Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

T. Aura
Aalto University
M. Sethi
Ericsson

March 11, 2019

Nimble out-of-band authentication for EAP (EAP-NOOB)

draft-aura-eap-noob-05

Abstract

Extensible Authentication Protocol (EAP) provides support for multiple authentication methods. This document defines the EAP-NOOB authentication method for nimble out-of-band (OOB) authentication and key derivation. This EAP method is intended for bootstrapping all kinds of Internet-of-Things (IoT) devices that have a minimal user interface and no pre-configured authentication credentials. The method makes use of a user-assisted one-directional OOB channel between the peer device and authentication server.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
2. Terminology 4
3. EAP-NOOB protocol 5
  3.1. Protocol overview 5
  3.2. Protocol messages and sequences 8
    3.2.1. Initial Exchange 8
    3.2.2. OOB Step 10
    3.2.3. Completion Exchange 11
    3.2.4. Waiting Exchange 13
  3.3. Protocol data fields 15
    3.3.1. Peer identifier, realm and NAI 15
    3.3.2. Message data fields 17
  3.4. Fast reconnect and rekeying 22
    3.4.1. Persistent EAP-NOOB association 22
    3.4.2. Reconnect Exchange 23
    3.4.3. User reset 26
  3.5. Key derivation 27
  3.6. Error handling 30
    3.6.1. Invalid messages 32
    3.6.2. Unwanted peer 32
    3.6.3. State mismatch 32
    3.6.4. Negotiation failure 32
    3.6.5. Cryptographic verification failure 33
    3.6.6. Application-specific failure 33
4. IANA Considerations 34
  4.1. Cryptosuites 34
  4.2. Message Types 34
  4.3. Error codes 35
  4.4. Domain name reservation considerations 36
5. Implementation Status 37
  5.1. Implementation with wpa_supplicant and hostapd 37
  5.2. Protocol modeling 38
6. Security considerations 38
  6.1. Authentication principle 38
  6.2. Identifying correct endpoints 40
  6.3. Trusted path issues and misbinding attacks 40
  6.4. Peer identifiers and attributes 41
  6.5. Identity protection 42
  6.6. Downgrading threats 43
  6.7. Recovery from loss of last message 44
  6.8. EAP security claims 45

7. References 47
7.1. Normative references 47
7.2. Informative references 48
Appendix A. Exchanges and events per state 50
Appendix B. Application-specific parameters 51
Appendix C. ServerInfo and PeerInfo contents 52
Appendix D. EAP-NOOB roaming 54
Appendix E. OOB message as URL 55
Appendix F. Example messages 56
Appendix G. TODO list 58
Appendix H. Version history 58
Appendix I. Acknowledgments 60
Authors' Addresses 60

1. Introduction

This document describes a method for registration, authentication and key derivation for network-connected ubiquitous computing devices, such as consumer and enterprise appliances that are part of the Internet of Things (IoT). These devices may be off-the-shelf hardware that is sold and distributed without any prior registration or credential-provisioning process. Thus, the device registration in a server database, ownership of the device, and the authentication credentials for both network access and application-level security must all be established at the time of the device deployment. Furthermore, many such devices have only limited user interfaces that could be used for their configuration. Often, the interfaces are limited to either only input (e.g. camera) or output (e.g. display screen). The device configuration is made more challenging by the fact that the devices may exist in large numbers and may have to be deployed or re-configured nimbly based on user needs.

More specifically, the devices may have the following characteristics:

- no pre-established relation with a specific server or user,
- no pre-provisioned device identifier or authentication credentials,
- limited user interface and configuration capabilities.

Many proprietary OOB configuration methods exits for specific IoT devices. The goal of this specification is to provide an open standard and a generic protocol for bootstrapping the security of network-connected appliances, such as displays, printers, speakers, and cameras. The security bootstrapping in this specification makes use of a user-assisted out-of-band (OOB) channel. The device authentication relies on user having physical access to the device, and the of the key exchange security is based on the assumption that attackers are not able to observe or modify the messages conveyed through the OOB channel. We follow the common approach taken in pairing protocols: performing a Diffie-Hellman key exchange over the insecure network and authenticating the established key with the help of the OOB channel in order to prevent impersonation and man-in-the-middle (MitM) attacks.

The solution presented here is intended for devices that have either an input or output interface, such as a camera, microphone, display screen, speakers or blinking LED light, which is able to send or receive dynamically generated messages of tens of bytes in length. Naturally, this solution may not be appropriate for very small sensors or actuators that have no user interface at all or for devices that are inaccessible to the user. We also assume that the OOB channel is at least partly automated (e.g. camera scanning a bar code) and, thus, there is no need to absolutely minimize the length of the data transferred through the OOB channel. This differs, for example, from Bluetooth simple pairing [BluetoothPairing], where it is critical to minimize the length of the manually transferred or compared codes. Since the OOB messages are dynamically generated, we do not support static printed registration codes. This also prevents attacks where a static secret code would be leaked.

2. 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 [RFC2119].

In addition, this document frequently uses the following terms as they have been defined in [RFC5216]:

authenticator The entity initiating EAP authentication.

peer The entity that responds to the authenticator. In [IEEE-802.1X], this entity is known as the supplicant.

server The entity that terminates the EAP authentication method with the peer. In the case where no backend authentication server is used, the EAP server is part of the authenticator. In the case where the authenticator operates in pass-through mode, the EAP server is located on the backend authentication server.

3. EAP-NOOB protocol

This section defines the EAP-NOOB protocol. The protocol is a generalized version of the original idea presented by Sethi et al. [Sethi14].

3.1. Protocol overview

One EAP-NOOB protocol execution spans multiple EAP conversations, called Exchanges. This is necessary to leave time for the OOB message to be delivered, as will be explained below.

The overall protocol starts with the Initial Exchange, in which the server allocates an identifier to the peer, and the server and peer negotiate the protocol version and cryptosuite (i.e. cryptographic algorithm suite), exchange nonces and perform an Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) key exchange. The user-assisted OOB Step then takes place. This step requires only one out-of-band message either from the peer to the server or from the server to the peer. While waiting for the OOB Step action, the peer MAY probe the server by reconnecting to it with EAP-NOOB. If the OOB Step has already taken place, the probe leads to the Completion Exchange, which completes the mutual authentication and key confirmation. On the other hand, if the OOB Step has not yet taken place, the probe leads to the Waiting Exchange, and the peer will perform another probe after a server-defined minimum waiting time. The Initial Exchange and Waiting Exchange always end in EAP-Failure, while the Completion Exchange may result in EAP-Success. Once the peer and server have performed a successful Completion Exchange, both endpoints store the created association in persistent storage, and the OOB Step is not repeated. Thereafter, creation of new temporal keys, ECDHE rekeying, and updates of cryptographic algorithms can be achieved with the Reconnect Exchange.

```
                                       OOB Output/Initial Exchange/
                                                  Waiting Exchange
                                                    .-----.
                                                    |     v
        .------------------.   Initial       .------------------.
        |                  |   Exchange      |                  |
     .->| 0. Unregistered  |---------------->|1. Waiting for OOB|
     |  |                  |                 |                  |
     |  '------------------'                 '------------------'
     |                                         |      |      ^
    User Reset                 Completion      |      |      |
     |                         Exchange        |     OOB   OOB
     |<-------.      .-------------------------'    Input  Reject/
     |        |      |                                |    Initial
     |        |      |                                |    Exchange
     |        |      v                                v      |
     |  .------------------.   Completion    .------------------.
     |  |                  |   Exchange      |                  |
     |  |  4. Registered   |<----------------| 2. OOB Received  |
     |  |                  |                 |                  |
     |  '------------------'                 '------------------'
     |        |      ^
     |  Mobility/    |
     |  Timeout/   Reconnect
     |  Failure    Exchange
     |        |      |
     |        v      |
     |  .-----------------.
     |  |                 |
     '--| 3. Reconnecting |
        |                 |
        '-----------------'
```

Figure 1: EAP-NOOB server-peer association state machine

Figure 1 shows the association state machine, which is the same for the server and for the peer. (For readability, only the main state transitions are shown. The complete table of transitions can be found in Appendix A.) When the peer initiates the EAP-NOOB method, the server chooses the ensuing message exchange based on the combination of the server and peer states. The EAP server and peer are initially in the Unregistered state, in which no state information needs to be stored. Before a successful Completion Exchange, the server-peer association state is ephemeral in both the server and peer (ephemeral states 0..2), and either endpoint may cause the protocol to fall back to the Initial Exchange. After the Completion Exchange has resulted in EAP-Success, the association state becomes persistent (persistent states 3..4). Only user reset or memory failure can cause the return of the server or the peer from the persistent states to the ephemeral states and to the Initial Exchange.

The server MUST NOT repeat a successful OOB Step with the same peer except if the association with the peer is explicitly reset by the user or lost due to failure of the persistent storage in the server. More specifically, once the association has entered the Registered state, the server MUST NOT delete the association or go back to states 0..2 without explicit user approval. Similarly, the peer MUST NOT repeat the OOB Step unless the user explicitly deletes from the peer the association with the server or resets the peer to the Unregistered state. The server and peer MAY implement user reset of the association by deleting the state data from that endpoint. If an endpoint continues to store data about the association after the user reset, its behavior SHOULD be equivalent to having deleted the association data.

It can happen that the peer accidentally or through user reset loses its persistent state and reconnects to the server without a previously allocated peer identifier. In that case, the server MUST treat the peer as a new peer. The server MAY use auxiliary information, such as the PeerInfo field received in the Initial Exchange, to detect multiple associations with the same peer. However, it MUST NOT delete or merge redundant associations without user or application approval because EAP-NOOB internally has no secure way of verifying that the two peers are the same physical device. Similarly, the server might lose the association state because of a memory failure or user reset. In that case, the only way to recover is that the user resets also the peer.

A special feature of the EAP-NOOB method is that the server is not assumed to have any a-priori knowledge of the peer. Therefore, the peer initially uses the generic identity string "noob@eap-noob.net" as its network access identifier (NAI). The server then allocates a server-specific identifier to the peer. The generic NAI serves two purposes: firstly, it tells the server that the peer supports and expects the EAP-NOOB method and, secondly, it allows routing of the EAP-NOOB sessions to a specific authentication server in the AAA architecture.

EAP-NOOB is an unusual EAP method in that the peer has to have multiple EAP conversations with the server before it can receive EAP-Success. The reason is that, while EAP allows delays between the request-response pairs, e.g. for repeated password entry, the user delays in OOB authentication can be much longer than in password trials. In particular, EAP-NOOB supports also peers with no input capability in the user interface. Since user cannot initiate the protocol in these devices, they have to perform the Initial Exchange opportunistically and hope for the OOB Step to take place within a timeout period (NoobTimeout), which is why the timeout needs to be several minutes rather than seconds. For example, consider a printer (peer) that outputs the OOB message on paper, which is then scanned for the server. To support such high-latency OOB channels, the peer and server perform the Initial Exchange in one EAP conversation, then allow time for the OOB message to be delivered, and later perform the Waiting and Completion Exchanges in different EAP conversations.

3.2. Protocol messages and sequences

This section defines the EAP-NOOB exchanges, which correspond to EAP conversations. The protocol messages in each exchange and the data members in each message are listed in the diagrams below.

Each EAP-NOOB exchange begins with the authenticator sending an EAP-Request/Identity packet to the peer. From this point on, the EAP conversation occurs between the server and the peer, and the authenticator acts as a pass-through device. The peer responds to the authenticator with an EAP-Response/Identity packet, containing the network access identifier (NAI), which conveys both the peer identity and the current peer state as defined in Section 3.3.1.

After receiving the NAI, the server chooses the EAP-NOOB exchange, i.e. the ensuing message sequence, based on the combination of the peer and server states. The peer recognizes the exchange based on the message type field (Type) of the EAP-NOOB request received from the server. The available exchanges are defined in the following subsections. Each exchange comprises one or more EAP requests-response pairs and ends in either EAP-Failure, indicating that authentication is not (yet) successful, or in EAP-Success.

3.2.1. Initial Exchange

Upon receiving the EAP-Response/Identity from the peer, if either the peer or the server is in the Unregistered (0) state and the other is in one of the ephemeral states (0..2), the server chooses the Initial Exchange.

The Initial Exchange comprises two EAP-NOOB request-response pairs, one for version, cryptosuite and parameter negotiation and the other for the ECDHE key exchange. The first EAP-NOOB request (Type=1) from the server contains a newly allocated PeerId for the peer and an optional Realm. The server allocates a new PeerId in the Initial Exchange regardless of any old PeerId in the username part of the received NAI. The server also sends in the request a list of the protocol versions (Vers) and cryptosuites (Cryptosuites) it supports, an indicator of the OOB channel directions it supports (Dirs), and a ServerInfo object. The peer chooses one of the versions and cryptosuites. The peer sends a response (Type=1) with the selected protocol version (Verp), the received PeerId, the selected cryptosuite (Cryptosuitep), an indicator of the OOB channel directions selected by the peer (Dirp), and a PeerInfo object. In the second EAP-NOOB request and response (Type=2), the server and peer exchange the public components of their ECDHE keys and nonces (PKs,Ns,PKp,Np). The ECDHE keys MUST be based on the negotiated cryptosuite i.e. Cryptosuitep. The Initial Exchange ends always with EAP-Failure from the server because the authentication cannot yet be completed.

```
     EAP Peer                                          EAP Server
        |                                                   |
        |<----------- EAP-Request/Identity -|               |
        |                                                   |
        |                                                   |
        |------------ EAP-Response/Identity -------------->|
        |      (NAI=noob|PeerId+sX@eap-noob.net)            |
        |                                                   |
        |                                                   |
        |<----------- EAP-Request/EAP-NOOB ----------------|
        |      (Type=1,Vers,PeerId,[Realm],                 |
        |       Cryptosuites,Dirs,ServerInfo)               |
        |                                                   |
        |                                                   |
        |------------ EAP-Response/EAP-NOOB -------------->|
        |      (Type=1,Verp,PeerId,Cryptosuitep,            |
        |        Dirp,PeerInfo)                             |
        |                                                   |
        |                                                   |
        |<----------- EAP-Request/EAP-NOOB ----------------|
        |      (Type=2,PeerId,PKs,Ns,[SleepTime])           |
        |                                                   |
        |                                                   |
        |------------ EAP-Response/EAP-NOOB -------------->|
        |      (Type=2,PeerId,PKp,Np)                       |
        |                                                   |
        |                                                   |
        |<----------- EAP-Failure -------------------------|
        |                                                   |
```

Figure 2: Initial Exchange

At the conclusion of the Initial Exchange, both the server and the peer move to the Waiting for OOB (1) state.

3.2.2. OOB Step

The OOB Step, labeled as OOB Output and OOB Input in Figure 1, takes place after the Initial Exchange. Depending on the direction negotiated, the peer or the server outputs the OOB message shown in Figure 3 or Figure 4, respectively. The data fields are the PeerId, the secret nonce Noob, and the cryptographic fingerprint Hoob. The contents of the data fields are defined in Section 3.3.2. The OOB message is delivered to the other endpoint via a user-assisted OOB channel.

```
     EAP Peer                                        EAP Server
        |                                                 |
        |=================OOB============================>|
        |             (PeerId,Noob,Hoob)                  |
        |                                                 |
```

Figure 3: OOB Step, from peer to EAP server

```
     EAP Peer                                        EAP Server
        |                                                 |
        |<=================OOB============================|
        |             (PeerId,Noob,Hoob)                  |
        |                                                 |
```

Figure 4: OOB Step, from EAP server to peer

The receiver of the OOB message MUST compare the received value of the fingerprint Hoob with a value that it computes locally. If the values are equal, the receiver moves to the OOB Received (2) state. Otherwise, the receiver MUST reject the OOB message. For usability reasons, the receiver SHOULD indicate the acceptance or rejection of the OOB message to the user. The receiver SHOULD reject invalid OOB messages without changing its state, until an application-specific number of invalid messages (OobRetries) has been reached, after which the receiver SHOULD consider it an error and go back to the Unregistered (0) state.

The server or peer MAY send multiple OOB messages with different Noob values while in the Waiting for OOB (1) state. The OOB sender SHOULD remember the Noob values until they expire and accept any one of them in the following Completion Exchange. The Noob values sent by the server expire after an application-dependent timeout (NoobTimeout), and the server MUST NOT accept Noob values older than that in the Completion Exchange. The RECOMMENDED value for NoobTimeout is 3600 seconds if there are no application-specific reasons for making it shorter or longer. The Noob values sent by the peer expire as defined in Section 3.2.4.

The OOB receiver does not accept further OOB messages after it has accepted one and moved to the OOB Received (2) state. However, the receiver MAY buffer redundant OOB messages in case OOB message expiry or similar error detected in the Completion Exchange causes it to return to the Waiting for OOB (1) state. It is RECOMMENED that the OOB receiver notifies the user about redundant OOB messages, but it MAY also discard them silently.

The sender will typically generate a new Noob, and therefore a new OOB message, at constant time intervals (NoobInterval). The RECOMMENDED interval is NoobInterval = NoobTimeout / 2, so that the two latest values are always accepted. However, the timing of the Noob generation may also be based on user interaction or on implementation considerations.

Even though not recommended (see Section 3.3), this specification allows both directions to be negotiated (Dirp=3) for the OOB channel. In that case, both sides SHOULD output the OOB message, and it is up to the user to deliver one of them.

The details of the OOB channel implementation including the message encoding are defined by the application. Appendix E gives an example of how the OOB message can be encoded as a URL that may be embedded in a QR code and NFC tag.

3.2.3. Completion Exchange

After the Initial Exchange, if both the server and the peer support the peer-to-server direction for the OOB channel, the peer SHOULD initiate the EAP-NOOB method again after an applications-specific waiting time in order to probe for completion of the OOB Step. Also, if both sides support the server-to-peer direction of the OOB exchange and the peer receives the OOB message, it SHOULD initiate the EAP-NOOB method immediately. Once the server receives the EAP-Response/Identity, if one of the server and peer is in the OOB Received (2) state and the other is either in the Waiting for OOB (1) or OOB Received (2) state, the OOB Step has taken place and the server SHOULD continue with the Completion Exchange.

The Completion Exchange comprises one or two EAP-NOOB request-response pairs. If the peer is in the Waiting for OOB (1) state, the OOB message has been sent in the peer-to-server direction. In that case, only one request-response pair (Type=4) takes place. In the request, the server sends the NoobId value, which the peer uses to identify the exact OOB message received by the server. On the other hand, if the peer is in the OOB Received (2) state, the direction of the OOB message is from server to peer. In that case, two request-response pairs (Type=8 and Type=4) are needed. The purpose of the first request-response pair (Type=8) is that it enables the server to discover NoobId, which identifies the exact OOB message received by the peer. The server returns the same NoobId to the peer in the latter request.

In the last and sometimes only request-response pair (Type=4) of the Completion Exchange, the server and peer exchange message authentication codes. Both sides MUST compute the keys Kms and Kmp as defined in Section 3.5 and the message authentication codes MACs and MACp as defined in Section 3.3.2. Both sides MUST compare the received message authentication code with a locally computed value. If the peer finds that it has received the correct value of MACs and the server finds that it has received the correct value of MACp, the Completion Exchange ends in EAP-Success. Otherwise, the endpoint where the comparison fails indicates this with an error message (error code 4001, see Section 3.6.1) and the Completion Exchange ends in EAP-Failure.

After successful Completion Exchange, both the server and the peer move to the Registered (4) state. They also derive the output keying material and store the persistent EAP-NOOB association state as defined in Section 3.4 and Section 3.5.

It is possible that the OOB message expires before it is received. In that case, the sender of the OOB message no longer recognizes the NoobId that it receives in the Completion Exchange. Another reason why the OOB sender might not recognize the NoobId is if the received OOB message was spoofed and contained an attacker-generated Noob value. The recipient of an unrecognized NoobId indicates this with an error message (error code 2003, see Section 3.6.1) and the Completion Exchange ends in EAP-Failure. The recipient of the error message 2003 moves back to the Waiting for OOB (1) state. This state transition is shown as OOB Reject in Figure 1 (even though it really is a specific type of failed Completion Exchange). The sender of the error message, on the other hand, stays in its previous state.

Although it is not expected to occur in practice, poor user interface design could lead to two OOB messages delivered simultaneously, one from the peer to the server and the other from the server to the peer. The server detects this event in the beginning of the Completion Exchange by observing that both the server and peer are in the OOB Received state (2). In that case, as a tiebreaker, the server MUST behave as if only the server-to-peer message had been delivered.

```
EAP Peer                                               EAP Server
   |                                                          |
   |<----------- EAP-Request/Identity -|                      |
   |                                                          |
   |                                                          |
   |------------ EAP-Response/Identity -------------->|
   |       (NAI=PeerId+sX@eap-noob.net)                       |
   |                                                          |
   |                                                          |
   |<----------- [ EAP-Request/EAP-NOOB ] ------------|
   |       (Type=8,PeerId)                                    |
   |                                                          |
   |                                                          |
   |------------ [ EAP-Response/EAP-NOOB ] ---------->|
   |       (Type=8,PeerId,NoobId)                             |
   |                                                          |
   |                                                          |
   |<----------- EAP-Request/EAP-NOOB ----------------|
   |       (Type=4,PeerId,NoobId,MACs)                        |
   |                                                          |
   |                                                          |
   |------------ EAP-Response/EAP-NOOB -------------->|
   |       (Type=4,PeerId,MACp)                               |
   |                                                          |
   |                                                          |
   |<----------- EAP-Success -------------------------|
   |                                                          |
```

Figure 5: Completion Exchange

3.2.4. Waiting Exchange

As explained in Section 3.2.3, the peer SHOULD probe the server for completion of the OOB Step. Once the server receives the EAP-Response/Identity, if both the server and peer states are in the Waiting for OOB (1) state, the server will continue with the Waiting Exchange (Type=3). The main purpose of this exchange is to inform the peer that the server has not yet received a peer-to-server OOB message.

In order to limit the rate at which peers probe the server, the server MAY send to the peer either in the Initial Exchange or in the Waiting Exchange a minimum time to wait before probing the server again. A peer that has not received an OOB message MUST wait at least the server-specified minimum waiting time in seconds (SleepTime) before initiating EAP again with the same server. The peer uses the latest SleepTime value that it has received in or after the Initial Exchange. If the server has not sent any SleepTime value, the peer SHOULD wait for an application-specified minimum time (SleepTimeDefault).

After the Waiting Exchange, the peer MUST discard (from its local ephemeral storage) Noob values that it has sent to the server in OOB messages that are older than the application-defined timeout NoobTimeout (see Section 3.2.2). The peer SHOULD discard such expired Noob values even if the probing failed, e.g. because of failure to connect to the EAP server or incorrect HMAC. The timeout of peer-generated Noob values is defined like this in order to allow the peer to probe the server once after it has waited for the server-specified SleepTime.

If the server and peer have negotiated to use only the server-to-peer direction for the OOB channel (Dirp=2), the peer SHOULD nevertheless probe the server. The purpose of this is to keep the server informed about the peers that are still waiting for OOB messages. The server MAY set SleepTime to a high number (3600) to prevent the peer from probing the server frequently.

```
        EAP Peer                                         EAP Server
          |                                                   |
          |<----------- EAP-Request/Identity -|               |
          |                                                   |
          |                                                   |
          |------------ EAP-Response/Identity --------------->|
          |      (NAI=PeerId+s1@eap-noob.net)                 |
          |                                                   |
          |                                                   |
          |<----------- EAP-Request/EAP-NOOB -----------------|
          |      (Type=3,PeerId,[SleepTime])                  |
          |                                                   |
          |                                                   |
          |------------ EAP-Response/EAP-NOOB --------------->|
          |      (Type=3,PeerId)                              |
          |                                                   |
          |                                                   |
          |<----------- EAP-Failure --------------------------|
          |                                                   |
```

Figure 6: Waiting Exchange

3.3. Protocol data fields

This section defines the various identifiers and data fields used in the EAP-NOOB protocol.

3.3.1. Peer identifier, realm and NAI

The server allocates a new peer identifier (PeerId) for the peer in the Initial Exchange. The peer identifier MUST follow the syntax of the utf8-username specified in [RFC7542]; however, it MUST NOT exceed 60 bytes in length and MUST NOT contain the character '+'. The server MUST generate the identifiers in such a way that they do not repeat and cannot be guessed by the peer or third parties before the server sends them to the peer in the Initial Exchange. One way to generate the identifiers is to choose a random 16-byte identifier and to base64url encode it without padding [RFC4648] into a 22-character string. Another way to generate the identifiers is to choose a random 22-character alphanumeric string. It is not advisable to use identifiers longer than this because they result in longer OOB messages.

