servers. While this works in an IPv4 network, it doesn't work for IPv6 associations and may have other problems in an environment with mixed use of IPv4 and IPv6. Specifically, the NTP Project has received a report where the generated IPv6 hash decoded to the IPv4 address of a different machine on the system peer's network.

This proposal offers a way for a system to generate a REFID for a system peer that communicates over IPv6 that does not conflict with a valid IPv4-based REFID.

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 September 15, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
2. Augmenting the IPv6 REFID Hash	3
3. Potential Problems	3
4. Questions	4
5. Acknowledgements	4
6. IANA Considerations	4
7. Security Considerations	4
8. Normative References	4
Author's Address	4

1. Introduction

RFC 5905 [RFC5905], section 7.3, "Packet Header Variables", defines the value to be used as the REFID for network associations. It says:

If using the IPv4 address family, the identifier is the four-octet IPv4 address. If using the IPv6 family, it is the first four octets of the MD5 hash of the IPv6 address. ...

Often, the REFID is simplistically and incorrectly used to identify upstream servers. While this works in an IPv4 network, it doesn't work for IPv6 associations and may have other problems in an environment with mixed use of IPv4 and IPv6. Specifically, the NTP Project has received a report where the generated IPv6 hash decoded to the IPv4 address of a different machine on the system peer's network.

This proposal offers a way for a system to generate a REFID for a system peer that communicates over IPv6 that does not conflict with a valid IPv4-based REFID.

1.1. 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].

2. Augmenting the IPv6 REFID Hash

When generating a REFID based on a network system peer, the NTPv4 specification says:

If using the IPv4 address family, the identifier is the four-octet IPv4 address. If using the IPv6 family, it is the first four octets of the MD5 hash of the IPv6 address. ...

This means that the IPv4 representation of the IPv6 hash would be: b1.b2.b3.b4 . The proposal is that the system MAY also use 255.b2.b3.b4 as its REFID.

When using the REFID to check for a timing loop for an IPv6 association, if the code that checks the first four-octets of the hash fails to match then the code must check again, using 0xFF as the first octet of the hash.

3. Potential Problems

There is a 1 in 16,777,216 chance that the REFID hashes of two IPv6 addresses will be identical, producing a false-positive loop detection. With a sufficient number of servers, the risk of this problem becomes a non-issue. The use of the "REFID Suggestion" extension field is also a way to mitigate this potential situation.

Unrealistically, if only two instances of NTP are communicating via IPv6 and one side implements this new IPv4 REFID hash and the other side does not, the "other side" will not be able to detect this loop condition. In this case, the two machines will slowly increase their Stratum until they reach S16 and become unsynchronized. This situation is considered to be unrealistic because the only current way this could happen would be for there to only be these two instances of NTP available as time sources in a misconfigured "orphan mode" setup. There is no risk of this happening in an NTP network with 3 or more time sources, or in a properly-configured "time island" setup.

4. Questions

Should we ask IANA to allocate a pseudo Extension Field Type of 0xFFFF (for example) so the proposed "I-Do" exchange can report whether or not the "IPv6 REFID Hash" is supported?

5. Acknowledgements

The author wishes to acknowledge Dan Mahoney (and perhaps others) for suggesting the idea of using an "impossible" first-octet value to indicate an IPv6 refid hash. The author wishes to acknowledge the contributions of Joey Saccadonuts.

6. IANA Considerations

This memo makes no requests of IANA.

7. Security Considerations

Additional information TBD

8. 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>>.
- [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>>.

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwttime.org

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2016

H. Stenn
Network Time Foundation
March 14, 2016

Network Time Protocol Last Extension Field
draft-stenn-ntp-last-extension-00

Abstract

NTPv4 is defined by RFC 5905 [RFC5905], and it and earlier versions of the NTP Protocol have supported symmetric private key MAC authentication. MACs pre-date the Extension Fields introduced in RFC 5905 [RFC5905], and as the number of Extension Fields grows there is an increasing chance of ambiguity when deciding if the "next" set of data is an Extension Field or a MAC. This proposal defines a new Extension Field which is used to signify that it is the last Extension Field in the packet. If present, any subsequent data SHOULD be considered to be a legacy MAC.

