ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 25, 2021
R. Marin
University of Murcia
D. Garcia
University of Oviedo
January 21, 2021

EAP-based Authentication Service for CoAP
draft-marin-ace-wg-coap-eap-07

Abstract

This document describes an authentication service that uses EAP transported by means of CoAP messages with the following purposes:

o Authenticate a CoAP-enabled device that enters a new security domain against a AAA infrastructure through a domain Controller.

o Bootstrap key material to protect CoAP messages exchanged between them.

o Enable the establishment of Security Associations between them.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
  1.1. Requirements Language 3
2. General Architecture 3
3. General Flow Operation 4
  3.1. EAP over CoAP: Running an OSCORE Security Association . . 4
  3.2. The SeqNum Option 7
4. Key Derivation for protecting CoAP messages 8
  4.1. Deriving the OSCORE Security Context 8
5. Use Case Scenario 10
6. Discussion 11
  6.1. CoAP as EAP lower-layer 11
  6.2. Size of the EAP lower-layer vs EAP method size 12
  6.3. Controller as the CoAP Client 12
  6.4. Possible Optimizations 12
    6.4.1. Empty Token 13
    6.4.2. Removing SeqNum Option 13
    6.4.3. Further re-authentication 14
7. Security Considerations 14
  7.1. Authorization 14
  7.2. Cryptographic suite selection 14
  7.3. Freshness of the key material 15
8. IANA Considerations 15
9. Acknowledgments 15
10. References 15
  10.1. Normative References 15
  10.2. Informative References 18
Appendix A. CoAP-EAP with DTLS 18
  A.1. Deriving DTLS_PSK 20
Authors' Addresses 20

1. Introduction

The goal of this document is to describe an authentication service that uses the Extensible Authentication Protocol (EAP) [RFC3748]. The authentication service is built on top of the Constrained Application Protocol (CoAP) [RFC7252] and allows authenticating two CoAP endpoints by using EAP without the need of additional protocols to bootstrap a security association between them.

In particular, the document describes how CoAP can be used as a constrained link-layer independent EAP lower-layer [RFC3748] to transport EAP between a CoAP server (EAP peer) and the CoAP client (EAP authenticator) using CoAP messages. The CoAP client MAY contact with a backend AAA infrastructure to complete the EAP negotiation as described in the EAP specification [RFC3748].

The assumption is that the EAP method transported in CoAP MUST generate cryptographic material [RFC5247] . In this way, the CoAP messages can be protected. There are two approaches that we have considered in this document:

- o To define how the OSCORE security association can be established based on the cryptographic material generated from the EAP authentication.

- o To establish a DTLS security association using the exported cryptographic material after a successful EAP authentication. [I-D.ohba-core-eap-based-bootstrapping]

This document also provides some comments about implementation of a proof-of-concept of this preliminary idea

1.1. Requirements Language

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

2. General Architecture

The Figure 1 shows the architecture defined in this document. Basically a node acting as the EAP peer wants to be authenticated by using EAP. At the time of writing this document, we have considered a model where the EAP peer will act as CoAP server for this service and the EAP authenticator will act as CoAP client and MAY interact with a backend AAA infrastructure, which will place the EAP server and contain the information required to authenticate the CoAP client. The rationale behind this decision, as we will expand later, is that EAP requests go always from the EAP authenticator and the EAP peer and the EAP responses from the EAP peer to the EAP authenticator. Nevertheless, a model where the EAP peer act as CoAP client and the EAP authenticator as CoAP server can be also analyzed in the future.

```
       +------------+        +------------+       +--------------+
       | EAP peer/  |        | EAP auth./ |       |  EAP server/ |
       | CoAP server|+------+| CoAP client|+-----+|  AAA server  |
       +------------+  CoAP  +------------+  AAA  +--------------+
```

Figure 1: CoAP EAP Architecture

3. General Flow Operation

The authentication service uses CoAP as transport layer for EAP. In other words, CoAP becomes an EAP lower-layer (in EAP terminology). In general, it is assumed that, since the EAP authenticator may implement an AAA client to interact with the AAA infrastructure, this endpoint will have more resources or, at least, it is not a so constrained device. We describe two different sequence flow. First, it is shown in Figure 2 where the OSCORE is used at the end of EAP authentication. The diagram in Figure 5 shows the flow when DTLS is used to protect CoAP messages at the end of the EAP authentication. As an example, both diagrams show the usage of a generic EAP method that we call EAP-X as authentication mechanism. (NOTE: any EAP method which is able to export cryptographic material should be valid).