When the peer is in one of the states 1..2, the peer MUST use the PeerId that the server assigned to it in the latest Initial Exchange. When the peer is in one of the persistent states 3..4, it MUST use the PeerId from its persistent EAP-NOOB association. When the peer is in the Unregistered (0) state, it MUST use the default value "noob" as its PeerId.

The server MAY also assign a realm (Realm) to the peer by in the Initial Exchange or Reconnect Exchange. The realm value MUST follow the syntax of the utf8-realm specified in [RFC7542]. When the peer is in one of the states 1..2, the peer MUST use the Realm that the server assigned to it the latest Initial Exchange, if one was assigned. When the peer is in one of the persistent states 3..4, it MUST use the Realm from its persistent EAP-NOOB association, if one is stored in the association. When the peer is in the Unregistered (0) state, or when the peer is in one of the other states 1..4 and the server has not assigned it a realm, the peer SHOULD use the default realm "eap-noob.net". However, the user or application MAY provide a different default realm to the peer.

To compose its NAI [RFC7542], the peer uses the PeerId in the user part and the Realm as the realm part. When no server-assigned PeerId or Realm is available, the default value is used instead. In the Unregistered (0) state, the peer must create the NAI as the concatenation of the PeerId, the symbol "@", and the Realm. In the other states 1..4, the peer MUST create the NAI as the concatenation of the PeerId, the string "+s", a single numeric character indicating the state of the peer, and the Realm.

Note that a new peer typically sends the NAI "noob@eap-noob.net" in its first EAP-Response/Identity message, and it is always acceptable for a peer that is in the Unregistered (0) state to use this default NAI. The purpose of the state indicator "+sX" is to inform the server about the peer state without the cost of an additional round trip. The server uses this information together with the server state to decide on the type of the exchange and, thus, the type of the next EAP-Request. The lack of a state indicator in the NAI means that that peer is in state 0.

The purpose of the server-assigned realm is to enable more flexible routing of the EAP sessions over the AAA infrastructure, including roaming scenarios (see Appendix D). Moreover, some Authenticators or AAA servers use the assigned Realm to determine peer-specific connection parameters, such as isolating the peer to a specific VLAN. The possibility to configure a different default realm enables registration of new devices while roaming. It also enables manufacturers to set up their own AAA servers for bootstrapping of new peer devices.

The peer's PeerId and Realm are ephemeral until a successful Completion Exchange takes place. Thereafter, the values become parts of the persistent EAP-NOOB association, until the user resets the peer and the server or until a new Realm is assigned in the Reconnect Exchange.

3.3.2. Message data fields

Table 1 defines the data fields in the protocol messages. The in-band messages are formatted as JSON objects [RFC8259] in UTF-8 encoding. The JSON member names are in the left-hand column of the table.

Data field	Description
Vers, Verp	EAP-NOOB protocol versions supported by the EAP server, and the protocol version chosen by the peer. Vers is a JSON array of unsigned integers, and Verp is an unsigned integer. Example values are "[1]" and "1", respectively.
PeerId	Peer identifier as defined in Section 3.3.1.
Realm	Peer realm as defined in Section 3.3.1.
Peer State "+sX"	Peer state indicator as defined in Section 3.3.1.
Type	EAP-NOOB message type. The type is an integer in the range 0..8. EAP-NOOB requests and the corresponding responses share the same type value.
PKs, PKp	The public components of the ECDHE keys of the server and peer. PKs and PKp are sent in the JSON Web Key (JWK) format [RFC7517]. Detailed format of the JWK object is defined by the cryptosuite.
Cryptosuites, Cryptosuitep	The identifiers of cryptosuites supported by the server and of the cryptosuite selected by the peer. The server-supported cryptosuites in Cryptosuites are formatted as a JSON array of the identifier integers. The server MUST send a nonempty array with no repeating elements, ordered by decreasing priority. The peer MUST respond with exactly one suite in the Cryptosuitep value, formatted as an identifier integer. The registration of cryptosuites is

	specified in Section 4.1. Example values are "[1]" and "1", respectively.
Dirs, Dirp	The OOB channel directions supported by the server and the directions selected by the peer. The possible values are 1=peer-to-server, 2=server-to-peer, 3=both directions.
Dir	The actual direction of the OOB message (1 =peer-to-server, 2=server-to-peer). This value is not sent over any communication channel but it is included in the computation of the cryptographic fingerprint Hoob.
Ns, Np	32-byte nonces for the Initial Exchange.
ServerInfo	This field contains information about the server to be passed from the EAP method to the application layer in the peer. The information is specific to the application or to the OOB channel and it is encoded as a JSON object of at most 500 bytes. It could include, for example, the access-network name and server name or a Uniform Resource Locator (URL) [RFC4266] or some other information that helps the user to deliver the OOB message to the server through the out-of-band channel.
PeerInfo	This field contains information about the peer to be passed from the EAP method to the application layer in the server. The information is specific to the application or to the OOB channel and it is encoded as a JSON object of at most 500 bytes. It could include, for example, the peer brand, model and serial number, which help the user to distinguish between devices and to deliver the OOB message to the correct peer through the out-of-band channel.
SleepTime	The number of seconds for which peer MUST NOT start a new execution of the EAP-NOOB method with the authenticator, unless the peer receives the OOB message or the peer is reset by the user. The server can use this field to limit the rate at which peers probe it. SleepTime is an unsigned integer in the range 0..3600.

Noob	16-byte secret nonce sent through the OOB channel and used for the session key derivation. The endpoint that received the OOB message uses this secret in the Completion Exchange to authenticate the exchanged key to the endpoint that sent the OOB message.
Hoob	16-byte cryptographic fingerprint (i.e. hash value) computed from all the parameters exchanged in the Initial Exchange and in the OOB message. Receiving this fingerprint over the OOB channel guarantees the integrity of the key exchange and parameter negotiation. Hence, it authenticates the exchanged key to the endpoint that receives the OOB message.
NoobId	16-byte identifier for the OOB message, computed with a one-way function from the nonce Noob in the message.
MACs, MACp	Message authentication codes (HMAC) for mutual authentication, key confirmation, and integrity check on the exchanged information. The input to the HMAC is defined below, and the key for the HMAC is defined in Section 3.5.
Ns2, Np2	32-byte Nonces for the Reconnect Exchange.
KeyingMode	Integer indicating the key derivation method. 0 in the Completion Exchange, and 1..3 in the Reconnect Exchange.
PKs2, PKp2	The public components of the ECDHE keys of the server and peer for the Reconnect Exchange. PKp2 and PKs2 are sent in the JSON Web Key (JWK) format [RFC7517]. Detailed format of the JWK object is defined by the cryptosuite.
MACs2, MACp2	Message authentication codes (HMAC) for mutual authentication, key confirmation, and integrity check on the Reconnect Exchange. The input to the HMAC is defined below, and the key for the HMAC is defined in Section 3.5.
ErrorCode	Integer indicating an error condition. Defined in Section 4.3.

ErrorInfo	Textual error message for logging and debugging purposes. UTF-8 string of at most 500 bytes.

Table 1: Message data fields

It is RECOMMENDED for servers to support both OOB channel directions (Dirs=3), unless the type of the OOB channel limits them to one direction (Dirs=1 or Dirs=2). On the other hand, it is RECOMMENDED that the peer selects only one direction (Dirp=1 or Dirp=2) even when both directions (Dirp=3) would be technically possible. The reason is that, if value 3 is negotiated, the user may be presented with two OOB messages, one for each direction, even though only one of them needs to be delivered. This can be confusing to the user. Nevertheless, the EAP-NOOB protocol is designed to cope also with selected value 3, in which case it uses the first delivered OOB message. In the unlikely case of simultaneously delivered OOB messages, the protocol prioritizes the server-to-peer direction.

The nonces in the in-band messages (Ns, Np, Ns2, Np2) are 32-byte fresh random byte strings, and the secret nonce Noob is a 16-byte fresh random byte string. All the nonces are generated by the endpoint that sends the message.

The fingerprint Hoob and the identifier NoobId are computed with the cryptographic hash function specified in the negotiated cryptosuite and truncated to the 16 leftmost bytes of the output. The message authentication codes (MACs, MACp, MACs2, MACp2) are computed with the HMAC function [RFC2104] based on the same cryptographic hash function and truncated to the 32 leftmost bytes of the output.

The inputs to the hash function for computing the fingerprint Hoob and to the HMAC for computing MACs, MACp, MACs2 and MACp2 are JSON arrays containing a fixed number (17) of elements. The array elements MUST be copied to the array verbatim from the sent and received in-band messages. When the element is a JSON object, its members MUST NOT be reordered or re-encoded. Whitespace MUST NOT be added anywhere in the JSON structure. Implementers should check that their JSON library copies the elements as UTF-8 strings and does not modify then in any way, and that it does not add whitespace to the HMAC input.

The inputs for computing the fingerprint and message authentication codes are the following:

```
Hoob = H(Dir,Vers,Verp,PeerId,Cryptosuites,Dirs,ServerInfo,Cryptos
uitep,Dirp,[Realm],PeerInfo,0,PKs,Ns,PKp,Np,Noob).

NoobId = H("NoobId",Noob).

KzId = H("KzId",Kz,Cryptosuitep).

MACs = HMAC(Kms; 2,Vers,Verp,PeerId,Cryptosuites,Dirs,ServerInfo,C
ryptosuitep,Dirp,[Realm],PeerInfo,0,PKs,Ns,PKp,Np,Noob).

MACp = HMAC(Kmp; 1,Vers,Verp,PeerId,Cryptosuites,Dirs,ServerInfo,C
ryptosuitep,Dirp,[Realm],PeerInfo,0,PKs,Ns,PKp,Np,Noob).

MACs2 = HMAC(Kms2; 2,Vers,Verp,PeerId,Cryptosuites,"",[ServerInfo]
,Cryptosuitep,"",[Realm],[PeerInfo],KeyingMode,[PKs2],Ns2,[PKp2],N
p2,KzId)

MACp2 = HMAC(Kmp2; 1,Vers,Verp,PeerId,Cryptosuites,"",[ServerInfo]
,Cryptosuitep,"",[Realm],[PeerInfo],KeyingMode,[PKs2],Ns2,[PKp2],N
p2,KzId)
```

Missing input values are represented by empty strings "" in the array. The values indicated with "" above are always empty strings. Realm is included in the computation of MACs and MACp if it was sent or received in the preceding Initial Exchange. Each of the values in brackets for the computation of Macs2 and Macp2 MUST be included if it was sent or received in the same Reconnect Exchange; otherwise the value is replaced by an empty string "".

The parameter Dir indicates the direction in which the OOB message containing the Noob value is being sent (1=peer-to-server, 2=server-to-peer). This field is included in the Hoob input to prevent the user from accidentally delivering the OOB message back to its originator in the rare cases where both OOB directions have been negotiated. The keys (Kms, Kmp, Kms2, Kmp2) for the HMACs are defined in Section 3.5.

The nonces (Ns, Np, Ns2, Np2, Noob) and the hash value (NoobId) MUST be base64url encoded [RFC4648] when they are used as input to the cryptograhic functions H or HMAC. These values and the message authentication codes (MACs, MACp, MACs2, MACp2) MUST also be base64url encoded when they are sent in the in-band messages. The values Noob and Hoob in the OOB channel MAY be base64url encoded if that is appropriate for the application and the OOB channel. All base64url encoding is done without padding. The base64url encoded values will naturally consume more space than the number of bytes specified above (22-character string for a 16-byte nonce and 43-character string for a 32-byte nonce or message authentication code). In the key derivation in Section 3.5, on the other hand, the unencoded nonces (raw bytes) are used as input to the key derivation function.

The ServerInfo and PeerInfo are JSON objects with UTF-8 encoding. The length of either encoded object as a byte array MUST NOT exceed 500 bytes. The format and semantics of these objects MUST be defined by the application that uses the EAP-NOOB method.

3.4. Fast reconnect and rekeying

EAP-NOOB implements Fast Reconnect ([RFC3748], section 7.2.1) that avoids repeated use of the user-assisted OOB channel.

The rekeying and the Reconnect Exchange may be needed for several reasons. New EAP output values MSK and EMSK may be needed because of mobility or timeout of session keys. Software or hardware failure or user action may also cause the authenticator, EAP server or peer to lose its non-persistent state data. The failure would typically be detected by the peer or authenticator when session keys no longer are accepted by the other endpoint. Change in the supported cryptosuites in the EAP server or peer may also cause the need for a new key exchange. When the EAP server or peer detects any one of these events, it MUST change from the Registered to Reconnecting state. These state transitions are labeled Mobility/Timeout/Failure in Figure 1. The EAP-NOOB method will then perform the Reconnect Exchange next time when EAP is triggered.

3.4.1. Persistent EAP-NOOB association

To enable rekeying, the EAP server and peer store the session state in persistent memory after a successful Completion Exchange. This state data, called "persistent EAP-NOOB association", MUST include at least the data fields shown in Table 2. They are used for identifying and authenticating the peer in the Reconnect Exchange. When a persistent EAP-NOOB association exists, the EAP server and peer are in the Registered state (4) or Reconnecting state (3), as shown in Figure 1.

Data field	Value	Type
PeerId	Peer identifier allocated by server	UTF-8 string (typically 22 bytes)
Verp	Negotiated protocol version	integer
Cryptosuitep	Negotiated cryptosuite	integer
CryptosuitepPrev (at peer only)	Previous cryptosuite	integer
Realm	Optional realm assigned by server (default value is "eap-noob.net")	UTF-8 string
Kz	Persistent key material	32 bytes
KzPrev (at peer only)	Previous Kz value	32 bytes
KzId	Kz identifier	16 bytes
KzIdPrev (at peer only)	Previous Kz identifier	16 bytes

Table 2: Persistent EAP-NOOB association

3.4.2. Reconnect Exchange

When the server receives the EAP-Response/Identity, the server chooses the Reconnect Exchange if the peer is in the Reconnecting (3) state and the server itself is in the Registered (4) or Reconnecting (3) state. The peer MUST NOT initiate EAP-NOOB when the peer is in Registered state.

The Reconnect Exchange comprises three EAP-NOOB request-response pairs, one for cryptosuite and parameter negotiation, another for the nonce and ECDHE key exchange, and the last one for exchanging message authentication codes. In the first request and response (Type=5) the server and peer negotiate a protocol version and cryptosuite in the same way as in the Initial Exchange. The server SHOULD NOT offer and the peer MUST NOT accept protocol versions or cryptosuites that it knows to be weaker than the one currently in the Cryptosuitep field of the persistent EAP-NOOB association. The server SHOULD NOT needlessly change the cryptosuites it offers to the same peer because peer devices may have limited ability to update their persistent storage. However, if the peer has different values in the Cryptosuitep and CryptosuitepPrev fields, it SHOULD also accept offers that are not weaker than CryptosuitepPrev. Note that Cryptosuitep and CryptosuitePrev from the persistent EAP-NOOB association are only used to support the negotiation as described above; all actual cryptographic operations use the negotiated cryptosuite. The request and response (Type=5) MAY additionally contain PeerInfo and ServerInfo objects.

The server then determines the KeyingMode (defined in Section 3.5) based on changes in the negotiated cryptosuite and whether it desires to achieve forward secrecy or not. The server SHOULD only select KeyingMode 3 when the negotiated cryptosuite differs from the Cryptosuitep in the server’s persistent EAP-NOOB association, although it is technically possible to select this values without changing the cryptosuite. In the second request and response (Type=6), the server informs the peer about the KeyingMode, and the server and peer exchange nonces (Ns2, Np2). When KeyingMode is 2 or 3 (rekeying with ECDHE), they also exchange public components of ECDHE keys (PKs2, PKp2). The server ECDHE key MUST be fresh, i.e. not previously used with the same peer, and the peer ECDHE key SHOULD be fresh, i.e. not previously used.

In the third and final request and response (Type=7), the server and peer exchange message authentication codes. Both sides MUST compute the keys Kms2 and Kmp2 as defined in Section 3.5 and the message authentication codes MACs2 and MACp2 as defined in Section 3.3.2. Both sides MUST compare the received message authentication code with a locally computed value.

The rules by which the peer compares the received MACs2 are non-trivial because, in addition to authenticating the current exchange, MACs2 may confirm the success or failure of a recent cryptosuite upgrade. The peer processes the final request (Type=7) as follows:

1. The peer first compares the received MACs2 value with one it computed using the Kz stored in the persistent EAP-NOOB association. If the received and computed values match, the peer deletes any data stored in the CryptosuitepPrev and KzPrev fields of the persistent EAP-NOOB association. It does this because the received MACs2 confirms that the peer and server share the same Cryptosuitep and Kz, and any previous values must no longer be accepted.

2. If, on the other hand, the peer finds that the received MACs2 value does not match the one it computed locally with Kz, the

peer checks whether the KzPrev field in the persistent EAP-NOOB association stores a key. If it does, the peer repeats the key derivation (Section 3.5) and local MACs2 computation (Section 3.3.2) using KzPrev in place of Kz. If this second computed MACs2 matches the received value, the match indicates synchronization failure caused by the loss of the last response (Type=7) in a previously attempted cryptosuite upgrade. In this case, the peer rolls back that upgrade by overwriting Cryptosuitep with CryptosuitepPrev and Kz with KzPrev in the persistent EAP-NOOB association. It also clears the CryptosuitepPrev and KzPrev fields.

3. If the received MACs2 matched one of the locally computed values, the peer proceeds to send the final response (Type=7). The peer also moves to the Registered (4) state. When KeyingMode is 1 or 2, the peer stops here. When KeyingMode is 3, the peer also updates the persistent EAP-NOOB association with the negotiated Cryptosuitep and the newly-derived Kz value. To prepare for possible synchronization failure caused by the loss of the final response (Type=7) during cryptosuite upgrade, the peer copies the old Cryptosuitep and Kz values in the persistent EAP-NOOB association to the CryptosuitepPrev and KzPrev fields.

4. Finally, if the peer finds that the received MACs2 does not match either of the two values that it computed locally (or one value if no KzPrev was stored), the peer sends an error message (error code 4001, see Section 3.6.1), which causes the the Reconnect Exchange to end in EAP-Failure.

The server rules for processing the final message are simpler than the peer rules because the server does not store previous keys and it never rolls back a cryptosuite upgrade. Upon receiving the final response (Type=7), the server compares the received value of MACp2 with one it computes locally. If the values match, the Reconnect Exchange ends in EAP-Success. When KeyingMode is 3, the server also updates Cryptosuitep and Kz in the persistent EAP-NOOB association. On the other hand, if the server finds that the values do not match, it sends an error message (error code 4001), and the Reconnect Exchange ends in EAP-Failure.

The endpoints MAY send updated Realm, ServerInfo and PeerInfo objects in the Reconnect Exchange. When there is no update to the values, they SHOULD omit this information from the messages. If the Realm was sent, each side updates Realm in the persistent EAP-NOOB association when moving to the Registered (4) state.

```
        EAP Peer                                         EAP Server
          |                                                   |
          |<----------- EAP-Request/Identity -|               |
          |                                                   |
          |                                                   |
          |------------ EAP-Response/Identity -------------->|
          |      (NAI=PeerId+s3@eap-noob.net)                 |
          |                                                   |
          |                                                   |
          |<----------- EAP-Request/EAP-NOOB ----------------|
          |      (Type=5,Vers,PeerId,Cryptosuites,            |
          |       [Realm],[ServerInfo])                       |
          |                                                   |
          |                                                   |
          |------------ EAP-Response/EAP-NOOB -------------->|
          |      (Type=5,Verp,PeerId,Cryptosuitep,[PeerInfo])|
          |                                                   |
          |                                                   |
          |<----------- EAP-Request/EAP-NOOB ----------------|
          |      (Type=6,PeerId,KeyingMode,KzId,[PKs2],Ns2)   |
          |                                                   |
          |                                                   |
          |------------ EAP-Response/EAP-NOOB -------------->|
          |      (Type=6,PeerId,[PKp2],Np2)                   |
          |                                                   |
          |                                                   |
          |<----------- EAP-Request/EAP-NOOB ----------------|
          |      (Type=7,PeerId,MACs2)                        |
          |                                                   |
          |                                                   |
          |------------ EAP-Response/EAP-NOOB -------------->|
          |      (Type=7,PeerId,MACp2)                        |
          |                                                   |
          |                                                   |
          |<----------- EAP-Success -------------------------|
          |                                                   |
```

Figure 7: Reconnect Exchange

3.4.3. User reset

As shown in the association state machine in Figure 1, the only specified way for the association to return from the Registered state (4) to the Unregistered state (0) is through user-initiated reset. After the reset, a new OOB message will be needed to establish a new association between the EAP server and peer. Typical situations in which the user reset is required are when the other side has accidentally lost the persistent EAP-NOOB association data, or when the peer device is decommissioned.

The server could detect that the peer is in the Registered or Reconnecting state but the server itself is in one of the ephemeral states 0..2 (including situations where the server does not recognize the PeerId). In this case, effort should be made to recover the persistent server state, for example, from a backup storage - especially if many peer devices are similarly affected. If that is not possible, the EAP server SHOULD log the error or notify an administrator. The only way to continue from such a situation is by having the user reset the peer device.

On the other hand, if the peer is in any of the ephemeral states 0..2, including the Unregistered state, the server will treat the peer as a new peer device and allocate a new PeerId to it. The PeerInfo can be used by the user as a clue to which physical device has lost its state. However, there is no secure way of matching the "new" peer with the old PeerId without repeating the OOB Step. This situation will be resolved when the user performs the OOB Step and, thus, identifies the physical peer device. The server user interface MAY support situations where the "new" peer is actually a previously registered peer that has been reset by a user or otherwise lost its persistent data. In those cases, the user could choose to merge new peer identity with the old one in the server. The alternative is to treat the device just like a new peer.

3.5. Key derivation

EAP-NOOB derives the EAP output values MSK and EMSK and other secret keying material from the output of an Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) algorithm following the NIST specification [NIST-DH]. In NIST terminology, we use a C(2, 0, ECC CDH) scheme, i.e. two ephemeral keys and no static keys. In the Initial and Reconnect Exchanges, the server and peer compute the ECDHE shared secret Z as defined in section 6.1.2.2 of the NIST specification [NIST-DH]. In the Completion and Reconnect Exchanges, the server and peer compute the secret keying material from Z with the single-step key derivation function (KDF) defined in section 5.8.1 of the NIST specification. The hash function H for KDF is taken from the negotiated cryptosuite.

KeyingMode	Description
0	Completion Exchange (always with ECDHE)
1	Reconnect Exchange, rekeying without ECDHE
2	Reconnect Exchange, rekeying with ECHDE, no change in cryptosuite
3	Reconnect Exchange, rekeying with ECDHE, new cryptosuite negotiated

Table 3: Keying modes

The key derivation has three different modes (KeyingMode), which are specified in Table 3. Table 4 defines the inputs to KDF in each KeyingMode.

In the Completion Exchange (KeyingMode=0), the input Z comes from the preceding Initial exchange. KDF takes some additional inputs (OtherInfo), for which we use the concatenation format defined in section 5.8.1.2.1 of the NIST specification [NIST-DH]. OtherInfo consists of the AlgorithmId, PartyUInfo, PartyVInfo, and SuppPrivInfo fields. The first three fields are fixed-length bit strings, and SuppPrivInfo is a variable-length string with a one-byte Datalength counter. AlgorithmId is the fixed-length 8-byte ASCII string "EAP-NOOB". The other input values are the server and peer nonces. In the Completion Exchange, the inputs also include the secret nonce Noob from the OOB message.

In the simplest form of the Reconnect Exchange (KeyingMode=1), fresh nonces are exchanged but no ECDHE keys are sent. In this case, input Z to the KDF is replaced with the shared key Kz from the persistent EAP-NOOB association. The result is rekeying without the computational cost of the ECDHE exchange, but also without forward secrecy.

When forward secrecy is desired in the Reconnect Exchange (KeyingMode=2 or KeyingMode=3), both nonces and ECDHE keys are exchanged. Input Z is the fresh shared secret from the ECDHE exchange with PKs2 and PKp2. The inputs also include the shared secret Kz from the persistent EAP-NOOB associatiob. This binds the rekeying output to the previously authenticated keys.

KeyingMode	KDF input field	Value	Length (bytes)
0 Completion	Z	ECDHE shared secret from PKs and PKp	variable
	AlgorithmId	"EAP-NOOB"	8
	PartyUInfo	Np	32
	PartyVInfo	Ns	32
	SuppPubInfo	(not allowed)	
	SuppPrivInfo	Noob	16
1 Reconnect, rekeying without ECDHE	Z	Kz	32
	AlgorithmId	"EAP-NOOB"	8
	PartyUInfo	Np2	32
	PartyVInfo	Ns2	32
	SuppPubInfo	(not allowed)	
	SuppPrivInfo	(null)	0
2 or 3 Reconnect, rekeying, with ECDHE, same or new cryptosuite	Z	ECDHE shared secret from PKs2 and PKp2	variable
	AlgorithmId	"EAP-NOOB"	8
	PartyUInfo	Np2	32
	PartyVInfo	Ns2	32
	SuppPubInfo	(not allowed)	
	SuppPrivInfo	Kz	32

Table 4: Key derivation input

Table 5 defines how the output bytes of KDF are used. In addition to the EAP output values MSK and EMSK, the server and peer derive another shared secret key AMSK, which MAY be used for application-layer security. Further output bytes are used internally by EAP-NOOB for the message authentication keys (Kms, Kmp, Kms2, Kmp2).