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 September 15, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	2
2. The Last Extension Field Extension Field	2
3. Acknowledgements	3
4. IANA Considerations	3
5. Security Considerations	4
6. Normative References	4
Author's Address	4

1. Introduction

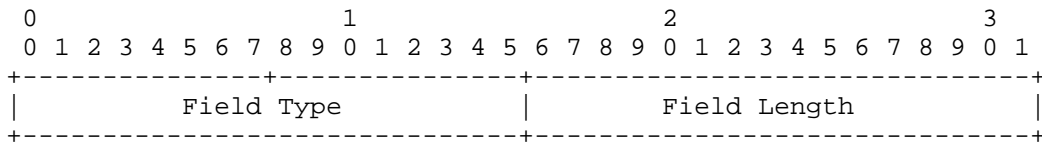
NTPv4 is defined by RFC 5905 [RFC5905], and it and earlier versions of the NTP Protocol have supported symmetric private key MAC authentication. MACs pre-date the Extension Fields introduced in RFC 5905 [RFC5905], and as the number of Extension Fields grows there is an increasing chance of ambiguity when deciding if the "next" set of data is an Extension Field or a MAC. This proposal defines a new Extension Field which is used to signify that it is the last Extension Field in the packet. If present, any subsequent data SHOULD be considered to be a legacy MAC.

1.1. 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].

2. The Last Extension Field Extension Field

Now that multiple extension fields are a possibility, and the chance that additional packet data could be an Extension Field or an old-style MAC, having a means to indicate that there are no more Extension Fields in an NTP packet, and any subsequent data MUST be something else, almost certainly an old-style MAC, is a valuable facility.



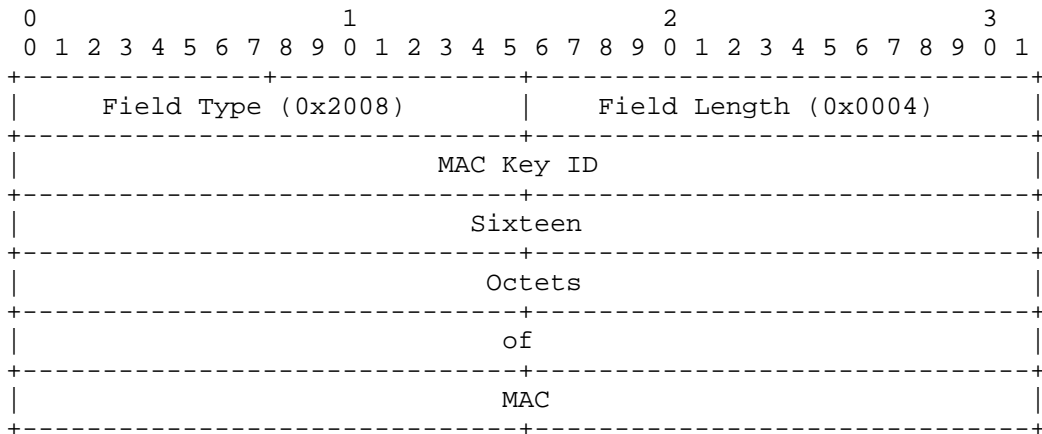
NTP Extension Field: Last Extension Field

Field Type: TBD (Recommendation for IANA: 0x2008 (Last Extension Field, MAC OPTIONAL))

Field Length: 4

Payload: None.

Example:



Example: NTP Extension Field: Last Extension Field

3. Acknowledgements

The author wishes to acknowledge the contributions of Joey Saccadonuts.

4. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Types 0x0007 (I-Do), 0x2007 (I-Do, MAC OPTIONAL), 0x4007 (I-Do Response), and 0x6007 (I-Do Response, MAC OPTIONAL) for this proposal.

5. Security Considerations

Additional information TBD

6. 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>>.
- [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>>.
- [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>>.

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwttime.org

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 26, 2019

H. Stenn
Network Time Foundation
March 25, 2019

Network Time Protocol Leap Smear REFID
draft-stenn-ntp-leap-smear-refid-02

Abstract

Leap Seconds are part of UTC. NTP timestamps are based on POSIX timestamps, which require each day to have exactly 86,400 seconds per day. Some applications and environments choose to "smear" leap second corrections over a period that can last up to 24 hours' time, and implement NTP servers that offer smeared time to clients asking them for the time.

Both NTP clients and operators have no current way to tell if an NTP server is offering leap-smeared time or not. This is a problem.

Similarly, an NTP server may choose to offer leap-smeared time to clients that do not appear to know that a leap event is in-process. This is a problem.

This proposal offers a mechanism that provides a simple and clean solution to problems, by giving a way that clients (and operators) can trivially ask for leap-smeared time and detect a server that is offering leap-smeared time.

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 <https://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 September 26, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
2. Leap Smear REFID	3
3. Acknowledgements	5
4. IANA Considerations	5
5. Security Considerations	5
6. Normative References	5
Author's Address	5

1. Introduction

Leap Seconds are applied as needed to UTC in order to keep its time of day close to UT1's mean solar time.

