Workgroup: ACE Working Group
Internet-Draft: draft-ietf-ace-wg-coap-eap-03
Published: 26 July 2021
Intended Status: Standards Track
Expires: 27 January 2022
Authors: R. Marin-Lopez   D. Garcia-Carrillo
         University of Murcia   University of Oviedo

EAP-based Authentication Service for CoAP

Abstract

This document specifies an authentication service that uses the Extensible Authentication Protocol (EAP) transported employing Constrained Application Protocol (CoAP) messages. As such, it defines an EAP lower-layer based on CoAP called CoAP-EAP. One of the primer goals is to authenticate a CoAP-enabled device (EAP peer) that intends to join a security domain managed by a domain Controller (EAP authenticator). Secondly, it allows deriving key material to protect CoAP messages exchanged between them based on Object Security for Constrained RESTful Environments (OSCORE), enabling the establishment of a security association between them. This document also provides guidelines on how to generate key material for other types of security associations.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction
  1.1. Requirements Language
2. General Architecture
3. CoAP-EAP Operation
  3.1. General flow of operation
  3.2. Error handling
    3.2.1. Non-responding endpoint
    3.2.2. Messages with /.well-known/a
  3.3. Managing the State of the Service
    3.3.1. Deleting the state
    3.3.2. Reauthentication
4. Cryptosuite negotiation and key derivation
  4.1. Cryptographic suite negotiation
  4.2. Deriving the OSCORE Security Context
  4.3. Deriving DTLS PSK
5. Discussion
  5.1. CoAP as EAP lower layer
  5.2. Size of the EAP lower layer vs EAP method size
  5.3. Controller as the CoAP Client
6. Security Considerations
  6.1. Authorization
  6.2. Freshness of the key material
  6.3. Channel Binding support
  6.4. Additional Security Consideration
7. IANA Considerations
8. Acknowledgments
9. References
  9.1. Normative References
  9.2. Informative References
Appendix A. Examples of Use Case Scenario
  A.1. Example 1: CoAP-EAP in ACE
  A.2. Example 2: Multi-domain with AAA infrastructures
  A.3. Example 3: Single domain with AAA infrastructure
  A.4. Example 4: Single domain without AAA infrastructure
  A.5. Other use cases
    A.5.1. CoAP-EAP for network access control
    A.5.2. CoAP-EAP for service authentication
Authors' Addresses

1. Introduction

This document specifies an authentication service that uses the Extensible Authentication Protocol (EAP) [RFC3748] and is built on top of the Constrained Application Protocol (CoAP) [RFC7252] called CoAP-EAP. CoAP-EAP allows authenticating two CoAP endpoints by using EAP, and to establish an OSCORE security association between them.

More specifically, this document specifies how CoAP can be used as a constrained, link-layer independent, EAP lower layer [RFC3748] to transport EAP messages between a CoAP server (acting as EAP peer) and a CoAP client (acting as EAP authenticator) using CoAP messages. The CoAP client has the option of contacting a backend AAA infrastructure to complete the EAP negotiation as described in the EAP specification [RFC3748].

EAP methods transported in CoAP MUST generate cryptographic material [RFC5247]. This way, CoAP messages are protected after the authentication. After CoAP-EAP's operation, Object Security for Constrained RESTful Environments (OSCORE) is established between endpoints of the service. In addition, using the key material derived from the authentication, this document specifies how to establish other types of security associations:

* An OSCORE [RFC8613] security association is established based on the cryptographic material generated from the EAP authentication.

* A DTLS security association MAY be established using the exported cryptographic material after a successful EAP authentication.

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Readers are expected to be familiar with the terms and concepts of described in CoAP [RFC7252], EAP [RFC3748]and OSCORE [RFC8613].

2. General Architecture

Figure 1 illustrates the architecture defined in this document. Basically, an IoT device, acting as the EAP peer, wants to be authenticated by using EAP to join a domain that is managed by a Controller (EAP authenticator). The IoT device will act a CoAP server for this service, and the EAP authenticator as a CoAP client. The rationale behind this decision, as expanded later, is that EAP requests go always from the EAP server to the EAP peer. Accordingly, the EAP responses go from the EAP peer to the EAP server.