The Completion Exchange (KeyingMode=0) produces the shared secret Kz, which the server and peer store in the persistent EAP-NOOB association. When a new cryptosuite is negotiated in the Reconnect Exchange (KeyingMode=3), it similarly produces a new Kz. In that case, the server and peer update both the cryptosuite and Kz in the persistent EAP-NOOB association. Additionally, the peer stores the previous Cryptosuitep and Kz values in the CryptosuitepPrev and KzPrev fields of the persistent EAP-NOOB association.

KeyingMode	KDF output bytes	Used as	Length (bytes)
0 Completion	0..63 64..127 128..191 192..223 224..255 256..287 288..319	MSK EMSK AMSK MethodId Kms Kmp Kz	64 64 64 32 32 32 32
1 or 2 Reconnect, rekeying without ECDHE, or with ECDHE and unchanged cryptosuite	0..63 64..127 128..191 192..223 224..255 256..287	MSK EMSK AMSK MethodId Kms2 Kmp2	64 64 64 32 32 32
3 Reconnect, rekeying with ECDHE, new cryptosuite	0..63 64..127 128..191 192..223 224..255 256..287 288..319	MSK EMSK AMSK MethodId Kms2 Kmp2 Kz	64 64 64 32 32 32 32

Table 5: Key derivation output

Finally, every EAP method must export a Server-Id, Peer-Id and Session-Id [RFC5247]. In EAP-NOOB, the exported Peer-Id is the PeerId which the server has assigned to the peer. The exported Server-Id is a zero-length string (i.e. null string) because EAP-NOOB neither knows nor assigns any server identifier. The exported Session-Id is created by concatenating the Type-Code xxx (TBA) with the MethodId, which is obtained from the KDF output as shown in Table 5.

3.6. Error handling

Various error conditions in EAP-NOOB are handled by sending an error notification message (Type=0) instead of the expected next EAP request or response message. Both the EAP server and the peer may send the error notification, as shown in Figure 8 and Figure 9. After sending or receiving an error notification, the server MUST send an EAP-Failure (as required by [RFC3748] section 4.2). The notification MAY contain an ErrorInfo field, which is a UTF-8 encoded text string with a maximum length of 500 bytes. It is used for sending descriptive information about the error for logging and debugging purposes.

```
     EAP Peer                                               EAP Server
       ...                                                     ...
        |                                                       |
        |<----------- EAP-Request/EAP-NOOB ----------------|
        |        (Type=0,[PeerId],ErrorCode,[ErrorInfo])        |
        |                                                       |
        |                                                       |
        |<----------- EAP-Failure -------------------------|
        |                                                       |
```

Figure 8: Error notification from server to peer

```
     EAP Peer                                               EAP Server
       ...                                                     ...
        |                                                       |
        |------------ EAP-Response/EAP-NOOB -------------->|
        |        (Type=0,[PeerId],ErrorCode,[ErrorInfo])        |
        |                                                       |
        |                                                       |
        |<----------- EAP-Failure -------------------------|
        |                                                       |
```

Figure 9: Error notification from peer to server

After the exchange fails due to an error notification, the server and peer set the association state as follows. In the Initial Exchange, both the sender and recipient of the error notification MUST set the association state to the Unregistered (0) state. In the Waiting and Completion Exchanges, each side MUST remain in its old state as if the failed exchange had not taken place, with the exception that the recipient of error code 2003 processes it as specified in Section 3.2.3. In the Reconnect Exchange, both sides MUST set the association state to the Reconnecting (3) state.

Errors that occur in the OOB channel are not explicitly notified in-band.

3.6.1. Invalid messages

If the NAI structure is invalid, the server SHOULD send the error code 1001 to the peer. The recipient of an EAP-NOOB request or response SHOULD send the following error codes back to the sender: 1002 if it cannot parse the message as a JSON object or the top-level JSON object has missing or unrecognized members; 1003 if a data field has an invalid value, such as an integer out of range, and there is no more specific error code available; 1004 if the received message type was unexpected in the current state; 2004 if the PeerId has an unexpected value; 2003 if the NoobId is not recognized; 2005 if the KzId is not recognized; and 1007 if the ECDHE key is invalid.

3.6.2. Unwanted peer

The preferred way for the EAP server to rate limit EAP-NOOB connections from a peer is to use the SleepTime parameter in the Waiting Exchange. However, if the EAP server receives repeated EAP-NOOB connections from a peer which apparently should not connect to this server, the server MAY indicate that the connections are unwanted by sending the error code 2001. After receiving this error message, the peer MAY refrain from reconnecting to the same EAP server and, if possible, both the EAP server and peer SHOULD indicate this error condition to the user or server administrator. However, in order to avoid persistent denial of service, the peer is not required to stop entirely from reconnecting to the server.

3.6.3. State mismatch

In the states indicated by "-" in Figure 10 in Appendix A, user action is required to reset the association state or to recover it, for example, from backup storage. In those cases, the server sends the error code 2002 to the peer. If possible, both the EAP server and peer SHOULD indicate this error condition to the user or server administrator.

3.6.4. Negotiation failure

If there is no matching protocol version, the peer sends the error code 3001 to the server. If there is no matching cryptosuite, the peer sends the error code 3002 to the server. If there is no matching OOB direction, the peer sends the error code 3003 to the server.

In practice, there is no way of recovering from these errors without software or hardware changes. If possible, both the EAP server and peer SHOULD indicate these error conditions to the user.

3.6.5. Cryptographic verification failure

If the receiver of the OOB message detects an unrecognized PeerId or incorrect fingerprint (Hoob) in the OOB message, the receiver MUST remain in the Waiting for OOB state (1) as if no OOB message was received. The receiver SHOULD indicate the failure to accept the OOB message to the user. No in-band error message is sent.

Note that if the OOB message was delivered from the server to the peer and the peer does not recognize the PeerId, the likely cause is that the user has unintentionally delivered the OOB message to the wrong peer device. If possible, the peer SHOULD indicate this to the user; however, the peer device may not have the capability for many different error indications to the user and it MAY use the same indication as in the case of an incorrect fingerprint.

The rationale for the above is that the invalid OOB message could have been presented to the receiver by mistake or intentionally by a malicious party and, thus, it should be ignored in the hope that the honest user will soon deliver a correct OOB message.

If the EAP server or peer detects an incorrect message authentication code (MACs, MACp, MACs2, MACp2), it sends the error code 4001 to the other side. As specified in the beginning of Section 3.6, the failed Completion Exchange will not result in server or peer state changes while error in the Reconnect Exchange will put both sides to the Reconnecting (3) state and thus lead to another reconnect attempt.

The rationale for this is that the invalid cryptographic message may have been spoofed by a malicious party and, thus, it should be ignored. In particular, a spoofed message on the in-band channel should not force the honest user to perform the OOB Step again. In practice, however, the error may be caused by other failures, such as a software bug. For this reason, the EAP server MAY limit the rate of peer connections with SleepTime after the above error. Also, there SHOULD be a way for the user to reset the peer to the Unregistered state (0), so that the OOB Step can be repeated at the last resort.

3.6.6. Application-specific failure

Applications MAY define new error messages for failures that are specific to the application or to one type of OOB channel. They MAY also use the generic application-specific error code 5001, or the error codes 5002 and 5004, which have been reserved for indicating invalid data in the ServerInfo and PeerInfo fields, respectively. Additionally, anticipating OOB channels that make use of a URL, the error code 5003 has been reserved for indicating invalid server URL.

4. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the EAP-NOOB protocol, in accordance with [RFC8126].

The EAP Method Type number for EAP-NOOB needs to be assigned.

This memo also requires IANA to create new registries as defined in the following subsections.

4.1. Cryptosuites

Cryptosuites are identified by an integer. Each cryptosuite MUST specify an ECDHE curve for the key exchange, encoding of the ECDHE public key as a JWK object, and a cryptographic hash function for the fingerprint and HMAC computation and key derivation. The hash value output by the cryptographic hash function MUST be at least 32 bytes in length. The following suites are defined by EAP-NOOB:

Cryptosuite	Algorithms
1	ECDHE curve Curve25519 [RFC7748], public-key format [RFC7518] Section 6.2.1, hash function SHA-256 [RFC6234]

Table 6: EAP-NOOB cryptosuites

An example of Cryptosuite 1 public-key encoded as a JWK object is given below (line breaks are for readability only).

"jwk":{"kty":"EC","crv":"Curve25519","x":"3p7bfXt9wbTTW2HC7OQ1Nz-
DQ8hbeGdNrfx-FG-IK08"}

Assignment of new values for new cryptosuites MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC8126].

4.2. Message Types

EAP-NOOB request and response pairs are identified by an integer Message Type. The following Message Types are defined by EAP-NOOB:

Message Type	Used in Exchange	Purpose
1	Initial	Version, cryptosuite and parameter negotiation
2	Initial	Exchange of ECDHE keys and nonces
3	Waiting	Indication to peer that the server has not yet received an OOB message
4	Completion	Authentication and key confirmation with HMAC
5	Reconnect	Version, cryptosuite, and parameter negotiation
6	Reconnect	Exchange of ECDHE keys and nonces
7	Reconnect	Authentication and key confirmation with HMAC
8	Completion	NoobId discovery
0	Error	Error notification

Table 7: EAP-NOOB

Assignment of new values for new Message Types MUST be done through IANA with "Expert Review" as defined in [RFC8126].

4.3. Error codes

The error codes defined by EAP-NOOB are listed in Table 8.

Error code	Purpose
1001	Invalid NAI
1002	Invalid message structure
1003	Invalid data
1004	Unexpected message type
1007	Invalid ECDHE key
2001	Unwanted peer
2002	State mismatch, user action required
2003	Unrecognized OOB message identifier
2004	Unexpected peer identifier
2005	Unrecognized Kz identifier
3001	No mutually supported protocol version
3002	No mutually supported cryptosuite
3003	No mutually supported OOB direction
4001	HMAC verification failure
5001	Application-specific error
5002	Invalid server info
5003	Invalid server URL
5004	Invalid peer info
6001-6999	Private and experimental use

Table 8: EAP-NOOB error codes

Assignment of new error codes MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC8126], with the exception of the range 6001-6999, which is reserved for "Private Use" and "Experimental Use".

4.4. Domain name reservation considerations

"eap-noob.net" should be registered as a special-use domain. The considerations required by [RFC6761] for registering this special-use domain name are the following:

- o Users: Non-admin users are not expected to encounter this name or recognize it as special. AAA administrators may need to recognize the name.

- o Application Software: Application software is not expected to recognize this domain name as special.

- o Name Resolution APIs and Libraries: Name resolution APIs and libraries are not expected to recognize this domain name as special.

- o Caching DNS Servers: Caching servers are not expected to recognize this domain name as special.

- o Authoritative DNS Servers: Authoritative DNS servers MUST respond to queries for eap-noob.net with NXDOMAIN.

- o DNS Server Operators: Except for the authoritative DNS server, there are no special requirements for the operators.

- o DNS Registries/Registrars: There are no special requirements for DNS registrars.

5. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

5.1. Implementation with wpa_supplicant and hostapd

- o Responsible Organization: Aalto University

- o Location: <https://github.com/tuomaura/eap-noob>

- o Coverage: This implementation includes all of the features described in the current specification. The implementation supports two dimensional QR codes and NFC as example out-of-band (OOB) channels

- o Level of Maturity: Alpha

- o Version compatibility: Version 03 of the draft implemented

- o Licensing: BSD

- o Contact Information: Tuomas Aura, tuomas.aura@aalto.fi

5.2. Protocol modeling

The current EAP-NOOB specification has been modeled with the mCRL2 formal specification language [mcrl2]. The model <https://github.com/tuomaura/eap-noob/tree/master/protocolmodel/mcrl2> was used mainly for simulating the protocol behavior and for verifying basic safety and liveness properties as part of the specification process. For example, we verified the correctness of the tiebreaking mechanism when two OOB messages are received simultaneously, one in each direction. We also verified that a man-in-the-middle attacker cannot cause persistent failure by spoofing a finite number of messages in the Reconnect Exchange. Additionally, the protocol has been modeled with the ProVerif [proverif] tool. This model <https://github.com/tuomaura/eap-noob/tree/master/protocolmodel/proverif> was used to verify security properties such as mutual authentication.

6. Security considerations

EAP-NOOB is an authentication and key derivation protocol and, thus, security considerations can be found in most sections of this specification. In the following, we explain the protocol design and highlight some other special considerations.

6.1. Authentication principle

EAP-NOOB establishes a shared secret with an authenticated ECDHE key exchange. The mutual authentication in EAP-NOOB is based on two separate features, both conveyed in the OOB message. The first authentication feature is the secret nonce Noob. The peer and server use this secret in the Completion Exchange to mutually authenticate the session key previously created with ECDHE. The message authentication codes computed with the secret nonce Noob are alone sufficient for authenticating the key exchange. The second authentication feature is the integrity-protecting fingerprint Hoob. Its purpose is to prevent impersonation and man-in-the-middle attacks even in situations where the attacker is able to eavesdrop the OOB channel and the nonce Noob is compromised. In some human-assisted OOB channels, such as sound burst or user-transferred URL, it may be easier to detect tampering than spying of the OOB message, and such applications benefit from the second authentication feature.

The additional security provided by the cryptographic fingerprint Hoob is somewhat intricate to understand. The endpoint that receives the OOB message uses Hoob to verify the integrity of the ECDHE exchange. Thus, the OOB receiver can detect impersonation and man-in-the-middle attacks on the in-band channel. The other endpoint, however, is not equally protected because the OOB message and fingerprint are sent only in one direction. Some protection to the OOB sender is afforded by the fact that the user may notice the failure of the association at the OOB receiver and therefore reset the OOB sender. Other device-pairing protocols have solved similar situations by requiring the user to confirm to the OOB sender that the association was accepted by the OOB receiver, e.g. by pressing an "confirm" button on the sender side. Applications MAY implement EAP-NOOB in this way. Nevertheless, since EAP-NOOB was designed to work with strictly one-directional OOB communication and the fingerprint is only the second authentication feature, the EAP-NOOB specification does not mandate such explicit confirmation to the OOB sender.

To summarize, EAP-NOOB uses the combined protection of the secret nonce Noob and the cryptographic fingerprint Hoob, both conveyed in the OOB message. The secret nonce Noob alone is sufficient for mutual authentication, unless the attacker can eavesdrop it from the OOB channel. Even if an attacker is able to eavesdrop the secret nonce Noob, it nevertheless cannot perform a full man-in-the-middle attack on the in-band channel because the mismatching fingerprint would alert the OOB receiver, which would reject the OOB message. The attacker that eavesdropped the secret nonce can impersonate the OOB receiver to the OOB sender. In this case, the association will appear to be complete only on the OOB sender side, and such situations have to be resolved by the user by resetting the OOB sender to the initial state.

The expected use cases for EAP-NOOB are ones where it replaces a user-entered access credentials in IoT appliances. In wireless network access without EAP, the user-entered credential is often a passphrase that is shared by all the network stations. The advantage of an EAP-based solution, including EAP-NOOB, is that it establishes a different master secret for each peer device, which makes the system more resilient against device compromise than if there were a common master secret. Additionally, it is possible to revoke the security association for an individual device on the server side.

Forward secrecy in EAP-NOOB is optional. The Reconnect Exchange in EAP-NOOB provides forward secrecy only if both the server and peer send their fresh ECDHE keys. This allows both the server and the peer to limit the frequency of the costly computation that is required for forward secrecy. The server MAY adjust the frequency of its attempts at ECDHE rekeying based on what it knows about the peer’s computational capabilities.

The users delivering the OOB messages will often authenticate themselves to the EAP server, e.g. by logging into a secure web page. In this case, the server can reliably associate the peer device with the user account. Applications that make use of EAP-NOOB can use this information for configuring the initial owner of the freshly-registered device.

6.2. Identifying correct endpoints

Potential weaknesses in EAP-NOOB arise from the fact that the user must identify physically the correct peer device. If the attacker is able to trick the user into delivering the OOB message to or from the wrong peer device, the server may create an association with the wrong peer. This reliance on user in identifying the correct endpoints is an inherent property of user-assisted out-of-band authentication.

It is, however, not possible to exploit accidental delivery of the OOB message to the wrong device when the user makes a mistake. This is because the wrong peer device would not have prepared for the attack by performing the Initial Exchange with the server. In comparison, simpler solutions where the master key is transferred to the device via the OOB channel are vulnerable to opportunistic attacks if the user mistakenly delivers the master key to more than one device.

One mechanism that can mitigate user mistakes is certification of peer devices. The certificate can convey to the server authentic identifiers and attributes of the peer device. Compared to a fully certificate-based authentication, however, EAP-NOOB can be used without trusted third parties and does not require the user to know any identifier of the peer device; physical access to the device is sufficient.

Similarly, the attacker can try to trick the user to deliver the OOB message to the wrong server, so that the peer device becomes associated with the wrong server. Since the EAP server is typically online and accessed through a web user interface, the attack would be akin to phishing attacks where the user is tricked to accessing the wrong URL and wrong web page.

6.3. Trusted path issues and misbinding attacks

Another potential threat is spoofed user input or output on the peer device. When the user is delivering the OOB message to or from the correct peer device, a trusted path between the user and the peer device is needed. That is, the user must communicate directly with an authentic operating system and EAP-NOOB implementation in the peer device and not with a spoofed user interface. Otherwise, a Registered device that is under the control of the attacker could emulate the behavior of an unregistered device. The secure path can be implemented, for example, by having the user pressing a reset button to return the device to the Unregistered state and a trusted UI. The problem with such trusted paths is that they are not standardized across devices.

Another potential consequence of spoofed UI is the misbinding attack where the user tries to register the correct but compromised device, and that device tricks the user into registering another device instead. For example, a compromised device might have a malicious full-screen app running, which presents to the user QR codes copied, in real time, from another device's screen. If the unwitting user scans the QR code and delivers the OOB message in it to the server, the wrong device may become registered in the server. Such misbinding vulnerabilities arise because the user does not have any secure way of verifying that the in-band cryptographic handshake and the out-of-band physical access are terminated at the same physical device. Sethi et al. [Sethi19] analyze the binding threat against device-pairing protocols and also EAP-NOOB. Essentially, all protocols where the authentication relies on the user's physical access to the device are vulnerable to misbinding, including EAP-NOOB.

A standardized trusted path for communicating directly with the trusted computing base in a physical device would mitigate the misbinding threat, but such paths rarely exist in practice. Careful asset tracking can also prevent most misbinding attacks because the PeerInfo sent in-band by the wrong device will not match expected values. Device certification by the manufacturer can further strengthen the asset tracking.

6.4. Peer identifiers and attributes

The PeerId value in the protocol is a server-allocated identifier for its association with the peer and SHOULD NOT be shown to the user because its value is initially ephemeral. Since the PeerId is allocated by the server and the scope of the identifier is the single server, the so-called identifier squatting attacks, where a malicious peer could reserve another peer's identifier, are not possible in EAP-NOOB. The server SHOULD assign a random or pseudo-random PeerId to each new peer. It SHOULD NOT select the PeerId based on any peer characteristics that it may know, such as the peer's link-layer network address.

User reset or failure in the OOB Step can cause the peer to perform many Initial Exchanges with the server and to allocate many PeerIds and to store the ephemeral protocol state for them. The peer will typically only remember the latest one. EAP-NOOB leaves it to the implementation to decide when to delete these ephemeral associations. There is no security reason to delete them early, and the server does not have any way to verify that the peers are actually the same one. Thus, it is safest to store the ephemeral states for at least one day. If the OOB messages are sent only in the server-to-peer direction, the server SHOULD NOT delete the ephemeral state before all the related Noob values have expired.

After completion of EAP-NOOB, the server may store the PeerInfo data, and the user may use it to identify the peer and its properties, such as the make and model or serial number. A compromised peer could lie in the PeerInfo that it sends to the server. If the server stores any information about the peer, it is important that this information is approved by the user during or after the OOB Step. Without verification by the user or authentication with vendor certificates on the application level, the PeerInfo is not authenticated information and should not be relied on.

One possible use for the PeerInfo field is EAP channel binding ([RFC3748] Section 7.15). That is, the PeerInfo may include data items that bind the EAP-NOOB association and exported keys to properties of the authenticator or the access link, such as the SSID and BSSID of the wireless network (see Appendix C).

6.5. Identity protection

The PeerInfo field contains identifiers and other information about the peer device (see Appendix C), and the peer sends this information in plaintext to the EAP server before the server authentication in EAP-NOOB has been completed. While the information refers to the peer device and not directly to the user, it may be better for user privacy to avoid sending unnecessary information. In the Reconnect Exchange, the optional PeerInfo SHOULD be omitted unless some critical data has changed.

Peer devices that randomize their layer-2 address to prevent tracking can do this whenever the user resets the EAP-NOOB association. During the lifetime of the association, the PeerId is a unique identifier that can be used to track the peer in the access network. Later versions of this specification may consider updating the PeerId at each Reconnect Exchange. In that case, it is necessary to consider how the authenticator and access-network administrators can recognize and blacklist misbehaving peer devices and how to avoid loss of synchronization between the server and the peer if messages are lost during the identifier update.

6.6. Downgrading threats

The fingerprint Hoob protects all the information exchanged in the Initial Exchange, including the cryptosuite negotiation. The message authentication codes MACs and MACp also protect the same information. The message authentication codes MACs2 and MACp2 protect information exchanged during key renegotiation in the Reconnect Exchange. This prevents downgrading attacks to weaker cryptosuites as long as the possible attacks take more time than the maximum time allowed for the EAP-NOOB completion. This is typically the case for recently discovered cryptanalytic attacks.

As an additional precaution, the EAP server and peer SHOULD check for downgrading attacks in the Reconnect Exchange. As long as the server or peer saves any information about the other endpoint, it MUST also remember the previously negotiated cryptosuite and MUST NOT accept renegotiation of any cryptosuite that is known to be weaker than the previous one, such as a deprecated cryptosuite.

Integrity of the direction negotiation cannot be verified in the same way as the integrity of the cryptosuite negotiation. That is, if the OOB channel used in an application is critically insecure in one direction, a man-in-the-middle attacker could modify the negotiation messages and thereby cause that direction to be used. Applications that support OOB messages in both directions SHOULD therefore ensure that the OOB channel has sufficiently strong security in both directions. While this is a theoretical vulnerability, it could arise in practice if EAP-NOOB is deployed in unexpected applications. However, most devices acting as the peer are likely to support only one direction of exchange, in which case interfering with the direction negotiation can only prevent the completion of the protocol.

The long-term shared key material Kz in the persistent EAP-NOOB association is established with an ECDHE key exchange when the peer and server are first associated. It is a weaker secret than a manually configured random shared key because advances in cryptanalysis against the used ECDHE curve could eventually enable the attacker to recover Kz. EAP-NOOB protects against such attacks by allowing cryptosuite upgrades in the Reconnect Exchange and by updating shared key material Kz whenever the cryptosuite is upgraded. We do not expect the cryptosuite upgrades to be frequent, but if one becomes necessary, the upgrade can be made without manual resetting and reassociation of the peer devices.

6.7. Recovery from loss of last message

The EAP-NOOB Completion Exchange, as well as the Reconnect Exchange with cryptosuite update, result in a persistent state change that should take place either on both endpoints or on neither; otherwise, the result is a state mismatch that requires user action to resolve. The state mismatch can occur if the final EAP response of the exchanges is lost. In the Completion Exchange, the loss of the final response (Type=4) results in the peer moving to Registered (4) state and creating a persistent EAP-NOOB association while the server stays in an ephemeral state (1 or 2). In the Reconnect Exchange, the loss of the final response (Type=7) results in the peer moving to the Registered (4) state and updating its persistent key material Kz while the server stays in the Reconnecting (3) state and keeps the old key material.

The state mismatch is an example of a unavoidable problem in distributed systems: it is theoretically impossible to guarantee synchronous state changes in endpoints that communicate asynchronously. The protocol will always have one critical message that may get lost, so that one side commits to the state change and the other side does not. In EAP, the critical message is the final response from the peer to the server. While the final response is normally followed by EAP-Success, [RFC3748] section 4.2 states that the peer MAY assume that the EAP-Success was lost and the authentication was successful. Furthermore, EAP methods in the peer do not receive notification of the EAP-Success message from the parent EAP state machine [RFC4137]. For these reasons, EAP-NOOB on the peer side commits to a state change already when it sends the final response.

