Workgroup: Internet Engineering Task Force
Internet-Draft: draft-rieckers-emu-eap-ute-01
Published: 22 September 2022
Intended Status: Standards Track
Expires: 26 March 2023
Authors: J.-F. Rieckers
DFN

User-assisted Trust Establishment (EAP-UTE)

Abstract

The Extensible Authentication Protocol (EAP) provides support for multiple authentication methods. This document defines the EAP-UTE authentication method for a User-assisted Trust Establishment between the peer and the server. The EAP method is intended for bootstrapping Internet-of-Things (IoT) devices without preconfigured authentication credentials. The trust establishment is achieved by transmitting a one-directional out-of-band (OOB) message between the peer and the server to authenticate the in-band exchange. The peer must have a secondary input or output interface, such as a display, camera, microphone, speaker, blinking light, or light sensor, so that dynamically generated messages with tens of bytes in length can be transmitted or received.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rieckers-emu-eap-ute/.

Discussion of this document takes place on the EAP Method Update (emu) Working Group mailing list (mailto:emu@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/emu/.

Status of This Memo

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

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

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

This Internet-Draft will expire on 26 March 2023.

Copyright Notice

Table of Contents

1. Introduction

This document describes a method for registration, authentication, and key derivation for network-connected devices, especially with low computational power and small or no interaction interfaces, such as devices that are part of the Internet of Things (IoT). These devices may come without preconfigured trust anchors or have no possibility to receive a network configuration that enables them to connect securely to a network.

This document uses the basic design principle behind the EAP-NOOB method described in [RFC9140] and aims to improve some key elements of the protocol to better address the needs for IoT devices. This is mainly achieved by using CBOR with numeric keys instead of JSON to encode the exchanged messages and by modifying the message flow.

TODO: The EAP-UTE protocol also allows extensions, they are still TBD. Basically, the messages can just include additional fields with newly defined meanings.

The possible problems of EAP-NOOB are discussed in [I-D.draft-rieckers-emu-eap-noob-observations]. This document provides a specification which aims to address these concerns.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

TODO frequently used terms

authenticator

peer

server

3. EAP-UTE protocol

This section defines the EAP-UTE method.

3.1. Protocol Overview

TODO: The introduction text is basically copied from RFC9140. Should be reworded.

The EAP-UTE method execution spans two or more EAP conversations, called Exchanges in this specification. Each Exchange consists of several EAP request-response pairs. In order to give the user time to deliver the OOB message between the peer and the server, at least two separate EAP conversations are needed.

The overall protocol starts with a version and cryptosuite negotiation and peer detection. Depending on the current state of the peer and server, different exchanges are selected.

If the server or the peer are in the unregistered state, peer and server exchange nonces and keys for the Ephemeral Elliptic Curve Diffie-Hellman. This is called the Initial Exchange. The Initial Exchange results in an EAP-Failure, since neither the server nor the peer are authenticated.

After the Initial Exchange, the user-assisted step of trust establishment takes place. The user delivers one OOB message either from the peer to the server or from the server to the peer.

While peer and server are waiting for completion of the OOB Step, the peer **MAY** probe the server by reconnecting, to check for successful transmission of the OOB message. This probe request will result in a Waiting Exchange and EAP-Failure, if the server has not yet received the OOB message.

If either the server or the peer have received the OOB message, the probe request will result in a Completion Exchange. In the Completion Exchange, peer and server exchange message authentication codes over the previous in-band messages and the OOB message. 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.

After a successful Completion Exchange, the peer and server can use the Reconnect Exchange, to create a new association with new cryptographic bindings. The user-assisted OOB step is not necessary, since the peer and server can infer the mutual authentication by using the persistent data stored after the Completion Exchange.