It is worth noting that the CoAP client (EAP authenticator) MAY interact with a backend AAA infrastructure when EAP pass-through mode is used, which will place the EAP server in the AAA server that contains the information required to authenticate the EAP peer.

```
               IoT device           Controller
             +------------+        +------------+
             | EAP peer/  |        | EAP auth./ |
             | CoAP server|+------+| CoAP client|
             +------------+  CoAP  +------------+
```

Figure 1: CoAP-EAP Architecture

3. CoAP-EAP Operation

The sequence flow can be seen in Figure 2. It shows how EAP-X, a generic EAP method, is used as an authentication mechanism.

(NOTE: any EAP method that exports cryptographic material is valid. For example, EAP-MD5 cannot be used since it does not export key material).

3.1. General flow of operation

The first step to run CoAP-EAP is for the IoT device to discover the Controller, and that it implements the CoAP-EAP service. To do so, we can rely on any mechanism that allows us to perform this discovery. For example, the discovery mechanism of CoAP. To access the authentication service, this document defines the well-known URI "/.well-known/a" (to be assigned by IANA). This URI is referring to the authentication service that is present in both the IoT device and the Controller.

For the rest of this section, the IoT device is referred to as the EAP peer and the Controller as the EAP authenticator.

In the CoAP-EAP flow of operation, the EAP peer always starts the authentication process by sending the first message. When the EAP peer discovers the presence of the EAP authenticator and wants to start the authentication, it sends a Non-Confirmable "POST /.well-known/a" request (Step 0). This message will carry the option called 'No-Response' [RFC7967]. The rationale of this option is to avoid waiting for a response that is not needed. So, the use of this option will indicate to the EAP authenticator that the EAP peer is ready to start the authentication process.

This message is the only instance where the EAP authenticator acts as a CoAP server and the EAP peer as a CoAP client. After this, the exchange continues with the EAP authenticator as a CoAP client and the EAP peer as a CoAP server. A discussion about the CoAP roles is in Section 5.3

Once the EAP authenticator is aware of the intention of the EAP peer of starting an authentication process, sends a Confirmable "POST /.well-known/a" request to the EAP peer(Step 1). After receiving the first POST, the EAP peer (CoAP server) assigns a resource to the ongoing authentication process, and answers with an Acknowledgment with a piggybacked response containing the resource identifier in the Location-Path (and/or Location-Query) Options (Step 2). The path /.well-known/a at this point will not be used. The name of the resource is selected by the CoAP server as it pleases. The only condition is that both end points have to correctly process the following message based on the selected value. As an example, the CoAP server MAY follow the the naming "/a/x". Where "a" is the general name of the authentication service; "x" represents the resource established to process the following message of the authentication. This naming, as example, will be used in the figures of this document. The EAP peer (CoAP server) will only have one authentication session with that particular EAP authenticator (CoAP client) and it will not process EAP authentications in parallel (with the same EAP authenticator) to save resources.

In this exchange (Step 1 and 2), the EAP-Request/Identity (EAP Req/Id) and EAP-Response/Identity (EAP Resp/Id) messages are exchanged between the EAP authenticator and the EAP peer. Upon the reception of the EAP Req/Id message, the EAP authenticator sends this message to the EAP server. The EAP server is in charge of steering the conversation, choosing the EAP method to be used (e.g. EAP-X). In this exchange we can also see the cryptosuite negotiation. If the cryptosuite is not sent, the default cryptosuite is used. The first message (Step 1) carries, concatenated to the EAP Req/Id, a CBOR array containing a list with the cryptosuites, and the CoAP server in the next message will confirm by sending a choice. The details of the cryptosuite negotiation are discussed in Section 4.1.

NOTE: Since only an ongoing EAP authentication is permitted per EAP authenticator and EAP is a lock-step protocol, a CoAP Token of a constant value can be used throughout the authentication process. An empty Token MAY be considered to reduce bytes.

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 in a Confirmable message to send EAP requests to the EAP peer. The EAP peer will use a piggybacked response in the Acknowledgment message to carry the EAP response. EAP requests and responses are represented in Figure 2 using the nomenclature (EAP-X-Req n) and (EAP-X-Resp n) respectively. 'X' indicates the EAP method use. 'n' indicates the identifier of the EAP message, which is shown monotonically increasing from 1 in this flow of operation.

When a Confirmable POST message arrives (e.g, /a/x) carrying an EAP request message, if processed correctly, returns an EAP Response. Along with the EAP response, a new resource is created (e.g, /a/y) and the ongoing resource (i.e., /a/x) is erased. This EAP response will go in a CoAP piggybacked response that will also indicate the new resource in the Location-Path (and/or Location-Query) Options. In case there is an error processing a legitimate message, the server will return an (4.00 Bad Request). There is a discussion about error handling in Section 3.2.