The best available solution to the loss of the critical message is to keep trying. EAP retransmission behavior defined in Section 4.3 of [RFC3748] suggests 3-5 retransmissions. In the absence of an attacker, this would be sufficient to reduce the probability of failure to an acceptable level. However, a determined attacker on the in-band channel can drop the final EAP-Response message and all subsequent retransmissions. In the Completion Exchange (KeyingMode=0) and in the Reconnect Exchange with cryptosuite upgrade (KeyingMode=3), this could result in state mismatch and persistent denial of service until user resets the peer state.

EAP-NOOB implements its own recovery mechanism that allows unlimited retries of the Reconnect Exchange. When the DoS attacker eventually stops dropping packets on the in-band channel, the protocol will recover. The logic for this recovery mechanism is specified in Section 3.4.2.

EAP-NOOB does not implement the same kind of retry mechanism in the Completion Exchange. The reason is that there is always a user involved in the initial association process, and the user can repeat the OOB Step to complete the association after the DoS attacker has left. On the other hand, Reconnect Exchange needs to work without user involvement.

6.8. EAP security claims

EAP security claims are defined in section 7.2.1 of [RFC3748]. The security claims for EAP-NOOB are listed in Table 9.

Security property	EAP-NOOB claim
Authentication mechanism	ECDHE key exchange with out-of-band authentication
Protected cryptosuite negotiation	yes
Mutual authentication	yes
Integrity protection	yes
Replay protection	yes
Key derivation	yes
Key strength	The specified cryptosuites provide key strength of at least 128 bits.
Dictionary attack protection	yes
Fast reconnect	yes
Cryptographic binding	not applicable
Session independence	yes
Fragmentation	no
Channel binding	yes (The ServerInfo and PeerInfo can be used to convey integrity-protected channel properties such as network SSID or peer MAC address.)

Table 9: EAP security claims

7. References

7.1. Normative references

[NIST-DH] Barker, E., Chen, L., Roginsky, A., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publication 800-56A Revision 2 , May 2013, <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <https://www.rfc-editor.org/info/rfc3748>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <https://www.rfc-editor.org/info/rfc5247>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6761] Cheshire, S. and M. Krochmal, "Special-Use Domain Names", RFC 6761, DOI 10.17487/RFC6761, February 2013, <https://www.rfc-editor.org/info/rfc6761>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <https://www.rfc-editor.org/info/rfc7542>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

7.2. Informative references

[BluetoothPairing] Bluetooth, SIG, "Simple pairing whitepaper", Technical report , 2007.

[EUI-48] Institute of Electrical and Electronics Engineers, "802-2014 IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.", IEEE Standard 802-2014. , June 2014.

[IEEE-802.1X] Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Port-Based Network Access Control", IEEE Standard 802.1X-2004. , December 2004.

[mcrl2] Groote, J. and M. Mousavi, "Modeling and analysis of communicating systems", The MIT press , 2014, <https://mitpress.mit.edu/books/modeling-and-analysis-communicating-systems>.

[proverif]
Blanchet, B., Smyth, B., Cheval, V., and M. Sylvestre, "ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial", The MIT press , 2018, <http://prosecco.gforge.inria.fr/personal/bblanche/proverif/>.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, DOI 10.17487/RFC2904, August 2000, <https://www.rfc-editor.org/info/rfc2904>.

[RFC4137] Vollbrecht, J., Eronen, P., Petroni, N., and Y. Ohba, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator", RFC 4137, DOI 10.17487/RFC4137, August 2005, <https://www.rfc-editor.org/info/rfc4137>.

[RFC4266] Hoffman, P., "The gopher URI Scheme", RFC 4266, DOI 10.17487/RFC4266, November 2005, <https://www.rfc-editor.org/info/rfc4266>.

[RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, DOI 10.17487/RFC5216, March 2008, <https://www.rfc-editor.org/info/rfc5216>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[Sethi14] Sethi, M., Oat, E., Di Francesco, M., and T. Aura, "Secure Bootstrapping of Cloud-Managed Ubiquitous Displays", Proceedings of ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp 2014), pp. 739-750, Seattle, USA , September 2014, <http://dx.doi.org/10.1145/2632048.2632049>.

[Sethi19] Sethi, M., Peltonen, A., and T. Aura, "Misbinding Attacks on Secure Device Pairing", 2019, <https://arxiv.org/abs/1902.07550>.

Appendix A. Exchanges and events per state

Figure 10 shows how the EAP server chooses the exchange type depending on the server and peer states. In the state combinations marked with hyphen "-", there is no possible exchange and user action is required to make progress. Note that peer state 4 is omitted from the table because the peer never connects to the server when the peer is in that state. The table also shows the handling of errors in each exchange. A notable detail is that the recipient of error code 2003 moves to state 1.

peer states	exchange chosen by server	next peer and server states
server state: Unregistered (0)		
0..2	Initial Exchange	both 1 (0 on error)
3	-	no change, notify user
server state: Waiting for OOB (1)		
0	Initial Exchange	both 1 (0 on error)
1	Waiting Exchange	both 1 (no change on error)
2	Completion Exchange	both 4 (A)
3	-	no change, notify user
server state: OOB Received (2)		
0	Initial Exchange	both 1 (0 on error)
1	Completion Exchange	both 4 (B)
2	Completion Exchange	both 4 (A)
3	-	no change, notify user
server state: Reconnecting (3) or Registered (4)		
0..2	-	no change, notify user
3	Reconnect Exchange	both 4 (3 on error)

(A) peer to 1 on error 2003, no other changes on error
(B) server to 1 on error 2003, no other changes on error

Figure 10: How server chooses the exchange type

Figure 11 lists the local events that can take place in the server or peer. Both the server and peer output and accept OOB messages in association state 1, leading the receiver to state 2. Communication errors and timeouts in states 0..2 lead back to state 0, while similar errors in states 3..4 lead to state 3. Application request for rekeying (e.g. to refresh session keys or to upgrade cryptosuite) also takes the association from state 3..4 to state 3. User can always reset the association state to 0. Recovering association data, e.g. from a backup, leads to state 3.

server/ peer state	possible local events on server and peer	next state
1	OOB Output*	1
1	OOB Input*	2 (1 on error)
0..2	Timeout/network failure	0
3..4	Timeout/network failure	3
3..4	Rekeying request	3
0..4	User resets peer state	0
0..4	Association state recovery	3

Figure 11: Local events on server and peer

Appendix B. Application-specific parameters

Table 10 lists OOB channel parameters that need to be specified in each application that makes use of EAP-NOOB. The list is not exhaustive and is included for the convenience of implementors only.

Parameter	Description
OobDirs	Allowed directions of the OOB channel
OobMessageEncoding	How the OOB message data fields are encoded for the OOB channel
SleepTimeDefault	Default minimum time in seconds that the peer should sleep before the next Waiting Exchange
OobRetries	Number of received OOB messages with invalid Hoob after which the receiver moves to Unregistered (0) state
NoobTimeout	How many seconds the sender of the OOB message remembers the sent Noob value. The RECOMMENDED value is 3600 seconds.
ServerInfoMembers	Required members in ServerInfo
PeerInfoMembers	Required members in PeerInfo

Table 10: OOB channel characteristics

Appendix C. ServerInfo and PeerInfo contents

The ServerInfo and PeerInfo fields in the Initial Exchange and Reconnect Exchange enable the server and peer, respectively, send information about themselves to the other endpoint. They contain JSON objects whose structure may be specified separately for each application and each type of OOB channel. ServerInfo and PeerInfo MAY contain auxiliary data needed for the OOB channel messaging and for EAP channel binding. Table 11 lists some suggested data fields for ServerInfo.

Data field	Description
ServerName	String that may be used to aid human identification of the server.
ServerURL	Prefix string when the OOB message is formatted as URL, as suggested in Appendix E.
SSIDList	List of wireless network identifier (SSID) strings used for roaming support, as suggested in Appendix D. JSON array of UTF-8 encoded SSID strings.
Base64SSIDList	List of wireless network identifier (SSID) strings used for roaming support, as suggested in Appendix D. JSON array of SSIDs, each of which is base64url encoded without padding. Peer SHOULD send at most one of the fields SSIDList and Base64SSIDList in PeerInfo, and the server SHOULD ignore SSIDList if Base64SSIDList is included.

Table 11: Suggested ServerInfo data fields

PeerInfo typically contains auxiliary information for identifying and managing peers on the application level at the server end. Table 12 lists some suggested data fields for PeerInfo.

Data field	Description
PeerName	String that may be used to aid human identification of the peer.
Manufacturer	Manufacturer or brand string.
Model	Manufacturer-specified model string.
SerialNumber	Manufacturer-assigned serial number.
MACAddress	Peer link-layer identifier (EUI-48) in the 12-digit base-16 form [EUI-48]. The string MAY include additional colon ':' or dash '-' characters that MUST be ignored by the server.
SSID	Wireless network SSID for channel binding. The SSID is a UTF-8 string.
Base64SSID	Wireless network SSID for channel binding. The SSID is base64url encoded. Peer SHOULD send at most one of the fields SSID and Base64SSID in PeerInfo, and the server SHOULD ignore SSID if Base64SSID is included.
BSSID	Wireless network BSSID (EUI-48) in the 12-digit base-16 form [EUI-48]. The string MAY include additional colon ':' or dash '-' characters that MUST be ignored by the server.

Table 12: Suggested PeerInfo data fields

Appendix D. EAP-NOOB roaming

AAA architectures [RFC2904] allow for roaming of network-connected appliances that are authenticated over EAP. While the peer is roaming in a visited network, authentication still takes place between the peer and an authentication server at its home network. EAP-NOOB supports such roaming by assigning a Realm to the peer. After the Realm has been assigned, the peer's NAI enables the visited network to route the EAP session to the peer's home AAA server.

A peer device that is new or has gone through a hard reset should be connected first to the home network and establish an EAP-NOOB association with its home AAA server before it is able to roam.

After that, it can perform the Reconnect Exchange from the visited network.

Alternatively, the device may provide some method for the user to configure the Realm of the home network. In that case, the EAP-NOOB association can be created while roaming. The device will use the user-assigned Realm in the Initial Exchange, which enables the EAP messages to be routed correctly to the home AAA server.

While roaming, the device needs to identify the networks where the EAP-NOOB association can be used to gain network access. For 802.11 access networks, the server MAY send a list of SSID strings in the ServerInfo JSON object in a member called either SSIDList or Base64SSIDList. The list is formated as explained in Table 11. If present, the peer MAY use this list as a hint to determine the networks where the EAP-NOOB association can be used for access authorization, in addition to the access network where the Initial Exchange took place.

Appendix E. OOB message as URL

While EAP-NOOB does not mandate any particular OOB communication channel, typical OOB channels include graphical displays and emulated NFC tags. In the peer-to-server direction, it may be convenient to encode the OOB message as a URL, which is then encoded as a QR code for displays and printers or as an NDEF record for NFC tags. A user can then simply scan the QR code or NFC tag and open the URL, which causes the OOB message to be delivered to the authentication server. The URL MUST specify the https protocol i.e. secure connection to the server, so that the man-in-the-middle attacker cannot read or modify the OOB message.

The ServerInfo in this case includes a JSON member called ServerUrl of the following format with maximum length of 60 characters:

```
https://<host>[:<port>]/[<path>]
```

To this, the peer appends the OOB message fields (PeerId, Noob, Hoob) as a query string. PeerId is provided to the peer by the server and might be a 22-character string. The peer base64url encodes, without padding, the 16-byte values Noob and Hoob into 22-character strings. The query parameters MAY be in any order. The resulting URL is of the following format:

```
https://<host>[:<port>]/[<path>]?P=<PeerId>&N=<Noob>&H=<Hoob>
```

The following is an example of a well-formed URL encoding the OOB message (without line breaks):

https://example.com/Noob?P=ZrD7qkczNoHGbGcN2bN0&N=rMinS0-F4EfCU8D9ljx
X_A&H=QvnMp4UGxuQVFaXPW_14UW

Appendix F. Example messages

The message examples in this section are generated with Curve25519 ECDHE test vectors specified in section 6.1 of [RFC7748] (server=Alice, peer=Bob). The direction of the OOB channel negotiated is 1 (peer-to-server). The JSON messages are as follows (line breaks are for readability only).

====== Initial Exchange ======

Identity response:
noob@eap-noob.net

EAP request (type 1):
{"Type":1,"Vers":[1],"PeerId":"07KRU6OgqX0HIeRFldnbSW","Realm":"no
ob.example.com","Cryptosuites":[1],"Dirs":3,"ServerInfo":{"Name":"
Example","Url":"https://noob.example.com/sendOOB"}}

EAP response (type 1):
{"Type":1,"Verp":1,"PeerId":"07KRU6OgqX0HIeRFldnbSW","Cryptosuitep
":1,"Dirp":1,"PeerInfo":{"Make":"Acme","Type":"None","Serial":"DU-
9999","SSID":"Noob1","BSSID":"6c:19:8f:83:c2:80"}}

EAP request (type 2):
{"Type":2,"PeerId":"07KRU6OgqX0HIeRFldnbSW","PKs":{"kty":"EC","crv
":"Curve25519","x":"hSDwCYkwp1R0i33ctD73Wg2_Og0mOBr066SpjqqbTmo"},
"Ns":"PYO7NVd9Af3BxEri1MI6hL8Ck49YxwCjSRPqlC1SPbw","SleepTime":60}

EAP response (type 2):
{"Type":2,"PeerId":"07KRU6OgqX0HIeRFldnbSW","PKp":{"kty":"EC","crv
":"Curve25519","x":"3p7bfXt9wbTTW2HC7OQ1Nz-DQ8hbeGdNrfx-FG-
IK08"},"Np":"HIvB6g0n2btpxEcU7YXnWB-451ED6L6veQQd6ugiPFU"}

====== Waiting Exchange ======

Identity response:
07KRU6OgqX0HIeRFldnbSW+s1@noob.example.com

EAP request (type 3):
{"Type":3,"PeerId":"07KRU6OgqX0HIeRFldnbSW","SleepTime":60}

EAP response (type 3):
{"Type":3,"PeerId":"07KRU6OgqX0HIeRFldnbSW"}

====== OOB Step ======

```
   Identity response:
      P=07KRU6OgqX0HIeRFldnbSW&N=x3JlolaPciK4Wa6XlMJxtQ&H=faqWz68trUrBTK
      AnioZMQA

   ====== Completion Exchange ======

   Identity response:
      07KRU6OgqX0HIeRFldnbSW+s2@noob.example.com

   EAP request (type 8):
      {"Type":8,"PeerId":"07KRU6OgqX0HIeRFldnbSW"}

   EAP response (type 8):
      {"Type":8,"PeerId":"07KRU6OgqX0HIeRFldnbSW","NoobId":"U0OHwYGCS4nE
      kzk2TPIE6g"}

   EAP request (type 4):
      {"Type":4,"PeerId":"07KRU6OgqX0HIeRFldnbSW","NoobId":"U0OHwYGCS4nE
      kzk2TPIE6g","MACs":"Y5NfKQkZTbRW3sEFhWy0Bv0ic2wsMnaA6xGqtUmQqmc"}

   EAP response (type 4):
      {"Type":4,"PeerId":"07KRU6OgqX0HIeRFldnbSW","MACp":"ddY225rN3lYzo7
      qZNPStbVO1HRdNnTx0Rit6_8xEh7A"}

   ====== Reconnect Exchange ======

   Identity response:
      07KRU6OgqX0HIeRFldnbSW+s3@noob.example.com

   EAP request (type 5):
      {"Type":5,"Vers":[1],"PeerId":"07KRU6OgqX0HIeRFldnbSW","Cryptosuit
      es":[1],"Realm":"noob.example.com","ServerInfo":{"Name":"Example",
      "Url":"https://noob.example.com/sendOOB"}}

   EAP response (type 5):
      {"Type":5,"Verp":1,"PeerId":"07KRU6OgqX0HIeRFldnbSW","Cryptosuitep
      ":1,"PeerInfo":{"Make":"Acme","Type":"None","Serial":"DU-
      9999","SSID":"Noob1","BSSID":"6c:19:8f:83:c2:80"}}

   EAP request (type 6):
      {"Type":6,"PeerId":"07KRU6OgqX0HIeRFldnbSW","PKs2":{"kty":"EC","cr
      v":"Curve25519","x":"hSDwCYkwp1R0i33ctD73Wg2_Og0mOBr066SpjqqbTmo"}
      ,"Ns2":"RDLahHBlIgnmL_F_xcynrHurLPkCsrp3G3B_S82WUF4"}

   EAP response (type 6):
      {"Type":6,"PeerId":"07KRU6OgqX0HIeRFldnbSW","PKp2":{"kty":"EC","cr
      v":"Curve25519","x":"3p7bfXt9wbTTW2HC7OQ1Nz-DQ8hbeGdNrfx-FG-
      IK08"},"Np2":"jN0_V4P0JoTqwI9VHHQKd9ozUh7tQdc9ABd-j6oTy_4"}
```

EAP request (type 7):
{"Type":7,"PeerId":"07KRU6OgqX0HIeRFldnbSW","MACs2":"_pXDF4-
7uBKXKqVKKB6U-GP9EDnGCNOMdkyfEQp_iwA"}

EAP response (type 7):
{"Type":7,"PeerId":"07KRU6OgqX0HIeRFldnbSW","MACp2":"qSUH4zA0VzMqU
2O1U-JJTqwGRXGB8i3bggasYL6o1uU"}

Appendix G. TODO list

o Change the way KzId is generated and document its use in Reconnect Exchange.

Appendix H. Version history

o Version 01:

  * Fixed Reconnection Exchange.

  * URL examples.

  * Message examples.

  * Improved state transition (event) tables.

o Version 02:

  * Reworked the rekeying and key derivation.

  * Increased internal key lengths and in-band nonce and HMAC lengths to 32 bytes.

  * Less data in the persistent EAP-NOOB association.

  * Updated reference [NIST-DH] to Revision 2 (2013).

  * Shorter suggested PeerId format.

  * Optimized the example of encoding OOB message as URL.

  * NoobId in Completion Exchange to differentiate between multiple valid Noob values.

  * List of application-specific parameters in appendix.

  * Clarified the equivalence of Unregistered state and no state.

  * Peer SHOULD probe the server regardless of the OOB channel direction.

  * Added new error messages.

  * Realm is part of the persistent association and can be updated.

  * Clarified error handling.

  * Updated message examples.

  * Explained roaming in appendix.

  * More accurate definition of timeout for the Noob nonce.

  * Additions to security considerations.

- Version 03:

  * Clarified reasons for going to Reconnecting state.

  * Included Verp in persistent state.

  * Added appendix on suggested ServerInfo and PeerInfo fields.

  * Exporting PeerId and SessionId.

  * Explicitly specified next state after OOB Step.

  * Clarified the processing of an expired OOB message and unrecognized NoobId.

  * Enabled protocol version upgrade in Reconnect Exchange.

  * Explained handling of redundant received OOB messages.

  * Clarified where raw and base64url encoded values are used.

  * Cryptosuite must specify the detailed format of the JWK object.

  * Base64url encoding in JSON strings is done without padding.

  * Simplified explanation of PeerId, Realm and NAI.

  * Added error codes for private and experimental use.

  * Updated the security considerations.

o Version 04:

  * Recovery from synchronization failure due to lost last response.

o Version 05:

  * Kz identifier added to help recovery from lost last messages.

  * Error message codes changed for better structure.

  * Improved security considerations section.

Appendix I. Acknowledgments

Aleksi Peltonen modeled the protocol specification with the mCRL2 formal specification language. Shiva Prasad TP and Raghavendra MS implemented parts of the protocol with wpa_supplicant and hostapd. Their inputs helped us in improving the specification.

The authors would also like to thank Rhys Smith and Josh Howlett for providing valuable feedback as well as new use cases and requirements for the protocol. Thanks to Eric Rescorla, Darshak Thakore, Stefan Winter and Hannes Tschofenig for interesting discussions in this problem space.

Authors' Addresses

Tuomas Aura
Aalto University
Aalto 00076
Finland

EMail: tuomas.aura@aalto.fi

Mohit Sethi
Ericsson
Jorvas 02420
Finland

EMail: mohit@piuha.net

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: September 12, 2019

F. Gont
SI6 Networks / UTN-FRH
I. Arce
Quarkslab

March 11, 2019

Security and Privacy Implications of Numeric Identifiers Employed in Network Protocols

draft-gont-predictable-numeric-ids-03

Abstract

This document performs an analysis of the security and privacy implications of different types of "numeric identifiers" used in IETF protocols, and tries to categorize them based on their interoperability requirements and the associated failure severity when such requirements are not met. It describes a number of algorithms that have been employed in real implementations to meet such requirements and analyzes their security and privacy properties. Additionally, it provides advice on possible algorithms that could be employed to satisfy the interoperability requirements of each identifier type, while minimizing the security and privacy implications, thus providing guidance to protocol designers and protocol implementers. Finally, it provides recommendations for future protocol specifications regarding the specification of the aforementioned numeric identifiers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
2. Terminology 4
3. Threat Model 5
4. Issues with the Specification of Identifiers 5
5. Timeline of Vulnerability Disclosures Related to Some Sample Identifiers 6
5.1. IPv4/IPv6 Identification 6
5.2. TCP Initial Sequence Numbers (ISNs) 7
6. Protocol Failure Severity 9
7. Categorizing Identifiers 9
8. Common Algorithms for Identifier Generation 11
8.1. Category #1: Uniqueness (soft failure) 11
8.1.1. Simple Randomization Algorithm 11
8.1.2. Another Simple Randomization Algorithm 12
8.2. Category #2: Uniqueness (hard failure) 13
8.3. Category #3: Uniqueness, constant within context (soft-failure) 13
8.4. Category #4: Uniqueness, monotonically increasing within context (hard failure) 14
8.4.1. Predictable Linear Identifiers Algorithm 14
8.4.2. Per-context Counter Algorithm 16
8.4.3. Simple Hash-Based Algorithm 18
8.4.4. Double-Hash Algorithm 20
8.4.5. Random-Increments Algorithm 21
9. Common Vulnerabilities Associated with Identifiers 23
9.1. Category #1: Uniqueness (soft failure) 23
9.2. Category #2: Uniqueness (hard failure) 23
9.3. Category #3: Uniqueness, constant within context (soft

failure) 23
9.4. Category #4: Uniqueness, monotonically increasing within context (hard failure) 24
10. Security and Privacy Requirements for Identifiers 25
11. IANA Considerations 26
12. Security Considerations 26
13. Acknowledgements 26
14. References 26
14.1. Normative References 26
14.2. Informative References 27
Authors' Addresses 31

1. Introduction

Network protocols employ a variety of numeric identifiers for different protocol entities, ranging from DNS Transaction IDs (TxIDs) to transport protocol numbers (e.g. TCP ports) or IPv6 Interface Identifiers (IIDs). These identifiers usually have specific properties that must be satisfied such that they do not result in negative interoperability implications (e.g. uniqueness during a specified period of time), and associated failure severities when such properties are not met, ranging from soft to hard failures.

For more than 30 years, a large number of implementations of the TCP/IP protocol suite have been subject to a variety of attacks, with effects ranging from Denial of Service (DoS) or data injection, to information leakage that could be exploited for pervasive monitoring [RFC7528]. The root of these issues has been, in many cases, the poor selection of identifiers in such protocols, usually as a result of an insufficient or misleading specification. While it is generally trivial to identify an algorithm that can satisfy the interoperability requirements for a given identifier, there exists practical evidence that doing so without negatively affecting the security and/or privacy properties of the aforementioned protocols is prone to error.

For example, implementations have been subject to security and/or privacy issues resulting from:

- Predictable TCP sequence numbers
- Predictable transport protocol numbers
- Predictable IPv4 or IPv6 Fragment Identifiers
- Predictable IPv6 IIDs
- Predictable DNS TxIDs

Recent history indicates that when new protocols are standardized or new protocol implementations are produced, the security and privacy properties of the associated identifiers tend to be overlooked and inappropriate algorithms to generate identifier values are either suggested in the specification or selected by implementators. As a result, we believe that advice in this area is warranted.

This document contains a non-exhaustive survey of identifiers employed in various IETF protocols, and aims to categorize such identifiers based on their interoperability requirements, and the associated failure severity when such requirements are not met. Subsequently, it analyzes several algorithms that have been employed in real implementation to meet such requirements and analyzes their security and privacy properties, and provides advice on possible algorithms that could be employed to satisfy the interoperability requirements of each category, while minimizing the associated security and privacy implications. Finally, it provides recommendations for future protocol specifications regarding the specification of the aforementioned numeric identifiers.

