Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 25, 2017

T. Aura
Aalto University
M. Sethi
Ericsson

May 24, 2017

# Nimble out-of-band authentication for EAP (EAP-NOOB) draft-aura-eap-noob-02

## 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

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

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

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

This Internet-Draft will expire on November 25, 2017.

## Copyright Notice

Table of Contents

## 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 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 of 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 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 or display screen, 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. 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 [SimplePairing], 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 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 Elliptic Curve Diffie-Hellman (ECDH) key exchange. The user-assisted OOB Step then takes place. This step involves 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 parties store the created association in persistent storage, and the OOB Step is not repeated. Thereafter, creation of new temporal keys, ECDH rekeying, and updates of cryptographic algorithms can be achieved with the Reconnect Exchange.

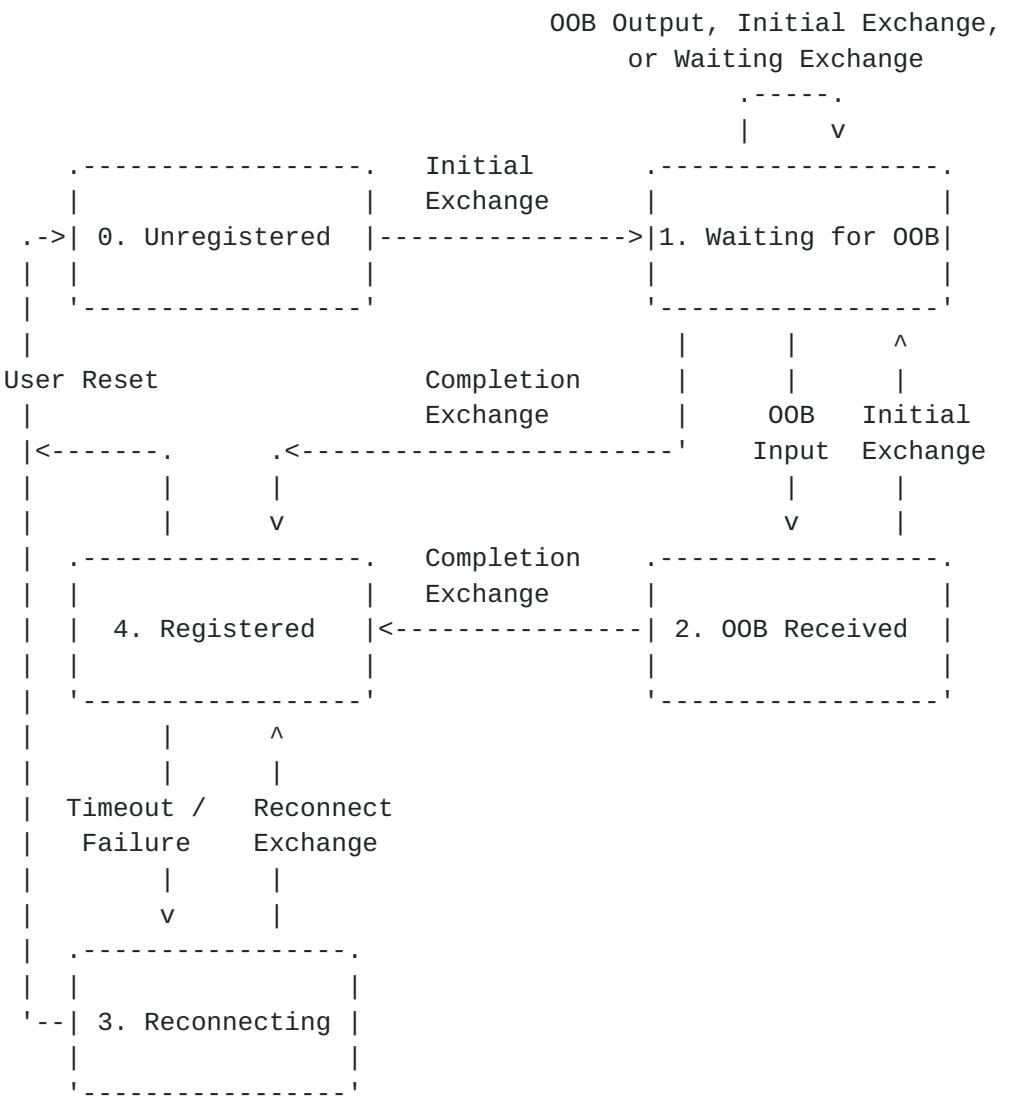


Figure 1: EAP-NOOB server-peer association state machine

Figure 1 shows the association state machine, which is the same for the server and for the peer. When the client 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 party 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), and only user reset or memory failure can cause the return of the server or the peer to the ephemeral states and to the Initial Exchange.

The server MUST NOT repeat the 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. In particular, 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 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 they continue to store data about the association after the user reset, their 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 such multiple association of 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 the NAI. The server then allocates a server-specific identifier to the peer. After that, the network access identifier NAI is a concatenation of the server-allocated peer identifier and the generic suffix "@eap-noob.net". This suffix 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 connect to the server two or more times 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 these output-only devices cannot be told to perform the protocol at the right moment, 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) from which the OOB message is printed as a bar code on paper and then scanned with a camera phone and communicated to the server. To support such devices and slow OOB channels, the peer in EAP-NOOB first contacts the server in the Initial Exchange, then disconnects for some time, and later continues with the Waiting and Completion Exchanges.

## 3.2. Protocol messages and sequences

This section defines the EAP-NOOB exchanges. The protocol messages communicated 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). The peer MUST compose the NAI as defined in Section 3.3. Essentially, if the peer has no previous peer identifier (PeerId), it uses the fixed NAI string "noob@eap-noob.net", and if it has received a PeerId from the server, it creates the NAI by concatenating the PeerId, a state indicator "+sX", and the fixed suffix string "@eap-noob.net".

After receiving the NAI, the server chooses the EAP-NOOB exchange, i.e. the ensuing message sequence, based on the combination of the client and server states. The client 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 two 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, algorithm and parameter negotiation and the other for the ECDH 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

protocol versions supported (Vers), cryptosuites (Cryptosuites), an indicator of the OOB channel directions supported by the server (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 ECDH keys and nonces (PKs,Ns,PKp,Np). The ECDH keys MUST be based on the negotiated cryptosuite. The Initial Exchange ends with EAP-Failure from the server because the authentication cannot yet be completed.