When the authentication is completed correctly, the EAP server sends an EAP Success message, and both CoAP endpoints will share a Master Session Key (MSK)[RFC5295].

Using the MSK, an OSCORE security association is established between EAP peer and EAP authenticator, which serves as key confirmation of the MSK and to provide integrity and authentication to the last exchange (Steps 7 and 8). From that point, any exchange between both CoAP endpoints is protected with OSCORE. Before sending the EAP success to the EAP peer, the EAP authenticator can derive the OSCORE Security Context. The EAP peer can do so when it receives the message containing the EAP success, and the session liftime, to verify the message from the EAP authenticator and generate the answer (Step 8). The establishment of the OSCORE Security Context is discussed in Section 4.2. The Session Lifetime, in seconds, is represented with a unsigned integer in a CBOR array. The disposition of the information is the same to the cryptosuite negotiation () see Section 4.1).

```
         EAP peer                                  EAP Auth.
      (CoAP server)                             (CoAP client)
      -------------                             -------------
           | NON [0x6af5] POST /.well-known/a        |
        0) | Token (0xab), No-Response               |
           |---------------------------------------->|
           |        CON [0x7654] POST /.well-known/a |
           |                            Token (0xac) |
           |        Payload (EAP Req/Id||Cryptosuite)|
        1) |<----------------------------------------|
           | ACK [0x7654]                            |
           | Token (0xac)                            |
           | 2.01 Created Location-Path [/a/x]       |
           | Payload (EAP Resp/Id||Cryptosuite)      |
        2) |---------------------------------------->|
           |                  CON [0x8694] POST /a/x |
           |                            Token (0xac) |
           |                     Payload EAP-X-Req 1 |
        3) |<----------------------------------------|
           | ACK [0x8694]                            |
           | Token (0xac)                            |
           | 2.01 Created Location-Path [/a/y]       |
           | Payload EAP-X-Resp 1                    |
        4) |---------------------------------------->|
                              ....

           |                CON [0x9869] POST /a/y   |
           |                            Token (0xac) |
           |                   Payload EAP-X-Req (n) |
        5) |<----------------------------------------|
           | ACK [0x9869]                            |
           | Token (0xac)                            |
           | 2.01 Created Location-Path [/a/z]       | MSK
           | Payload EAP-X-Resp (n)                  |  |
        6) |---------------------------------------->|  v
           |                CON [0x7811] POST /a/z   |OSCORE
           |                  Token (0xac), OSCORE   |CONTEXT
 MSK       | Payload (EAP success||Session-Lifetime) |(*)
  |     7) |<----------------------------------------|
  v        | ACK [0x7811]                            |
OSCORE  (*)| Token (0xac), OSCORE                    |
CONTEXT 8) |---------------------------------------->|
           (*) Protected with OSCORE
```

Figure 2: CoAP-EAP flow of operation

3.2. Error handling

This section elaborates how different errors are handled, from a non-responding endpoint, dealing lost messages and initial POST messages arriving arriving out of place.

In general, the CoAP engine, in the client and server, will take care of retransmissions, dealing with lost messages sent out of place, and sending the errors when a client is trying to access a resource that does not exist in the CoAP server. But, there are cases where the messages can arrive out of place, and the CoAP engine may not recognize them as old or as a retransmission, and send them to the CoAP application.

3.2.1. Non-responding endpoint

When CoAP-EAP starts, a state is created that stores all information related to the process. If by any reason one of the entities becomes non-responding, the state should be only kept form a period of time before it is removed. According to CoAP, EXCHANGE_LIFETIME considers the time it takes until a client stops expecting a response to a Confirmable request. A timer is reseted every time a message is sent. If EXCHANGE_LIFETIME has passed waiting the next message, both entities will delete the CoAP-EAP state if the authentication process has not finished correctly.

3.2.2. Messages with /.well-known/a

The reception of messages containing /.well-known/a needs some additional considerations, as the resource is always available in both entities. This message can be sent out of the expected order by the EAP peer as well as by the EAP authenticator for reasons such as network delays.