3.1. EAP over CoAP: Running an OSCORE Security Association

When the EAP peer discovers the presence of the EAP authenticator and wants to start the authentication, it can send a Non-Confirmable "POST /b" request to the node (Step 0). This message, will carry an option developed from the work on [RFC7967] called no response. The rationale of this option is to avoid waiting for a response if it is not needed. So the use of this option will allow signaling the intention of the EAP peer to start the authentication process, as a trigger mechanism. Immediately after that, the EAP authenticator will start the authentication service. It is worth noting that the EAP authenticator MAY decide to start the authentication without waiting for the trigger message if it has knowledge about the presence of the EAP peer, for instance, through a previous authentication (re-authentication).

In any case, to perform the authentication service, the CoAP client (EAP authenticator) sends a Confirmable "POST /b" request to the CoAP Server (Step 1). This POST message contains a new option SeqNum that holds a sequence number randomly chosen by the CoAP client. This SeqNum is used to provide ordered and reliable delivery of messages involved during the whole authentication. In general, when a CoAP request with EAP message is received, the CoAP client considers a valid message if only if its sequence number is the expected value.

The sequence number is monotonically incremented by 1 so that the CoAP server can know what it is the next expected sequence number.

After receiving the first POST, the CoAP server assigns a resource and answers with an Acknowledgment with the piggy-backed resource identifier (Uri-Path) (Step 2). It is assumed that the CoAP server will only have an ongoing authentication and will not process simultaneous EAP authentications in parallel to save resources. In these two messages, the EAP Req/Id and Rep/ID are exchanged between the EAP authenticator and the EAP peer. The EAP Req/Id message is forwarded by the EAP authenticator, when EAP is in pass-through mode, to the local AAA server that is in charge of steering the conversation, choosing the EAP method to be used (e.g. EAP-X) if the user is local or sending the EAP messages to the home AAA of the EAP peer. At this point, the CoAP server has created a resource for the EAP authentication. The resource identifier value will be used together to relate all the EAP conversation between both CoAP endpoints. Since, only an ongoing EAP authentication is permitted and EAP is a lock-step protocol a Token of a constant value and 1 byte can be used throughout the authentication process. This also allows to save bytes through the link.

From now on, the EAP authenticator and the EAP peer will exchange EAP packets related to the EAP method, transported in the CoAP message payload (Steps 3,4,5,6). The EAP authenticator will use the POST method to send EAP requests to the EAP peer. The EAP peer will use a Piggy-backed response in the Acknowledgment message to carry the EAP response. At the end of the message exchanges, if everything has gone well, the EAP authenticator is able to send an EAP Success message and both CoAP endpoints will share a Master Session Key (MSK) ([RFC5295])

To establish a security association that will confirm to the EAP peer that EAP authenticator received the MSK from the AAA sever, as well as to the EAP authenticator that the EAP peer derived the MSK correctly, both entities engage in the establishment of a security association. In the context of constrained devices [RFC7228] and networks we consider protocols that are designed for these cases. Concretely, we show here in the diagram the establishment of the OSCORE security association. This is shown in Steps 7 and 8. From that point any exchange between both CoAP endpoints are protected with OSCORE. Before sending the EAP success to the EAP peer, the EAP authenticator is able to derive the OSCORE Security Context, to confirm the establishment of the security association. The details of the establishment of the OSCORE Security Context are discussed in Section 4.1