2. Terminology

Identifier:
A data object in a protocol specification that can be used to definetely distinguish a protocol object (a datagram, network interface, transport protocol endpoint, session, etc) from all other objects of the same type, in a given context. Identifiers are usually defined as a series of bits and represented using integer values. We note that different identifiers may have additional requirements or properties depending on their specific use in a protocol. We use the term "identifier" as a generic term to refer to any data object in a protocol specification that satisfies the identification property stated above.

Failure Severity:
The consequences of a failure to comply with the interoperability requirements of a given identifier. Severity considers the worst potential consequence of a failure, determined by the system damage and/or time lost to repair the failure. In this document we define two types of failure severity: "soft" and "hard".

Hard Failure:
A hard failure is a non-recoverable condition in which a protocol does not operate in the prescribed manner or it operates with excessive degradation of service. For example, an established TCP connection that is aborted due to an error condition constitutes, from the point of view of the transport protocol, a hard failure,

since it enters a state from which normal operation cannot be recovered.

Soft Failure:
A soft failure is a recoverable condition in which a protocol does not operate in the prescribed manner but normal operation can be resumed automatically in a short period of time. For example, a simple packet-loss event that is subsequently recovered with a retransmission can be considered a soft failure.

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

3. Threat Model

Throughout this document, we assume an attacker does not have physical or logical device to the device(s) being attacked. We assume the attacker can simply send any traffic to the target devices, to e.g. sample identifiers employed by such devices.

4. Issues with the Specification of Identifiers

While assessing protocol specifications regarding the use of identifiers, we found that most of the issues discussed in this document arise as a result of one of the following:

- Protocol specifications which under-specify the requirements for their identifiers
- Protocol specifications that over-specify their identifiers
- Protocol implementations that simply fail to comply with the specified requirements

A number of protocol implementations (too many of them) simply overlook the security and privacy implications of identifiers. Examples of them are the specification of TCP port numbers in [RFC0793], the specification of TCP sequence numbers in [RFC0793], or the specification of the DNS TxID in [RFC1035].

On the other hand, there are a number of protocol specifications that over-specify some of their associated protocol identifiers. For example, [RFC4291] essentially results in link-layer addresses being embedded in the IPv6 Interface Identifiers (IIDs) when the interoperability requirement of uniqueness could be achieved in other ways that do not result in negative security and privacy implications [RFC7721]. Similarly, [RFC2460] suggests the use of a global counter for the generation of Fragment Identification values, when the interoperability properties of uniqueness per {Src IP, Dst IP} could be achieved with other algorithms that do not result in negative security and privacy implications.

Finally, there are protocol implementations that simply fail to comply with existing protocol specifications. For example, some popular operating systems (notably Microsoft Windows) still fail to implement randomization of transport protocol ephemeral ports, as specified in [RFC6056].

5. Timeline of Vulnerability Disclosures Related to Some Sample Identifiers

This section contains a non-exhaustive timeline of vulnerability disclosures related to some sample identifiers and other work that has led to advances in this area. The goal of this timeline is to illustrate:

- o That vulnerabilities related to how the values for some identifiers are generated and assigned have affected implementations for an extremely long period of time.

- o That such vulnerabilities, even when addressed for a given protocol version, were later reintroduced in new versions or new implementations of the same protocol.

- o That standardization efforts that discuss and provide advice in this area can have a positive effect on protocol specifications and protocol implementations.

5.1. IPv4/IPv6 Identification

December 1998:
[Sanfilippo1998a] finds that predictable IPv4 Identification values can be leveraged to count the number of packets sent by a target node. [Sanfilippo1998b] explains how to leverage the same vulnerability to implement a port-scanning technique known as dumb/idle scan. A tool that implements this attack is publicly released.

November 1999:
[Sanfilippo1999] discusses how to leverage predictable IPv4 Identification to uncover the rules of a number of firewalls.

November 1999:
[Bellovin2002] explains how the IPv4 Identification field can be exploited to count the number of systems behind a NAT.

December 2003:
[Zalewski2003] explains a technique to perform TCP data injection attack based on predictable IPv4 identification values which requires less effort than TCP injection attacks performed with bare TCP packets.

November 2005:
[Silbersack2005] discusses shortcoming in a number of techniques to mitigate predictable IPv4 Identification values.

October 2007:
[Klein2007] describes a weakness in the pseudo random number generator (PRNG) in use for the generation of the IP Identification by a number of operating systems.

June 2011:
[Gont2011] describes how to perform idle scan attacks in IPv6.

November 2011:
Linux mitigates predictable IPv6 Identification values [RedHat2011] [SUSE2011] [Ubuntu2011].

December 2011:
[I-D.ietf-6man-predictable-fragment-id-08] describes the security implications of predictable IPv6 Identification values, and possible mitigations.

May 2012:
[Gont2012] notes that some major IPv6 implementations still employ predictable IPv6 Identification values.

June 2015:
[I-D.ietf-6man-predictable-fragment-id-08] notes that some popular host and router implementations still employ predictable IPv6 Identification values.

5.2. TCP Initial Sequence Numbers (ISNs)

September 1981:
[RFC0793], suggests the use of a global 32-bit ISN generator, whose lower bit is incremented roughly every 4 microseconds. However, such an ISN generator makes it trivial to predict the ISN that a TCP will use for new connections, thus allowing a variety of attacks against TCP.

February 1985:

[Morris1985] was the first to describe how to exploit predictable TCP ISNs for forging TCP connections that could then be leveraged for trust relationship exploitation.

April 1989:
[Bellovin1989] discussed the security implications of predictable ISNs (along with a range of other protocol-based vulnerabilities).

February 1995:
[Shimomura1995] reported a real-world exploitation of the attack described in 1985 (ten years before) in [Morris1985].

May 1996:
[RFC1948] was the first IETF effort, authored by Steven Bellovin, to address predictable TCP ISNs. The same concept specified in this document for TCP ISNs was later proposed for TCP ephemeral ports [RFC6056], TCP Timestamps, and eventually even IPv6 Interface Identifiers [RFC7217].

March 2001:
[Zalewski2001] provides a detailed analysis of statistical weaknesses in some ISN generators, and includes a survey of the algorithms in use by popular TCP implementations.

May 2001:
Vulnerability advisories [CERT2001] [USCERT2001] are released regarding statistical weaknesses in some ISN generators, affecting popular TCP/IP implementations.

March 2002:
[Zalewski2002] updates and complements [Zalewski2001]. It concludes that "while some vendors [...] reacted promptly and tested their solutions properly, many still either ignored the issue and never evaluated their implementations, or implemented a flawed solution that apparently was not tested using a known approach". [Zalewski2002].

February 2012:
[RFC6528], after 27 years of Morris' original work [Morris1985], formally updates [RFC0793] to mitigate predictable TCP ISNs.

August 2014:
[I-D.eddy-rfc793bis-04], the upcoming revision of the core TCP protocol specification, incorporates the algorithm specified in [RFC6528] as the recommended algorithm for TCP ISN generation.

6. Protocol Failure Severity

Section 2 defines the concept of "Failure Severity" and two types of failures that we employ throughout this document: soft and hard.

Our analysis of the severity of a failure is performed from the point of view of the protocol in question. However, the corresponding severity on the upper application or protocol may not be the same as that of the protocol in question. For example, a TCP connection that is aborted may or may not result in a hard failure of the upper application: if the upper application can establish a new TCP connection without any impact on the application, a hard failure at the TCP protocol may have no severity at the application level. On the other hand, if a hard failure of a TCP connection results in excessive degradation of service at the application layer, it will also result in a hard failure at the application.

7. Categorizing Identifiers

This section includes a non-exhaustive survey of identifiers, and proposes a number of categories that can accommodate these identifiers based on their interoperability requirements and their failure modes (soft or hard)

Identifier	Interoperability Requirements	Failure Severity
IPv6 Frag ID	Uniqueness (for IP address pair)	Soft/Hard (1)
IPv6 IID	Uniqueness (and constant within IPv6 prefix) (2)	Soft (3)
TCP SEQ	Monotonically-increasing	Hard (4)
TCP eph. port	Uniqueness (for connection ID)	Hard
IPv6 Flow L.	Uniqueness	None (5)
DNS TxID	Uniqueness	None (6)

Table 1: Survey of Identifiers

Notes:

(1)
While a single collision of Fragment ID values would simply lead to a single packet drop (and hence a "soft" failure), repeated collisions at high data rates might trash the Fragment ID space, leading to a hard failure [RFC4963].

(2)
While the interoperability requirements are simply that the Interface ID results in a unique IPv6 address, for operational reasons it is typically desirable that the resulting IPv6 address (and hence the corresponding Interface ID) be constant within each network [I-D.ietf-6man-default-iids] [RFC7217].

(3)
While IPv6 Interface IDs must result in unique IPv6 addresses, IPv6 Duplicate Address Detection (DAD) [RFC4862] allows for the detection of duplicate Interface IDs/addresses, and hence such Interface ID collisions can be recovered.

(4)
In theory there are no interoperability requirements for TCP sequence numbers, since the TIME-WAIT state and TCP's "quiet time" take care of old segments from previous incarnations of the connection. However, a widespread optimization allows for a new incarnation of a previous connection to be created if the Initial Sequence Number (ISN) of the incoming SYN is larger than the last sequence number seen in that direction for the previous incarnation of the connection. Thus, monotonically-increasing TCP sequence numbers allow for such optimization to work as expected [RFC6528].

(5)
The IPv6 Flow Label is typically employed for load sharing [RFC7098], along with the Source and Destination IPv6 addresses. Reuse of a Flow Label value for the same set {Source Address, Destination Address} would typically cause both flows to be multiplexed into the same link. However, as long as this does not occur deterministically, it will not result in any negative implications.

(6)
DNS TxIDs are employed, together with the Source Address, Destination Address, Source Port, and Destination Port, to match DNS requests and responses. However, since an implementation knows which DNS requests were sent for that set of {Source Address, Destination Address, Source Port, and Destination Port, DNS TxID}, a collision of TxID would result, if anything, in a small performance penalty (the response would be discarded when it

is found that it does not answer the query sent in the corresponding DNS query).

Based on the survey above, we can categorize identifiers as follows:

Cat #	Category	Sample Proto IDs
1	Uniqueness (soft failure)	IPv6 Flow L., DNS TxIDs
2	Uniqueness (hard failure)	IPv6 Frag ID, TCP ephemeral port
3	Uniqueness, constant within context (soft failure)	IPv6 IIDs
4	Uniqueness, monotonically increasing within context (hard failure)	TCP ISN

Table 2: Identifier Categories

We note that Category #4 could be considered a generalized case of category #3, in which a monotonically increasing element is added to a constant (within context) element, such that the resulting identifiers are monotonically increasing within a specified context. That is, the same algorithm could be employed for both #3 and #4, given appropriate parameters.

8. Common Algorithms for Identifier Generation

The following subsections describe common algorithms found for Protocol ID generation for each of the categories above.

8.1. Category #1: Uniqueness (soft failure)

8.1.1. Simple Randomization Algorithm

```
/* Ephemeral port selection function */
id_range = max_id - min_id + 1;
next_id = min_id + (random() % id_range);
count = next_id;

do {
    if(check_suitable_id(next_id))
        return next_id;

    if (next_id == max_id) {
        next_id = min_id;
    } else {
        next_id++;
    }

    count--;
} while (count > 0);

return ERROR;
```

Note:

random() is a function that returns a pseudo-random unsigned integer number of appropriate size. Note that the output needs to be unpredictable, and typical implementations of POSIX random() function do not necessarily meet this requirement. See [RFC4086] for randomness requirements for security.

The function check_suitable_id() can check, when possible, whether this identifier is e.g. already in use. When already used, this algorithm selects the next available protocol ID.

All the variables (in this and all the algorithms discussed in this document) are unsigned integers.

8.1.2. Another Simple Randomization Algorithm

The following pseudo-code illustrates another algorithm for selecting a random identifier in which, in the event the identifier is found to be not suitable (e.g., already in use), another identifier is selected randomly:

```
       id_range = max_id - min_id + 1;
       next_id = min_id + (random() % id_range);
       count = id_range;

       do {
           if(check_suitable_id(next_id))
               return next_id;

           next_id = min_id + (random() % id_range);
           count--;
       } while (count > 0);

       return ERROR;
```

This algorithm might be unable to select an identifier (i.e., return "ERROR") even if there are suitable identifiers available, when there are a large number of identifiers "in use".

8.2. Category #2: Uniqueness (hard failure)

One of the most trivial approaches for achieving uniqueness for an identifier (with a hard failure mode) is to implement a linear function. As a result, all of the algorithms described in Section 8.4 are of use for complying the requirements of this identifier category.

8.3. Category #3: Uniqueness, constant within context (soft-failure)

The goal of this algorithm is to produce identifiers that are constant for a given context, but that change when the aforementioned context changes.

Keeping one value for each possible "context" may in many cases be considered too onerous in terms of memory requirements. As a workaround, the following algorithm employs a calculated technique (as opposed to keeping state in memory) to maintain the constant identifier for each given context.

In the following algorithm, the function F() provides (statelessly) a constant identifier for each given context.

```
/* Protocol ID selection function  */
id_range = max_id - min_id + 1;

counter = 0;

do {
    offset = F(CONTEXT, counter, secret_key);
    next_id = min_id + (offset % id_range);

    if(check_suitable_id(next_id))
        return next_id;

    counter++;

} while (counter <= MAX_RETRIES);

return ERROR;
```

The function F() provides a "per-CONTEXT" constant identifier for a given context. 'offset' may take any value within the storage type range since we are restricting the resulting identifier to be in the range [min_id, max_id] in a similar way as in the algorithm described in Section 8.1.1. Collisions can be recovered by incrementing the 'counter' variable and recomputing F().

The function F() should be a cryptographic hash function like SHA-256 [FIPS-SHS]. Note: MD5 [RFC1321] is considered unacceptable for F() [RFC6151]. CONTEXT is the concatenation of all the elements that define a given context. For example, if this algorithm is expected to produce identifiers that are unique per network interface card (NIC) and SLAAC autoconfiguration prefix, the CONTEXT should be the concatenation of e.g. the interface index and the SLAAC autoconfiguration prefix (please see [RFC7217] for an implementation of this algorithm for the generation of IPv6 IIDs).

The secret should be chosen to be as random as possible (see [RFC4086] for recommendations on choosing secrets).

8.4. Category #4: Uniqueness, monotonically increasing within context (hard failure)

8.4.1. Predictable Linear Identifiers Algorithm

One of the most trivial ways to achieve uniqueness with a low identifier reuse frequency is to produce a linear sequence. This obviously assumes that each identifier will be used for a similar period of time.

For example, the following algorithm has been employed in a number of operating systems for selecting IP fragment IDs, TCP ephemeral ports, etc.

```
/* Initialization at system boot time. Could be random */
next_id = min_id;
id_inc= 1;

/* Identifier selection function */
count = max_id - min_id + 1;

do {
    if (next_id == max_id) {
        next_id = min_id;
    }
    else {
        next_id = next_id + id_inc;
    }

    if (check_suitable_id(next_id))
        return next_id;

    count--;

} while (count > 0);

return ERROR;
```

Note:
check_suitable_id() is a function that checks whether the resulting identifier is acceptable (e.g., whether its in use, etc.).

For obvious reasons, this algorithm results in predicable sequences. If a global counter is used (such as "next_id" in the example above), a node that learns one protocol identifier can also learn or guess values employed by past and future protocol instances. On the other hand, when the value of increments is known (such as "1" in this case), an attacker can sample two values, and learn the number of identifiers that were generated in-between.

Where identifier reuse would lead to a hard failure, one typical approach to generate unique identifiers (while minimizing the security and privacy implications of predictable identifiers) is to obfuscate the resulting protocol IDs by either:

o Replace the global counter with multiple counters (initialized to a random value) o Randomizing the "increments"

Avoiding global counters essentially means that learning one identifier for a given context (e.g., one TCP ephemeral port for a given {src IP, Dst IP, Dst Port}) is of no use for learning or guessing identifiers for a different context (e.g., TCP ephemeral ports that involve other peers). However, this may imply keeping one additional variable/counter per context, which may be prohibitive in some environments. The choice of id_inc has implications on both the security and privacy properties of the resulting identifiers, but also on the corresponding interoperability properties. On one hand, minimizing the increments (as in "id_inc = 1" in our case) generally minimizes the identifier reuse frequency, albeit at increased predictability. On the other hand, if the increments are randomized predictability of the resulting identifiers is reduced, and the information leakage produced by global constant increments is mitigated.

8.4.2. Per-context Counter Algorithm

One possible way to achieve similar (or even lower) identifier reuse frequency while still avoiding predictable sequences would be to employ a per-context counter, as opposed to a global counter. Such an algorithm could be described as follows:

```
/* Initialization at system boot time. Could be random */
id_inc= 1;

/* Identifier selection function */
count = max_id - min_id + 1;

if(lookup_counter(CONTEXT) == ERROR){
     create_counter(CONTEXT);
}

next_id= lookup_counter(CONTEXT);

do {
    if (next_id == max_id) {
        next_id = min_id;
    }
    else {
        next_id = next_id + id_inc;
    }

    if (check_suitable_id(next_id)){
        store_counter(CONTEXT, next_id);
        return next_id;
    }

    count--;

} while (count > 0);

store_counter(CONTEXT, next_id);
return ERROR;
```

NOTE:

lookup_counter() returns the current counter for a given context, or an error condition if such a counter does not exist.

create_counter() creates a counter for a given context, and initializes such counter to a random value.

store_counter() saves (updates) the current counter for a given context.

check_suitable_id() is a function that checks whether the resulting identifier is acceptable (e.g., whether its in use, etc.).

Essentially, whenever a new identifier is to be selected, the algorithm checks whether there there is a counter for the corresponding context. If there is, such counter is incremented to obtain the new identifier, and the new identifier updates the corresponding counter. If there is no counter for such context, a new counter is created an initialized to a random value, and used as the new identifier.

This algorithm produces a per-context counter, which results in one linear function for each context. Since the origin of each "line" is a random value, the resulting values are unknown to an off-path attacker.

This algorithm has the following drawbacks:

o If, as a result of resource management, the counter for a given context must be removed, the last identifier value used for that context will be lost. Thus, if subsequently an identifier needs to be generated for such context, that counter will need to be recreated and reinitialized to random value, thus possibly leading to reuse/collistion of identifiers.

o If the identifiers are predictable by the destination system (e.g., the destination host represents the context), a vulnerable host might possibly leak to third parties the identifiers used by other hosts to send traffic to it (i.e., a vulnerable Host B could leak to Host C the identifier values that Host A is using to send packets to Host B). Appendix A of [RFC7739] describes one possible scenario for such leakage in detail.

8.4.3. Simple Hash-Based Algorithm

The goal of this algorithm is to produce monotonically-increasing sequences, with a randomized initial value, for each given context. For example, if the identifiers being generated must be unique for each {src IP, dst IP} set, then each possible combination of {src IP, dst IP} should have a corresponding "next_id" value.

Keeping one value for each possible "context" may in many cases be considered too onerous in terms of memory requirements. As a workaround, the following algorithm employs a calculated technique (as opposed to keeping state in memory) to maintain the random offset for each possible context.

In the following algorithm, the function F() provides (statelessly) a random offset for each given context.

```
/* Initialization at system boot time. Could be random. */
counter = 0;

/* Protocol ID selection function  */
id_range = max_id - min_id + 1;
offset = F(CONTEXT, secret_key);
count = id_range;

do {
    next_id = min_id +
           (counter + offset) % id_range;

    counter++;

    if(check_suitable_id(next_id))
        return next_id;

    count--;

} while (count > 0);

return ERROR;
```

The function F() provides a "per-CONTEXT" fixed offset within the identifier space. Both the 'offset' and 'counter' variables may take any value within the storage type range since we are restricting the resulting identifier to be in the range [min_id, max_id] in a similar way as in the algorithm described in Section 8.1.1. This allows us to simply increment the 'counter' variable and rely on the unsigned integer to wrap around.

The function F() should be a cryptographic hash function like SHA-256 [FIPS-SHS]. Note: MD5 [RFC1321] is considered unacceptable for F() [RFC6151]. CONTEXT is the concatenation of all the elements that define a given context. For example, if this algorithm is expected to produce identifiers that are monotonically-increasing for each set (Source IP Address, Destination IP Address), the CONTEXT should be the concatenation of these two values.

The secret should be chosen to be as random as possible (see [RFC4086] for recommendations on choosing secrets).

It should be noted that, since this algorithm uses a global counter ("counter") for selecting identifiers, if an attacker could, e.g., force a client to periodically establish a new TCP connection to an attacker-controlled machine (or through an attacker-observable routing path), the attacker could substract consecutive source port values to obtain the number of outgoing TCP connections established globally by the target host within that time period (up to wrap-around issues and five-tuple collisions, of course).

8.4.4. Double-Hash Algorithm

A trade-off between maintaining a single global 'counter' variable and maintaining 2**N 'counter' variables (where N is the width of the result of F()) could be achieved as follows. The system would keep an array of TABLE_LENGTH integers, which would provide a separation of the increment of the 'counter' variable. This improvement could be incorporated into the algorithm from Section 8.4.3 as follows:

```
/* Initialization at system boot time */
for(i = 0; i < TABLE_LENGTH; i++)
    table[i] = random();

id_inc = 1;


/* Protocol ID selection function */
id_range = max_id - min_id + 1;
offset = F(CONTEXT, secret_key1);
index = G(CONTEXT, secret_key2);
count = id_range;

do {
    next_id = min_id + (offset + table[index]) % id_range;
    table[index] = table[index] + id_inc;

    if(check_suitable_id(next_id))
        return next_id;

   count--;

} while (count > 0);

return ERROR;
```

'table[]' could be initialized with random values, as indicated by the initialization code in pseudo-code above. The function G() should be a cryptographic hash function. It should use the same CONTEXT as F(), and a secret key value to compute a value between 0 and (TABLE_LENGTH-1). Alternatively, G() could take an "offset" as input, and perform the exclusive-or (XOR) operation between all the bytes in 'offset'.

The array 'table[]' assures that successive identifiers for a given context will be monotonically-increasing. However, the increments space is separated into TABLE_LENGTH different spaces, and thus identifier reuse frequency will be (probabilistically) lower than that of the algorithm in Section 8.4.3. That is, the generation of identifier for one given context will not necessarily result in increments in the identifiers for other contexts.

It is interesting to note that the size of 'table[]' does not limit the number of different identifier sequences, but rather separates the *increments* into TABLE_LENGTH different spaces. The identifier sequence will result from adding the corresponding entry of 'table[]' to the variable 'offset', which selects the actual identifier sequence (as in the algorithm from Section 8.4.3).

An attacker can perform traffic analysis for any "increment space" into which the attacker has "visibility" -- namely, the attacker can force a node to generate identifiers where G(offset) identifies the target "increment space". However, the attacker's ability to perform traffic analysis is very reduced when compared to the predictable linear identifiers (described in Section 8.4.1) and the hash-based identifiers (described in Section 8.4.3). Additionally, an implementation can further limit the attacker's ability to perform traffic analysis by further separating the increment space (that is, using a larger value for TABLE_LENGTH) and/or by randomizing the increments.

8.4.5. Random-Increments Algorithm

This algorithm offers a middle ground between the algorithms that select ephemeral ports randomly (such as those described in Section 8.1.1 and Section 8.1.2), and those that offer obfuscation but no randomization (such as those described in Section 8.4.3 and Section 8.4.4).

```
/* Initialization code at system boot time. */
next_id = random();        /* Initialization value */
id_inc = 500;        /* Determines the trade-off */

/* Identifier selection function */
id_range = max_id - min_id + 1;

count = id_range;

do {
    /* Random increment */
    next_id = next_id + (random() % id_inc) + 1;

    /* Keep the identifier within acceptable range */
    next_id = min_id + (next_id % id_range);

    if(check_suitable_id(next_id))
       return next_id;

    count--;
} while (count > 0);

return ERROR;
```

This algorithm aims at producing a monotonically increasing sequence of identifiers, while avoiding the use of fixed increments, which would lead to trivially predictable sequences. The value "id_inc" allows for direct control of the trade-off between the level of obfuscation and the ID reuse frequency. The smaller the value of "id_inc", the more similar this algorithm is to a predicable, global monotonically-increasing ID generation algorithm. The larger the value of "id_inc", the more similar this algorithm is to the algorithm described in Section 8.1.1 of this document.

When the identifiers wrap, there is the risk of collisions of identifiers (i.e., identifier reuse). Therefore, "id_inc" should be selected according to the following criteria:

- It should maximize the wrapping time of the identifier space.
- It should minimize identifier reuse frequency.
- It should maximize obfuscation.

Clearly, these are competing goals, and the decision of which value of "id_inc" to use is a trade-off. Therefore, the value of "id_inc" should be configurable so that system administrators can make the trade-off for themselves.

9. Common Vulnerabilities Associated with Identifiers

This section analyzes common vulnerabilities associated with the generation of identifiers for each of the categories identified in Section 7.

