Internet Engineering Task Force
Internet-Draft
Obsoletes: 7822 (if approved)
Intended status: Standards Track
Expires: September 22, 2018

H. Stenn
D. Mills
Network Time Foundation

March 21, 2018

Network Time Protocol Version 4 (NTPv4) Extension Fields
draft-stenn-ntp-extension-fields-06

Abstract

Network Time Protocol version 4 (NTPv4) defines the optional usage of extension fields. An extension field, as defined in RFC 5905 [RFC5905] and RFC 5906 [RFC5906], resides after the end of the NTP header and supplies optional capabilities or information that is not conveyed in the standard NTP header. This document updates RFC 5905 [RFC5905] by clarifying some points regarding NTP extension fields and their usage with legacy Message Authentication Codes (MACs).

This proposal deprecates RFC 7822 [RFC7822].

Table of Contents

1. Introduction

An NTP packet consists of a set of fixed fields that may be followed by optional fields. Two types of optional fields are defined: extension fields (EFs) as defined in Section 7.5 of RFC 5905 [RFC5905], and legacy Message Authentication Codes (legacy MACs).

If a legacy MAC is used, it resides at the end of the packet. This field can be either a 4-octet crypto-NAK or data that has traditionally been 16, 20 or 24 octets long.

Additional information about the content of a MAC is specified in RFC 5906 [RFC5906], but since that RFC is Informational an implementor that was not planning to provide Autokey would likely never read that document. The result of this would be interoperability problems, at least. To address this problem, this proposal also includes copying and clarifying some of the content of RFC 5906 and putting it into RFC 5905. Because there is a reasonable expectation that RFC 5906

will be deprecated, this document does not propose changes or updates to RFC 5906.

NTP extension fields are defined in RFC 5905 [RFC5905] as a generic mechanism that allows the addition of future extensions and features without modifying the NTP header format (Section 16 of RFC 5905 [RFC5905]).

With the knowledge and experience we have gained over time, it has become clear that simplifications, clarifications, and improvements can be made to the NTP specification around EFs and MACs.

This proposal adjusts the requirements around EFs and MACs, allows EFs to be on 4-octet boundaries of any acceptable length, and provides methods to disambiguate packet parsing in the unexpected and unlikely case where an implementation would choose to send a packet that could be ambiguously parsed by the receiver.

This proposal deprecates RFC 7822 [RFC7822].

This document better specifies and clarifies extension fields as well as the requirements and parsing of a legacy MAC, with changes to address errors found after the publication of RFC 5905 [RFC5905] with respect to extension fields. Specifically, this document updates Section 7.5 of RFC 5905 [RFC5905], clarifying the relationship between extension fields and MACs, and expressly defines the behavior of a host that receives an unknown extension field.

2. Conventions Used in This Document

2.1. 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].

2.2. Terms and Abbreviations

EF - Extension Field

MAC - Message Authentication Code

NTPv4 - Network Time Protocol, Version 4 RFC 5905 [RFC5905]

3. NTP MAC - RFC 5906 Update

This document copies and updates some information in RFC 5906 [RFC5906] and puts it in to RFC 5905, as follows:

3.1. RFC5906 Section 4. - Autokey Cryptography

This section describes some of the cryptography aspects of Autokey. The third paragraph describes the use of 128- and 160-bit message digests. The enumeration of 128- and 160-bit message digests is not meant to be limiting - other message digest lengths MAY be implemented. This paragraph also describes some of the recommended semantic ranges of the key ID. This information belongs in RFC 5905. The key ID value is particularly significant because it provide additional disambiguation protection when deciding if the next data portion is either a legacy MAC or an extension field.

3.2. RFC5906 Section 10. - Autokey Protocol Messages

This section describes the extension field format, including initial flag bits, a Code field, and 8-bit Field Type, and the 16-bit Length. This proposal expands and clarifies this information and puts it into RFC 5905.

This section says "The reference implementation discards any packet with a field length of more than 1024 characters." but this is no longer true.

3.3. RFC5906 Section 11.5. - Error Recovery

This section describes the crypto-NAK, which should be described in RFC 5905.