The server MAY assign a realm to the peer by sending the optional Realm field in the Initial Exchange. In that case, the peer MUST use the assigned Realm (together with the allocated PeerId) to construct the NAI for the following Waiting, Completion, and Reconnect Exchanges with the server. The peer MUST remember the assigned values until a new Initial Exchange or return to Unregistered state. Some Authenticators or AAA servers use the assigned Realm to determine client-specific connection parameters, such as isolating the peer to a specific VLAN.

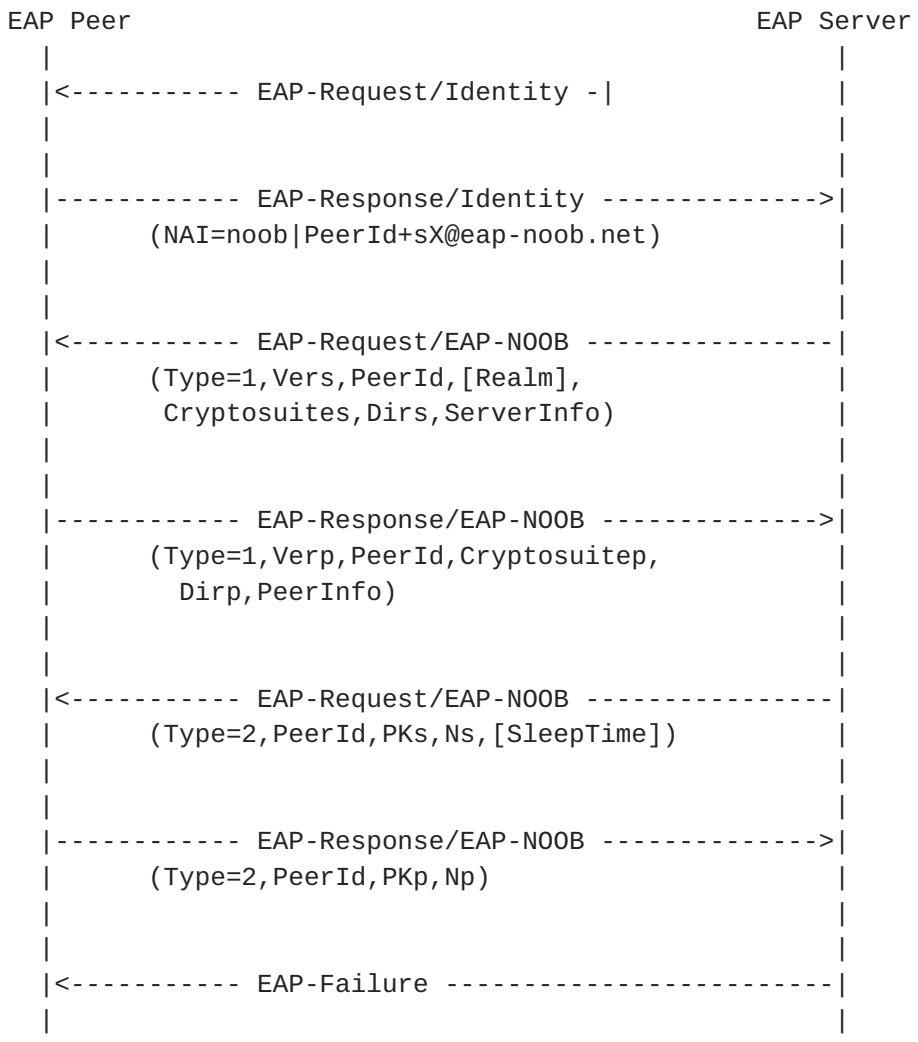


Figure 2: Initial Exchange

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, shown 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 containing the PeerId, the secret nonce Noob, and the cryptographic fingerprint Hoob, as defined in Section 3.3. This message is then delivered to the other party via a user-assisted OOB channel. The details of the OOB channel are defined by the application.

The receiver of the OOB message MUST compare the received value of the fingerprint Hoob with a value that it computes locally, and it

MUST reject OOB messages with invalid Hoob. 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 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 sender will typically generate a new Noob, and therefore a new OOB message, at constant 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 this case, both sides SHOULD output the OOB message, and it is up to the user to deliver one of them.

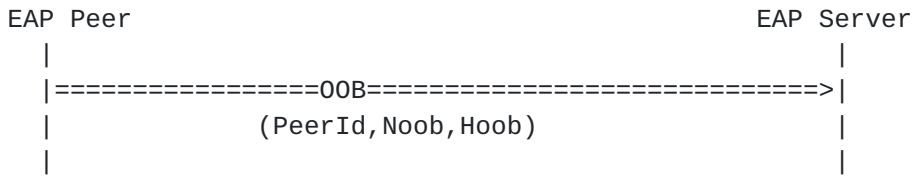


Figure 3: OOB Step, from peer to EAP server

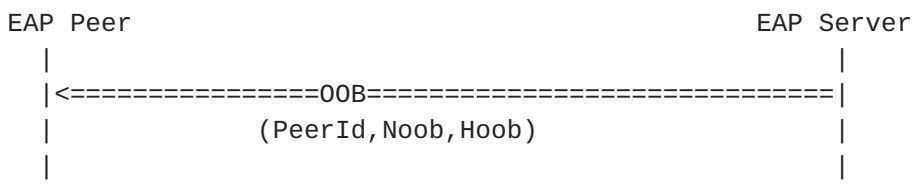


Figure 4: OOB Step, from EAP server to peer

### 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. With the first request, the server discovers 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. Both sides MUST compare the received message authentication code with a locally computed value. If the EAP server finds that it has received the correct value of MACp, the Completion Exchange ends in EAP-Success; otherwise, in EAP-Failure.

While 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 by observing that both the server and peer are in the OOB Received state (2). In that case, the server MUST behave as if only the server-to-peer message was delivered.