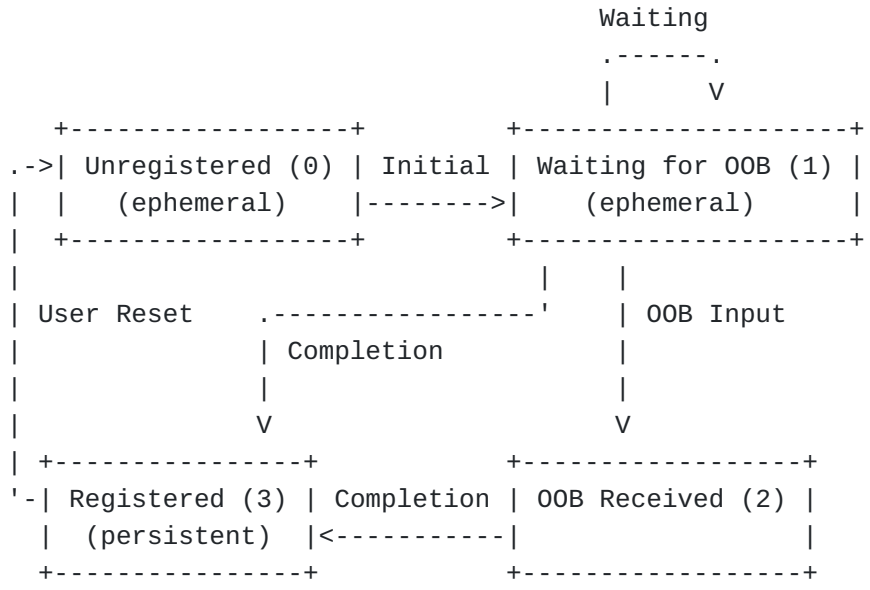

Figure 1: EAP-UTE Server-Peer Association State Machine

3.2. Messages

3.2.1. General Message format

All EAP-UTE messages consist of a CBOR Sequence with the following elements

type: integer to indicate the type of the message

payload: a byte-string containing a CBOR encoded map

additional: an optional byte-string containing additional information, e.g. a message authentication code, encoded as CBOR map

Remark from the author:
This format is just a first draft. It allows a very simple MAC calculation, since the MACs can just consist of the concatenated previous messages. This also allows an easy addition of extensions, since the extension payloads are automatically included in the MAC calculation, if they are part of the CBOR payload. Additionally, the additional section allows for extensions that do not need integrity protection, e.g. for referal to a different server in the background.

The message payloads are encoded in CBOR [RFC8949] as maps. In Table 1 the different message fields, their assigned mapkey and the type are listed.

Mapkey	Type	Label	Description
1	Array of Integers	Versions	The versions supported by the server. For this document the version is 1
2	Integer	Version	The version selected by the peer
3	Array	Ciphers*	The ECDHE curves and Hash algorithms supported by the server.
4	Array	Cipher*	The ECHDE curve and Hash algorithm selected by the peer
5	Integer	Directions	The OOB-Directions supported by the server. 0x01 for peer-to-server, 0x02 for server-to-peer, 0x03 for both
6	Integer	Direction	The OOB-Direction selected by the peer. **SHOULD** be either 0x01 or 0x02, but **MAY** be 0x03 for both directions
7	Map	ServerInfo	Information about the server, e.g. a URL for OOB-message-submission
8	Map	PeerInfo	Information about the peer, e.g. manufacturer/serial number
9	bytes	Nonce_P	Peer Nonce
10	bytes	Nonce_S	Server Nonce
11	Map	Key_P**	Peer's ECDHE key according to the chosen cipher
12	Map	Key_S**	Server's ECDHE key
13	null/ bytes	MAC_S***	Indication that Server MAC is included (null value) and byte string in additional section
14	null/ bytes	MAC_P***	Indication that Peer MAC is included (null value) and byte string in additional section
15	text	PeerId	Peer Identifier
16	bytes	OOB-Id	Identifier of the OOB message
17	int	RetryInterval	Number of seconds to wait after a failed Completion Exchange
18	Map	AdditionalServerInfo	

Mapkey	Type	Label	Description
			Additional information about the server. TODO: not sure about this yet.

Table 1: Mapkeys for CBOR encoding

*: The Ciphers field consists of an array of two arrays. The first array contains all supported ECDHE curves using the identifiers from the COSE Elliptic Curves registry. The server **MUST NOT** offer and the peer **MUST NOT** accept curves not suited for ECDH. The second array contains all supported Hash algorithms using the indentifiers from the COSE Algorithms registry. The server **MUST** only offer and the peer **MUST** only accept Hash algorithms. The Cipher field consists of an array of two items, first the selected ECDHE curve, second the selected Hash algorithm.

**: The peer and server Key are encoded as COSE key [RFC8152].

***: The inclusion of MAC_S or MAC_P map keys with a NULL map value indicate that the MAC value is included in the additional field of the CBOR sequence. The MAC field is encoded with the same map key as byte string, its length is determined by the used cryptosuite.

A message **MUST NOT** contain both MAC_S and MAC_P, only one of these values can be present in a message.

3.2.1.1. Thoughts about the message format

EAP-NOOB [RFC9140] uses JSON as encoding. Problems of using JSON are discussed in section 2.1 of [I-D.draft-rieckers-emu-eap-noob-observations].

For this specification, the following encodings have been considered:

*Static encoding
This allows a minimal number of bytes and requires a minimal amount of parsing, since the format and order of the message fields is specified exactly. However, this encoding severely affects the extensibility, unless a specific extension format is used. The specification of this protocol also has optional fields in some message types, so this would also have to be addressed.

*CBOR with static fields (e.g. Array)
This approach has a slightly higher number of bytes than the static encoding, but allows an easier extensibility. The required fields can be specified, so the order of the protocol field is static and a parser has minimal effort to parse the protocol fields. However, this might be problematic in future protocol

versions, when new fields are introduced. Like with static encoding, this also requires a mechanism for optional fields in the different message types.

*CBOR map with numeric keys
To mitigate the problems of optional fields while keeping the parsing effort low, CBOR maps with numeric keys can be used. All protocol fields are identified by a unique identifier, specified in this document. A parser can simply loop through the CBOR map. Since CBOR maps have a canonical order, minimal implementations may rely on this fact to parse the information needed.