3.4. RFC5906 Section 13. - IANA Consideration

This section lists the Autokey-related Extension Field Types, including Flag Bits, Codes, and Field Types, which should be described in RFC 5905, or perhaps in some other document.

4. NTP Extension Fields - RFC 5905 Update

This document updates Section 7.5 of RFC 5905 [RFC5905] as follows:

4.1. OLD: RFC5905 7.5 - NTP Extension Field Format

In NTPv4, one or more extension fields can be inserted after the header and before the MAC, which is always present when an extension field is present. Other than defining the field format, this

document makes no use of the field contents. An extension field contains a request or response message in the format shown in Figure 14.

```
 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          |
+-------------------------------+-------------------------------+
.                                                               .
.                             Value                             .
.                                                               .
+-------------------------------+-------------------------------+
|                       Padding (as needed)                     |
+---------------------------------------------------------------+
```

Figure 14: Extension Field Format

All extension fields are zero-padded to a word (four octets) boundary. The Field Type field is specific to the defined function and is not elaborated here. While the minimum field length containing required fields is four words (16 octets), a maximum field length remains to be established.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including the Padding field.

4.2. NEW: RFC5905 Section 7.5 - NTP Extension Field Format

In NTPv4, one or more extension fields can be inserted after the header and before the possibly optional legacy MAC. A MAC SHOULD be present when an extension field is present. A MAC is always present in some form when NTP packets are authenticated. This MAC SHOULD be either a legacy MAC or a MAC-EF. It MAY be both. Other than defining the field format, this document makes no use of the field contents. An extension field contains a request or response message in the format shown in Figure 14.

```
 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         |
+-------------------------------+-------------------------------+
.                                                               .
.                             Value                             .
.                                                               .
+-------------------------------+-------------------------------+
|                       Padding (as needed)                     |
+---------------------------------------------------------------+
```

Figure 14: Extension Field Format

All extension fields are zero-padded to a word (four octet) boundary. The Field Type is specific to the defined function and detailed information about the Field Type is not elaborated here. The minimum size of an Extension Field is a 32-bit word (4 octets), and while the maximum extension field size MUST be 65532 octets or less, an NTP packet SHOULD NOT exceed the network MTU.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including any Padding octets. The bottom two bits of the Field Length SHOULD be zero, and the size of the extension field SHOULD end on a 32-bit (4 octet) boundary. [RFC5905 Section 7.5 says "All extension fields are zero-padded to a word (four octets) boundary." but does not use 'MUST' language. Is it overkill to reiterate this requirement here? Should we use SHOULD or MUST regarding the bottom two bits or the boundary of the EF? It is possible, down the road, that we might find some use for those bottom 2 bits, even if we require a 32-bit boundary on the last octet of an EF.]

The Field Type contains the following sub-elements:

```
 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
+---------------+---------------+-------------------------------+
|R|E|      Code |      Type     |        (Field Length)         |
+-------------------------------+-------------------------------+
```

Field Type Format

Where the following Field Type flags are defined:

R: 0 for a "Query", 1 for a "Response"

E: 0 for "OK", 1 for an "Error". Unused, and will be deprecated.

[The 'R' flag is currently used by Autokey, and by the proposed I-DO extension field. This flag is used after the packet is accepted.]

[The 'E' flag was proposed for use by Autokey, after the packet was accepted. As it was never used and no other use-cases have been identified, we are recommending this flag be deprecated at some point in the future.]

[The EF Code subtype is currently used by RFC 5906, Autokey [RFC5906]. The EF Code subtype is used by the proposed Extended Information EF proposal, and is expected to be used by the NTS Extension Field, at least.]

The Autokey EF currently uses the most Code values - 10 of them, which equates to the least-significant 4 bits of the high-order octet. It is possible that additional flag bits will be allocated; in the past, the high-order 2 bits were reserved, and for a time two additional bits were proposed. Make no assumptions about the unused bits in this octet.

The Field Type, Value, and Padding fields are specific to the defined function and are not elaborated here; appropriate Field Type flags, the EF Code, and EF Type values are defined in an IANA registry, and the Length, Value, and Padding values are defined by the document referred to by the registry. If a host receives an extension field with an unknown Field Type, the host SHOULD ignore the extension field and MAY drop the packet altogether, depending on local policy.