9.1. Category #1: Uniqueness (soft failure)

Possible vulnerabilities associated with identifiers of this category are:

- o Use of trivial algorithms (e.g. global counters) that generate predictable identifiers
- o Use of flawed PRNGs.

Since the only interoperability requirement for these identifiers is uniqueness, the obvious approach to generate them is to employ a PRNG. An implementer should consult [RFC4086] regarding randomness requirements for security, and consult relevant documentation when employing a PRNG provided by the underlying system.

Use algorithms other than PRNGs for generating identifiers of this category is discouraged.

9.2. Category #2: Uniqueness (hard failure)

As noted in Section 8.2 this category typically employs the same algorithms as Category #4, since a monotonically-increasing sequence tends to minimize the identifier reuse frequency. Therefore, the vulnerability analysis of Section 9.4 applies to this case.

9.3. Category #3: Uniqueness, constant within context (soft failure)

There are two main vulnerabilities that may be associated with identifiers of this category:

1. Use algorithms or sources that result in predictable identifiers
2. Employing the same identifier across contexts in which constantcy is not required

At times, an implementation or specification may be tempted to employ a source for the identifier which is known to provide unique values. However, while unique, the associated identifiers may have other properties such as being predictable or leaking information about the node in question. For example, as noted in [RFC7721], embedding link-layer addresses for generating IPv6 IIDs not only results in predictable values, but also leaks information about the manufacturer of the network interface card.

On the other hand, using an identifier across contexts where constantcy is not required can be leveraged for correlation of activities. On of the most trivial examples of this is the use of IPv6 IIDs that are constant across networks (such as IIDs that embed the underlying link-layer address).

9.4. Category #4: Uniqueness, monotonically increasing within context (hard failure)

A simple way to generalize algorithms employed for generating identifiers of Category #4 would be as follows:

```
    /* Identifier selection function */
    count = max_id - min_id + 1;

    do {
                linear(CONTEXT)= linear(CONTEXT) + increment();
        next_id= offset(CONTEXT) + linear(CONTEXT);

        if(check_suitable_id(next_id))
           return next_id;

        count--;
    } while (count > 0);

    return ERROR;
```

Essentially, an identifier (next_id) is generated by adding a linear function (linear()) to an offset value, which is unknown to the attacker, and constant for given context.

The following aspects of the algorithm should be considered:

- o For the most part, it is the offset() function that results in identifiers that are unpredictable by an off-path attacker. While the resulting sequence will be monotonically-increasing, the use of an offset value that is unknown to the attacker makes the resulting values unknown to the attacker.

- o The most straightforward "stateless" implementation of offset would be that in which offset() is the result of a cryptographically-secure hash-function that takes the values that identify the context and a "secret" (not shown in the figure above) as arguments.

- Another possible (but stateful) approach would be to simply generate a random offset and store it in memory, and then look-up the corresponding context when a new identifier is to be selected. The algorithm in Section 8.4.2 is essentially an implementation of this type.

- The linear function is incremented according to increment(). In the most trivial case increment() could always return the constant "1". But it could also possibly return small integers such the increments are randomized.

Considering the generic algorithm illustrated above we can identify the following possible vulnerabilities:

- If the offset value spans more than the necessary context, identifiers could be unnecessarily predictable by other parties, since the offset value would be unnecessarily leaked to them. For example, an implementation that means to produce a per-destination counter but replaces offset() with a constant number (i.e., employs a global counter), will unnecessarily result in predictable identifiers.

- The function linear() could be seen as representing the number of identifiers that have so far been generated for a given context. If linear() spans more than the necessary context, the "increments" could be leaked to other parties, thus disclosing information about the number of identifiers that have so far been generated. For example, an implementation in which linear() is implemented as a single global counter will unnecessarily leak information the number of identifiers that have been produced.

- increment() determines how the linear() is incremented for each identifier that is selected. In the most trivial case, increment() will return the integer "1". However, an implementation may have increment() return a "small" integer value such that even if the current value employed by the generator is guessed (see Appendix A of [RFC7739]), the exact next identifier to be selected will be slightly harder to identify.

10. Security and Privacy Requirements for Identifiers

Protocol specifications that specify identifiers should:

1. Clearly specify the interoperability requirements for selecting the aforementioned identifiers.

2. Provide a security and privacy analysis of the aforementioned identifiers.

3. Recommend an algorithm for generating the aforementioned identifiers that mitigates security and privacy issues, such as those discussed in Section 9.

11. IANA Considerations

There are no IANA registries within this document. The RFC-Editor can remove this section before publication of this document as an RFC.

12. Security Considerations

The entire document is about the security and privacy implications of identifiers.

13. Acknowledgements

The authors would like to thank (in alphabetical order) Steven Bellovin, Joseph Lorenzo Hall, Gre Norcie, and Martin Thomson, for providing valuable comments on earlier versions of this document.

The authors would like to thank Diego Armando Maradona for his magic and inspiration.

14. References

14.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[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>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

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

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/info/rfc4862>.

[RFC5722] Krishnan, S., "Handling of Overlapping IPv6 Fragments", RFC 5722, DOI 10.17487/RFC5722, December 2009, <https://www.rfc-editor.org/info/rfc5722>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February 2012, <https://www.rfc-editor.org/info/rfc6528>.

[RFC7098] Carpenter, B., Jiang, S., and W. Tarreau, "Using the IPv6 Flow Label for Load Balancing in Server Farms", RFC 7098, DOI 10.17487/RFC7098, January 2014, <https://www.rfc-editor.org/info/rfc7098>.

[RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, DOI 10.17487/RFC7217, April 2014, <https://www.rfc-editor.org/info/rfc7217>.

14.2. Informative References

[Bellovin1989]
Bellovin, S., "Security Problems in the TCP/IP Protocol Suite", Computer Communications Review, vol. 19, no. 2, pp. 32-48, 1989, <https://www.cs.columbia.edu/~smb/papers/ipext.pdf>.

[Bellovin2002]
Bellovin, S., "A Technique for Counting NATted Hosts", IMW'02 Nov. 6-8, 2002, Marseille, France, 2002.

[CERT2001]
CERT, "CERT Advisory CA-2001-09: Statistical Weaknesses in TCP/IP Initial Sequence Numbers", 2001, <http://www.cert.org/advisories/CA-2001-09.html>.

[CPNI-TCP]
Gont, F., "Security Assessment of the Transmission Control Protocol (TCP)", United Kingdom's Centre for the Protection of National Infrastructure (CPNI) Technical Report, 2009, <http://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf>.

[FIPS-SHS]
FIPS, "Secure Hash Standard (SHS)", Federal Information Processing Standards Publication 180-4, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

[Fyodor2004]
Fyodor, "Idle scanning and related IP ID games", 2004, <http://www.insecure.org/nmap/idlescan.html>.

[Gont2011]
Gont, F., "Hacking IPv6 Networks (training course)", Hack In Paris 2011 Conference Paris, France, June 2011.

[Gont2012]
Gont, F., "Recent Advances in IPv6 Security", BSDCan 2012 Conference Ottawa, Canada. May 11-12, 2012, May 2012.

[I-D.eddy-rfc793bis-04]
Eddy, W., "Transmission Control Protocol Specification", draft-eddy-rfc793bis-04 (work in progress), August 2014.

[I-D.gont-6man-flowlabel-security]
Gont, F., "Security Assessment of the IPv6 Flow Label", draft-gont-6man-flowlabel-security-03 (work in progress), March 2012.

[I-D.ietf-6man-default-iids]
Gont, F., Cooper, A., Thaler, D., and S. LIU, "Recommendation on Stable IPv6 Interface Identifiers", draft-ietf-6man-default-iids-16 (work in progress), September 2016.

[I-D.ietf-6man-predictable-fragment-id-08]
Gont, F., "Security Implications of Predictable Fragment Identification Values", draft-ietf-6man-predictable-fragment-id-08 (work in progress), June 2015.

[Joncheray1995]
Joncheray, L., "A Simple Active Attack Against TCP", Proc. Fifth Usenix UNIX Security Symposium, 1995.

[Klein2007]
Klein, A., "OpenBSD DNS Cache Poisoning and Multiple O/S Predictable IP ID Vulnerability", 2007, <http://www.trusteer.com/files/OpenBSD_DNS_Cache_Poisoning_and_Multiple_OS_Predictable_IP_ID_Vulnerability.pdf>.

[Morris1985]
Morris, R., "A Weakness in the 4.2BSD UNIX TCP/IP Software", CSTR 117, AT&T Bell Laboratories, Murray Hill, NJ, 1985, <https://pdos.csail.mit.edu/~rtm/papers/117.pdf>.

[RedHat2011]
RedHat, "RedHat Security Advisory RHSA-2011:1465-1: Important: kernel security and bug fix update", 2011, <https://rhn.redhat.com/errata/RHSA-2011-1465.html>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

[RFC1948] Bellovin, S., "Defending Against Sequence Number Attacks", RFC 1948, DOI 10.17487/RFC1948, May 1996, <https://www.rfc-editor.org/info/rfc1948>.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <https://www.rfc-editor.org/info/rfc4963>.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <https://www.rfc-editor.org/info/rfc5927>.

[RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <https://www.rfc-editor.org/info/rfc6056>.

[RFC7528] Higgs, P. and J. Piesing, "A Uniform Resource Name (URN) Namespace for the Hybrid Broadcast Broadband TV (HbbTV) Association", RFC 7528, DOI 10.17487/RFC7528, April 2015, <https://www.rfc-editor.org/info/rfc7528>.

[RFC7707] Gont, F. and T. Chown, "Network Reconnaissance in IPv6 Networks", RFC 7707, DOI 10.17487/RFC7707, March 2016, <https://www.rfc-editor.org/info/rfc7707>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC7739] Gont, F., "Security Implications of Predictable Fragment Identification Values", RFC 7739, DOI 10.17487/RFC7739, February 2016, <https://www.rfc-editor.org/info/rfc7739>.

[Sanfilippo1998a]
Sanfilippo, S., "about the ip header id", Post to Bugtraq mailing-list, Mon Dec 14 1998, <http://seclists.org/bugtraq/1998/Dec/48>.

[Sanfilippo1998b]
Sanfilippo, S., "Idle scan", Post to Bugtraq mailing-list, 1998, <http://www.kyuzz.org/antirez/papers/dumbscan.html>.

[Sanfilippo1999]
Sanfilippo, S., "more ip id", Post to Bugtraq mailing-list, 1999, <http://www.kyuzz.org/antirez/papers/moreipid.html>.

[Shimomura1995]
Shimomura, T., "Technical details of the attack described by Markoff in NYT", Message posted in USENET's comp.security.misc newsgroup Message-ID: <3g5gkl$5j1@ariel.sdsc.edu>, 1995, <http://www.gont.com.ar/docs/post-shimomura-usenet.txt>.

[Silbersack2005]
Silbersack, M., "Improving TCP/IP security through randomization without sacrificing interoperability", EuroBSDCon 2005 Conference, 2005, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.4542&rep=rep1&type=pdf>.

[SUSE2011]
SUSE, "SUSE Security Announcement: Linux kernel security update (SUSE-SA:2011:046)", 2011, <http://lists.opensuse.org/opensuse-security-announce/2011-12/msg00011.html>.

[Ubuntu2011]
Ubuntu, "Ubuntu: USN-1253-1: Linux kernel vulnerabilities", 2011, <http://www.ubuntu.com/usn/usn-1253-1/>.

[USCERT2001]
US-CERT, "US-CERT Vulnerability Note VU#498440: Multiple TCP/IP implementations may use statistically predictable initial sequence numbers", 2001, <http://www.kb.cert.org/vuls/id/498440>.

[Zalewski2001]
Zalewski, M., "Strange Attractors and TCP/IP Sequence Number Analysis", 2001, <http://lcamtuf.coredump.cx/oldtcp/tcpseq.html>.

[Zalewski2002]
Zalewski, M., "Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later", 2001, <http://lcamtuf.coredump.cx/newtcp/>.

[Zalewski2003]
Zalewski, M., "A new TCP/IP blind data injection technique?", 2003, <http://lcamtuf.coredump.cx/ipfrag.txt>.

Authors' Addresses

Fernando Gont
SI6 Networks / UTN-FRH
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
Email: fgont@si6networks.com
URI: http://www.si6networks.com

Ivan Arce
Quarkslab

Email: iarce@quarkslab.com
URI: https://www.quarkslab.com

Network Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: January 9, 2017

R. Salz
Akamai Technologies

July 08, 2016

No MTI Crypto without Public Review
draft-rsalz-drbg-speck-wap-wep-01

Abstract

Cryptography is becoming more important to the IETF and its protocols, and more IETF protocols are using, or looking at, cryptography to increase privacy on the Internet [RFC7258].

This document specifies a proposed best practice for any mechanism (or data format) that uses cryptography; namely, that RFCs cannot specify an algorithm as mandatory-to-implement (MTI) unless that algorithm has had reasonable public review. This document also "sketches out" a rough definition around what such a review would look like.

Status of This Memo

Copyright Notice

Table of Contents

1. Terminology 2
2. Introduction 2
3. Why is Cryptography Hard? 3
4. Things to avoid 4
5. Why limit to MTI? 4
6. How to Do it Right 5
6.1. Public Review 6
7. Acknowledgements 6
8. References 6
8.1. Normative References 6
8.2. Informative References 6
Author's Address 7

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 RFC 2119 [RFC2119].

The term mandatory to implement (MTI) is used in this document to describe a cryptographic algorithm that is listed as a MUST in an RFC.

The term "snake oil" is used as a pejorative for something which appears to do its job acceptably, but actually does not; see https://en.wikipedia.org/wiki/Snake_oil_%28cryptography%29 . It is a goal of the IETF that we never be misled into being, or mistakenly taken as, snake oil salesman.

2. Introduction

Cryptography is becoming more important to the IETF and its protocols, and more IETF protocols are using, or looking at, cryptography to increase privacy on the Internet [RFC7258].

This document specifies a proposed best practice for any protocol (or data format) that uses cryptography. Namely, that such RFCs cannot specify an algorithm as mandatory-to-implement (MTI) unless that algorithm has had reasonable public review. This document also "sketches out" a rough definition around what such a review would look like.

3. Why is Cryptography Hard?

Cryptography is hard because it is not like traditional IETF protocol deployments. In this classic situation, if one party implements a protocol incorrectly, it usually becomes obvious as interoperability suffers or completely fails. But with cryptography, one party can have implementation defects, or known exploitable weaknesses, that expose the entire communication stream to an attacker. Open source and code reviews are not a panacea here, but using only widely-accepted cryptographic mechanisms (e.g., avoiding facilities like https://en.wikipedia.org/wiki/Dual_EC_DRBG) will reduce the attack surface.

Cryptography is hard because subtle design characteristics can have disastrous consequences. For example, the US Digital Signature Algorithm requires the random nonce to be protected and never re-used. If those requirements are not met, the private key can be leaked.

Cryptography is hard because adversaries design new attacks and refine existing ones. Attacks get better over time; they never get worse. For example, it is now de riguer to protect against CPU timing attacks, even when the device is only viewable over a network. A recent paper [acoustic] (XXX reference) can identify a private key if your smartphone is just laid next to an innocuous charging device. We understand power differential attacks, timing attacks, and perhaps cache line attacks; we now have to think about RFI emissions from our phone.

Cryptography is hard because the order of operations can matter. It is not intuitively obvious to most developers, which should come first among signing, compression and encryption. This issues was first raised in Spring of 2001 [davis] but was only addressed in TLS by [RFC7366] more than a dozen years later.

Getting the cryptography right is important because the Internet, and therefore the work of the IETF, has become a tempting target for all types of attackers, from individual "script kiddies," through criminal commercial botnet and phishing ventures, up to national-scale adversaries operating on behalf of their nation-state.

4. Things to avoid

"Sunlight is said to be the best of disinfectants; electric light the most efficient policeman." - Louis Brandeis, _Other People's Money and How Bankers Use it,_ first published as a set of articles in _Harper's Weekly_ in 1914.

Cryptography that is developed in private, such as among an industry consortium is a bad idea. Notable examples of this include:

- A5/1 and A5/2 for GSM-based mobile phones.

- WEP and WPA for WiFi access.

- SSLv2, while published, was developed by a private group at an Internet startup. It had security flaws that had global effects decades later, see https://drownattack.com/ .

It is hard to get good public review of patented cryptography, unless there is a strongly compelling need. For example, decades ago RSA was the only practial public-key mechanism available and it was therefore studied pretty extensively.

Part of the concern about patented cryptography is that the patent-holder has every incentive to provide that their system is good, while the rest of the world generally has little interest in proving that their commercial venture is bad. Examples of this include:

- Algebraic Eraser, prior to its presentation at IETF-xx, received little public interest.

- There is not a great deal of study about NTRU.

Both of these items are "lattice cryptography" and that might also be a reason for lack of review; the field might not have much interest yet.

- XXX STILL MORE NEEDED

5. Why limit to MTI?

There is an argument that any new RFC not classified as "historical" should not specify or recommend insufficiently-reviewed cryptography, whether it MTI or not. This document limits itself to MTI for a couple of reasons.

- o Informational RFCs often document how to interoperate with other systems, and this is useful. As examples of this, see the Internet-Drfats on scrypt and [RFC7693].

- o Putting insufficiently-reviewed algorithms into an RFC can be one way to spur interest in getting more reviews. This MUST NOT be the primary motivation for inclusion, but it can be a useful side-effect, and might lead to future "promotion" to MTI. Note that waiting through draft and last-call state, then claiming "nobody broke it" MUST NOT be used as the rationale; this is using the IETF to host a "proof by contest."

- o Drawing a strict boundary just around MTI is a tractable problem. Drawing a similar boundary around all potential IETF uses of cryptography is bound to have mistakes and errors, any one of which can has the potential to make the IETF look bad, if not incompetent.

- o Requiring MTI to have public review also pressures everyone to conform and raise the bar. Imagine a hypothetical national security body that has a new cryptographic algorithm, Military Top-secret Encryption, or MITE. If MITE is not MTI, then that government might be hard-pressed to get it accepted into off-the-shell offerings. If it is MTI without sufficient review, then they have good reason to keep flaws in existing cryptography private. To avoid both situations, the that government should work to get MITE as an MTI, and would now have the burden to make sure it receives sufficient analysis.

6. How to Do it Right

Cryptographic agility, [RFC7696], is probably a MUST. While it has its detractors, there are no known (to the author) practical considerations to evolving a deployed based to stronger crypto, while still maintaining interoperability with existing entities. This requires being able to make informed choices about when to use old weak crypto, and when to use the "latest and greatest," and while not much software, and essentially no end-users, are capable of making that choice, it seems sadly the best we can do.

NIST is an important reference for crypto algorithms. Yes, they have made mistakes (DUAL_EC_DRBG), but so has the IETF (opaque-prf) in the same area. But they have run respected international contests and their output receives heavy scrutiny.

The second consideration is to avoid temptation and premature optimization. Do not adopt an algorithm just because it seems "small and fast" or comes from "someone I respect."

6.1. Public Review

What constitutes sufficient public review? It is hard to say. This section attempts to provide some guidelines.

An open competition, such as those that led to AES (XXX ref) and SHA-3 (XXX ref) seem to be good, even when they come from sources that are under widespread suspicion, like the US Government. These efforts, like the Password Hashing Competition https://password-hashing.net/ , had wide international participation and analysis by many noted exports.

Papers presented in the various Crypto conferences (XXX need list) are good. Same for various Usenix workshops.

Proof by contest - "Nobody's Claimed my $200 reward" - are generally useless, for a number of reasons. They tend to be promoted by amateur cryptographers as a way to get attention, and if someone actually looks at them they are always cracked. Numerical analysis is a better approach, albeit much harder work. Contests designed to show the amount of "brute-force" work needed, such as the old RSA factoring challenges, can be useful. But they do not show, for example, if the cryptography under test is fundamentally flawed or not.

Public review is also a natural fit for the IETF, which takes "rough consensus and running code" as an axiom. Theory reduced to practice is much easier, and much less of a limited academic exercise, to review.

7. Acknowledgements

Thanks to Stephen Farrell for instigating this.

8. References

8.1. 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>.

8.2. Informative References

[acoustic]
Technion and Tel Aviv University, Weizmann Institute of Science, and Tel Aviv University, "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis", December 2013, <http://www.tau.ac.il/˜tromer/papers/acoustic-20131218.pdf>.

[davis] "Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML.", Usenix Proc. Usenix Tech. Conf., June 2001, <http://world.std.com/˜dtd/sign_encrypt/sign_encrypt7.PDF>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7366] Gutmann, P., "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7366, DOI 10.17487/RFC7366, September 2014, <http://www.rfc-editor.org/info/rfc7366>.

[RFC7693] Saarinen, M-J., Ed. and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)", RFC 7693, DOI 10.17487/RFC7693, November 2015, <http://www.rfc-editor.org/info/rfc7693>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <http://www.rfc-editor.org/info/rfc7696>.

Author's Address

Rich Salz
Akamai Technologies

Email: rsalz@akamai.com

Network Working Group J. Vcelak
Internet-Draft CZ.NIC
Intended status: Standards Track S. Goldberg
Expires: July 2, 2019 Boston University
D. Papadopoulos
HKUST
S. Huque
Salesforce
D. Lawrence
Dyn
December 29, 2018

NSEC5, DNSSEC Authenticated Denial of Existence
draft-vcelak-nsec5-08

Abstract

The Domain Name System Security Extensions (DNSSEC) introduced two resource records (RR) for authenticated denial of existence: the NSEC RR and the NSEC3 RR. This document introduces NSEC5 as an alternative mechanism for DNSSEC authenticated denial of existence. NSEC5 uses verifiable random functions (VRFs) to prevent offline enumeration of zone contents. NSEC5 also protects the integrity of the zone contents even if an adversary compromises one of the authoritative servers for the zone. Integrity is preserved because NSEC5 does not require private zone-signing keys to be present on all authoritative servers for the zone, in contrast to DNSSEC online signing schemes like NSEC3 White Lies.

Ed note

Text inside square brackets ([]) is additional background information, answers to frequently asked questions, general musings, etc. They will be removed before publication. This document is being collaborated on in GitHub at <https://github.com/fcelda/nsec5-draft>. The most recent version of the document, open issues, etc should all be available there. The authors gratefully accept pull requests.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Rationale 3
  1.2. Requirements 6
  1.3. Terminology 6
2. Backward Compatibility 6
3. How NSEC5 Works 7
4. NSEC5 Algorithms 8
5. The NSEC5KEY Resource Record 9
  5.1. NSEC5KEY RDATA Wire Format 9
  5.2. NSEC5KEY RDATA Presentation Format 9
6. The NSEC5 Resource Record 9
  6.1. NSEC5 RDATA Wire Format 10
  6.2. NSEC5 Flags Field 10
  6.3. NSEC5 RDATA Presentation Format 11
7. The NSEC5PROOF Resource Record 11
  7.1. NSEC5PROOF RDATA Wire Format 11
  7.2. NSEC5PROOF RDATA Presentation Format 12
8. Types of Authenticated Denial of Existence with NSEC5 12
  8.1. Name Error Responses 12
  8.2. No Data Responses 13
    8.2.1. No Data Response, Opt-Out Not In Effect 13

8.2.2. No Data Response, Opt-Out In Effect 14
8.3. Wildcard Responses 14
8.4. Wildcard No Data Responses 14
9. Authoritative Server Considerations 15
9.1. Zone Signing 15
9.1.1. Precomputing Closest Provable Encloser Proofs 16
9.2. Zone Serving 17
9.3. NSEC5KEY Rollover Mechanism 18
9.4. Secondary Servers 18
9.5. Zones Using Unknown NSEC5 Algorithms 18
9.6. Dynamic Updates 18
10. Resolver Considerations 19
11. Validator Considerations 19
11.1. Validating Responses 19
11.2. Validating Referrals to Unsigned Subzones 20
11.3. Responses With Unknown NSEC5 Algorithms 20
12. Special Considerations 20
12.1. Transition Mechanism 20
12.2. Private NSEC5 keys 20
12.3. Domain Name Length Restrictions 21
13. Implementation Status 21
14. Performance Considerations 21
15. Security Considerations 21
15.1. Zone Enumeration Attacks 21
15.2. Compromise of the Private NSEC5 Key 22
15.3. Key Length Considerations 22
15.4. NSEC5 Hash Collisions 22
16. IANA Considerations 23
17. Contributors 23
18. References 24
18.1. Normative References 24
18.2. Informative References 25
Appendix A. Examples 27
A.1. Name Error Example 27
A.2. No Data Example 29
A.3. Delegation to an Unsigned Zone in an Opt-Out span Example 30
A.4. Wildcard Example 31
A.5. Wildcard No Data Example 32
Appendix B. Change Log 33
Authors' Addresses 34

1. Introduction

1.1. Rationale

NSEC5 provides an alternative mechanism for authenticated denial of existence for the DNS Security Extensions (DNSSEC). NSEC5 has two key security properties. First, NSEC5 protects the integrity of the zone contents even if an adversary compromises one of the authoritative servers for the zone. Second, NSEC5 prevents offline zone enumeration, where an adversary makes a small number of online DNS queries and then processes them offline in order to learn all of the names in a zone. Zone enumeration can be used to identify routers, servers or other "things" that could then be targeted in more complex attacks. An enumerated zone can also be a source of probable email addresses for spam, or as a "key for multiple WHOIS queries to reveal registrant data that many registries may have legal obligations to protect" [RFC5155].