On the contrary, if DTLS is used (see Figure 5), a DTLS_PSK is derived from the MSK. Moreover, exchanges between both CoAP endpoints are protected with DTLS from that point.

```
            EAP peer                                  EAP Auth.
          (CoAP server)                             (CoAP client)
          -------------                             -------------
               |                                         |
               | NON [0x6af5]                            |
               | POST /b                                 |
               | No-Response                             |
           0)  | Token (0xab)                            |
               |---------------------------------------->|
               |                                         |
               |                            CON [0x7654] |
               |                                 POST /b |
               |                               SeqNum(x) |
               |                            Token (0xac) |
               |                      Payload EAP Req/Id |
            1) |<----------------------------------------|
               |                                         |
               | ACK [0x7654]                            |
               | SeqNum(x)                               |
               | Token (0xac)                            |
               | 2.01 Created                            |
               | Uri-Path [/b/5]                         |
               | Payload EAP Rep/Id                      |
            2) |---------------------------------------->|
               |                                         |
               |                            CON [0x8694] |
               |                               POST /b/5 |
               |                             SeqNum(x+1) |
               |                            Token (0xac) |
               |                     Payload EAP-X MSG 1 |
            3) |<----------------------------------------|
               |                                         |
               | ACK [0x8694]                            |
               | Token (0xac)                           |
               | SeqNum(x+1)                             |
               | 2.04 Changed                            |
               | Payload EAP-X MSG 2                     |
            4) |---------------------------------------->|
                                  ....

               |                                         |
               |                            CON [0x9869] |
```

```
               |                                POST /b/5 |
               |                             SeqNum(x+n/2)|
               |                             Token (0xac) |
               |                   Payload EAP-X MSG (n-1) |
            5) |<-----------------------------------------|
               |                                          |
               | ACK [0x9869]                             |
               | SeqNum(x+n/2)                            |
               | Token (0xac)                             |
               | 2.04 Changed                             |
               | Payload EAP-X MSG (n)                    |  MSK
            6) |----------------------------------------->|   |
               |                                          |   V
               |                        CON [0x7811]      | OSCORE
               |                        POST /b/5         | CONTEXT
               |                        SeqNum(x+n/2+1)   |
               |                        Token (0xac)      | (*)
               |                        OSCORE Option     |
               |                        EAP success       |
     MSK    7) |<-----------------------------------------|
      |        |                                          |
      V    (*) | ACK [0x7811]                             |
    OSCORE     | SeqNum(x+n/2+1)                          |
    CONTEXT    | Token (0xac)                             |
               | OSCORE Option                            |
               | 2.04 Changed                             |
            8) |----------------------------------------->|

               (*) Protected with OSCORE
```

Figure 2: CoAP-EAP with OSCORE option

3.2. The SeqNum Option

A new SeqNum option is defined in this document for establishing the ordering guarantee of the EAP exchange. Following guidelines in [RFC7252] this option is:

1. Format opaque (sequence of bytes).

2. Critical

3. Safe to Forward

4. No cacheable and Not part of the Cache-Key

5. Not repeatable

The number of the option will be determined by this previous decisions.

1. Critical (C = 1)

2. Safe to Forward (1)

3. NoCacheKey (111)

The number of the SeqNum option will fit this pattern: xxx11111

```
 0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|           | NoCacheKey| U | C |
+---+---+---+---+---+---+---+---+
```

Figure 3: Auth Option Number Mask

The option number is TBD.

The resultant SeqNum option is:

No.	C	U	N	R	Name	Format	Length	Default
TBD	x		x		SeqNum	uint	0-16	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable

Figure 4: SeqNum option

4. Key Derivation for protecting CoAP messages

As a result of a successful EAP authentication, both CoAP server and CoAP client share a Master Key Session (MSK). The assumption is that MSK is a fresh key so any derived key from the MSK will be also fresh. We have considered the derivation of either the OSCORE Security Context or pre-shared key that can be used for a DTLS negotiation (DTLS_PSK) (in the Appendix)

4.1. Deriving the OSCORE Security Context

Key material needed to derive the OSCORE Security Context, from the MSK can be done as follows. First, HKDF SHA-256 [RFC5869] is mandatory to implement. We assume the use of the default algorithms

HKDF SHA-256 and AES-CCM-16-64-128. The extract phase of HKDF produces a pseudo-random key (that we refer to here as RK) that is used to generate the OSCORE Security Context in the Expand phase. The derivation is done as follows:

RK = HMAC-SHA-256(MSK)

Where:

- o MSK is the Master Session Key derived from the EAP method.

- o RK is the Random Key that is generated from the MSK in the Extract phase.

Discussions about the use of the MSK for the key derivation are done in Section Section 7.

Based on the RK generated from the MSK, we can now generate the Master Secret and Master Salt. The key derivation is performed as follows:

Master_Secret = HKDF(RK, "IETF_OSCORE_MASTER_SECRET", length)

where:

- o The RK is exported in the Extract Phase, previously commented.

- o "IETF_OSCORE_MASTER_SECRET" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).

- o length is the length of the Master_Secret. We set the length to 32 bytes

The Master Salt can be derived as follows:

Master_Salt = HKDF(PK, "IETF_OSCORE_MASTER_SALT", length)

where:

- o The RK is exported in the Extract Phase, previously commented.