When this message arrives to the other entity during an ongoing authentication, and it is not recognized by the CoAP engine as an old or retransmitted message, arriving to the CoAP-EAP application, we describe the behavior depending on the entity receiving this message. If this message is from the EAP authenticator to the peer, being a Confirmable POST message, will honor the request and answer, in this case with a CoAP Reset message, that will indicate that, since there is an authentication ongoing, the EAP authenticator is not in disposition of processing this message. This exchange is illustrated in Figure 3.

```
     EAP peer                                  EAP Auth.
   (CoAP server)            .                (CoAP client)
   -------------            .               -------------
        |                                         |
        |        CON [0x7654] POST /.well-known/a |
        |                            Token (0xac) |
        |        Payload (EAP Req/Id||Cryptosuite)|
        |<----------------------------------------|
        |                                         |
        | RST [0x7654]                            |
        | 0.00 Code                               |
        |---------------------------------------->|
```

Figure 3: /.well-known/a during ongoing authentication from the EAP authenticator

In case the message goes from the EAP peer to the authenticator, being a Non-confirmable POST message with the No-Response Option, there is no need to further process this message, and can be silently ignored. This exchange is illustrated in Figure 4.

```
     EAP peer                                  EAP Auth.
   (CoAP server)            .                (CoAP client)
   -------------            .               -------------
        |                                         |
        | NON [0x6af5] POST /.well-known/a        |
        | Token (0xab), No-Response               |
        |---------------------------------------->| --+
        |                   .                     |   |
        |                   .                     |   v
                                                  (Ignored)
```

Figure 4: /.well-known/a during ongoing authentication from the EAP peer

When this message arrives to the CoAP-EAP application, and there is no authentication ongoing it will understand that a new authentication process is starting. In case the message goes from the EAP authenticator to the peer, and no prior Non-confirmable /.well-known/a message was sent by the EAP peer, it will send a Reset message indicating that is not in disposition to process this message, as it has to be the EAP peer the one to initiate the process. This exchange is illustrated in Figure 5.

```
      EAP peer                                 EAP Auth.
     (CoAP server)                           (CoAP client)
     -------------                          -------------
           |                                         |
           |        CON [0x7654] POST /.well-known/a |
           |                            Token (0xac) |
           |        Payload (EAP Req/Id||Cryptosuite)|
           |<----------------------------------------|
           |                                         |
           | RST [0x7654]                            |
           | 0.00 Code                               |
           |---------------------------------------->|
```

Figure 5: /.well-known/a with no ongoing authentication from the EAP authenticator

If this message is from the EAP peer to the authenticator, receiving a Non-confirmable /.well-known/a message, it will understand it as the start of the process by the EAP peer, and start a normal process of authentication as depicted in the general flow of operation in Figure 2.

3.3. Managing the State of the Service

The management of the CoAP-EAP state, can be done in different ways. The Controller MAY choose to delete it as explained in Section 3.3.1. On the other hand, the IoT device may need to renew the CoAP-EAP state because the key material is close to expire, as elaborated in Section 3.3.2.

3.3.1. Deleting the state

There are situations where the current CoAP-EAP state might need to be removed. For instance due to its expiration or a forced removal if the IoT device needs to be expelled from the security domain. This exchange is illustrated in Figure 6. If the Controller, which implements the CoAP client in this exchange, deems necessary the removal of the state, it can send a DELETE command to the CoAP server, referencing the last CoAP-EAP state resource given by the CoAP server, whose identifier will be the last one received with the ACK of the EAP success message (/a/z in Figure 2) This message will be protected with the OSCORE security association to prevent forgery. Upon reception of this message, the CoAP server sends piggybacked response to the client with the Code 2.02 Deleted, which is also protected with the OSCORE security association.

```
      EAP peer                                  EAP Auth.
    (CoAP server)                             (CoAP client)
    -------------                             -------------
         |                                         |
         |                CON [0x7654] DELETE /a/z |
         |                            Token (0xac) |
         |                                  OSCORE |
         |<----------------------------------------|
         |                                         |
         | ACK [0x7654]                            |
         | Token (0xac)                            |
         | 2.02 Deleted                            |
         | OSCORE                                  |
         |---------------------------------------->|
```

Figure 6: Deleting state

If the ACK from the CoAP server does not arrive, after the maximum retransmission attempts, the CoAP client will remove the state from its side.