RFC 5905 [RFC5905] and earlier versions of NTP are the overwhelming method of distributing time on networks. The timescale used by NTP is based on POSIX which, for better or worse, ignores any instances where there are not the ordinary 86,400 seconds per day.

Leap Seconds will continue to exist for the foreseeable future, and similarly, POSIX can be expected to ignore leap seconds for the foreseeable future.

Different applications have different requirements for the stability of time during the application of a leap second. Some applications are tolerant of a fast application of the correction, while other applications prefer to "smear" the leap second over a longer period, where the time reported by leap-second aware servers is gradually applied so there is no abrupt change to time during the processing of a leap second.

Additionally, some use-cases for calculating elapsed time (a "difference clock") that use POSIX timestamps are greatly complicated in the possible presence of a leap-second corrections. If the presence of leap-smeared time is of greater value than legally-correct time, leap smearing is the choice some administrators will take.

1.1. Requirements Language

2. Leap Smear REPID

All NTP time values are represented in twos-complement format, with bits numbered in big-endian (as described in Appendix A of [RFC0791]) fashion from zero starting at the left, or high-order, position.

```

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
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Seconds                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Fraction                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

NTP Timestamp Format (32:32)

This format provides coverage for 136 years' time to a precision of 232 picoseconds. If a leap-second addition is being completely smeared just before the stroke of the next POSIX second then the smear correction will be (0,1). If this was the only way to apply a leap smear correction then we could simply use an unsigned value to represent the correction. But while the first popular leap smear implementation applied the correction over an appropriate number of hours' time before the actual leap second so the system time was corrected at the stroke of 00:00, that meant that the difference between system time and UTC spent half of the duration of the smear application at [.5,1) "off" of correct time. The second popular implementation of the leap smear applied the first half-second correction before the stroke of 00:00 for a correction range of (0,.5] and the last half-second correction starting at the stroke of 00:00 for a [-.5,0) correction range. This also means we need a signed value to represent the amount of correction.

If a system implements the leap-smear REFID, the REFID of a system that is supplying smeared time to client requests while leap-smear correction is active would be 254.b1.b2.b3, where the three octets (b1, b2, and b3) are a 2:22 formatted value, yielding precision to 238 nanoseconds, or about a quarter of a microsecond.

Note that if an NTP server decides to offer smeared time corrections to clients, it SHOULD only offer this time in response to CLIENT time requests. There is something to be said for further only offering smeared time to CLIENT time requests that show an LI value of 0, and perhaps 3. The reason for this is that if a client knows a leap second is pending, it can be expected to know how to process that leap second. An NTP server that is offering smeared time SHOULD NOT send smeared time in any peer exchanges. Also, CLIENT machines SHOULD NOT be distributing time (smeared or otherwise) to other systems.

We also note that during the application of a leap smear, the REFID from a system offering smeared time cannot provide detection of a timing loop. This is not expected to be a problem because time server systems are not expected to make CLIENT connections with each other, so they should not be receiving smeared time. Moreso, if a time server is configured to make CLIENT connections to a server that offers smeared time, with the mechanism described here it can detect when it is getting smeared time, and either ignore time from that source, or "undo" the leap smear correction and use the corrected time for that sample.

This proposal is not an attempt to justify servers offering leap smeared time. Its purpose is to make it easy to identify when a

client is receiving smeared time, and provide the client a way to know the amount of smear correction as of the latest successful poll.

3. Acknowledgements

The author wishes to acknowledge the contributions of Juergen Perlinger.

4. IANA Considerations

This memo requests that IANA allocate a pseudo Extension Field Type of 0xFEFF so the proposed "I-Do" exchange can report whether or not this server can offer leap smeared time in response to CLIENT time requests, identifying the amount of correction using the above REFID.

5. Security Considerations

No special or unusual security issues have been identified that are directly related to this proposal.

Additional information TBD.

6. Normative References

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

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwttime.org

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 26, 2019

H. Stenn
Network Time Foundation
March 25, 2019

Network Time Protocol Suggested REFID Extension Field
draft-stenn-ntp-suggest-refid-05