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

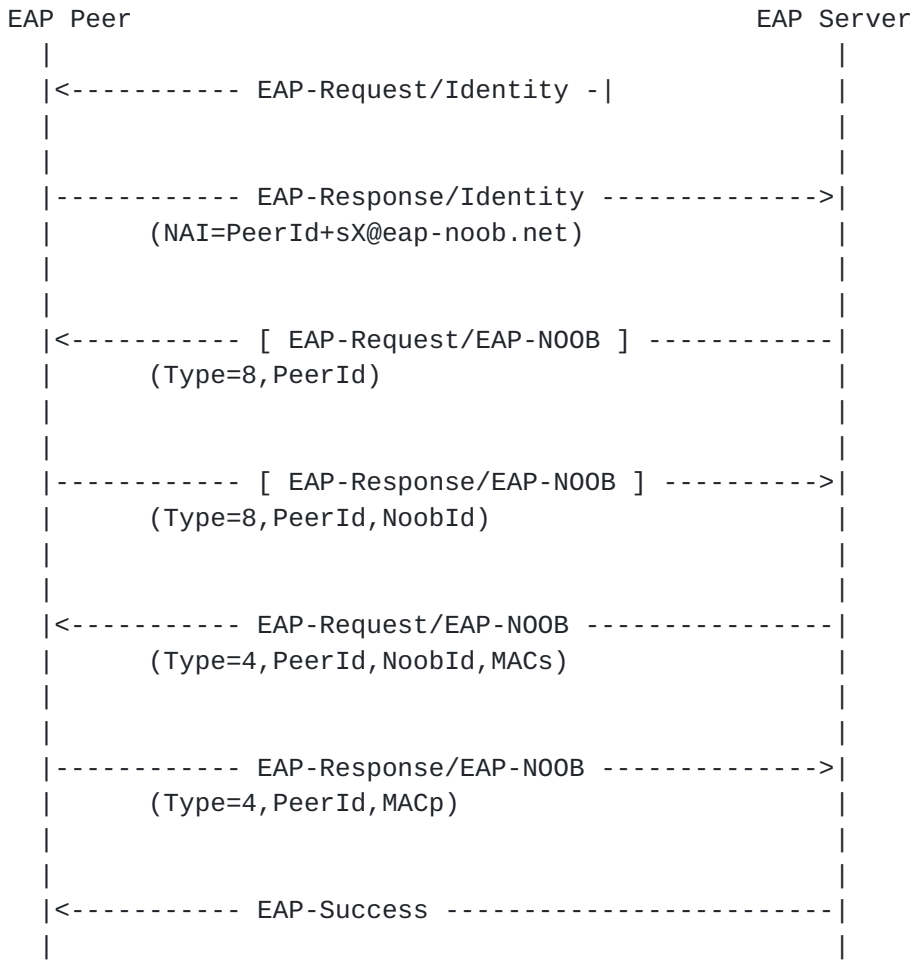


Figure 5: Completion Exchange

### 3.2.4. Waiting Exchange

As explained in Section 3.2.3, the peer SHOULD probe the server for completion of the OOB Step. If both the server and client states are Waiting for OOB (1), the server will continue with the Waiting

Exchange (message Type=3). The main purpose of this exchange is to indicate to 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 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.

After the Waiting Exchange, the peer MUST discard 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 MAC. The timeout of 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.

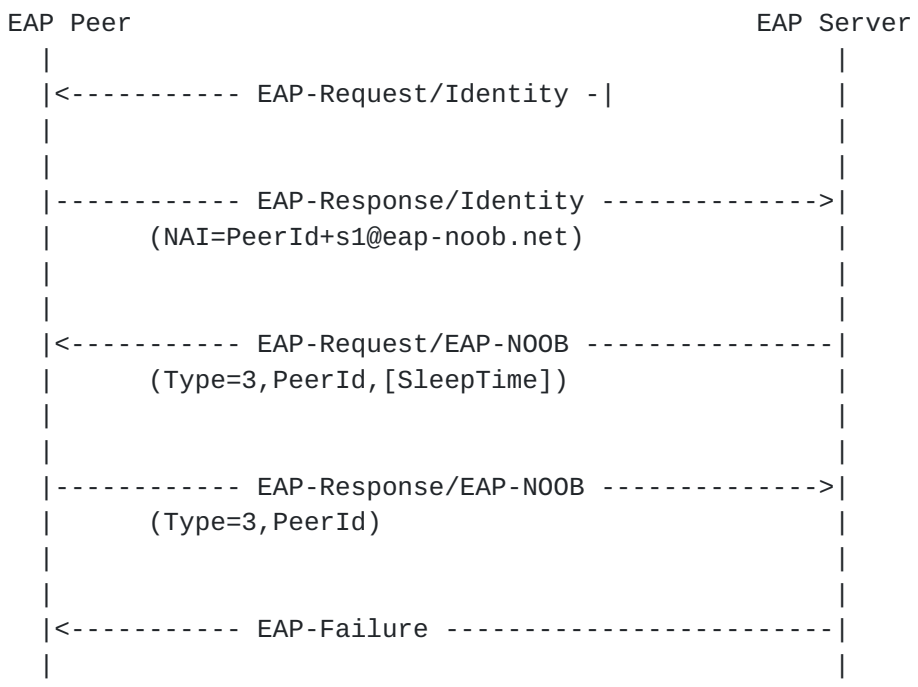


Figure 6: Waiting Exchange

## 3.3. Message data fields

Table 1 defines the data fields in the protocol messages. The in-band messages are formatted as JSON objects [RFC7159] in UTF-8 encoding. The JSON member names are in the left-hand column of 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. Currently, the only defined values are "[1]" and "1", respectively. |
| PeerId | Peer identifier. If the peer does not yet have a peer identifier or it does not remember one, it uses the NAI "noob@eap-noob.net" in the Initial Exchange. The server then assigns an identifier to the peer and sends it in the first server-to-peer request of the Initial Exchange. The assigned identifier is ephemeral until a successful Completion Exchange takes |