All other DNSSEC mechanisms for authenticated denial of existence either fail to preserve integrity against a compromised server, or fail to prevent offline zone enumeration.

When offline signing with NSEC is used [RFC4034], an NSEC chain of all existing domain names in the zone is constructed and signed offline. The chain is made of resource records (RRs), where each RR represents two consecutive domain names in canonical order present in the zone. The authoritative server proves the non-existence of a name by presenting a signed NSEC RR which covers the name. Because the authoritative server does not need not to know the private zone-signing key, the integrity of the zone is protected even if the server is compromised. However, the NSEC chain allows for easy zone enumeration: N queries to the server suffice to learn all N names in the zone (see e.g., [nmap-nsec-enum], [nsec3map], and [ldns-walk]).

When offline signing with NSEC3 is used [RFC5155], the original names in the NSEC chain are replaced by their cryptographic hashes. Offline signing ensures that NSEC3 preserves integrity even if an authoritative server is compromised. However, offline zone enumeration is still possible with NSEC3 (see e.g., [nsec3walker], [nsec3gpu]), and is part of standard network reconnaissance tools (e.g., [nmap-nsec3-enum], [nsec3map]).

When online signing is used, the authoritative server holds the private zone-signing key and uses this key to synthesize NSEC or NSEC3 responses on the fly (e.g. NSEC3 White Lies (NSEC3-WL) or Minimally-Covering NSEC, both described in [RFC7129]). Because the synthesized response only contains information about the queried name (but not about any other name in the zone), offline zone enumeration is not possible. However, because the authoritative server holds the private zone-signing key, integrity is lost if the authoritative server is compromised.

Scheme	Integrity vs network attacks?	Integrity vs compromised auth. server?	Prevents offline zone enumeration?	Online crypto?
Unsigned	NO	NO	YES	NO
NSEC	YES	YES	NO	NO
NSEC3	YES	YES	NO	NO
NSEC3-WL	YES	NO	YES	YES
NSEC5	YES	YES	YES	YES

NSEC5 prevents offline zone enumeration and also protects integrity even if a zone's authoritative server is compromised. To do this, NSEC5 replaces the unkeyed cryptographic hash function used in NSEC3 with a verifiable random function (VRF) [I-D.irtf-cfrg-vrf] [MRV99]. A VRF is the public-key version of a keyed cryptographic hash. Only the holder of the private VRF key can compute the hash, but anyone with public VRF key can verify the correctness of the hash.

The public VRF key is distributed in an NSEC5KEY RR, similar to a DNSKEY RR, and is used to verify NSEC5 hash values. The private VRF key is present on all authoritative servers for the zone, and is used to compute hash values. For every query that elicits a negative response, the authoritative server hashes the query on the fly using the private VRF key, and also returns the corresponding precomputed NSEC5 record(s). In contrast to the online signing approach [RFC7129], the private key that is present on all authoritative servers for NSEC5 cannot be used to modify the zone contents.

Like online signing approaches, NSEC5 requires the authoritative server to perform online public key cryptographic operations for every query eliciting a denying response. This is necessary; [nsec5] proved that online cryptography is required to prevent offline zone enumeration while still protecting the integrity of zone contents against network attacks.

NSEC5 is not intended to replace NSEC or NSEC3. It is an alternative mechanism for authenticated denial of existence. This document specifies NSEC5 based on the VRFs in [I-D.irtf-cfrg-vrf] over the FIPS 186-3 P-256 elliptic curve and over the the Ed25519 elliptic curve. A formal cryptographic proof of security for NSEC5 is in [nsec5ecc].

1.2. Requirements

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].

1.3. Terminology

The reader is assumed to be familiar with the basic DNS and DNSSEC concepts described in [RFC1034], [RFC1035], [RFC4033], [RFC4034], and [RFC4035]; subsequent RFCs that update them in [RFC2136], [RFC2181], [RFC2308], [RFC5155], and [RFC7129]; and DNS terms in [RFC7719].

The reader should also be familiar with verifiable random functions (VRFs) as defined in [I-D.irtf-cfrg-vrf].

The following terminology is used through this document:

Base32hex: The "Base 32 Encoding with Extended Hex Alphabet" as specified in [RFC4648]. The padding characters ("=") are not used in the NSEC5 specification.

Base64: The "Base 64 Encoding" as specified in [RFC4648].

QNAME: The domain name being queried (query name).

Private NSEC5 key: The private key for the verifiable random function (VRF).

Public NSEC5 key: The public key for the VRF.

NSEC5 proof: A VRF proof. The holder of the private NSEC5 key (e.g., authoritative server) can compute the NSEC5 proof for an input domain name. Anyone who knows the public VRF key can verify that the NSEC5 proof corresponds to the input domain name.

NSEC5 hash: A cryptographic digest of an NSEC5 proof. If the NSEC5 proof is known, anyone can compute its corresponding NSEC5 hash.

NSEC5 algorithm: A triple of VRF algorithms that compute an NSEC5 proof (VRF_prove), verify an NSEC5 proof (VRF_verify), and process an NSEC5 proof to obtain its NSEC5 hash (VRF_proof2hash).

2. Backward Compatibility

The specification describes a protocol change that is not backward compatible with [RFC4035] and [RFC5155]. An NSEC5-unaware resolver will fail to validate responses introduced by this document.

To prevent NSEC5-unaware resolvers from attempting to validate the responses, new DNSSEC algorithms identifiers are introduced in Section 16 which alias existing algorithm numbers. The zones signed according to this specification MUST use only these algorithm identifiers, thus NSEC5-unaware resolvers will treat the zone as insecure.

3. How NSEC5 Works

With NSEC5, the original domain name is hashed using a VRF [I-D.irtf-cfrg-vrf] using the following steps:

1. The domain name is processed using a VRF keyed with the private NSEC5 key to obtain the NSEC5 proof. Anyone who knows the public NSEC5 key, normally acquired via an NSEC5KEY RR, can verify that a given NSEC5 proof corresponds to a given domain name.

2. The NSEC5 proof is then processed using a publicly-computable VRF proof2hash function to obtain the NSEC5 hash. The NSEC5 hash can be computed by anyone who knows the input NSEC5 proof.

The NSEC5 hash determines the position of a domain name in an NSEC5 chain.

To sign a zone, the private NSEC5 key is used to compute the NSEC5 hashes for each name in the zone. These NSEC5 hashes are sorted in canonical order [RFC4034], and each consecutive pair forms an NSEC5 RR. Each NSEC5 RR is signed offline using the private zone-signing key. The resulting signed chain of NSEC5 RRs is provided to all authoritative servers for the zone, along with the private NSEC5 key.

To prove non-existence of a particular domain name in response to a query, the server uses the private NSEC5 key to compute the NSEC5 proof and NSEC5 hash corresponding to the queried name. The server then identifies the NSEC5 RR that covers the NSEC5 hash, and responds with this NSEC5 RR and its corresponding RRSIG signature RRset, as well as a synthesized NSEC5PROOF RR that contains the NSEC5 proof corresponding to the queried name.

To validate the response, the client verifies the following items:

- The client uses the public NSEC5 key, normally acquired from the NSEC5KEY RR, to verify that the NSEC5 proof in the NSEC5PROOF RR corresponds to the queried name.

- The client uses the VRF proof2hash function to compute the NSEC5 hash from the NSEC5 proof in the NSEC5PROOF RR. The client verifies that the NSEC5 hash is covered by the NSEC5 RR.

- The client verifies that the NSEC5 RR is validly signed by the RRSIG RRset.

4. NSEC5 Algorithms

The algorithms used for NSEC5 authenticated denial are independent of the algorithms used for DNSSEC signing. An NSEC5 algorithm defines how the NSEC5 proof and the NSEC5 hash are computed and validated.

The NSEC5 proof corresponding to a name is computed using ECVRF_prove(), as specified in [I-D.irtf-cfrg-vrf]. The input to ECVRF_prove() is a public NSEC5 key followed by a private NSEC5 key followed by an RR owner name in [RFC4034] canonical wire format. The output NSEC5 proof is an octet string.

An NSEC5 hash corresponding to a name is computed from its NSEC5 proof using ECVRF_proof2hash(), as specified in [I-D.irtf-cfrg-vrf]. The input to VRF_proof2hash() is an NSEC5 proof as an octet string. The output NSEC5 hash is either an octet string, or INVALID.

An NSEC5 proof for a name is verified using ECVRF_verify(), as specified in [I-D.irtf-cfrg-vrf]. The input is the NSEC5 public key, followed by an NSEC5 proof as an octet string, followed by an RR owner name in [RFC4034] canonical wire format. The output is either VALID or INVALID.

This document defines the EC-P256-SHA256 NSEC5 algorithm as follows:

- The VRF is the ECVRF algorithm using the ECVRF-P256-SHA256 ciphersuite specified in [I-D.irtf-cfrg-vrf].

- The public key format to be used in the NSEC5KEY RR is defined in Section 4 of [RFC6605] and thus is the same as the format used to store ECDSA public keys in DNSKEY RRs.
  [NOTE: This specification does not compress the elliptic curve point used for the public key, but we do compress curve points in every other place we use them. The NSEC5KEY record can be shrunk by 31 additional octets by encoding the public key with point compression.]

This document defines the EC-ED25519-SHA512 NSEC5 algorithm as follows:

- The VRF is the EC-VRF algorithm using the ECVRF-ED25519-SHA512 ciphersuite specified in [I-D.irtf-cfrg-vrf].

- o The public key format to be used in the NSEC5KEY RR is defined in Section 3 of [RFC8080] and thus is the same as the format used to store Ed25519 public keys in DNSKEY RRs.

[NOTE: Could alternatively have the EC-ED25519-SHA512 NSEC5 ciphersuite use the EC-VRF-ED25519-SHA512-ELLIGATOR2 ciphersuite specified in [I-D.irtf-cfrg-vrf].]

5. The NSEC5KEY Resource Record

The NSEC5KEY RR stores a public NSEC5 key. The key allows clients to validate an NSEC5 proof sent by a server.

5.1. NSEC5KEY RDATA Wire Format

The RDATA for the NSEC5KEY RR is as shown below:

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Algorithm   |                   Public Key                  /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Algorithm is a single octet identifying the NSEC5 algorithm.

Public Key is a variable-sized field holding public key material for NSEC5 proof verification.

5.2. NSEC5KEY RDATA Presentation Format

The presentation format of the NSEC5KEY RDATA is as follows:

The Algorithm field is represented as an unsigned decimal integer.

The Public Key field is represented in Base64 encoding. Whitespace is allowed within the Base64 text.

6. The NSEC5 Resource Record

The NSEC5 RR provides authenticated denial of existence for an RRset or domain name. One NSEC5 RR represents one piece of an NSEC5 chain, proving existence of the owner name and non-existence of other domain names in the part of the hashed domain space that is covered until the next owner name hashed in the RDATA.

6.1. NSEC5 RDATA Wire Format

The RDATA for the NSEC5 RR is as shown below:

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Key Tag           |     Flags     |  Next Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Next Hashed Owner Name                    /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                         Type Bit Maps                         /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Key Tag field contains the key tag value of the NSEC5KEY RR that validates the NSEC5 RR, in network byte order. The value is computed from the NSEC5KEY RDATA using the same algorithm used to compute key tag values for DNSKEY RRs. This algorithm is defined in [RFC4034].

The Flags field is a single octet. The meaning of individual bits of the field is defined in Section 6.2.

The Next Length field is an unsigned single octet specifying the length of the Next Hashed Owner Name field in octets.

The Next Hashed Owner Name field is a sequence of binary octets. It contains an NSEC5 hash of the next domain name in the NSEC5 chain.

Type Bit Maps is a variable-sized field encoding RR types present at the original owner name matching the NSEC5 RR. The format of the field is equivalent to the format used in the NSEC3 RR, described in [RFC5155].

6.2. NSEC5 Flags Field

The following one-bit NSEC5 flags are defined:

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|           |W|O|
+-+-+-+-+-+-+-+-+
```

O - Opt-Out flag

W - Wildcard flag

All the other flags are reserved for future use and MUST be zero.

The Opt-Out flag has the same semantics as in NSEC3. The definition and considerations in [RFC5155] are valid, except that NSEC3 is replaced by NSEC5.

The Wildcard flag indicates that a wildcard synthesis is possible at the original domain name level (i.e., there is a wildcard node immediately descending from the immediate ancestor of the original domain name). The purpose of the Wildcard flag is to reduce the maximum number of RRs required for an authenticated denial of existence proof from (at most) three to (at most) two, as originally described in [I-D.gieben-nsec4] Section 7.2.1.

6.3. NSEC5 RDATA Presentation Format

The presentation format of the NSEC5 RDATA is as follows:

The Key Tag field is represented as an unsigned decimal integer.

The Flags field is represented as an unsigned decimal integer.

The Next Length field is not represented.

The Next Hashed Owner Name field is represented as a sequence of case-insensitive Base32hex digits without any whitespace and without padding.

The Type Bit Maps representation is equivalent to the representation used in NSEC3 RR, described in [RFC5155].

7. The NSEC5PROOF Resource Record

The NSEC5PROOF record is not to be included in the zone file. The NSEC5PROOF record contains the NSEC5 proof, proving the position of the owner name in an NSEC5 chain.

7.1. NSEC5PROOF RDATA Wire Format

The RDATA for the NSEC5PROOF RR is shown below:

```
                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Key Tag            |       Owner Name Hash         /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Key Tag field contains the key tag value of the NSEC5KEY RR that validates the NSEC5PROOF RR, in network byte order.

Owner Name Hash is a variable-sized sequence of binary octets encoding the NSEC5 proof of the owner name of the RR.

7.2. NSEC5PROOF RDATA Presentation Format

The presentation format of the NSEC5PROOF RDATA is as follows:

The Key Tag field is represented as an unsigned decimal integer.

The Owner Name Hash is represented in Base64 encoding. Whitespace is allowed within the Base64 text.

8. Types of Authenticated Denial of Existence with NSEC5

This section summarizes all possible types of authenticated denial of existence. For each type the following lists are included:

1. Facts to prove: the minimum amount of information that an authoritative server must provide to a client to assure the client that the response content is valid.

2. Authoritative server proofs: the names for which the NSEC5PROOF RRs are synthesized and added into the response along with the NSEC5 RRs matching or covering each such name. These records together prove the listed facts.

3. Validator checks: the individual checks that a validating server is required to perform on a response. The response content is considered valid only if all of the checks pass.

If NSEC5 is said to match a domain name, the owner name of the NSEC5 RR has to be equivalent to an NSEC5 hash of that domain name. If an NSEC5 RR is said to cover a domain name, the NSEC5 hash of the domain name must sort in canonical order between that NSEC5 RR's Owner Name and Next Hashed Owner Name.

8.1. Name Error Responses

Facts to prove:

Non-existence of the domain name that explictly matches the QNAME.

Non-existence of the wildcard that matches the QNAME.

Authoritative server proofs:

NSEC5PROOF for closest encloser and matching NSEC5 RR.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Closest encloser is in the zone.

The NSEC5 RR matching the closest encloser has its Wildcard flag cleared.

The NSEC5 RR matching the closest encloser does not have NS without SOA in the Type Bit Map.

The NSEC5 RR matching the closest encloser does not have DNAME in the Type Bit Map.

Next closer name is not in the zone.

8.2. No Data Responses

The processing of a No Data response for DS QTYPE differs if the Opt-Out is in effect. For DS QTYPE queries, the validator has two possible checking paths. The correct path can be simply decided by inspecting if the NSEC5 RR in the response matches the QNAME.

Note that the Opt-Out is valid only for DS QTYPE queries.

8.2.1. No Data Response, Opt-Out Not In Effect

Facts to prove:

Existence of an RRset explicitly matching the QNAME.

Non-existence of QTYPE RRset matching the QNAME.

Non-existence of CNAME RRset matching the QNAME.

Authoritative server proofs:

NSEC5PROOF for the QNAME and matching NSEC5 RR.

Validator checks:

QNAME is in the zone.

NSEC5 RR matching the QNAME does not have QTYPE in Type Bit Map.

NSEC5 RR matching the QNAME does not have CNAME in Type Bit Map.

8.2.2. No Data Response, Opt-Out In Effect

Facts to prove:

The delegation is not covered by the NSEC5 chain.

Authoritative server proofs:

NSEC5PROOF for closest provable encloser and matching NSEC5 RR.

Validator checks:

Closest provable encloser is in zone.

Closest provable encloser covers (not matches) the QNAME.

NSEC5 RR matching the closest provable encloser has Opt-Out flag set.

8.3. Wildcard Responses

Facts to prove:

A signed positive response to the QNAME demonstrating the existence of the wildcard (label count in RRSIG is less than in QNAME), and also providing closest encloser name.

Non-existence of the domain name matching the QNAME.

Authoritative server proofs:

A signed positive response for the wildcard expansion of the QNAME.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Next closer name is not in the zone.

8.4. Wildcard No Data Responses

Facts to prove:

The existence of the wildcard at the closest encloser to the QNAME.

Non-existence of both the QTYPE and of the CNAME type that matches QNAME via wildcard expansion.

Authoritative server proofs:

NSEC5PROOF for source of synthesis (i.e., wildcard at closest encloser) and matching NSEC5 RR.

NSEC5PROOF for next closer name and covering NSEC5 RR.

Validator checks:

Closest encloser to the QNAME exists.

NSEC5 RR matching the wildcard label prepended to the closest encloser, and which does not have the bits corresponding to the QTYPE and CNAME types set it the type bitmap.

9. Authoritative Server Considerations

9.1. Zone Signing

Zones using NSEC5 MUST satisfy the same properties as described in Section 7.1 of [RFC5155], with NSEC3 replaced by NSEC5. In addition, the following conditions MUST be satisfied as well:

- o If the original owner name has a wildcard label immediately descending from the original owner name, the corresponding NSEC5 RR MUST have the Wildcard flag set in the Flags field. Otherwise, the flag MUST be cleared.
- o The zone apex MUST include an NSEC5KEY RRset containing a NSEC5 public key allowing verification of the current NSEC5 chain.

The following steps describe one possible method to properly add required NSEC5 related records into a zone. This is not the only such existing method.

1. Select an algorithm for NSEC5 and generate the public and private NSEC5 keys.
2. Add an NSEC5KEY RR into the zone apex containing the public NSEC5 key.
3. For each unique original domain name in the zone and each empty non-terminal, add an NSEC5 RR. If Opt-Out is used, owner names of unsigned delegations MAY be excluded.

A. The owner name of the NSEC5 RR is the NSEC5 hash of the original owner name encoded in Base32hex without padding, prepended as a single label to the zone name.

B. Set the Key Tag field to be the key tag corresponding to the public NSEC5 key.

C. Clear the Flags field. If Opt-Out is being used, set the Opt-Out flag. If there is a wildcard label directly descending from the original domain name, set the Wildcard flag. Note that the wildcard can be an empty non-terminal (i.e., the wildcard synthesis does not take effect and therefore the flag is not to be set).

D. Set the Next Length field to a value determined by the used NSEC5 algorithm. Leave the Next Hashed Owner Name field blank.

E. Set the Type Bit Maps field based on the RRsets present at the original owner name.

4. Sort the set of NSEC5 RRs into canonical order.

5. For each NSEC5 RR, set the Next Hashed Owner Name field by using the owner name of the next NSEC5 RR in the canonical order. If the updated NSEC5 is the last NSEC5 RR in the chain, the owner name of the first NSEC5 RR in the chain is used instead.

The NSEC5KEY and NSEC5 RRs MUST have the same class as the zone SOA RR. Also the NSEC5 RRs SHOULD have the same TTL value as the SOA minimum TTL field.

Notice that a use of Opt-Out is not indicated in the zone. This does not affect the ability of a server to prove insecure delegations. The Opt-Out MAY be part of the zone-signing tool configuration.

9.1.1. Precomputing Closest Provable Encloser Proofs

Per Section 8, the worst-case scenario when answering a negative query with NSEC5 requires the authoritative server to respond with two NSEC5PROOF RRs and two NSEC5 RRs. One pair of NSEC5PROOF and NSEC5 RRs corresponds to the closest provable encloser, and the other pair corresponds to the next closer name. The NSEC5PROOF corresponding to the next closer name MUST be computed on the fly by the authoritative server when responding to the query. However, the NSEC5PROOF corresponding to the closest provable encloser MAY be precomputed and stored as part of zone signing.

Precomputing NSEC5PROOF RRs can halve the number of online cryptographic computations required when responding to a negative query. NSEC5PROOF RRs MAY be precomputed as part of zone signing as follows: For each NSEC5 RR, compute an NSEC5PROOF RR corresponding to the original owner name of the NSEC5 RR. The content of the precomputed NSEC5PROOF record MUST be the same as if the record was computed on the fly when serving the zone. NSEC5PROOF records are not part of the zone and SHOULD be stored separately from the zone file.

9.2. Zone Serving

This specification modifies DNSSEC-enabled DNS responses generated by authoritative servers. In particular, it replaces use of NSEC or NSEC3 RRs in such responses with NSEC5 RRs and adds NSEC5PROOF RRs.

According to the type of a response, an authoritative server MUST include NSEC5 RRs in the response, as defined in Section 8. For each NSEC5 RR in the response, a corresponding RRSIG RRset and an NSEC5PROOF MUST be added as well. The NSEC5PROOF RR has its owner name set to the domain name required according to the description in Section 8. The class and TTL of the NSEC5PROOF RR MUST be the same as the class and TTL value of the corresponding NSEC5 RR. The RDATA payload of the NSEC5PROOF is set according to the description in Section 7.1.

Notice that the NSEC5PROOF owner name can be a wildcard (e.g., source of synthesis proof in wildcard No Data responses). The name also always matches the domain name required for the proof while the NSEC5 RR may only cover (not match) the name in the proof (e.g., closest encloser in Name Error responses).

If NSEC5 is used, an answering server MUST use exactly one NSEC5 chain for one signed zone.

NSEC5 MUST NOT be used in parallel with NSEC, NSEC3, or any other authenticated denial of existence mechanism that allows for enumeration of zone contents, as this would defeat a principal security goal of NSEC5.

Similarly to NSEC3, the owner names of NSEC5 RRs are not represented in the NSEC5 chain and therefore NSEC5 records deny their own existence. The desired behavior caused by this paradox is the same as described in Section 7.2.8 of [RFC5155].

9.3. NSEC5KEY Rollover Mechanism

Replacement of the NSEC5 key implies generating a new NSEC5 chain. The NSEC5KEY rollover mechanism is similar to "Pre-Publish Zone Signing Key Rollover" as specified in [RFC6781]. The NSEC5KEY rollover MUST be performed as a sequence of the following steps:

1. A new public NSEC5 key is added into the NSEC5KEY RRset in the zone apex.

2. The old NSEC5 chain is replaced by a new NSEC5 chain constructed using the new key. This replacement MUST happen as a single atomic operation; the server MUST NOT be responding with RRs from both the new and old chain at the same time.

3. The old public key is removed from the NSEC5KEY RRset in the zone apex.

The minimum delay between steps 1 and 2 MUST be the time it takes for the data to propagate to the authoritative servers, plus the TTL value of the old NSEC5KEY RRset.

The minimum delay between steps 2 and 3 MUST be the time it takes for the data to propagate to the authoritative servers, plus the maximum zone TTL value of any of the data in the previous version of the zone.

9.4. Secondary Servers

This document does not define mechanism to distribute private NSEC5 keys. See Section 15.2 for security considerations for private NSEC5 keys.

9.5. Zones Using Unknown NSEC5 Algorithms

Zones that are signed with an unknown NSEC5 algorithm or with an unavailable private NSEC5 key cannot be effectively served. Such zones SHOULD be rejected when loading and servers SHOULD respond with RCODE=2 (Server failure) when handling queries that would fall under such zones.

9.6. Dynamic Updates

A zone signed using NSEC5 MAY accept dynamic updates [RFC2136]. The changes to the zone MUST be performed in a way that ensures that the zone satisfies the properties specified in Section 9.1 at any time. The process described in [RFC5155] Section 7.5 describes how to handle the issues surrounding the handling of empty non-terminals as well as Opt-Out.