On the basis of this discussion, this draft will use a CBOR map as message encoding. However, this is just a first draft and suggestions for other message formats are highly welcome.

3.2.2. Server greeting

*Message Type: 1

*Required Attributes:

-Versions

-Ciphers

-Directions

*Optional Attributes:

-ServerInfo

-RetryInterval?

3.2.3. Client greeting

*Message Type: 2

*Required Attributes:

-Version

-Cipher

-Nonce_P

-Key_P

*Optional Attributes:

-PeerId

-PeerInfo

-Direction

3.2.4. Server Keyshare

*Message Type: 3

*Required Attributes:

-Key_S

-Nonce_S

*Optional Attributes:

-MAC_S

-PeerId

-AdditionalServerInfo?

-RetryInterval?

3.2.5. Client Finished

*Message Type: 4

*Optional Attributes:

-MAC_P

3.2.6. Client Completion Request

*Message Type: 5

*Required Attributes:

-Nonce_P

-PeerId

*Optional Attributes:

-OOB-Id

3.2.7. Server Completion Response

*Message Type: 6

*Required Attributes:

-Nonce_S

-MAC_S

*Optional Attributes:

-OOB-Id

3.2.8. Client Keyshare

*Message Type: 7

*Required Attributes:

-PeerId

-Nonce_P

-Key_P

3.3. Protocol Sequence

After reception of the EAP-Response/Identity packet, the server always answers with a Server Greeting packet (Type 1). This Server Greeting contains the supported protocol versions, ciphers and OOB directions along with the ServerInfo.

Depending on the peer state, the peer chooses the next packet. If the peer is in the unregistered state and thus does not yet have an ephemeral or persistent state, it chooses the Client Greeting, which starts the Initial Handshake.

If the peer is in the Waiting for OOB or OOB Received state, the Initial Exchange has completed and the OOB step needs to take place. If the negotiated direction is from server to peer, the peer **SHOULD NOT** try to reconnect until the peer received an OOB message. If the negotiated direction is from peer to server, the peer can probe the server at regular intervals to check if the OOB message to the server has been delivered. The peer will send a Client Completion Request to initiate the Waiting/Completion Exchange.

If the peer is in the Registered state, it may choose between three different Reconnect Exchanges. If the peer wants a reconnect without new key exchanges, it will send a Client Completion Request, starting the Reconnect Exchange without ECDHE. If the peer wants to reconnect with new key exchanges, it will send a Client Key Share packet, which starts the Reconnect Exchange with new ECDHE exchange. If the peer wants to upgrade to a new protocol version or change the

used cipher suites, it will send a Client Greeting, starting the Upgrade exchange.

3.3.1. Initial Exchange

The Initial Exchange comprises of the following packets:

After the Server Greeting common to all exchanges, the peer sends a Client Greeting packet. The Client Greeting contains the peer's chosen protocol version, cipher and direction of the OOB message. The peer **MUST** only choose values for these fields offered by the server in it's Server Greeting. For Direction the peer **SHOULD** choose either 0x01 (peer-to-server) or 0x02 (server-to-peer) if the server offered 0x03 (both directions). Additionally, the Client Greeting contains PeerInfo, a nonce and the peer's ECDHE public key.

The server will then answer with a Server Keyshare packet. The packet contains a newly allocated PeerId, the server's nonce and the ECDHE public key.

The peer then answers with a Client Finished packet without any payload.

Since no authentication has yet been achieved, the server then answers with an EAP-Failure.