3.3.2. Reauthentication

When the CoAP-EAP state is close to expire, the IoT device MAY want to start a re-authentication process to renew the state. Since the initial authentication is usually taxing, it is assumed to be done only once over a long period of time. If further EAP 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 protocol (ERP) [RFC6696], EAP-NOOB [I-D.ietf-emu-eap-noob], or EAP-AKA' [RFC5448] MAY be used to avoid repeating the entire EAP exchange. The message flow for the reauthentication will be exactly the same as the one shown in Figure 2, using a more lightweight alternative to do a re-authentication. While the re-authentication is ongoing, two different CoAP-EAP states will be active. Once the reauthentication is completed and OSCORE is exchanged using the newly derived key material, the old CoAP-EAP state is deleted. If by any reason, the reauthentication fails to complete, the old CoAP-EAP state will be available until it expires.

4. Cryptosuite negotiation and key derivation

4.1. Cryptographic suite negotiation

How the cryptographic suite is selected is important. OSCORE runs after the EAP authentication, using the cryptosuite selected in the cryptosuite negotiation. To negotiate the cryptosuite, the protocol follows a simple approach by sending in the first message from the

Controller, a list in decreasing order or preference, with the identifiers of the supported cryptosuites. In the response to that message, the IoT device sends a response with the choice. To do this without resorting to the creation of a new CoAP option tailored to this purpose, by leveraging the fact that in the payload it is always expected at the beginning the EAP message, which by design specifies its own length. Following the EAP message, optionally there is a CBOR array that contains the cryptosuites. Be it the list of cryptosuites supported by the Controller, or the one chosen by the IoT device. An example of how the fields are arranged in the CoAP payload can be seen in Figure 7. An example of the exchange with the cryptosuite negotiation is shown in Figure 8, where can be appreciated the disposition of both EAP-Request/Identity and Response/Identity, followed by the CBOR array. It is worth noting that the CBOR array is also expected in the EAP Success, containing the Session Lifetime.

```
+-----+-----------+-------+------++-------------+
|Code |Identifier |Length |Data  ||Cyphersuites|
+-----+-----------+-------+------++-------------+
            EAP Packet             CBOR Array
```

Figure 7: How the cypersuites are sent in the CoAP payload

```
  EAP peer                                 EAP Auth.
(CoAP server)                            (CoAP client)
-------------                            -------------
     |                                         |
     |                  ...                    |
     |---------------------------------------->|
     |        CON [0x7654] POST /.well-known/a |
     |                             Token (0xac) |
     |  Payload (EAP Req/Id, CBORArray[0,1,2]) |
  1) |<----------------------------------------|
     | ACK [0x7654]                            |
     | Token (0xac)                            |
     | 2.01 Created Location-Path [/a/x]       |
     | Payload (EAP Resp/Id, CBORArray[0])     |
  2) |---------------------------------------->|
                        ...
```

Figure 8: Cryptosuite negotition in t CoAP-EAP flow

In case there is no CBOR Array stating the cryptosuites, the default cryptosuites are applied. If the Controller sends a restricted list of cryptosuites that is willing to accept, and the ones supported by the IoT device are not in that list, the IoT device will respond with a 4.00 Bad Request, expressing in the Payload the cryptosuites supported. Figure 9 illustrates this exchange.

```
      EAP peer                                   EAP Auth.
   (CoAP server)                               (CoAP client)
   -------------                               -------------
        |                                          |
        |                  ...                     |
        |----------------------------------------->|
        |        CON [0x7654] POST /.well-known/a  |
        |                             Token (0xac) |
        |    Payload (EAP Req/Id, CBORArray[1,2])  |
     1) |<-----------------------------------------|
        | ACK [0x7654]                             |
        | Token (0xac)                             |
        | 4.00 Bad Request                         |
        | Payload (CBORArray[0])                   |
     2) |----------------------------------------->|
```

Figure 9: Cryptosuite negotition failed

To avoid a downgrading attack, we will use the proposed cryptosuite list and the choice in the key derivation process, to bind them to the generated keys as explained in Section 4.2.

As a result of a successful EAP authentication, both the CoAP server and CoAP client share a Master Session Key (MSK). The MSK is a fresh key, so any key derived from the MSK will be also fresh. The CoAP-EAP exchange finishes with the establishment of an OSCORE security association for the CoAP-EAP service. The security level for the CoAP-EAP exchanges with OSCORE is with integrity. Due to the requirement of using OSCORE, the cryptosuite requirements are inherited from the ones established by OSCORE. This requires for the HKDF algorithm by default HKDF SHA-256 and the AEAD algorithm by default AES-CCM-16-64-128 to be mandatory to implement. The other cryptosuites supported and negotiated in the cryptosuite negotiation are, expressed as tuples of AEAD Algorithm and Hash Algorithm, the following:

0. AES-CCM-16-64-128, SHA-256

1. A128GCM, SHA-256

2. A256GCM, SHA-384

This specification uses the (HMAC)-based key derivation function (HKDF) defined in [RFC5869] to derive the necessary key material. Since the derivation will be done using the MSK, which is considered fresh key material, we will use the HKDF-Expand function, which we will shorten here as KDF. In addition to the generation of the OSCORE parameters, we consider the derivation of a pre-shared key that can be used for a DTLS security association (DTLS PSK).