- o "IETF_OSCORE_MASTER_SALT" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).

- o length is the length of the Master Salt. We set the length to 8 bytes.

The ID Context can be set to the Identity of the EAP peer.

5. Use Case Scenario

In the following, we explain a basic example about the usage of CoAP-EAP. There are 5 entities involved in the scenario:

- o 2 nodes (A and B), which are constrained devices.

- o 1 node D, which is considered a no so constrained device, such a phone, or a tablet or even a laptop.

- o 1 controller (C). The controller manages a domain where nodes can be deployed. It can be considered a more powerful machine than the nodes.

- o 1 AAA server (AAA). The AAA is an Authentication, Authorization and Accounting Server, which is not constrained.

Any node wanting to join the domain managed by the controller, MUST perform an CoAP-EAP authentication with the controller C. This authentication may involve an external AAA server. This means that A and B, once deployed, will perform this CoAP-EAP once as a bootstrapping phase to establish a security association with the controller C. Moreover, any other entity (i.e. node D) , which wants to join and establish communications with nodes under the controller C's domain must also do the same.

Let us assume that the node A wants to communicate with node B (e.g. to active a light switch). The overall process is divided in three phases. Let's start with node A. In the first phase, the node A (EAP peer) does not yet belong to the controller C's domain. Then, it communicates with controller C (EAP authenticator) and authenticates with CoAP-EAP, which, in turn, communicates with the AAA server to complete the authentication process. If the authentication is successful, key material is distributed to the controller C and derived by node A. This key material allows node A to establish a security association with controller C. Some authorization information coming from the AAA infrastructure may be also provided in this step. If authentication and authorization are correct, node A is enrolled in the controller C's domain during a period of time. In particular, [RFC5247] recommends 8 hours, though the AAA server can establish a different lifetime. In the same manner, B needs to perform the same process with CoAP-EAP to be part of the controller C's domain.

In the second phase, when node A wants to talk with node B, it contacts the controller C for authorization to access node B and

obtain all the required information to do that in a secure manner (e.g. keys, tokens, authorization information, etc.). It does NOT require the usage of CoAP-EAP. The details of this phase are out of scope of this document, but ACE framework can be used for this purpose [I-D.ietf-ace-oauth-authz]

In the third phase, the node A can access node B with the credentials and information obtained from the controller C in the second phase. This access can be repeated without contacting the controller, while the credentials given to A are still valid. The details of this phase are out of scope of this document.

It is worth noting that first phase with CoAP-EAP is ONLY required to join the controller C's domain. Once it is performed with success, the communications are local to the controller C's domain so there is no need to contact the external AAA server nor performing EAP authentication.

6. Discussion

6.1. CoAP as EAP lower-layer

In this section we discuss the suitability of the CoAP protocol as EAP lower layer, and review the requisites imposed by the EAP protocol to any protocol that transports EAP. The assumptions EAP makes about its lower layers can be found in section 3.1 of the rfc [RFC3748], which are enumerated next:

o Unreliable transport. EAP does not assume that lower layers are reliable.

o Lower layer error detection. EAP relies on lower layer error detection (e.g., CRC, Checksum, MIC, etc.)

o Lower layer security. EAP does not require security services from the lower layers.

o Minimum MTU. Lower layers need to provide an EAP MTU size of 1020 octets or greater.

o Possible duplication. EAP stipulates that, while desirable, it does not require for the lower layers to provide non-duplication.

o Ordering guarantees. EAP relies on lower layer ordering guarantees for correct operation.

Regarding the unreliable transport, although EAP assumes a non reliable transport, CoAP does provide a reliability mechanism through

the use of Confirmable messages. For the error detection, CoAP goes on top of UDP which provides a checksum mechanism over its payload. Lower layer security services are not required. About the minimum MTU of 1020 octets, CoAP assumes an upper bound of 1024 for its payload which covers the requirements for EAP. Regarding message ordering, we propose the use of a new CoAP option, the SeqNum option described in Section (Section 3.2), which will allow keep the order in which the different messages are exchanged.

Regarding the Token, we consider the use of a constant value using a small 1 byte Token. In fact, the EAP server will not send a new EAP request until it has processed the expected EAP response. Additionally, we are under the assumption that there will a single EAP authentication between the constrained device and the same Controller.