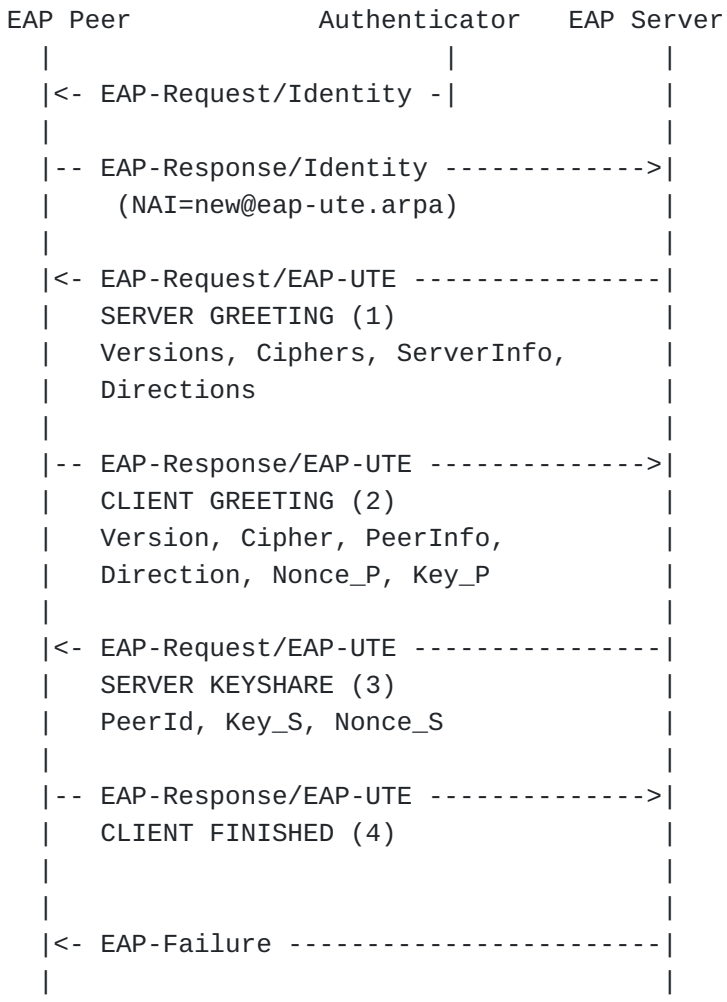

Figure 2: Initial Exchange

3.3.2. User-assisted out-of-band step

After the completed Initial Exchange, the peer or the server, depending on the negotiated direction, will generate an OOB message.

This message consists of a 16-byte freshly generated nonce (OOB-Nonce), the authentication value OOB-Auth and the PeerId. The calculation of the OOB-Auth field is described in Section 3.4.

The devices **MAY** also include the OOB-Id field, if size of the OOB message is not important.

Devices **SHOULD** export the message as a CBOR sequence of byte strings in the order PeerId, OOB-Nonce, OOB-Auth (, optionally OOB-Id). The format of the OOB message **MAY** be altered by the application, depending on the available interfaces.

3.3.3. Waiting Exchange

The Waiting Exchange is performed if neither the server nor the peer have received an OOB message yet.

The peer probes the server with a Client Completion Request. In this packet the peer omits the optional OOB-Id field. If the OOB message is delivered from the peer to the server, the server may have received an OOB message already. To allow the server to complete the association, the peer includes a nonce, along with the allocated PeerId. The nonce **MAY** be repeated for all Client Completion Requests while waiting for the completion, but **MUST** be recalculated if a new initial handshake is performed.

If the server did not receive an OOB message, it answers with an EAP-Failure.

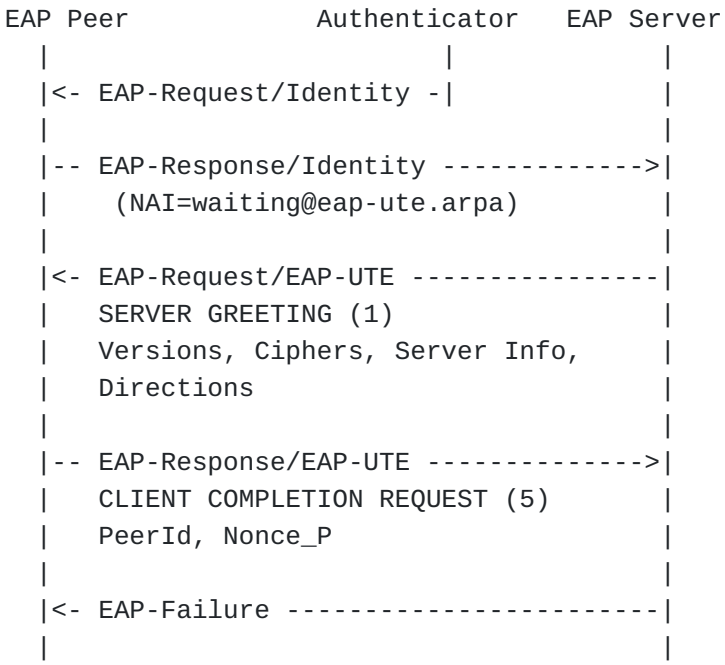

Figure 3: Waiting Exchange

3.3.4. Completion Exchange

The Completion Exchange is performed to finish the mutual trust establishment.

As in the Waiting Exchange, the peer probes the server with a Client Completion Request. The nonce of the previous Client Completion Requests which did not lead to a completion **MAY** be repeated. If the peer has received an OOB message, the peer will include the OOB-Id in the Completion Request. If the peer did not include an OOB-Id, the server will include the OOB-Id of its received OOB message. In

the unlikely case that both directions are negotiated and an OOB message is delivered from the peer to the server and from the server to the peer at the same time, as a tiebreaker, the OOB message from the server to the peer is chosen.

The server generates a new nonce, calculates MAC_S according to Section 3.4 and sends a Server Completion Response to the peer.

The peer will then calculate the MAC_P value and send a Client Finished message to the server.

After checking the MAC_P value, the server then answers with an EAP-Success.