4.2. Deriving the OSCORE Security Context

The derivation of the security context for OSCORE has a three purposes:

* Secure the last two messages of the CoAP-EAP exchange, while establishing the OSCORE security association

* Perform key confirmation

* Prevent a downgrading attack.

We can achieve this because, after a successful authentication, both the EAP authenticator and the peer share the MSK. From this MSK, necessary key material for the OSCORE context is derived. If the contexts match, both entities confirm they have the same Master Secret, and therefore they share the same MSK. We prevent a downgrading attack, by embedding into the derivation process, context information gathered from the cryptosuite negotiation. To do this, we concatenate to the label, the content of the cryptosuites negotiation (which we refer to here as CSO), using the null string for any part of the negotiation missing. If an attacker changes the value of the cryptosuite negotiation in any of the messages, the result will be different security contexts.

Key material needed to derive the OSCORE Security Context, from the MSK is done as follows:

The Master Secret can be derived by using the chosen cryptosuite and the KDF. The Master Secret can be derived as follows:

```
Master Secret = KDF(MSK, CSO | "OSCORE MASTER SECRET", length)
```

where:

* The algorithms are agreed in the cryptosuite negotiation.

* The MSK exported by the EAP method. Discussion about the use of the MSK for the key derivation is done in Section 6.

* CSO is the concatenation of the content of the cryptosuite negotiation, in the request and response. If any of the messages did not contain the CBOR array, the null string is used.

* "OSCORE MASTER SECRET" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).

* CSO and "OSCORE MASTER SECRET" are concatenated.

* length is the size of the output key material.

The Master Salt, similarly to the Master Secret, can be derived as follows:

```
Master Salt = KDF(MSK, CSO | "OSCORE MASTER SALT", length)
```

where:

* The algorithms are agreed in the cryptosuite negotiation.

* The MSK exported by the EAP method. Discussion about the use of the MSK for the key derivation is done in Section 6.

* CSO is the concatenation of the content of the cryptosuite negotiation, in the request and response. If any of the messages did not contain the CBOR array, the null string is used.

* "OSCORE MASTER SALT" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).

* CSO and "OSCORE MASTER SECRET" are concatenated.

* length is the size of the output key material.

Regarding the Recipient ID and Sender ID, as their purpose is to serve as identifiers of both entities involved in the exchange, these identifiers are generated as follows:

```
Recipient ID = KDF(MSK, "OSCORE RECIPIENT ID", length)
```

where:

* The algorithms are agreed in the cryptosuite negotiation.

* "OSCORE RECIPIENT ID" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).

* length is the size of the output. This value will be the maximum allowed length according to the limits established by OSCORE to Recipient ID, depending on the chosen cryptographic algorithms [RFC8613].

For the Sender ID, analogously to the Recipient ID:

Sender ID = KDF(MSK, "OSCORE SENDER ID", length)

where:

* The algorithms are agreed in the cryptosuite negotiation.

* "OSCORE RECIPIENT ID" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).

* length is the size of the output. The maximum value allowed is established by OSCORE to Sender ID, depending on the chosen cryptographic algorithms.

4.3. Deriving DTLS PSK

It is also possible to derive a pre-shared key for DTLS [RFC6347], refereed to here as "DTLS PSK", from the MSK between both CoAP endpoints if required. The length of the DTLS PSK will depend on the cryptosuite. To have a cryptographic material with sufficient length we will derive a key of 32 bytes that can be later truncated if needed:

DTLS PSK = KDF(MSK, "DTLS PSK", length).

where:

* MSK is exported by the EAP method.

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

* length is the size of the output key material.

5. Discussion

5.1. CoAP as EAP lower layer

This section discusses the suitability of the CoAP protocol as EAP lower layer, and reviews the requisites imposed by the EAP protocol to any protocol that transports EAP. What EAP expects from its lower layers can be found in section 3.1 of [RFC3748], which is elaborated next:

Unreliable transport. EAP does not assume that lower layers are reliable. CoAP provides a reliability mechanism through the use of Confirmable messages.