| | |
|---|---|
| | place. Thereafter, the peer identifier becomes permanent until the user resets the peer and the server. Resetting the server means deleting the association for the peer from the server database. 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 beforehand. One way to generate the identifiers is to choose a random 16-byte identifier and base64url encode it into a 22-character string. Another way to generate the identifiers is to choose a random 22-character alphanumeric string. |
| Realm | Peer realm. The server may assign a realm to the peer. Peer then uses this value as the realm part of its NAI. The realm value MUST follow the syntax of the uft8-realm specified in [RFC7542]. |
| Peer State "+sX" | This part of the NAI informs the server about the peer state. The server uses this information together with the server state to decide on the type of the exchange and, thus, of the type of the next EAP-Request. The peer appends the peer state to the PeerId to form the username part of the NAI. (Sending it in the EAP-Response/Identity message avoids an additional round trip for querying the peer state.) If the peer is in state 0, it MUST use the NAI "noob@eap-noob.net"; otherwise, the peer MUST create the NAI as the concatenation of the PeerId, the string "+s", a single digit indicating the state of the peer, and the string "@eap-noob.net". |
| 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 ECDH keys of the server and peer. PKs and PKp are sent in the JSON Web Key (JWK) format [RFC7517]. |

| | |
|---|---|
| Cryptosuites, Cryptosuitep | The identifiers of cryptosuites supported by the server and of the cryptosuite selected by the peer. The format is specified in Section 4.1. |
| Dirs, Dirp | OOB channel directions supported by the server and ones selected by the peer. The possible values are 1=peer-to-server, 2=server-to-peer, 3=both directions. Endpoints that are general-purpose computers or online services SHOULD support both directions. IoT devices with a limited user interface will mostly support only one direction. If the negotiated value is 3, the user may be presented with two OOB messages, one for each direction, even though the user needs to deliver only one of them. Since this can be confusing to the user, it is RECOMMENDED that the peer selects Dirp value 1 or 2. The EAP-NOOB protocol itself 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. |
| 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 and it is encoded as a JSON object of at most 500 bytes. It could include, for example, the 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 and it is encoded as a JSON object of at most 500 bytes. It could include, for example, the peer make, model and serial number that helps 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 it 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 party that received the OOB message uses this secret in the Completion Exchange to authenticate the exchanged key to the party that sent the OOB message. |
| Hoob | | 32-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 party that receives the OOB message. |
| NoobId | | 16-byte identifier for the OOB message, computed with a one-way function from the nonce Noob. |
| Ns2, Np2 | | 32-byte Nonces for the Reconnect Exchange. |
| MACs, MACp | | Message authentication codes 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. |
| PKs2, PKp2 | | The public components of the ECDH keys of the server and peer. These MUST be present if a new cryptosuite was negotiated. Otherwise, either party may omit the field. PKs2 and PKp2 are sent in the JSON Web Key (JWK) format [RFC7517]. |
| MACs2, MACp2 | | Message authentication codes 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. | |

Table 1: Message data fields

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 party that sends the message.

The fingerprint Hoob and the identifier NoobId are computed with the 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 (16) of members. The array member values MUST be copied to the array verbatim from the in-band messages, and space characters or whitespace MUST NOT be added anywhere in the JSON structure.

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,PKs,Ns,PKp,Np,Noob).

NoobId = H("NoobId",Noob).

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

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

MACs2 = HMAC(Kms2; 2,Vers,Verp,PeerId,Cryptosuites,"",[ServerInfo]
,Cryptosuitep,"",[Realm],[PeerInfo],[PKs2],Ns2,[PKp2],Np2,"")