The Length field is a 16-bit unsigned integer that indicates the length of the entire extension field in octets, including any Padding.

While the minimum field length of an EF that contains no value fields is one word (four octets), and the minimum field length of an EF that contains required fields is two words (8 octets), the maximum field length MUST NOT be longer than 65532 octets due to the maximum size of the data represented by the Length field, and SHOULD be small enough that the size of the NTP packet received by the client does not exceed the smallest MTU between the sender and the recipient. The bottom two bits of the Field Length SHOULD be zero and the EF data SHOULD be aligned to a 32-bit (4 octet) boundary.

4.3. NEW: RFC5905 Section 7.5.1 - Extension Fields and MACs

With the inclusion of additional Extension Fields, there is now a potential that a poorly-designed implementation would produce an ambiguous parsing in the presence of a legacy MAC. If an implementation offers even a modicum of care, there will be no

ambiguity when parsing an NTP packet that contains a legacy MAC from an existing implementation.

The first protection from this ambiguity comes from the fact that current conforming implementations only support the Autokey EF, which uses EF Type 2 and a legacy MAC. While the Experimental UDP Checksum Complement specified by RFC 7821 [RFC7821] uses EF Type 5, it specifically prohibits the use of a MAC, and the 0x2000 bit in its assigned EF specification of 0x2005 originally signified that a MAC is optional when this EF is seen. While the 0x2000 bit is no longer proposed as a means to flag that the MAC is optional, any usage of this EF with a Code field of either 0x2005 or 0x0005 can be trivially recognized as an Experimental UDP Checksum Complement EF, which does not, indeed, MUST NOT be followed by a MAC.

[As a side note, the requirement in RFC 7821 [RFC7821] that the UDP Checksum Complement EF must have a 28 octet length is demonstrably not needed if this proposal is accepted. It only needs 8 octets: 4 octets of EF header, 2 octets of must-be-zero padding, and 2 octets of Checksum Complement.]

If an implementation uses the LAST-EF extension field, the presence of this field means "I am the last EF in this NTP Packet. Any subsequent packet data MUST be a legacy MAC." In this case, there is no parsing ambiguity.

If a system sends its MAC as a MAC-EF and does not send a legacy MAC, there is no parsing ambiguity.

The only time there is a potential for a parsing ambiguity is when a legacy MAC is provided and neither of the previous two cases are present. Even in this case, there is minimal risk.

An implementation MAY choose to add padding to any EF, or at least any EF that comes before a legacy MAC, to avoid an EF that is 16, 20, or 24 octets in length. Doing this would generally avoid any risk of mis-parsing. But this should not be needed for the following reasons.

An Extension Field contains a 2-octet Field Type, a 2-octet Field Length, and any payload (data and/or padding). If the NTP Packet parsing is at a point where it is evaluating data after the base packet, one of the following situations exists:

> If the Field Length is not an even multiple of 4, we are not looking at an extension field. In this case, the only possibility of having a valid packet is if the data is part of a legacy MAC.

If the Field Length is valid, i.e., an even multiple of 4 octets, one of the following three cases must be present:

First, the Field Length will be less than the remaining data. This means subsequent data must parse as some number of Extension Fields, optionally followed by a legacy MAC.

Second, the Field Length will exactly match the remaining data.

The third case is where the Field Length is longer than the remaining packet data. In this case, the current parse cannot be a valid extension field, and if the packet is valid, the data must be a legacy MAC.

Semantic checking may also be done to validate a potential legacy MAC. A legacy MAC is a four-octet Key Identifier followed by a message digest. The usual message digest is 16 octets long but may be another size, depending on the digest algorithm. In the Reference Implementation and in implementations that follow the guidelines for the values of the Key Identifier, a Key Identifier between 1 and 65535, inclusive, is a symmetric key, while a Key Identifier that is > 65535 is an Autokey RFC 5906 [RFC5906], or similar. If the receiving system does not recognize the Key Identifier, the data CANNOT be a valid legacy MAC. If the receiving system recognizes the Key Identifier, then it also has knowledge of the digest algorithm and can make sure the digest payload is the proper length. If this is not the case, then the data CANNOT be a valid legacy MAC. In this case, it MIGHT be a valid extension field.