Abstract

NTP's Reference ID, or REFID, identifies the source of time in a timestamp or time packet. In NTP packets sent over the network the REFID is used to identify the "system peer", and in the long-term general case its fundamental purpose is to prevent a one-degree timing loop. Each instance of NTP decides for itself what REFID it will put in its outgoing packets, and there is currently no way for an external time source to tell or recommend this value in the case where that external time source is selected as the "system peer."

The SUGGESTED-REFID NTP Extension Field proposal is a backward-compatible way for a time source to tell its peers or clients "If you use me as your system peer, use this nonce as your REFID."

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 <https://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 September 26, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
2. The REFID	3
3. The Suggested REFID Extension Field	4
4. Generating and Sending a Nonce as the Suggested REFID Extension Field	4
5. Remembering a Nonce Suggested REFID Extension Field	5
6. The Suggested REFID Extension Field and Leap Smear REFIDs . .	5
7. Acknowledgements	6
8. IANA Considerations	6
9. Security Considerations	6
10. References	6
10.1. Normative References	6
10.2. Informative References	7
Author's Address	7

1. Introduction

NTP has been widely used through several revisions, with the latest being RFC 5905 [RFC5905]. A core component of the protocol and the algorithms is the Reference ID, or REFID, which is used to identify the time source. Traditionally, when the source of time was another system the REFID was the IPv4 address of that other system. If the remote system was using IPv6 for its connection, a 4 octet digest value of the IPv6 address was used. The general case core purpose of the REFID is to prevent a one-degree timing loop (where if A has several timing sources that include B, if B decides to get its time from A we don't want A then deciding to get its time from B). The REFID is considered to be "public data" and is a vital core-component of the base NTP packet. In an increasingly hostile Internet, knowledge of a system's time source is abusable information. If a system's REFID is the IPv4 address of its system peer, an attacker can try to use that information to send spoofed time packets to either or both the target or the target's server, attempting to cause a disruption in time service. There is also a clear use-case for having a special REFID for use if systems are exchanging leap-smeared time. This proposal is a backward-compatible way for a time source to tell its peers or clients "If you use me as your system peer, use

this nonce as your REFID." This nonce, a Suggested REFID, SHOULD be untraceable to the sending system. When used to hide the identity of a server, if the receiving system uses this Suggested REFID nonce instead of the IPv4 address as its REFID, this type of attack and information disclosure is prevented. When used to indicate that a system is either offering leap-smear time or is synchronized to a leap-smear time source, this information can be used to prevent unwanted synchronization to a source that is not offering the "flavor" of time we want, and, in the case where a leap smear correction continues into the next day, the second half of a leap smear correction can be applied in the expected manner.

This SUGGESTED-REFID NTP Extension Field proposal is a simple, clean, backward-compatible way for an external time source to request that the receiving system use the provided nonce in the case where the receiving system uses the sending system as its system peer.

1.1. 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].

2. The REFID

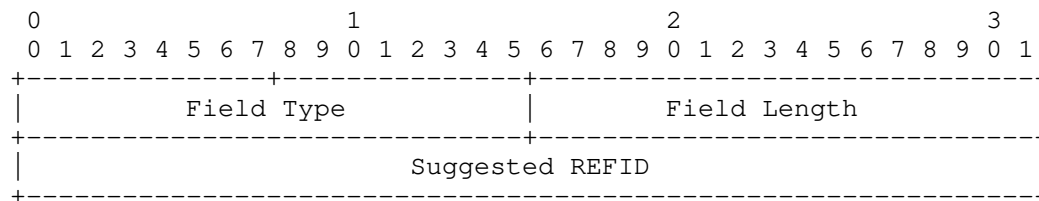
The core reason for the REFID in the NTP Protocol is to prevent a timing loop of degree 1. Put another way, if servers A and B are exchanging time with each other and server B decides to follow A as its system peer, the REFID that B will use must be able to identify server A. The interpretation of a REFID is based on the stratum, as documented in RFC 5905 [RFC5905], section 7.3, "Packet Header Variables". At Stratum 2+, which will be the case if servers A and B are exchanging packets over IPv4, if server B follows A, then B will have A's IPv4 address as its REFID. When A asks B for its time, A will see that B is synchronized to A because B will tell A that its REFID is A's IPv4 address, so when A sees its IP address as B's REFID, A knows that if it were to follow B for its time then there would be a timing loop. In this case, A will not select B as a potential source of time.

Another related use case for the REFID centers around the increasing use of leap-smearing time servers when the insertion (or any eventual deletion) of a leap second occurs. It is critical that operators and client systems be able to identify when a server is offering leap-smear time. Furthermore, with the current practice of smearing the insertion of a leap second starting at noon UTC on the day of the leap event and completing the smear at noon UTC on the day after the leap event, a server that is operating during a leap smear event must

be able to immediately identify if it should respond with either correct or leap-smear time.

3. The Suggested REFID Extension Field