6.2. Size of the EAP lower-layer vs EAP method size

Using CoAP as EAP lower layer guarantees a constrained transport for EAP in constrained environments. However, it is a fair to ask about the level of improvement taking into account the overload represented by the EAP method. In fact, if the EAP method is very taxing in the number of messages and the bytes sent over the networks the improvement achieved in the EAP lower-layer may be less significant ([coap-eap]). However, if the EAP method is lightweight and suitable for constrained networks (e.g. EAP-PSK, as a representative example of a lightweight EAP method) a constrained EAP lower-layer brings more benefits. This leads to the conclusion that possible next steps in this field could be also improving or designing new EAP methods that can be better adapted to the requirements of constrained devices and networks. Therefore, others EAP methods such as EAP-AKA or new lightweight EAP methods such as EAP-EDHOC [I-D.ingles-eap-edhoc] may benefit from a CoAP-based EAP lower-layer, as well as any new lightweight EAP method.

6.3. Controller as the CoAP Client

Due to the constrained capacities of the devices, to relieve them of the retransmission tasks, we set the Controller as the CoAP client, for the main exchange following the recommendations of the [I-D.ietf-lwig-coap] document to simplify the constrained device implementation.

6.4. Possible Optimizations

6.4.1. Empty Token

Assuming that the bootstrapping service runs before any other service, and that no other service will run concurrently until it has finished, we could use an Emtpy Token value to save resources, since there will be no other endpoint or CoAP exchange.

6.4.2. Removing SeqNum Option

An alternative to consider would be to try to rely on the Message ID values as a way of achieving the order delivery throughout the authentication exchange. Here we have two approximations: 1) Removing the option from the ACKs and 2) removing the option completely.

1. Since the ACKs are piggybacked by design, there is only 1 ongoing authentication process and the EAP exchange is done in a lockstep fashion, when we get a response we will get the same Message ID of the request and we can confirm the SeqNum of the Request.

2. An alternative to consider would be to try to solely rely on the Message ID values as a way of achieving the order delivery throughout the authentication exchange. Here we also have two approaches: A) To expect randomly generated Message IDs and B) set the Message ID to increase monotonically by 1.

    A. Regarding the use of the Message ID, their values in the requests sent by the Controller are generated randomly, as suggested by CoAP. The Controller selects a new Message ID value each time a new request is sent to the CoAP server, until the bootstrapping service finishes. Moreover, the Controller stores the last Message ID sent until correctly receiving the corresponding ACK. The CoAP server keeps track of the last received Message ID to identify retransmissions, and the previous Message IDs during the current bootstrapping to identify old messages. In general, a request is considered valid in terms of the Message ID if either this value matches the last value received, which means a retransmission of the last response is required, or the arrival of a new Message ID, which therefore represents a new message. If these rules do not apply (i.e., an old Message ID has been received), the CoAP server silently discards the request. This is possible because the bootstrapping service is designed as lockstep, i.e. the Controller will not send a new request until it has received the expected response. When the bootstrapping exchange finishes successfully, the CoAP server can free the tracked Message IDs, except for the

last received Message ID at the end of the bootstrapping, just in case a retransmission is required.

B. This case would avoid having to keep track of the already used Message IDs, monotonically increasing by 1 the message ID value once the first is randomly picked by the Controller.

6.4.3. Further re-authentication

Since the initial bootstrapping is usually taxing, it is assumed to be done only once over a long period of time. If further re-authentications for refreshing the key material are necessary, there are other methods that can be used to perform these re-authentications. For example, the EAP re-authentication (EAP-ERP) [RFC6696] can be used to avoid repeating the entire EAP exchange in few exchanges.

7. Security Considerations

There are some aspects to be considered such as how authorization is managed, how the cryptographic suite is selected and how the trust in the Controller is established.

7.1. Authorization

Authorization is part of the bootstrapping. It serves to establish whether the node can join and the set of conditions it has to adhere. The authorization data received from the AAA server can be delivered by the AAA protocol (e.g. Diameter). Providing more fine grained authorization data can be with the transport of SAML in RADIUS [RFC7833] After bootstrapping, additional authorization to operate in the security domain, e.g., access services offered by other nodes, can be taken care of by the solutions proposed in the ACE WG.

7.2. Cryptographic suite selection

How the cryptographic suit is selected is also important. To reduce the overhead of the protocol we use a default cryptographic suite. As OSCORE is assumed to run after the EAP authentication, the same default crypto-suite is used in this case as explained in the Key Derivation Section Section 4 The cryptographic suite is not negotiated. If the cryptographic suite to be used by the node is different from default, the AAA server will send the specific parameters to the Authenticator. If the cryptographic suite is not supported, the key derivation process would result in a security association failure.