MACp2 = HMAC(Kmp2; 1,Vers,Verp,PeerId,Cryptosuites,"",[ServerInfo]
,Cryptosuitep,"",[Realm],[PeerInfo],[PKs2],Ns2,[PKp2],Np2,"")
```

Missing input values are represented by empty strings "" in the array. The values indicated with "" are always empty strings. The values in brackets MUST be included if they were exchanged in the same Reconnect Exchange; otherwise these values are replaced by empty strings "".

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 in needed 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 for the HMACs are defined in the following section.

The nonces (Ns, Np, Ns2, Np2) and message authentication codes (MACs, MACp, MACs2, MACp2) in the in-band messages and in the cryptographic function inputs MUST be base64url encoded [RFC4648]. The values Noob and Hoob in the OOB channel MAY also be base64url encoded, if that is appropriate for the application and the used OOB channel.

The ServerInfo and PeerInfo are JSON object with UTF-8 encoding. The length of each 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. A timeout, software or hardware failure, or user action may cause the EAP server, peer or authenticator to lose its non-persistent state data such as the EAP output values MSK and EMSK. The failure would typically be detected by the peer or authenticator when the keys no longer are accepted by the other party. 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 such an event, it MUST change from the Registered to Reconnecting state. These state transitions are labeled 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 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) |
| Realm | Optional realm assigned by server (default value is "eap-noob.net") | UTF-8 string |
| Cryptosuitep | Negotiated cryptosuite | integer |
| Kz | Persistent key material | 32 bytes |

Table 2: Persistent EAP-NOOB association

### 3.4.2. Reconnect Exchange

The server chooses the Reconnect Exchange when 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 algorithm and parameter negotiation, another for the nonce and 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 cryptosuite in the same way as in the Initial Exchange. The messages MAY also contain PeerInfo and ServerInfo objects. In the second request and response (Type=6), the server and peer exchange the public components of ECDH keys and the nonces (PKs2,Ns2,PKp2,Np2). The server ECDH key MUST be fresh, i.e. not previously used with the same peer, and the client ECDH key SHOULD be fresh, i.e. not previously used. In the third and final request and response (Type=7), the server and peer exchange the message authentication codes (MACs2,MACp2). 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. Both sides MUST compare the received message authentication code with a locally computed value. If the EAP server finds that it has received

the correct value of MACp, the Completion Exchange ends in EAP-Success; otherwise, in EAP-Failure.

If the negotiated cryptosuite is the same as previously, the server MAY refuse to perform a new ECDH exchange by omitting PKs2, and the peer MAY refuse by omitting PKp2. If the server omits PKs2, the peer SHOULD also omit PKp2. When one or both public keys are not present, the new EAP output values are derived from the fresh nonces and the previously established shared key Kz, as defined in Section 3.5. The security property lost by refusing the ECDH exchange is forward secrecy.

The server and client MAY send updated Realm, ServerInfo and PeerInfo objects in the Reconnect Exchange. If there is no update to the values, they SHOULD omit this information from the messages.

Both sides MUST compare the received message authentication code with a locally computed value. If the EAP server finds that it has received the correct value of MACp2, the Reconnect Exchange ends in EAP-Success; otherwise, in EAP-Failure.

After successful Reconnect Exchange, both the server and the peer move to the Registered (4) state. If the Realm was updated or if a new cryptosuite and Kz were negotiated, they also update the persistent EAP-NOOB association with the changed values.

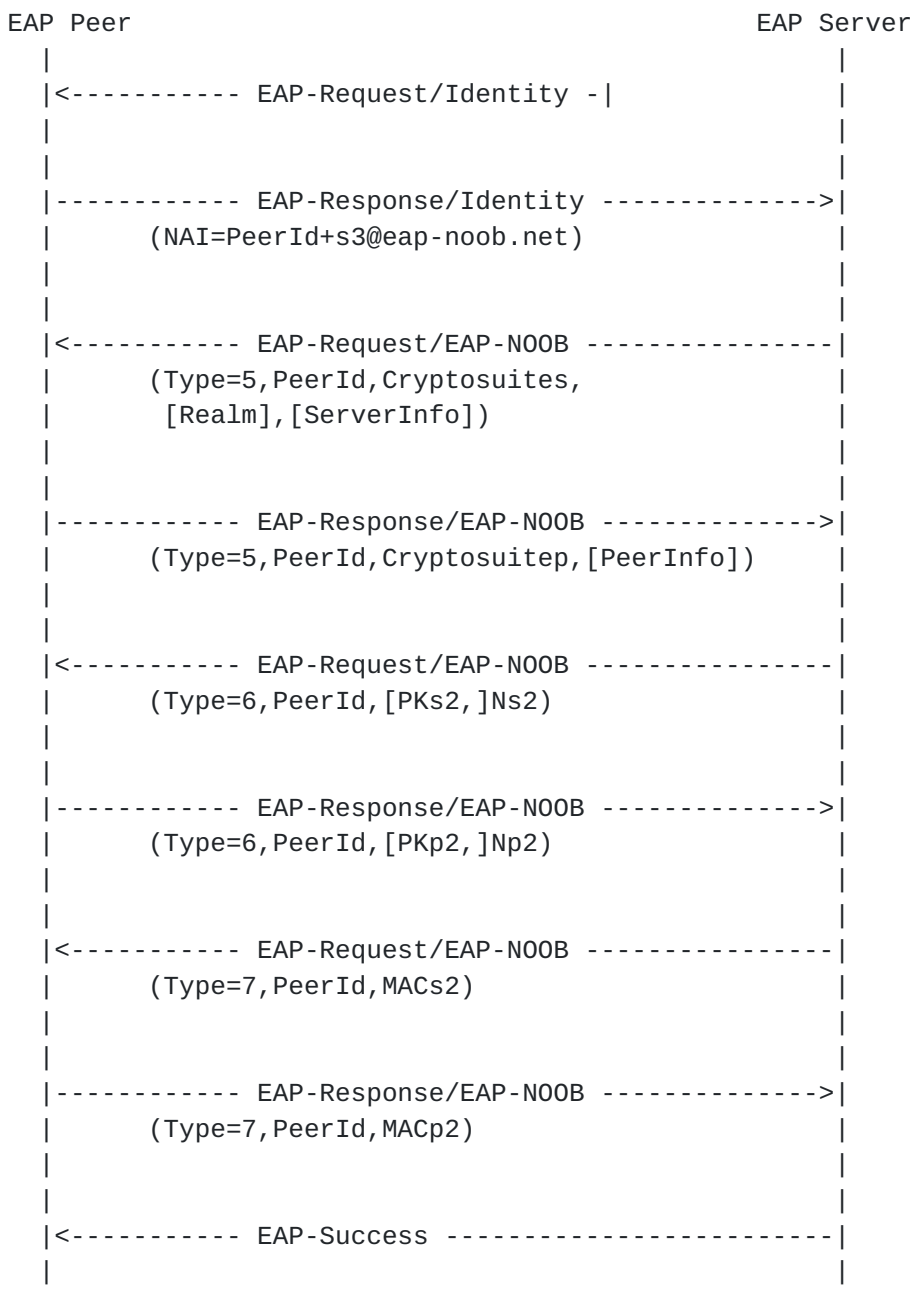


Figure 7: Reconnect Exchange

### 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 administrator 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 SHOULD support situations where the "new" peer is actually a previously registered peer that has been reset by a user or has otherwise lost the persistent EAP-NOOB association data and needs to be merged with the old peer data in the server.

## 3.5. Key derivation

EAP-NOOB derives the EAP output values MSK and EMSK and other secret keying material from the output of an Elliptic Curve Diffie-Hellman (ECDH) 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 Rekeying Exchange, the server and peer compute the ECDH shared secret Z as defined in section 6.1.2.2 of the NIST specification. In the Completion and Rekeying Exchange, 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.

Table 3 defines the inputs to the KDF. In the Completion Exchange, the input Z comes from the preceding Initial exchange, while the Rekeying Exchange uses the Z just created. The 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. OtherInfo consists of the AlgorithmId, PartyUInfo, PartyVInfo, and SuppPrivInfo fields. The three first ahve fixed length, and SuppPrivInfo have fixed length, and SuppPrivInfo has a one-byte

Datalength. AlgorithmId is the fixed-length 8-byte ASCII string "EAP-NOOB". The other input values are the server's and peer's nonces. In the Completion Exchange, the inputs also include the secret nonce Noob from the OOB message, while in the Rekeying Exchange, it is replaced by the shared secret Kz from the persistent EAP-NOOB association.

A special case of the rekeying occurs if no ECDH public keys were exchanged in the Reconnect Exchange (or if only one party sent its public key). 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 ECDH exchange, but also without forward secrecy.

| Exchange | KDF input field | Value | Length (bytes) |
|---|---|---|---|
| Completion | Z | ECDH shared secret from PKs and PKp | variable |
| | AlgorithmId | "EAP-NOOB" | 8 |
| | PartyUInfo | Np | 32 |
| | PartyVInfo | Ns | 32 |
| | SuppPubInfo | (not allowed) | |
| | SuppPrivInfo | Noob | 16 |
| | | | |
| Rekeying with ECDH | Z | ECDH shared secret from PKs2 and PKp2 | variable |
| | AlgorithmId | "EAP-NOOB" | |
| | PartyUInfo | Np2 | 32 |
| | PartyVInfo | Ns2 | 32 |
| | SuppPubInfo | (not allowed) | |
| | SuppPrivInfo | Kz | 32 |
| | | | |
| Rekeying without ECDH | Z | Kz | 32 |
| | AlgorithmId | "EAP-NOOB" | |
| | PartyUInfo | Np2 | 32 |
| | PartyVInfo | Ns2 | 32 |
| | SuppPubInfo | (not allowed) | |
| | SuppPrivInfo | (null) | 0 |

Table 3: Key derivation input

Table 3 defines how the output bytes of the 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 also produces the shared secret Kz, which the server and peer store in the persistent EAP-NOOB association. The Rekeying Exchange updates Kz only when a new cryptosuite is negotiated. In that case, the server and peer update both the cryptosuite and Kz in the persistent EAP-NOOB association.

| Exchange | KDF output bytes | Used as | Length (bytes) |
|---|---|---|---|
| Completion | 0..63 | MSK | 64 |
| | 64..127 | EMSK | 64 |
| | 128..191 | AMSK | 64 |
| | 192..223 | Kms | 32 |
| | 224..255 | Kmp | 32 |
| | 256..287 | Kz | 32 |
| | | | |
| Rekeying, no change in cryptosuite | 0..63 | MSK | 64 |
| | 64..127 | EMSK | 64 |
| | 128..191 | AMSK | 64 |
| | 192..223 | Kms2 | 32 |
| | 224..255 | Kmp2 | 32 |
| | | | |
| Rekeying, new cryptosuite negotiated | 0..63 | MSK | 64 |
| | 64..127 | EMSK | 64 |