Lower layer error detection. EAP relies on lower layer error detection (e.g., CRC, Checksum, MIC, etc.). CoAP goes on top of UDP which provides a checksum mechanism over its payload.

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

Minimum MTU. Lower layers need to provide an EAP MTU size of 1020 octets or greater. CoAP assumes an upper bound of 1024 for its payload which covers the requirements of EAP.

Ordering guarantees. EAP relies on lower layer ordering guarantees for correct operation. Regarding message ordering, every time a new message arrives at the authentication service hosted by the IoT device, a new resource is created and this is indicated in a 2.01 Created response code along with the name of the new resource via Location-Path or Location-Query. This way the application indicates that its state has advanced. Although the [RFC3748] states: "EAP provides its own support for duplicate elimination and retransmission", EAP is also reliant on lower layer ordering guarantees. In this regard, the RFC talks about possible duplication and says: "Where the lower layer is reliable, it will provide the EAP layer with a non-duplicated stream of packets. However, while it is desirable that lower layers provide for non-duplication, this is not a requirement". CoAP is providing a non-duplicated stream of packets and accomplish the "desirable" non-duplication. In addition, [RFC3748] says that when EAP runs over a reliable lower layer "the authenticator retransmission timer SHOULD be set to an infinite value, so that retransmissions do not occur at the EAP layer."

NOTE: This document does not assume any specific naming schema. The only requisite that both CoAP client and server MUST agree, is the establishment of a nomenclature indicates that the next URI used to refer to a resource, univocally points to the next expected EAP exchange.

Regarding the CoAP Token, CoAP-EAP will use one of a constant value throughout an authentication. This is because the EAP server will not send a new EAP request until it has processed the expected EAP response. Additionally, there will be a single EAP authentication between the IoT device and the same Controller. This would also enable the possibility of using an Empty Token to reduce the number of bytes.

As we can see, CoAP can fulfill the requirements of EAP to be considered suitable as lower layer.

5.2. Size of the EAP lower layer vs EAP method size

Regarding the impact an EAP lower layer will have to the total byte size of the whole exchange, there is a comparison with another network layer based EAP lower layer, PANA [RFC5191] in [coap-eap]. Comparing the EAP lower layer (alone) and taking into account EAP. On the one hand, at the EAP lower layer level, the usage of CoAP gives important benefits. On the other hand, when taking into account the EAP method overload, this reduction is less but still significant if the EAP method is lightweight (authors of [coap-eap] used EAP-PSK as a representative of a lightweight EAP method). If the EAP method is very taxing, the impact of the reduction in size of the EAP lower layer is less significant. This leads to the conclusion that possible next steps in this field could be designing new EAP methods that can be better adapted to the requirements of IoT devices and networks.

However, the impact of the EAP lower layer itself cannot be ignored, hence the proposal of using CoAP as lightweight protocol for this purpose. Other EAP methods such as EAP-AKA'[RFC5448] or new lightweight EAP methods such as EAP-NOOB [I-D.ietf-emu-eap-noob] or EAP-EDHOC [I-D.ingles-eap-edhoc] that can benefit from a CoAP-based EAP lower layer, as well as new ones that may be proposed in the future with IoT constraints in mind.

5.3. Controller as the CoAP Client

In general, it is assumed that, since the EAP authenticator (Controller) MAY implement an AAA client to interact with the AAA infrastructure. Hence, this endpoint will have more resources or, at least, will not be a so constrained device. Due to the constrained capacities of IoT devices, to relieve them of the retransmission tasks, the Controller is set as the CoAP client, for the main exchange following the recommendations of the [I-D.ietf-lwig-coap] document to simplify the IoT device implementation.

6. Security Considerations

There are some aspects to be considered such as how authorization is managed, the use of MSK as keying material and how the trust in the Controller is established. Additional considerations such as EAP channel binding as per [RFC6677] are also discussed here.

6.1. Authorization

Authorization is part of bootstrapping. It serves to establish whether the node can join and the set of conditions it has to adhere. The authorization data will be gathered from the organization that is responsible for the IoT device and sent to the EAP authenticator. In standalone mode, the authorization information will be in the Controller. If the pass-through mode is used, 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 information 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.