7.3. Freshness of the key material

In this design, we do not exchange nonces to provide freshness to the keys derived from the MSK. This is done under the assumption that the MSK and EMSK keys derived are fresh key material by the specifications of the EAP KMF. Since only one session key is derived from the MSK we do not have to concern ourselves with the generation of additional key material. In case we need to refresh the MSK, a re-authentication can be done, by running process again, using a more lightweight mechanism to derive additional key material such as ERP [RFC6696].

8. IANA Considerations

This document has no actions for IANA.

9. Acknowledgments

We would like to thank Pedro Moreno-Sanchez and Gabriel Lopez-Millan for the first review of this document. Also, we would like to thank Ivan Jimenez-Sanchez for the first proof-of-concept implementation of this idea.

We also thank for their valuables comments to Alexander Pelov and Laurent Toutain, specially for the potential optimizations of CoAP-EAP.

10. References

10.1. Normative References

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-36 (work in progress), November 2020.

[I-D.ietf-lwig-coap]
Kovatsch, M., Bergmann, O., and C. Bormann, "CoAP Implementation Guidance", draft-ietf-lwig-coap-06 (work in progress), July 2018.

[I-D.ingles-eap-edhoc]
Sanchez, E., Garcia-Carrillo, D., and R. Marin-Lopez, "EAP method based on EDHOC Authentication", draft-ingles-eap-edhoc-01 (work in progress), November 2020.

[I-D.kumar-dice-dtls-relay]
Kumar, S., Keoh, S., and O. Garcia-Morchon, "DTLS Relay for Constrained Environments", draft-kumar-dice-dtls-relay-02 (work in progress), October 2014.

[I-D.ohba-core-eap-based-bootstrapping]
Das, S. and Y. Ohba, "Provisioning Credentials for CoAP Applications using EAP", draft-ohba-core-eap-based-bootstrapping-01 (work in progress), March 2012.

[I-D.pelov-core-cosol]
Pelov, A., Toutain, L., and Y. Delibie, "Constrained Signaling Over LP-WAN", draft-pelov-core-cosol-01 (work in progress), February 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <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>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <https://www.rfc-editor.org/info/rfc4279>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

[RFC4615] Song, J., Poovendran, R., Lee, J., and T. Iwata, "The Advanced Encryption Standard-Cipher-based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE)", RFC 4615, DOI 10.17487/RFC4615, August 2006, <https://www.rfc-editor.org/info/rfc4615>.

[RFC4764] Bersani, F. and H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method", RFC 4764, DOI 10.17487/RFC4764, January 2007, <https://www.rfc-editor.org/info/rfc4764>.

[RFC5191] Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, DOI 10.17487/RFC5191, May 2008, <https://www.rfc-editor.org/info/rfc5191>.

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

[RFC5295] Salowey, J., Dondeti, L., Narayanan, V., and M. Nakhjiri, "Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK)", RFC 5295, DOI 10.17487/RFC5295, August 2008, <https://www.rfc-editor.org/info/rfc5295>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6345] Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., Ed., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", RFC 6345, DOI 10.17487/RFC6345, August 2011, <https://www.rfc-editor.org/info/rfc6345>.

[RFC6696] Cao, Z., He, B., Shi, Y., Wu, Q., Ed., and G. Zorn, Ed., "EAP Extensions for the EAP Re-authentication Protocol (ERP)", RFC 6696, DOI 10.17487/RFC6696, July 2012, <https://www.rfc-editor.org/info/rfc6696>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7833] Howlett, J., Hartman, S., and A. Perez-Mendez, Ed., "A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for the Security Assertion Markup Language (SAML)", RFC 7833, DOI 10.17487/RFC7833, May 2016, <https://www.rfc-editor.org/info/rfc7833>.

[RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <https://www.rfc-editor.org/info/rfc7967>.

10.2. Informative References

[cantcoap]
Mills, A., "Cantcoap implementation of CoAP", January 2013.

[coap-eap]
Garcia-Carrillo, D. and R. Marin-Lopez, "Lightweight CoAP-Based Bootstrapping Service for the Internet of Things - https://www.mdpi.com/1424-8220/16/3/358", March 2016.

[Contiki] Contiki, "Contiki: The Open Source OS for the Internet of Things", March 2014.