Since there is no way in the base NTP packet for "this" instance of an NTP server to tell the "other" instance what REFID it should use if the "other" instance decides to use "this" instance as its system peer, the best available way to convey this information is via an extension field.



NTP Extension Field: REFID Suggestion

Field Type: TBD (Recommendation for IANA: 0x0006 (Suggested REFID))

Field Length: 0x0008

Suggested REFID: The 4 octets of the suggested REFID. Random nonce REFID values SHOULD be 0xFDxxxxxx, where the bottom 3 octets SHOULD be random values.

Examples: When decoded as an IPv4 address, a random nonce suggested REFID would decode as 253.0.0.0 thru 253.255.255.255.

4. Generating and Sending a Nonce as the Suggested REFID Extension Field

A system that decides to send a nonce as a Suggested REFID extension field SHOULD generate a new Suggested REFID nonce for each new association. It MAY generate a new Suggested REFID nonce for any association in any response. In addition to remembering the IP-based REFID, the sender MUST also remember its most-recent Suggested REFID nonce.

Since the core NTPv4 and earlier protocols do not contain any way to tell the recipient what to use as a REFID and RFC 5905 [RFC5905] uses the IPv4 address of the sender as the REFID if the association is effected over an IPv4 connection, this means that an attacker can simply send an NTP client request to a server knowing that server's system peer will be returned as the REFID in the response packet. At

this point, an attacker can, if that REFID is an IPv4 address, begin to launch attacks at the target forging the putative IP of the target's time source, or the attacker can start forging packets to the putative time server claiming to be from the target, in an attempt to cause the time server to limit or deny time service to the target.

Using a nonce for the REFID that is only recognized by the sending machine effectively prevents this type of attack.

If servers S1, S2, and S3 are all exchanging time with each other and are all using the Suggested REFID mechanism, there is a 3 in 16,777,216 (2^{24}) chance that two different servers in the same group will happen to choose the same nonce, and that would produce a false-positive timing loop detection. If a nonce Suggested REFID is never changed, this false-positive condition will occur for potentially a long time. This small risk can be reduced by periodically generating a new Suggested REFID.

5. Remembering a Nonce Suggested REFID Extension Field

An NTP server keeps track of the IP address it uses to talk to its peers. If an NTP server chooses to send a Suggested REFID to an associated peer, the server MUST remember this value. When checking for a timing loop, the Suggested REFID must also be included in the list of tested REFID values.

A set of NTP servers that are acting as a group of time servers SHOULD be using peer associations (NTP mode 1 and 2 packets), and SHOULD NOT be using client/server (NTP mode 3 and 4) exchanges. Nevertheless, implementors should be aware that the recommendation against using client/server associations for time groups may be ignored, and should be conscious of the choices they make and the configuration options they offer in order to accomodate (or at least document) this situation.

6. The Suggested REFID Extension Field and Leap Smear REFIDs

The Suggested REFID can play an important part when a server has a client population that receives leap-smear time.

The current preferred behavior for servers that offer leap-smear time is to offer leap-smear time in response to appropriate client (mode 3) requests. There are two competing forces at play during this time:

- Clients that want correct time should get correct time.

- Clients that want leap-smeared time should get leap-smeared time.

An additional complication is that a leap-second insertion event begins at noon UTC, when the Leap Indicator is 1, but the smear is only halfway applied at midnight UTC, when the Leap Indicator changes back to 0. There is no simple way for the client to let its server(s) know that it is using leap-smeared time.

One simple way for the client to let its server(s) know that it is using and wants leap-smeared time is for the client to use a Leap Smear REFID [DRAFT-LEAP-SMEAR-REFID] in its client (mode 3) requests during the entire leap smear period.

7. Acknowledgements

The author wishes to acknowledge the contributions of Martin Burnicki and Sam Weiler.

8. IANA Considerations

This memo requests IANA to allocate NTP Extension Field Type 0x0006 (Suggested REFID) for this proposal.

9. Security Considerations

Adopting this proposal will provide a much needed mechanism by which cooperating systems can agree on a less trackable and less identifiable nonce for the REFID. It will also provide a means to properly and better handle leap-smearing events with populations where some clients want correct time and other clients want leap-smeared time, thus enabling better time synchronization.

No reports of adverse consequences of adopting this proposal have been received.

10. References

10.1. Normative References

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

[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>>.

10.2. Informative References

[DRAFT-I-DO]
Stenn, H., "draft-stenn-ntp-i-do", 2018.

[DRAFT-LEAP-SMEAR-REFID]
Stenn, H., "draft-stenn-ntp-leap-smear-refid", 2018.

Author's Address

Harlan Stenn
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: stenn@nwttime.org