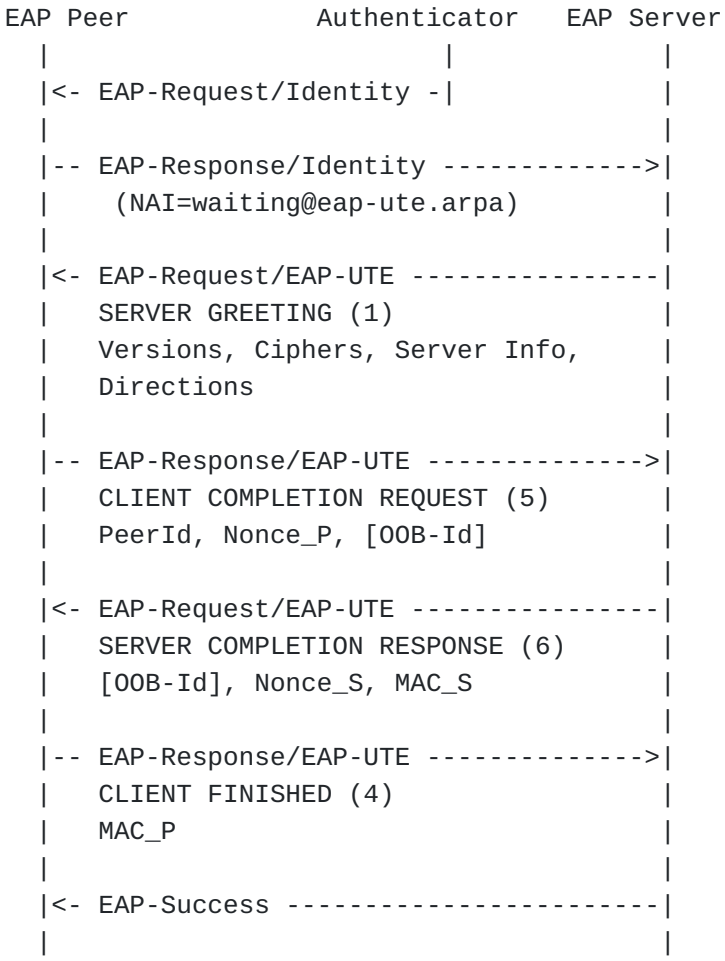

Figure 4: Completion Exchange

3.3.5. Reconnect Exchange

The Reconnect Exchange is performed if both the peer and the server are in the registered state.

For a reconnect without new exchanging of ECDHE keys, the peer will answer to the Server Greeting with a Client Completion Request, including the PeerId and a nonce.

To distinguish a Reconnect Exchange from a Waiting/Completion Exchange, the server will look up the saved states for the transmitted PeerId. If the server has a persistent state saved, it will choose the Reconnect Exchange, otherwise it will choose the Waiting Exchange.

The server will then generate a nonce and the MAC_S value according to Section 3.4 and send a Server Completion Response with the nonce and MAC_S value.

The peer then sends a Client Finished message, containing the computed MAC_P value.

The server then answers with an EAP-Success.

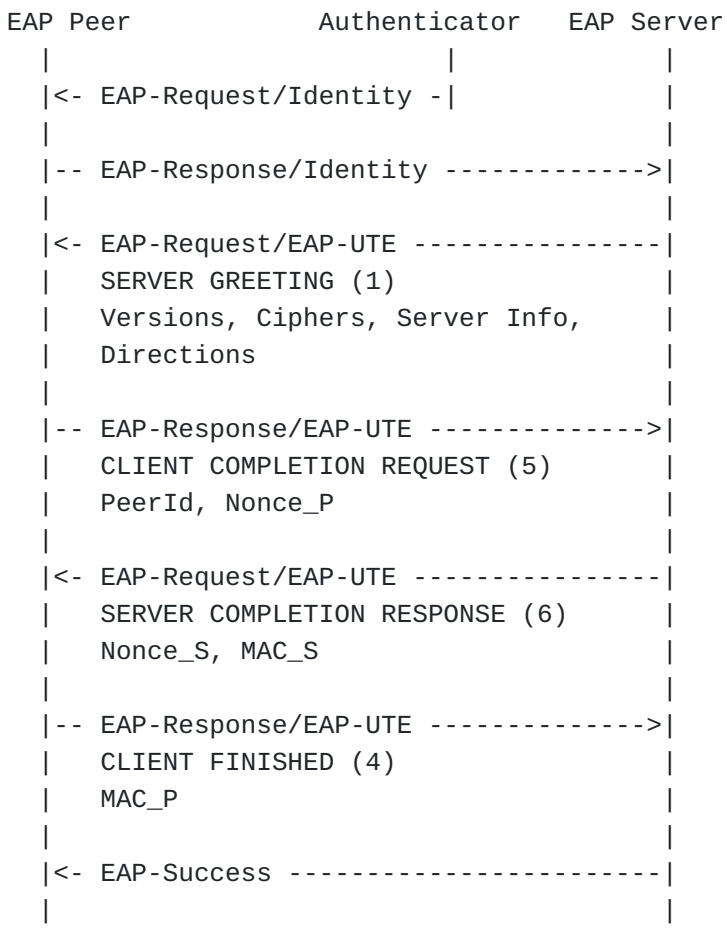

Figure 5: Reconnect Exchange without new ECDHE exchange

For a Reconnect Exchange with new ECDHE exchange, the peer will send a Client Keyshare in response to the Server Greeting. The Client Keyshare will include the PeerId, a nonce and a new ECDHE key.

The server will also generate a new ECDHE key, a nonce and compute MAC_S according to Section 3.4.