| | 128..191 | AMSK | 64 |
| | 192..223 | Kms2 | 32 |
| | 224..255 | Kmp2 | 32 |
| | 256..287 | Kz | 32 |
| | | | |

Table 4: Key derivation output

## 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 message. 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.

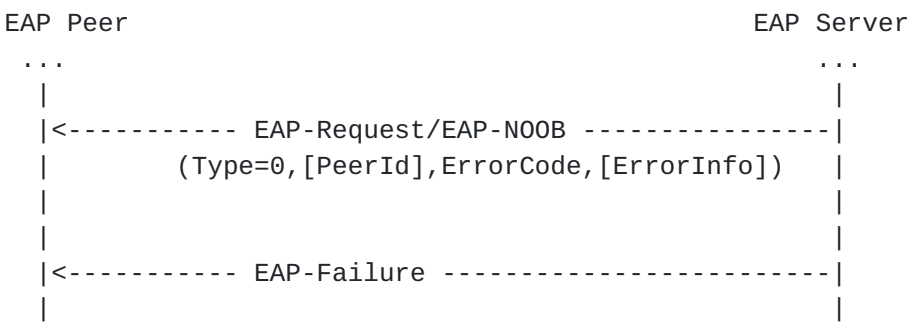


Figure 8: Error notification from server to peer

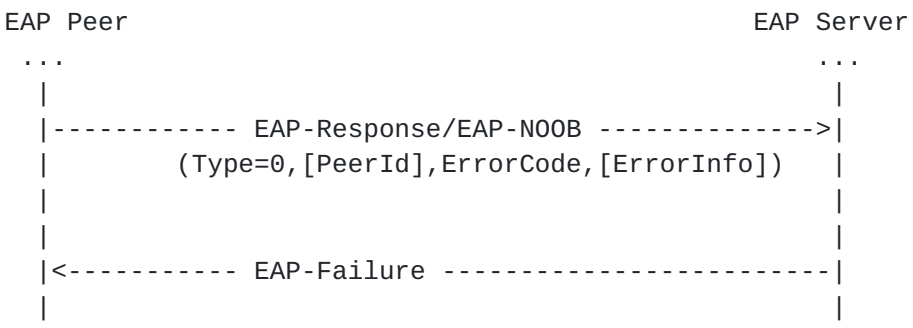


Figure 9: Error notification from peer to server

After an error notification, the server and peer set their 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 did not take place, with the exception that the peer checks for expiry of the Noob value as defined in Section 3.2.4. 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; 1004 if the

received message type was unexpected; 1005 if the PeerId has an unexpected value; 1006 if the NoobId is not recognized; and 1007 if the ECDH 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 is 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. 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.

### 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. In particular, user action is needed for changing the association from a persistent state to the Unregistered (0) state.

### 3.6.5. Cryptographic verification failure

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

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 destination. If possible, the peer SHOULD indicate this to the user; however, the peer device may not have capability for many different error indications and it MAY use the same method or error 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 recipient 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 software bug. For this reason, the EAP server MAY limit the rate of peer connections with SleepTime after the above error. Also, there MUST be a way for the user to reset the EAP server and peer to the Unregistered state (0), so that the OOB Step can be repeated.

### 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 5003, 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 have 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 [RFC5226].

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

An EAP server MUST supply one or more suggestions for cryptosuites as the Cryptosuites value in the Initial Exchange. They are formatted as a JSON array of the identifier integers. Each suite MUST appear only once in the array. The cryptosuites MUST be supplied in the order of priority. Peers MUST supply exactly one suite in the Cryptosuitep value, formatted as an identifier integer. The following suites are defined by EAP-NOOB:

| Cryptosuite | Algorithms |
|---|---|
| 1 | Curve25519 [RFC7748], SHA-256 [RFC6234] |

Table 5: EAP-NOOB cryptosuites

Assignment of new values for new cryptosuites MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC5226].

## 4.2. Error codes

The error codes defined by EAP-NOOB are listed in Table 6.

| Error code | Purpose |
|---|---|
| 1001 | Invalid NAI or peer state |
| 1002 | Invalid message structure |
| 1003 | Invalid data |
| 1004 | Unexpected message type |
| 1005 | Unexpected peer identifier |
| 1006 | Unrecognized OOB message identifier |
| 1007 | Invalid ECDH key |
| 2001 | Unwanted peer |
| 2002 | State mismatch, user action required |
| 3001 | No mutually supported protocol version |
| 3002 | No mutually supported cryptosuite |
| 3003 | No mutually supported OOB direction |
| 4001 | MAC verification failure |
| 5001 | Application-specific error |
| 5002 | Invalid server info |
| 5003 | Invalid server URL |
| 5004 | Invalid peer info |

Table 6: EAP-NOOB error codes

Assignment of new error codes MUST be done through IANA with "Specification Required" and "IESG Approval" as defined in [RFC5226].

### 4.3. 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 as follows:

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

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

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

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

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

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

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

## 5. 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.

### 5.1. Authentication principle

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 ECDH. The message authentication codes computed with the secret nonce Noob are alone sufficient for authenticating the key exchange. The OOB channel might, however, be vulnerable to eavesdropping of the OOB channel, which could lead to compromise of the secret nonce, which will then enable a man-in-the-middle attack on the in-band channel. This is why we include, as a second authentication feature, the integrity-protecting fingerprint Hoob in the OOB message. It is typically more difficult to spoof or alter messages on the human-assisted OOB channel, such as bar code, sound burst or user-transferred URL, than it is to spy on them.

The security provided by the cryptographic fingerprint is somewhat intricate to understand. The party that receives the OOB message uses Hoob to verify the integrity of the ECDH exchange. Thus, that party can detect man-in-the-middle attacks on the in-band channel. The other party, 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. Indeed, other device-pairing protocols have solved a similar situation by requiring the user to confirm to the OOB sender that the association was accepted by the OOB-receiver, e.g. by pressing an "accept" button on the sender. Since EAP-NOOB was designed to work strictly with one-directional OOB communication, it does not rely on such input 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. If an attacker is able to eavesdrop the secret nonce and performs a man-in-the-middle attack on the in-band channel, the mismatching fingerprint will alert the OOB receiver, which will reject the OOB message. In this case, the association will appear to be complete only on the OOB sender side. The user in many applications will detect this apparently one-sided association because the peer device does not appear registered on the server or network.

The expected use cases for EAP-NOOB are ones where it replaces a user-entered access credentials. In wireless network access for IoT devices, the user-entered credential is often a passphrase, which is shared by all the network stations. Like any other EAP-based solution, EAP-NOOB establishes a different master secret for each peer device, which is obviously more resilient to device compromise than 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 ECDH 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 should make its decision primarily based on what it knows about the peer's computational capabilities.

## 5.2. Identifying and naming peer devices

EAP-NOOB relies on physical possession or identification of the peer device and secure communication between the user and the server. The main remaining threat against EAP-NOOB is that the attacker performs a man-in-the-middle attack on the in-band channel and, during the protocol execution, tricks the user to deliver the OOB message to or from the wrong peer. The server will now be associated with that wrong peer. Similarly, the attacker could try to trick the user to accessing the wrong server in the OOB Step. This reliance on user in identifying the correct parties is an inherent property of out-of-band authentication.

One mechanism that can be used to mitigate user mistakes is certification of trusted servers and peer devices. For example, if used together with EAP-NOOB, vendor certificates could prevent accidental association with a rogue peer device. Compared to a fully certificate-based authentication, EAP-NOOB does not depend on trusted