It is RECOMMENDED that the server rejects all updates containing changes to the NSEC5 chain and its related RRSIG RRs, and performs itself any required alternations of the NSEC5 chain induced by the update. Alternatively, the server MUST verify that all the properties are satisfied prior to performing the update atomically.

10. Resolver Considerations

The same considerations as described in Section 9 of [RFC5155] for NSEC3 apply to NSEC5. In addition, as NSEC5 RRs can be validated only with appropriate NSEC5PROOF RRs, the NSEC5PROOF RRs MUST be all together cached and included in responses with NSEC5 RRs.

11. Validator Considerations

11.1. Validating Responses

The validator MUST ignore NSEC5 RRs with Flags field values other than the ones defined in Section 6.2.

The validator MAY treat responses as bogus if the response contains NSEC5 RRs that refer to a different NSEC5KEY.

According to a type of a response, the validator MUST verify all conditions defined in Section 8. Prior to making decision based on the content of NSEC5 RRs in a response, the NSEC5 RRs MUST be validated.

To validate a denial of existence, public NSEC5 keys for the zone are required in addition to DNSSEC public keys. Similarly to DNSKEY RRs, the NSEC5KEY RRs are present at the zone apex.

The NSEC5 RR is validated as follows:

1. Select a correct public NSEC5 key to validate the NSEC5 proof. The Key Tag value of the NSEC5PROOF RR must match with the key tag value computed from the NSEC5KEY RDATA.

2. Validate the NSEC5 proof present in the NSEC5PROOF Owner Name Hash field using the public NSEC5 key. If there are multiple NSEC5KEY RRs matching the key tag, at least one of the keys must validate the NSEC5 proof.

3. Compute the NSEC5 hash value from the NSEC5 proof and check if the response contains NSEC5 RR matching or covering the computed

NSEC5 hash. The TTL values of the NSEC5 and NSEC5PROOF RRs must be the same.

4. Validate the signature on the NSEC5 RR.

If the NSEC5 RR fails to validate, it MUST be ignored. If some of the conditions required for an NSEC5 proof are not satisfied, the response MUST be treated as bogus.

Notice that determining the closest encloser and next closer name in NSEC5 is easier than in NSEC3. NSEC5 and NSEC5PROOF RRs are always present in pairs in responses and the original owner name of the NSEC5 RR matches the owner name of the NSEC5PROOF RR.

11.2. Validating Referrals to Unsigned Subzones

The same considerations as defined in Section 8.9 of [RFC5155] for NSEC3 apply to NSEC5.

11.3. Responses With Unknown NSEC5 Algorithms

A validator MUST ignore NSEC5KEY RRs with unknown NSEC5 algorithms. The practical result of this is that zones signed with unknown algorithms will be considered bogus.

12. Special Considerations

12.1. Transition Mechanism

[TODO: The following information will be covered.]

- Transition to NSEC5 from NSEC/NSEC3

- Transition from NSEC5 to NSEC/NSEC3

- Transition to new NSEC5 algorithms

12.2. Private NSEC5 keys

This document does not define a format to store private NSEC5 keys. Use of a standardized and adopted format is RECOMMENDED.

The private NSEC5 key MAY be shared between multiple zones, however a separate key is RECOMMENDED for each zone.

12.3. Domain Name Length Restrictions

NSEC5 creates additional restrictions on domain name lengths. In particular, zones with names that, when converted into hashed owner names, exceed the 255 octet length limit imposed by [RFC1035] cannot use this specification.

The actual maximum length of a domain name depends on the length of the zone name and the NSEC5 algorithm used.

All NSEC5 algorithms defined in this document use 256-bit NSEC5 hash values. Such a value can be encoded in 52 characters in Base32hex without padding. When constructing the NSEC5 RR owner name, the encoded hash is prepended to the name of the zone as a single label which includes the length field of a single octet. The maximum length of the zone name in wire format using the 256-bit hash is therefore 202 octets (255 - 53).

13. Implementation Status

NSEC5 has been implemented for the Knot DNS authoritative server (version 1.6.4) and the Unbound recursive server (version 1.5.9). The implementations did not introduce additional library dependencies; all cryptographic primitives are already present in OpenSSL v1.0.2j, which is used by both implementations. The implementations support the full spectrum of negative responses, (i.e., NXDOMAIN, NODATA, Wildcard, Wildcard NODATA, and unsigned delegation) using the EC-P256-SHA256 algorithm. The code is deliberately modular, so that the EC-ED25519-SHA256 algorithm could be implemented by using the Ed25519 elliptic curve [RFC8080] as a drop-in replacement for the P256 elliptic curve. The authoritative server implements the optimization from Section 9.1.1 to precompute the NSEC5PROOF RRs matching each NSEC5 record.

14. Performance Considerations

The performance of NSEC5 has been evaluated in [nsec5ecc].

15. Security Considerations

15.1. Zone Enumeration Attacks

NSEC5 is robust to zone enumeration via offline dictionary attacks by any attacker that does not know the private NSEC5 key. Without the private NSEC5 key, that attacker cannot compute the NSEC5 proof that corresponds to a given domain name. The only way it can learn the NSEC5 proof value for a domain name is by querying the authoritative server for that name. Without the NSEC5 proof value, the attacker cannot learn the NSEC5 hash value. Thus, even an attacker that collects the entire chain of NSEC5 RR for a zone cannot use offline attacks to "reverse" that NSEC5 hash values in these NSEC5 RR and thus learn which names are present in the zone. A formal cryptographic proof of this property is in [nsec5] and [nsec5ecc].

15.2. Compromise of the Private NSEC5 Key

NSEC5 requires authoritative servers to hold the private NSEC5 key, but not the private zone-signing keys or the private key-signing keys for the zone.

The private NSEC5 key cannot be used to modify zone contents, because zone contents are signed using the private zone-signing key. As such, a compromise of the private NSEC5 key does not compromise the integrity of the zone. An adversary that learns the private NSEC5 key can, however, perform offline zone-enumeration attacks. For this reason, the private NSEC5 key need only be as secure as the DNSSEC records whose privacy (against zone enumeration) is being protected by NSEC5. A formal cryptographic proof of this property is in [nsec5] and [nsec5ecc].

To preserve this property of NSEC5, the private NSEC5 key MUST be different from the private zone-signing keys or key-signing keys for the zone.

15.3. Key Length Considerations

The NSEC5 key must be long enough to withstand attacks for as long as the privacy of the zone contents is important. Even if the NSEC5 key is rolled frequently, its length cannot be too short, because zone privacy may be important for a period of time longer than the lifetime of the key. For example, an attacker might collect the entire chain of NSEC5 RR for the zone over one short period, and then, later (even after the NSEC5 key expires) perform an offline dictionary attack that attempts to reverse the NSEC5 hash values present in the NSEC5 RRs. This is in contrast to zone-signing and key-signing keys used in DNSSEC; these keys, which ensure the authenticity and integrity of the zone contents, need to remain secure only during their lifetime.

15.4. NSEC5 Hash Collisions

If the NSEC5 hash of a QNAME collides with the NSEC5 hash of the owner name of an NSEC5 RR, it will be impossible to prove the non-existence of the colliding QNAME. However, the NSEC5 VRFs ensure that obtaining such a collision is as difficult as obtaining a collision in the SHA-256 hash function, requiring approximately 2^128 effort. Note that DNSSEC already relies on the assumption that a cryptographic hash function is collision-resistant, since these hash functions are used for generating and validating signatures and DS RRs. See also the discussion on key lengths in [nsec5].

16. IANA Considerations

This document updates the IANA registry "Domain Name System (DNS) Parameters" in subregistry "Resource Record (RR) TYPEs", by defining the following new RR types:

NSEC5KEY value TBD.

NSEC5 value TBD.

NSEC5PROOF value TBD.

This document creates a new IANA registry for NSEC5 algorithms. This registry is named "DNSSEC NSEC5 Algorithms". The initial content of the registry is:

0 is Reserved.

1 is EC-P256-SHA256.

2 is EC-ED25519-SHA256.

3-255 is Available for assignment.

This document updates the IANA registry "DNS Security Algorithm Numbers" by defining following aliases:

TBD is NSEC5-ECDSAP256SHA256 alias for ECDSAP256SHA256 (13).

TBD is NSEC5-ED25519, alias for ED25519 (15).

17. Contributors

This document would not be possible without help of Moni Naor (Weizmann Institute), Sachin Vasant (Cisco Systems), Leonid Reyzin (Boston University), and Asaf Ziv (Weizmann Institute) who contributed to the design of NSEC5. Ondrej Sury (CZ.NIC Labs), and Duane Wessels (Verisign Labs) provided advice on the implementation of NSEC5, and assisted with evaluating its performance.

18. References

18.1. Normative References

[FIPS-186-3]
National Institute for Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-3, June 2009.

[I-D.irtf-cfrg-vrf]
Goldberg, S., Reyzin, L., Papadopoulos, D., and J. Vcelak, "Verifiable Random Functions (VRFs)", draft-irtf-cfrg-vrf-03 (work in progress), September 2018.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <https://www.rfc-editor.org/info/rfc2136>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", RFC 2308, DOI 10.17487/RFC2308, March 1998, <https://www.rfc-editor.org/info/rfc2308>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <https://www.rfc-editor.org/info/rfc4035>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards", RFC 5114, DOI 10.17487/RFC5114, January 2008, <https://www.rfc-editor.org/info/rfc5114>.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, DOI 10.17487/RFC5155, March 2008, <https://www.rfc-editor.org/info/rfc5155>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, DOI 10.17487/RFC6605, April 2012, <https://www.rfc-editor.org/info/rfc6605>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8080] Sury, O. and R. Edmonds, "Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC", RFC 8080, DOI 10.17487/RFC8080, February 2017, <https://www.rfc-editor.org/info/rfc8080>.

18.2. Informative References

[I-D.gieben-nsec4] Gieben, R. and M. Mekking, "DNS Security (DNSSEC) Authenticated Denial of Existence", draft-gieben-nsec4-01 (work in progress), July 2012.

[ldns-walk] NLNetLabs, "ldns", 2015, <http://git.nlnetlabs.nl/ldns/tree/examples/ldns-walk.c>.

[MRV99] Michali, S., Rabin, M., and S. Vadhan, "Verifiable Random Functions", in FOCS, 1999.

[nmap-nsec-enum]
Bond, J., "nmap: dns-nsec-enum", 2011, <https://nmap.org/nsedoc/scripts/dns-nsec-enum.html>.

[nmap-nsec3-enum]
Nikolic, A. and J. Bond, "nmap: dns-nsec3-enum", 2011, <https://nmap.org/nsedoc/scripts/dns-nsec3-enum.html>.

[nsec3gpu]
Wander, M., Schwittmann, L., Boelmann, C., and T. Weis, "GPU-Based NSEC3 Hash Breaking", in IEEE Symp. Network Computing and Applications (NCA), 2014.

[nsec3map]
anonion0, "nsec3map with John the Ripper plugin", 2015, <https://github.com/anonion0/nsec3map.>.

[nsec3walker]
Bernstein, D., "Nsec3 walker", 2011, <http://dnscurve.org/nsec3walker.html>.

[nsec5] Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., and A. Ziv, "NSEC5: Provably Preventing DNSSEC Zone Enumeration", in NDSS'15, July 2014, <https://eprint.iacr.org/2014/582.pdf>.

[nsec5ecc]
Papadopoulos, D., Wessels, D., Huque, S., Vcelak, J., Naor, M., Reyzin, L., and S. Goldberg, "Can NSEC5 be Practical for DNSSEC Deployments?", in ePrint Cryptology Archive 2017/099, February 2017, <https://eprint.iacr.org/2017/099.pdf>.

[RFC6781] Kolkman, O., Mekking, W., and R. Gieben, "DNSSEC Operational Practices, Version 2", RFC 6781, DOI 10.17487/RFC6781, December 2012, <https://www.rfc-editor.org/info/rfc6781>.

[RFC7129] Gieben, R. and W. Mekking, "Authenticated Denial of Existence in the DNS", RFC 7129, DOI 10.17487/RFC7129, February 2014, <https://www.rfc-editor.org/info/rfc7129>.

[RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", RFC 7719, DOI 10.17487/RFC7719, December 2015, <https://www.rfc-editor.org/info/rfc7719>.

Appendix A. Examples

We use a small DNS zone to illustrate how negative responses are handled with NSEC5. For brevity, the class is not shown (defaults to IN) and the SOA record is shortened, resulting in the following zone file:

```
example.org.        SOA ( ... )
example.org.        NS  a.example.org

a.example.org.      A 192.0.2.1

c.example.org.      A 192.0.2.2
c.example.org.      TXT "c record"

d.example.org.      NS ns1.d.example.org

ns1.d.example.org.  A 192.0.2.4

g.example.org.      A 192.0.2.1
g.example.org.      TXT "g record"

*.a.example.org.    TXT "wildcard record"
```

Notice the delegation to an unsigned zone d.example.org served by ns1.d.example.org. (Note: if the d.example.org zone was signed, then the example.org zone have a DS record for d.example.org.)

Next we present example responses. All cryptographic values are shortened as indicated by "..." and ADDITIONAL sections have been removed.

A.1. Name Error Example

Consider a query for a type A record for a.b.c.example.org.

The server must prove the following facts:

- o Existence of closest encloser c.example.org.
- o Non-existence of wildcard at closest encloser *.c.example.org.
- o Non-existence of next closer b.c.example.org.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NXDOMAIN; id: 5937

;; QUESTION SECTION:
;; a.b.c.example.org.           IN      A

;; AUTHORITY SECTION:
example.org.         3600 IN SOA a.example.org. hostmaster.example.org. (
            2010111214 21600 3600 604800 86400 )

example.org.         3600 IN RRSIG  SOA 16 2 3600 (
            20170412024301 20170313024301 5137 example.org. rT231b1rH... )
```

This is an NSEC5PROOF RR for c.example.com. It's RDATA is the NSEC5 proof corresponding to c.example.com. (NSEC5 proofs are randomized values, because NSEC5 proof values are computed uses the EC-VRF from [I-D.irtf-cfrg-vrf].) Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
c.example.org.      86400 IN NSEC5PROOF 48566 Amgn22zUiZ9JVyaT...
```

This is a signed NSEC5 RR "matching" c.example.org, which proves the existence of closest encloser c.example.org. The NSEC5 RR has its owner name equal to the NSEC5 hash of c.example.org, which is O4K89V. (NSEC5 hash values are deterministic given the public NSEC5 key.) The NSEC5 RR also has its Wildcard flag cleared (see the "0" after the key ID 48566). This proves the non-existence of the wildcard at the closest encloser *.c.example.com. NSEC5 RRs are precomputed.

```
o4k89v.example.org. 86400 IN NSEC5   48566 0 0O49PI A TXT RRSIG
o4k89v.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. zDNTSMQNlz... )
```

This is an NSEC5PROOF RR for b.c.example.org. It's RDATA is the NSEC5 proof corresponding to b.c.example.com. This NSEC5PROOF RR must be computed on the fly.

```
b.c.example.org.    86400 IN NSEC5PROOF 48566 AuvvJqbUcEs8sCpY...
```

This is a signed NSEC5 RR "covering" b.c.example.org, which proves the non-existence of the next closer name b.c.example.org The NSEC5 hash of b.c.example.org, which is AO5OF, sorts in canonical order between the "covering" NSEC5 RR's Owner Name (which is 0O49PI) and Next Hashed Owner Name (which is BAPROH).

```
0o49pi.example.org. 86400 IN NSEC5      48566 0 BAPROH (
            NS SOA RRSIG DNSKEY NSEC5KEY )

0o49pi.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. 4HT1uj1YlMzO)
```

[TODO: Add discussion of CNAME and DNAME to the example?]

A.2. No Data Example

Consider a query for a type MX record for c.example.org.

The server must prove the following facts:

- o Existence of c.example.org. for any type other than MX or CNAME

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 38781

;; QUESTION SECTION:
;; c.example.org.    IN MX

;; AUTHORITY SECTION:
example.org.    3600 IN SOA     a.example.org. hostmaster.example.org. (
            2010111214 21600 3600 604800 86400 )

example.org.    3600 IN RRSIG   SOA 16 2 3600 20170412024301 20170313024301 5137
 example.org. /rT231b1rH/p
```

This is an NSEC5PROOF RR for c.example.com. Its RDATA corresponds to the NSEC5 proof for c.example.com. which is a randomized value. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
c.example.org. 86400 IN NSEC5PROOF 48566 Amgn22zUiZ9JVyaT
```

This is a signed NSEC5 RR "matching" c.example.org. with CNAME and MX Type Bits cleared and its TXT Type Bit set. This NSEC5 RR has its owner name equal to the NSEC5 hash of c.example.org. This proves the existence of c.example.org. for a type other than MX and CNAME. NSEC5 RR are precomputed.

```
o4k89v.example.org. 86400 IN NSEC5   48566 0 0O49PI A TXT RRSIG

o4k89v.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. zDNTSMQNlz/J)
```

A.3. Delegation to an Unsigned Zone in an Opt-Out span Example

Consider a query for a type A record for foo.d.example.org.

Here, d.example.org is a delegation to an unsigned zone, which lies within an Opt-Out span.

The server must prove the following facts:

- Non-existence of signature on next closer name d.example.org.
- Opt-out bit is set in NSEC5 record covering next closer name d.example.org.
- Existence of closest provable encloser example.org

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 45866

;; QUESTION SECTION:
;; foo.d.example.org.         IN A

;; AUTHORITY SECTION:
d.example.org.       3600  IN NS      ns1.d.example.org.
```

This is an NSEC5PROOF RR for d.example.org. Its RDATA is the NSEC5 proof corresponding to d.example.org. This NSEC5PROOF RR is computed on the fly.

```
d.example.org.      86400   IN      NSEC5PROOF      48566 A9FpmeH79q7g6VNW
```

This is a signed NSEC5 RR "covering" d.example.org with its Opt-out bit set (see the "1" after the key ID 48566). The NSEC5 hash of d.example.org (which is BLE8LR) sorts in canonical order between the "covering" NSEC5 RR’s Owner Name (BAPROH) and Next Hashed Owner Name (JQBMG4). This proves that no signed RR exists for d.example.org, but that the zone might contain a unsigned RR for a name whose NSEC5 hash sorts in canonical order between BAPROH and JQBMG4.

```
baproh.example.org. 86400 IN NSEC5   48566 1 JQBMG4 A TXT RRSIG

baproh.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1)
```

This is an NSEC5PROOF RR for example.com. It’s RDATA is the NSEC5 proof corresponding to example.com. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
   example.org.        86400 IN NSEC5PROOF      48566 AjwsPCJZ8zH/D0Tr
```

This is a signed NSEC5 RR "matching" example.org which proves the existence of a signed RRs for example.org. This NSEC5 RR has its owner name equal to the NSEC5 hash of example.org which is 0O49PI. NSEC5 RR are precomputed.

```
0o49pi.example.org. 86400 IN NSEC5   48566 0 BAPROH (
            NS SOA RRSIG DNSKEY NSEC5KEY)

0o49pi.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412034216 20170313034216 5137 example.org. 4HT1uj1YlMzO)
```

A.4. Wildcard Example

Consider a query for a type TXT record for foo.a.example.org.

The server must prove the following facts:

- o Existence of the TXT record for the wildcard *.a.example.org
- o Non-existence of the next closer name foo.a.example.org.

To do this, the server returns:

```
   ;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 53731

   ;; QUESTION SECTION:
   ;; foo.a.example.org.        IN TXT
```

This is a signed TXT record for the wildcard at a.example.org (number of labels is set to 3 in the RRSIG record).

```
;; ANSWER SECTION:
foo.a.example.org.      3600 IN TXT     "wildcard record"

foo.a.example.org.      3600 IN RRSIG   TXT 16 3 3600 (
            20170412024301 20170313024301 5137 example.org. aeaLgZ8sk+98)

;; AUTHORITY SECTION:
example.org.            3600 IN NS      a.example.org.

example.org.            3600 IN RRSIG   NS 16 2 3600 (
            20170412024301 20170313024301 5137 example.org. 8zuN0h2x5WyF)
```

This is an NSEC5PROOF RR for foo.a.example.org. This NSEC5PROOF RR must be computed on-the-fly.

```
  foo.a.example.org.     86400 IN NSEC5PROOF      48566 AjqF5FGGVso40Lda
```

This is a signed NSEC5 RR "covering" foo.a.example.org. The NSEC5 hash of foo.a.example.org is FORDMO and sorts in canonical order between the NSEC5 RR's Owner Name (which is BAPROH) and Next Hashed Owner Name (which is JQBMG4). This proves the non-existence of the next closer name foo.a.example.com. NSEC5 RRs are precomputed.

```
baproh.example.org. 86400 IN NSEC5   48566 1 JQBMG4 A TXT RRSIG
baproh.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
    20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1
```

A.5. Wildcard No Data Example

Consider a query for a type MX record for foo.a.example.org.

The server must prove the following facts:

- o Existence of wildcard at closest encloser *.a.example.org. for any type other than MX or CNAME.
- o Non-existence of the next closer name foo.a.example.org.

To do this, the server returns:

```
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 17332

;; QUESTION SECTION:
;; foo.a.example.org.           IN      MX

;; AUTHORITY SECTION:
example.org.       3600 IN SOA     a.example.org. hostmaster.example.org. (
            2010111214 21600 3600 604800 86400 )

example.org.       3600 IN RRSIG   SOA 16 2 3600 (
            20170412024301 20170313024301 5137 example.org. /rT231b1rH/p )
```

This is an NSEC5PROOF RR for *.a.example.com, with RDATA equal to the NSEC5 proof for *.a.example.com. Per Section 9.1.1, this NSEC5PROOF RR may be precomputed.

```
*.a.example.org.  86400 IN NSEC5PROOF      48566 Aq38RWWPhbs/vtih
```

This is a signed NSEC5 RR "matching" *.a.example.org with its CNAME and MX Type Bits cleared and its TXT Type Bit set. This NSEC5 RR has its owner name equal to the NSEC5 hash of *.a.example.org. NSEC5 RRs are precomputed.

```
mpu6c4.example.org. 86400 IN NSEC5   48566 0 O4K89V TXT RRSIG

mpu6c4.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. m3I75ttcWwVC )
```

This is an NSEC5PROOF RR for foo.a.example.com. This NSEC5PROOF RR must be computed on-the-fly.

```
foo.a.example.org.  86400 IN NSEC5PROOF      48566 AjqF5FGGVso40Lda
```

This is a signed NSEC5 RR "covering" foo.a.example.org. The NSEC5 hash of foo.a.example.org is FORDMO, and sorts in canonical order between this covering NSEC5 RR's Owner Name (which is BAPROH) and Next Hashed Owner Name (which is JQBMG4). This proves the existence of the wildcard at closest encloser *.a.example.org. for any type other than MX or CNAME. NSEC5 RRs are precomputed.

```
baproh.example.org. 86400 IN NSEC5   48566 1 JQBMG4 A TXT RRSIG

baproh.example.org. 86400 IN RRSIG   NSEC5 16 3 86400 (
            20170412024301 20170313024301 5137 example.org. fjTcoRKgdML1 )
```

Appendix B. Change Log

Note to RFC Editor: if this document does not obsolete an existing RFC, please remove this appendix before publication as an RFC.

pre 00 - initial version of the document submitted to mailing list only

00 - fix NSEC5KEY rollover mechanism, clarify NSEC5PROOF RDATA, clarify inputs and outputs for NSEC5 proof and NSEC5 hash computation.

01 - Add Performance Considerations section.

02 - Add elliptic curve based VRF. Add measurement of response sizes based on empirical data.

03 - Mention precomputed NSEC5PROOF Values in Performance Considerations section.

04 - Edit Rationale, How NSEC5 Works, and Security Consideration sections for clarity. Edit Zone Signing section, adding precomputation of NSEC5PROOFs. Remove RSA-based NSEC5 specification. Rewrite Performance Considerations and Implementation Status sections.

05 - Remove appendix specifying VRFs and add reference to draft-goldbe-vrf. Add Appendix A.

06 - Editorial changes. Minor updates to Section 8.1.

07 - Updated reference to [I-D.irtf-cfrg-vrf], updated VRF ciphersuites.

Authors' Addresses

Jan Vcelak
CZ.NIC
Milesovska 1136/5
Praha 130 00
CZ

EMail: jan.vcelak@nic.cz

Sharon Goldberg
Boston University
111 Cummington St, MCS135
Boston, MA 02215
USA

EMail: goldbe@cs.bu.edu

Dimitrios Papadopoulos
HKUST
Clearwater Bay
Hong Kong

EMail: dipapado@ust.hk

Shumon Huque
Salesforce
2550 Wasser Terr
Herndon, VA 20171
USA

EMail: shuque@gmail.com

David C Lawrence
Dyn
150 Dow Street, Tower Two
Manchester, NH 03101
USA

EMail: tale@dd.org