The peer will then calculate the MACs and keying material according to Section 3.4.3 and send a Client Finished message to the server, including its MAC_P value.

The server checks the MAC_P value and answers with an EAP-Success.

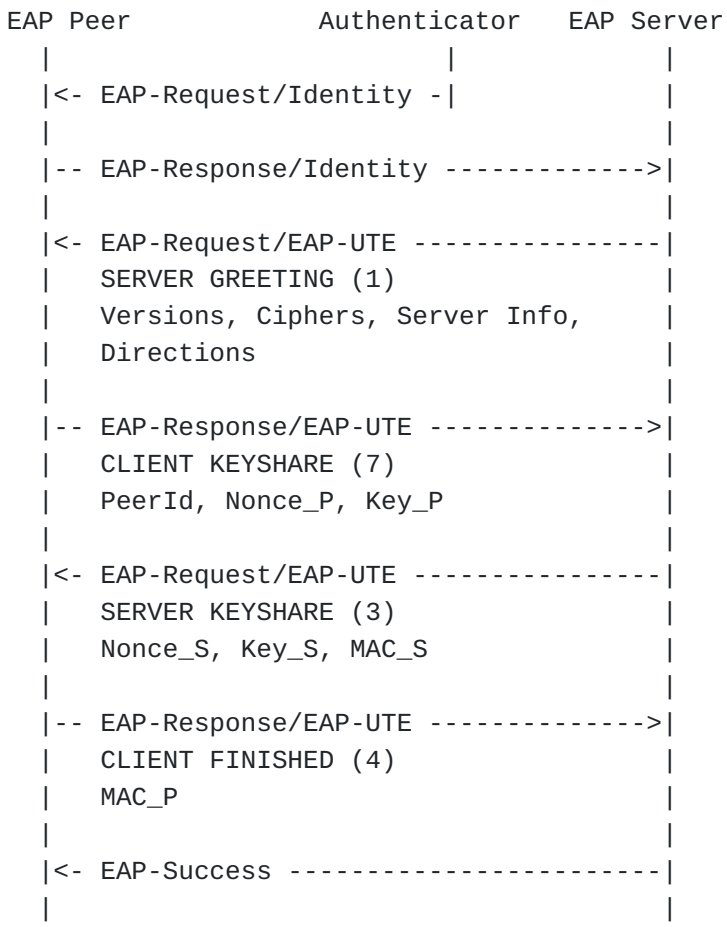

Figure 6: Reconnect Exchange with new ECDHE exchange

3.3.6. Upgrade Exchange

The Upgrade Exchange is performed to upgrade either the EAP-UTE version or the used cipher suite, or refresh the cryptographic keying material.

A client may choose to perform this exchange instead of a reconnect exchange. The client **SHOULD** only choose this if the server offers a

better version or cipher suite in its Server Greeting or if the current set of cryptographic keys has been used for an application specific amount of reconnect exchanges or time.

If the client cooses the Upgrade Exchange, it answers to the Server Greeting with a Client Greeting and includes the PeerId field. To distinguish the Upgrade Exchange from the Intial Exchange, the server will look up the PeerId and, if a persistent association is found, answer with its Server Keyshare, including the optional MAC_S field, calculated according to Section 3.4.

The peer will then calculate the MACs and keying material according to Section 3.4.3 and send a Client Finished message to the server, including its MAC_P value.

The server checks the MAC_P value of the client and answers with an EAP-Success. Afterwards the server updates the association stored for the client.