third parties and does not require the user to know the identifier of the peer device; physical access is sufficient.

The user could also accidentally deliver the OOB message to more than one peer device. This could, for example, occur if the OOB message is a bar code and the peer is a camera: the user could by mistake show the bar code first to the wrong camera. Such accidents in EAP-NOOB will not enable the wrong camera to compute the master key or to opportunistically eavesdrop the communication. This is because the wrong peer device would need to have performed a man-in-the middle attack on the in-band channel before the accident. In comparison, simpler solutions where the master key is transferred to the device via the OOB channel would be vulnerable to opportunistic attacks if the user mistakenly delivers the master key to more than one device.

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 make and model or serial number. A compromised peer could lie about this information 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 rigorous user checking, the PeerInfo is not authenticated information and should not be relied on. Therefore, it is better to include only minimal information about the peer in PeerInfo and to ask the user to name the peer devices. In many

applications, such as OOB authentication for ad-hoc wireless network access, it may be unnecessary to store any names for the peer device. Since the user delivering the OOB message will often communicate with the server over an authenticated channel, e.g. by logging into a secure web page, the user identity and user-given name can in those cases be reliably stored for the peer device. It is these user identities and user-given names that should be later used for access control and revocation.

Another reason to include only minimal information in the PeerInfo is potential privacy issues. The PeerInfo field is typically transmitted in plaintext between the peer and the authenticator. Although the PeerInfo sent by a new, unregistered device will not leak any information specifically about the user, it could reveal device identifiers and information about other device properties, which the user may want to avoid leaking at this point.

### 5.3. 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 downgrade 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 party, it SHOULD also remember the previously negotiated cryptosuite and not accept renegotiation of any cryptosuite that is known to be weaker than the previous one (e.g. a deprecated cryptosuite or the same ECDH field with a shorter key).

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 ECDH 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 ECDH curve could eventually enable the attacker to recover Kz. EAP-NOOB protect against such attacks by allowing cryptosuite upgrade in the Reconnect Exchange. We do not expect the upgrades to be frequent, but if one becomes necessary, the upgrade can be made without manual resetting and reassociation of the peer devices. During the algorithm upgrade, the shared key material Kz is also updated.

## 5.4. 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 7.

| Security property | EAP-NOOB claim |
|---|---|
| Authentication mechanism | ECDH 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 peer MAC address.) |

Table 7: EAP security claims

# 6. References

## 6.1. 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, <http://www.rfc-editor.org/info/rfc2104>.

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

[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, <http://www.rfc-editor.org/info/rfc3748>.

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

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

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[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, <http://www.rfc-editor.org/info/rfc6234>.

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

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

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

[RFC7542] DeKok, A., "The Network Access Identifier", RFC 7542, DOI 10.17487/RFC7542, May 2015, <http://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, <http://www.rfc-editor.org/info/rfc7748>.

## 6.2. Informative references

[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.

[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, <http://www.rfc-editor.org/info/rfc2904>.

[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>.

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

## 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.

| 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 (no change on error) |
| 3 | - | no change, notify user |
| server state: OOB Received (2) | | |
| 0 | Initial Exchange | both 1 (0 on error) |
| 1 | Completion Exchange | both 4 (no change on error) |
| 2 | Completion Exchange | both 4 (no change on error) |
| 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) |

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. The OOB message events have been marked with asterisk (*) to indicate that events are only possible if allowed by the negotiated OOB directions (Dirp). 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 algorithms) 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 8 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 8: OOB channel characteristics

## Appendix C. 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 in 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.

## Appendix D. 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 client-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 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 E. Example messages

The message examples in this section are generated with Curve25519 ECDH test vectors specified in section 6.1 of [RFC7748] (server=Alice, peer=Bob). The direction of the OOB channel negotiated is 2 (server-to-peer). 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":"qJjnijxpojI0dti6qGfOIb","Cryptosuit
es":[1],"Dirs":3,"ServerInfo":{"Name":"Example","Url":"https://exa
mple.com/Noob"},"realm":"noob.example.com"}

EAP request (type 1):

{"Type":1,"Vers":[1],"PeerId":"qJjnijxpojI0dti6qGfOIb","Cryptosuit
es":[1],"Dirs":3,"ServerInfo":{"Name":"Example","Url":"https://exa
mple.com/Noob"},"realm":"noob.example.com"}

EAP response (type 1):

{"Type":1,"Verp":1,"PeerId":"qJjnijxpojI0dti6qGfOIb","Cryptosuitep
":1,"Dirp":2,"PeerInfo":{"Make":"Acme","Type":"None","Serial":"DU-
8448","SSID":"Noob2","BSSID":"6c:19:8f:83:c2:90"}}

EAP request (type 2):

{"Type":2,"PeerId":"qJjnijxpojI0dti6qGfOIb","Ns":"1htC1L24K-
jkP9bEgDAEnnmK44ltlY1XwKooEOxs-5c","jwk":{"kty":"EC","crv":"Curve2
5519","x":"MCowBQYDK2VuAyEAhSDwCYkwp1R0i33ctD73Wg2_Og0mOBr066Spjqq
bTmo"},"SleepTime":60}

EAP response (type 2):

{"Type":2,"PeerId":"qJjnijxpojI0dti6qGfOIb","Np":"ppe-KZ_-Xdz8bwEb
_vfnny2dKkMepFbiLgf3xduVxxo","jwk":{"kty":"EC","crv":"Curve25519",
"x":"MCowBQYDK2VuAyEA3p7bfXt9wbTTW2HC7OQ1Nz-DQ8hbeGdNrfx-FG-
IK08"}}

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

Identity response:

qJjnijxpojI0dti6qGfOIb+s1@noob.example.com

EAP request (type 3):

{"Type":3,"PeerId":"qJjnijxpojI0dti6qGfOIb","SleepTime":60}

EAP response (type 3):

{"Type":3,"PeerId":"qJjnijxpojI0dti6qGfOIb"}

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

Identity response:

data:,P=qJjnijxpojI0dti6qGfOIb&N=sqfpPmEXh4iPx23oY0t_Lg&H=Y2MxZDc2
MDUzNTNkMTE3Mg

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

Identity response:

qJjnijxpojI0dti6qGfOIb+s2@noob.example.com