[hostapd] hostapd, "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator", March 2014.

Appendix A. CoAP-EAP with DTLS

Other possibility at our disposal is to do a DTLS handshake after the MSKs generation and continue the communication between endpoints using CoAP through DTLS as we can see at Figure 5. The Steps 0-6 are the same as the case with OSCORE, however, before continuing with Steps 7 and 8, the EAP authenticator starts the DTLS handshake with the EAP peer (Step 6'). To establish a DTLS Security Association, a key named DTLS-PSK is derived from MSK (see Section 4). In this case the CoAP client can start DTLS before sending the last message containing the EAP Success. Once DTLS is established, any posterior CoAP exchange is protected. Thus, OSCORE in this instance is not needed for key confirmation, since a successful DTLS negotiation confirms the possession of DTLS_PSK that, in turn, corroborates that both entities participated in the EAP authentication.

```
                EAP peer                                   EAP Auth.
              (COAP server)                              (COAP client)
              -------------                              -------------
                   |                                           |
                   | NON [0x6af5]                              |
                   | POST /b                                   |
                   | No-Response                               |
               0)  | Token (0xab)                              |
                   |------------------------------------------>|
```

```
                 |                                         |
                 |                          CON [0x7654] |
                 |                               POST /b |
                 |                             SeqNum(x) |
                 |                          Token (0xac) |
                 |                    Payload EAP Req/Id |
              1) |<-----------------------------------------|
                 |                                         |
                 | ACK [0x7654]                            |
                 | SeqNum(x)                               |
                 | Token (0xac)                            |
                 | 2.01 Created                            |
                 | Uri-Path [/b/5]                         |
                 | Payload EAP Rep/Id                      |
              2) |---------------------------------------->|
                 |                                         |
                 |                          CON [0x8694] |
                 |                             POST /b/5 |
                 |                           SeqNum(x+1) |
                 |                         Token (0xac)  |
                 |                     Payload EAP-X MSG 1 |
              3) |<-----------------------------------------|
                 |                                         |
                 | ACK [0x8694]                            |
                 | Token (0xac)                            |
                 | SeqNum(x+1)                             |
                 | 2.04 Changed                            |
                 | Payload EAP-X MSG 2                     |
              4) |---------------------------------------->|
                                 ....

                 |                                         |
                 |                          CON [0x9869] |
                 |                             POST /b/5 |
                 |                          SeqNum(x+n/2)|
                 |                          Token (0xac) |
                 |                 Payload EAP-X MSG (n-1) |
              5) |<-----------------------------------------|
                 |                                         |
                 | ACK [0x9869]                            |
                 | SeqNum(x+n/2)                           |
                 | Token (0xac)                            |
                 | 2.04 Changed                            |
                 | Payload EAP-X MSG (n)                   |
           MSK 6)|---------------------------------------->| MSK
              |  |                                         |  |
         DTLS_PSK|                                         | DTLS_PSK
                 |                                         |
```

```
                   |              DTLS HANDSHAKE           |
                   |          (Initiated by EAP Auth.)     |
               6') |<------------------------------------->|
                   |                                       |
                   |                          CON [0x7811] |
                   |                          POST /b/5    |
                   |                          Token (0xac) |
                   |                   Payload EAP Success | (*)
               7)  |<--------------------------------------|
                   |                                       |
                   | ACK [0x7811]                          |
               (*) | Token (0xac)                          |
                   | 2.04 Changed                          |
                8) |-------------------------------------->|

                   (*) Protected with DTLS
```

                     Figure 5: EAP over CoAP with DTLS

A.1. Deriving DTLS_PSK

In the second alternative, a DTLS_PSK is derived from the MSK between both CoAP endpoints. So far, DTLS_PSK will have also 16 byte length and it will derived from the RK (generated as done in Section Section 4) as follows:

DTLS_PSK = HKDF(RK, "IETF_DTLS_PSK" , length). This value is concatenated with the value of the Token Option value.

where:

- o RK is the Random Key generated in the Extract phase, from the MSK.
- o "IETF_DTLS_PSK" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).
- o length is the length of the DTLS_PSK (16 bytes).

Authors' Addresses

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 85 01
Email: rafa@um.es

Dan Garcia Carrillo
University of Oviedo
Calle Luis Ortiz Berrocal S/N, Edificio Polivalente
Gijon, Asturias 33203
Spain

Email: garciadan@uniovi.es