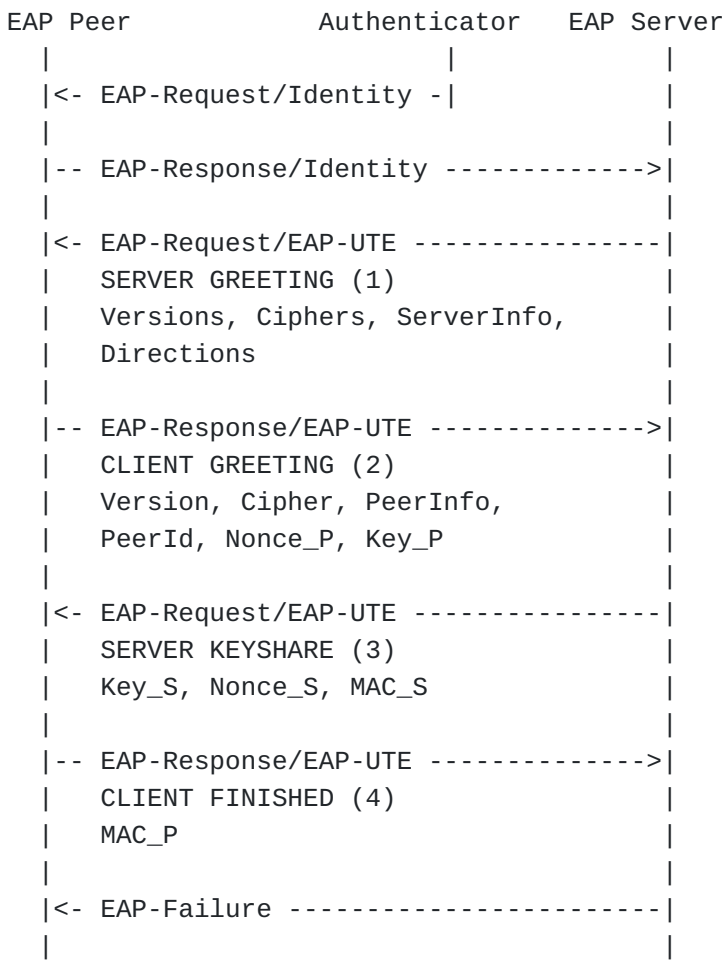

Figure 7: Upgrade Exchange

3.4. MAC and OOB calculation and Key derivation

3.4.1. MAC Calculation

For the MAC calculation, the exchanged messages up to the current message are concatenated into the "Messages" field. This field consists for each message of the CBOR sequence of the message type and the CBOR encoded message payload as byte-string. The optional MAC value at the end of the message is omitted.

For the following definition || denotes a concatenation.

Messages = Type_1 || Length_1 || Payload_1 || ... || Type_n || Length_n || Payload_n

The Messages field is calculated separately for each exchange, i.e. the Messages field for the Initial Exchange will include the Server Greeting, Client Greeting, Server Keyshare and Client Finished message.

The OOB-Auth field is calculated as hash over the concatenation of the Messages field of the Initial Exchange, the generated OOB-Nonce and the direction of the Out-of-band message. The length of the OOB-Auth field is determined by the used hash algorithm.

OOB-Auth = H(Messages || OOB-Nonce || Direction)

The OOB-Id, used to identify the used OOB message, is calculated over the string "OOB-Id" concatenated with the OOB-Auth field. The hash result is truncated to 16 bytes.

OOB-Id = H("OOB-Id" || OOB-Auth)[0..15]

For the calculation of the MAC_S and MAC_P value, the Messages field will only include the messages sent up to the point of the MAC calculation. For MAC_S this also includes the Server Keyshare/Server Completion Response message. For MAC_P the Client Finished message is omitted from the Messages field, so both MAC_P and MAC_S have the same input for the Messages field.

Depending on the performed Exchange, the MAC calculation differs. For the Completion Exchange, the MAC calculation includes the direction (0x01 for peer to server, 0x02 for server to peer), a Hash of the Messages field of the Initial Exchange, a hash of the Messages of the Completion exchange and the OOB-Nonce.

MAC_P = HMAC(K_p, 0x01 || H(Messages (Initial Exchange)) || H(Messages) || OOB-Nonce)

MAC_S = HMAC(K_s, 0x02 || H(Messages (Initial Exchange)) || H(Messages) || OOB-Nonce)

For the reconnect exchanges the MAC calculation will include only the direction and a hash of the Messages field of the Reconnect Exchange.

MAC_P = HMAC(K_p, 0x01 || H(Messages))

MAC_S = HMAC(K_s, 0x02 || H(Messages))

3.4.2. Key derivation

The key derivation is performed differently depending on the performed Exchange.

Performed Exchange	KDF input field	Value	Length (bytes)
Completion Exchange	Z	ECDHE shared secret from Key_P and Key_S	variable
	AlgorithmId	"EAP-UTE"	7
	PartyUInfo	Nonce_P	32
	PartyVInfo	Nonce_S	32
	SuppPrivInfo	OOB-Nonce	32
Reconnect Exchange without ECDHE	Z	Association_Key	32
	AlgorithmId	"EAP-UTE"	7
	PartyUInfo	Nonce_P	32
	PartyVInfo	Nonce_S	32
	SuppPrivInfo	(null)	0
Reconnect Exchange with new ECHDE exchange or cryptosuite change	Z	ECDHE shared secret from Key_P and Key_S	variable
	AlgorithmId	"EAP-UTE"	7
	PartyUInfo	Nonce_P	32
	PartyVInfo	Nonce_S	32
	SuppPrivInfo	Association_Key	32

Table 2

The output of the key derivation also depends on the used exchange method.

Performed Exchange	KDF output bytes	Used as	Length (bytes)
Completion Exchange or Upgrade Exchange	0..63	MSK	64

Performed Exchange	KDF output bytes	Used as	Length (bytes)
	64..127	EMSK	64
	128..191	AMSK	64
	192..223	MethodId	32
	224..255	K_s	32
	256..287	K_p	32
	288..319	Association_Key	32
Reconect exchanges	0..63	MSK	64
	64..127	EMSK	64
	128..191	AMSK	64
	192..223	MethodId	32
	224..255	K_s	32
	256..287	K_p	32

Table 3

3.4.3. Updating of keying materials

The client and server commit to new keying material at different positions in the protocol. If the final Client Finished message is lost, this leads to the client committing to the change and the server keeping the old state.

To circumvent this issue, the client will save the previous keying material until the change is authenticated by a following reconnect exchange.