It is trivial to parse the data after the base NTP packet and come up with a list of potential parsings. A local policy choice can specify the precedence of the parsing options in this case.

If none of the parsings validate, the packet fails authentication. An implementation has three local policy choices available if LAST-EF is not used and a legacy MAC may be provided. First, the implementation may specify EF-precedence. Second, the implementation may specify legacy-MAC-precedence. Finally, the implementation may specify "best fit" precedence. In this last case, the packet will meet one of the three following criteria: First, none of the parsings will match. Again, this is a case of failed authentication. Second, exactly one parsing will match and that parsing will be accepted. Third, multiple parsings will match, in which case the implementation may choose its behavior.

Additionally, most EFs will require a MAC. If there is a syntactically-valid parsing that does not include a MAC but previously scanned EFs require a MAC, then in a multiple-choice

parsing scenario where one of the choices does not include a MAC the "no MAC provided" choice SHOULD be eliminated.

Note well that this rare situation can be completely avoided by using LAST-EF, or by indicating that no legacy MAC will be used.

Finally, in many cases at least one side will know if a MAC is required or not. Client configurations of all types, unicast, broadcast, multicast, and manycast, that state that a key will be used to communicate with a server SHOULD reject packets claiming to be from the server that do not include a MAC. Symmetric associations also are configured with similar knowledge and requirements.

4.3.1. Legacy MAC/EF Parsing Pseudocode

Here are two potential pseudocode implementations showing how data after the base NTP packet could be analyzed to identify EFs and a possible legacy MAC.

Example 1: Generate a list of possible parsings:

```
struct pkt_parse {
 foo * ef_ptr;
 foo * legacy_mac;
 struct pkt_parse * next;
};

struct pkt_parse pkt_parse_chain = NULL;

EOPacket = address of last data in packet;
here = address of the EOBasePacket;
more_efs = 1;
while (1) {
    int candidate = 0;
    int ef_len = 0;

    if (EOPacket > here) {
        p = emalloc(pkt_parse);                 // *p is zeroed
        if (this could be a legacy MAC) {       // we know the keyid
            p->legacy_mac = here;
            candidate = 1;
        }
        if (more_efs && this could be an EF) {  // Length field valid
            p->ef_ptr = here;
            ef_len = (the length of the EF);
            here += ef_len;
            if (this is a LAST_EF) {
                more_efs = 0;
            }
            candidate = 1;
        } else {
            more_efs = 0;
        }
    }

    if (candidate) {
        p->next = pkt_parse_chain;
        pkt_parse_chain = p;
    } else {
        free(p);
        break;
    }
}
```

Example 1: Generate a list of possible parsings

and at this point we can scan thru the items in pkt_parse_chain to do deeper checks, throwing away the parsings that don't make sense.

This opens up more questions if we get multiple parsings and at least 1 of them is "valid". It's also perfectly reasonable to decide to produce a single parse based on precedence rules: Prefer legacy MAC, or prefer EF.