```
   EAP request (type 8):
      {"Type":8,"PeerId":"qJjnijxpojI0dti6qGfOIb"}

   EAP response (type 8):
      {"Type":8,"PeerId":"qJjnijxpojI0dti6qGfOIb","NoobId":"NzMxMTNkZGFl
      NjhiMmNmYg"}

   EAP request (type 4):
      {"Type":4,"PeerId":"qJjnijxpojI0dti6qGfOIb","NoobId":"NzMxMTNkZGFl
      NjhiMmNmYg","MACs":"3Kf-OEE0l_SMC1C2XYdQmg"}

   EAP response (type 4):
      {"Type":4,"PeerId":"qJjnijxpojI0dti6qGfOIb","MACp":"z6G--
      hg8U3dhSwjYJ8xIFA"}

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

   Identity response:
      qJjnijxpojI0dti6qGfOIb+s3@noob.example.com

   EAP request (type 5):
      {"Type":5,"Cryptosuites":[1],"PeerId":"qJjnijxpojI0dti6qGfOIb","Se
      rverInfo":{"Name":"Example","Url":"https://example.com/Noob"}}

   EAP response (type 5):
      {"Type":5,"PeerId":"qJjnijxpojI0dti6qGfOIb","Cryptosuitep":1,"Peer
      Info":{"Make":"Acme","Type":"None","Serial":"DU-
      8448","SSID":"Noob2","BSSID":"6c:19:8f:83:c2:90"}}

   EAP request (type 6):
      {"Type":6,"PeerId":"qJjnijxpojI0dti6qGfOIb","Ns":"RPmLvG79U-
      GfBZTbzbaYMhC1hEE4_lj9SEtrLAct-3w"}

   EAP response (type 6):
      {"Type":6,"PeerId":"qJjnijxpojI0dti6qGfOIb","Np":"u7ecFNRibJ0PnOn2
      zIXmpZrNwlykbqzfyGWZeaRq1MQ"}

   EAP request (type 7):
      {"Type":7,"PeerId":"qJjnijxpojI0dti6qGfOIb","MACs":"EL4hd-
      PMY_RL9pmfZqRffg"}

   EAP response (type 7):
      {"Type":7,"PeerId":"qJjnijxpojI0dti6qGfOIb","MACp":"rjMw0xRebtOYWH
      bgcFeGEw"}
```

## Appendix F. Version history

- Version 01:
  - Fixed Reconnection Exchange.
  - URL examples.
  - Message examples.
  - Improved state transition (event) tables.
- Version 02:
  - Reworked the rekeying and key derivation.
  - Increased internal key lengths and in-band nonce and MAC 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.

Authors' Addresses

Tuomas Aura
Aalto University
Aalto 00076
Finland

EMail: tuomas.aura@aalto.fi

Mohit Sethi
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

EMail: mohit@piuha.net