Upon Reception of the MAC_S value from the server in a Reconnect or Upgrade Exchange, the client will perform the following steps:

*The client will execute the key derivation using the current Association_Key and calculate the MAC_S value. If the received MAC_S value matches the locally computed one, the client purges the Prev_Association_Key, Prev_Cipher and Prev_Version values, if they are present. The previous values can be purged since this authentication proves that the server committed to the state change in a previous Upgrade Exchange. Afterwards the client calculates the MAC_P value and sends the Client Finished message.

*If the MAC_S value does not match, and the Prev_* values are empty, the client sends an error message and aborts the key derivation.

*The client will execute the key derivation using the Prev_Association_Key and calculate the MAC_S value. If the received MAC_S value matches the new locally computed MAC_S, this indicates that the server has not commited to a previous update of cryptographic keys in the last Upgrade Exchange. As a result, the client will move the values of Prev_Association_Key,

Prev_Cipher and Prev_Version values to Association_Key, Cipher and Version and pures the Prev_* values. Afterwards the client calculates the MAC_P value and sends the Client Finished message.

*If the second MAC_S value did not match either, the client sends an error message and aborts the key derivation.

*Finally, if the current exchange is an Upgrade Exchange, the client will save the newly generated Association_Key along with the current cipher and version into the persistent storage. The previous values of these fields are moved to the Prev_* slots.

3.5. Error handling

TBD

4. Security Considerations

This document has a lot of security considerations, however they remain TBD

4.1. EAP Security Claims

TODO. See [RFC3748], section 7.2.1

5. IANA Considerations

This document has IANA actions, if approved. What they are exactly needs to be defined in detail.

The EAP Method Type number for EAP-UTE needs to be assigned. The reference implementation will use 255 (Experimental) for now.

Like EAP-NOOB, this draft will probably use a .arpa domain, in this case probably eap-ute.arpa, as default NAI realm.

Additionally, the IANA should create registries for the message types and the message field mapkeys.

6. Implementation Status

Note to RFC Editor: Please remove this entire section before publication.

As of now, only a partial implementation exists.

6.1. Server Implementation of EAP-UTE

*Responsible Organization: DFN-Verein/University of Bremen

*Location: https://github.com/namib-project/eap-noob-ute-server

*Coverage: Only Initial and Completion Exchange implemented

*Level of Maturity: research

*Version Compatibility: Version 01 of the individual draft partially implemented

*Licensing: MIT / Apache 2.0

*Contact Information: Jan-Frederik Rieckers, rieckers@dfn.de

6.2. Client Implementation in ESP-IDF

*Responsible Organization: DFN-Verein/University of Bremen

*Location: https://github.com/namib-project/esp-idf/tree/namib/eap-ute

*Coverage: Only Initial and Completion Exchange implemented

*Level of Maturity: research

*Version Compatibility: Version 01 of the individual draft partially implemented

*Licensing: Apache 2.0

*Contact Information: Jan-Frederik Rieckers, rieckers@dfn.de

7. Differences to RFC 9140 (EAP-NOOB)

In this section the main differences between EAP-NOOB and EAP-UTE are discussed. Some problems of [RFC9140] are discussed in [I-D.draft-rieckers-emu-eap-noob-observations].

7.1. Different encoding

EAP-UTE uses CBOR instead of JSON. More text TBD.

7.2. Implicit transmission of peer state

In EAP-NOOB all EAP exchanges start with the same common handshake, which mainly serves the purpose of detecting the current peer state.

The server initiates the EAP conversation by sending a Type 1 message without any further content, to which the peer responds by sending its PeerId, if it was assigned, and its PeerState.

In EAP-UTE, this peer state transmission is done implicitly by the peer's choice of response to the Server Greeting.

This adds probably unnecessary bytes in the first packet from the server to the peer, since the peer already knows the server's supported versions, ciphers and the ServerInfo in the later exchanges, especially in the Waiting/Completion Exchange. However, this increased number of bytes is negligible in comparison to the elevated expense of an additional roundtrip, since this would significantly increase the authentication time, especially if the EAP packets are routed through a number of proxies.

7.3. Extensibility

The EAP-NOOB standard does not specify how to deal with unexpected labels in the message, which could be used to extend the protocol. This specification will explicitly allow extensions. They are still TBD.

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, <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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

8.2. Informative References

[I-D.draft-rieckers-emu-eap-noob-observations] Rieckers, J., "Observations about EAP-NOOB (RFC 9140)", Work in Progress, Internet-Draft, draft-rieckers-emu-eap-noob-observations-00, 7 March 2022, <https://www.ietf.org/archive/id/draft-rieckers-emu-eap-noob-observations-00.txt>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC9140] Aura, T., Sethi, M., and A. Peltonen, "Nimble Out-of-Band Authentication for EAP (EAP-NOOB)", RFC 9140, DOI 10.17487/RFC9140, December 2021, <https://www.rfc-editor.org/info/rfc9140>.

Acknowledgements

TBD

Author's Address

Jan-Frederik Rieckers
Deutsches Forschungsnetz | German National Research and Education Network
Alexanderplatz 1
10178 Berlin
Germany

Email: rieckers@dfn.de
URI: www.dfn.de