Example 2: Another possible way to handle EF/legacy-MAC parsing:

```
// We're at the end of the base NTP packet.
// A legacy MAC is allowed:
// - immediately after the base packet
// - immediately after one or more Autokey EFs (a non-issue, below)
// - immediately after a LAST-EF

ef_ok = 1;                             // An EF is allowed here
legacy_mac_ok = 1;                     // Legacy MAC allowed here
saw_mac = 0;                           // We haven't seen a MAC yet
authlen = LEN_PKT_NOMAC;               // Length of a base packet
leg_mac = rbufp->recv_length - authlen; // # bytes after base

while (leg_mac > 0) {                  // Data after base packet
        if (leg_mac % 4 != 0 || leg_mac < MIN_MAC_LEN) {
                return: Bad packet length;
        }

        // If ef_ok, this could be an EF or legacy MAC
        skeyid = ntohl(pkt[authlen / 4]);
        opcode = skeyid >> 16;
        len = skeyid & 0xffff;

        if (ef_ok && GET_EXT_FIELD_TYPE(opcode) == EF_FT_LAST) {
                if (leg_mac > MAX_MAC_LEN) {
                        return: Too much data after LAST_EF;
                }
                // Anything here MUST be a legacy MAC
                ef_ok = 0;
                legacy_mac_ok = 1;
        } else {
                if (4 == leg_mac && 0 == skeyid) {
                        break;   // Likely crypto-NAK
                }

                if (legacy_mac_ok && leg_mac <= MAX_MAC_LEN) {
                        int ksize;

                        // If we find a keyid, we know its alg/length
                        ksize = auth_findkeysize(skeyid);
                        if (   ksize != -1
                            && ksize == leg_mac
```

```
                                && (it validates)) {
                                    saw_mac = 1;
                                    break;
                            }
                            // If we got here, it can't be a valid
                            // legacy MAC.  It's still a potential EF.
                    }

                    if (!ef_ok) {
                            break;
                    }

                    // At this point, this SHOULD be an EF

                    if (   len % 4 != 0
                        || len < 4
                        || len + authlen > rbufp-> recv_length) {
                            return: Bad length;
                    }

                    switch (GET_EXT_FIELD_TYPE(opcode)) {
                    case EF_FT_AK:          // Autokey
                            // extract calling group name for later
                            break;
                    case EF_FT_LAST:        // LAST-EF
                            legacy_mac_ok = 1;
                            break;
                    default:
                            legacy_mac_ok = 0;
                            break;
                    }
            }

            authlen += len;
            leg_mac -= len;
    }

    if (leg_mac < 0) {
            return: Malformed packet
    }
```

Example 2: Another way to handle EF/legacy-MAC parsing

4.4. OLD: RFC5905 Section 9.2. - Peer Process Operations

...

FXMIT. ... This message includes the normal NTP header data shown in Figure 8, but with a MAC consisting of four octets of zeros. ...

4.5. NEW: RFC5905 Section 9.2. - Peer Process Operations

...

FXMIT. ... This message includes the normal NTP header data shown in Figure 8, but with a MAC consisting of four octets of zeros. This MAC can be a legacy MAC or a MAC-EF. If it's a MAC-EF, the crypto-NAK MUST be the only MAC in the MAC-EF payload. ...

5. Acknowledgements

The authors wish to acknowledge the contributions of Sam Weiler, Danny Mayer, and Tal Mizrahi.

6. IANA Considerations

This memo requests IANA to allocate the following bits in the NTP Extension Field Types table:

0x8000: R: Response (0: Request, 1: Response)

0x4000: E: Error (0: OK, 1: Error) - Unused, deprecation expected

The following table should be the functionally the same as the existing NTP Extension Field Table.

Field Type	Meaning
0x0000	crypto-NAK (with Field Length of 0)
0x0000	RESERVED: Permanently Unassigned
0x0001	RESERVED: Unassigned
0x0002	Autokey: No-Operation Request
0x8002	Autokey: No-Operation Response
0x0102	Autokey: Association Message Request
0x8102	Autokey: Association Message Response
0x0202	Autokey: Certificate Message Request
0x8202	Autokey: Certificate Message Response
0x0302	Autokey: Cookie Message Request
0x8302	Autokey: Cookie Message Response
0x0402	Autokey: Autokey Message Request
0x8402	Autokey: Autokey Message Response
0x0502	Autokey: Leapseconds Value Message Request
0x8502	Autokey: Leapseconds Value Message Response
0x0602	Autokey: Sign Message Request
0x8602	Autokey: Sign Message Response
0x0702	Autokey: IFF Identity Message Request
0x8702	Autokey: IFF Identity Message Response
0x0802	Autokey: GQ Identity Message Request
0x8802	Autokey: GQ Identity Message Response
0x0902	Autokey: MV Identity Message Request
0x8902	Autokey: MV Identity Message Response
0x0005	Checksum Complement
0x1005	Checksum Complement

Current Extension Fields

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, <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>.

[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>.

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

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

Authors' Addresses

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

Email: stenn@nwtime.org

David L. Mills
Network Time Foundation
P.O. Box 918
Talent, OR 97540
US

Email: mills@udel.edu