6.2. Freshness of the key material

In CoAP-EAP there is no nonce exchange to provide freshness to the keys derived from the MSK. The MSK and Extended Master Session Key (EMSK) keys according to the EAP Key Management Framework [RFC5247] are fresh key material. Since only one authentication is established per EAP authenticator, there is no need for generating additional key material. In case another authentication to be established, a re-authentication can be done, by running the process again, or using a more lightweight EAP method to derive additional key material as elaborated in Section 3.3.2.

6.3. Channel Binding support

According to the [RFC6677], channel binding related with EAP, is sent through the EAP method that supports it.

To satisfy the requirements of the document, we need to send the EAP lower layer identifier (To be assigned by IANA), in the EAP Lower-Layer Attribute if RADIUS is used.

6.4. Additional Security Consideration

Other security-related concerns can be how to ensure that the node joining the security domain can in fact trust the Controller. This issue is elaborated in the EAP Key Management Framework [RFC5247]. In particular, the constrained node knows it can trust the Controller because the key that is used to establish the security association is derived from the MSK. If the Controller has the MSK, it is clear the AAA Server of the node trusted the Controller, which can be considered as a trusted party.

7. IANA Considerations

Considerations for IANA regarding this document:

*Assignment of EAP lower layer identifier.

*Assignment of the URI /.well-known/a

8. Acknowledgments

We would like to thank as the reviewers of this work: Carsten Bormann, Mohit Sethi, Benjamin Kaduk, Alexandre Petrescu, Pedro Moreno-Sanchez and Eduardo Ingles-Sanchez.

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

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

9. References

9.1. 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>.

[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>.

[RFC6677] Hartman, S., Ed., Clancy, T., and K. Hoeper, "Channel-Binding Support for Extensible Authentication Protocol

(EAP) Methods", RFC 6677, DOI 10.17487/RFC6677, July 2012, <https://www.rfc-editor.org/info/rfc6677>.

[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>.

[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>.

[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>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

9.2. Informative References

[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.

[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)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-36, 16 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-ace-oauth-authz-36.txt>.

[I-D.ietf-emu-eap-noob] Aura, T., Sethi, M., and A. Peltonen, "Nimble out-of-band authentication for EAP (EAP-NOOB)", Work in Progress, Internet-Draft, draft-ietf-emu-eap-noob-05, 12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-emu-eap-noob-05.txt>.

[I-D.ietf-lwig-coap] Kovatsch, M., Bergmann, O., and C. Bormann, "CoAP Implementation Guidance", Work in Progress, Internet-Draft, draft-ietf-lwig-coap-06, 2 July 2018,

<http://www.ietf.org/internet-drafts/draft-ietf-lwig-coap-06.txt>.

[I-D.ingles-eap-edhoc] Sanchez, E., Garcia-Carrillo, D., and R. Marin-Lopez, "EAP method based on EDHOC Authentication", Work in Progress, Internet-Draft, draft-ingles-eap-edhoc-01, 2 November 2020, <https://www.ietf.org/internet-drafts/draft-ingles-eap-edhoc-01.txt>.

[lo-coap-eap] Garcia-Carrillo, D., Marin-Lopez, R., Kandasamy, A., and A. Pelov, "A CoAP-Based Network Access Authentication Service for Low-Power Wide Area Networks: LO-CoAP-EAP - https://www.mdpi.com/1424-8220/17/11/2646", November 2017.

[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>.

[RFC5448] Arkko, J., Lehtovirta, V., and P. Eronen, "Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA')", RFC 5448, DOI 10.17487/RFC5448, May 2009, <https://www.rfc-editor.org/info/rfc5448>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[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>.

[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>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/

RFC8824, June 2021, <https://www.rfc-editor.org/info/rfc8824>.

[TS133.501] ETSI, "5G; Security architecture and procedures for 5G System - TS 133 501 V15.2.0 (2018-10)", 2018.

Appendix A. Examples of Use Case Scenario

For a IoT device to act as a trustworthy entity within a security domain, certain key material is needed to be shared between the IoT device and the Controller.

Next, we elaborate on examples of different use case scenarios about the usage of CoAP-EAP. Generally, we are dealing with 4 entities:

*2 nodes (A and B), which are IoT devices. They are the EAP peers.

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

*1 AAA server (AAA) - Optional. The AAA is an Authentication, Authorization and Accounting Server, which is not constrained. Here, the Controller acts as EAP authenticator in pass-through mode.

Generally, any IoT device wanting to join the domain managed by the Controller MUST perform a CoAP-EAP authentication with the Controller (C). This authentication MAY involve an external AAA server. This means that A and B, once deployed, will run CoAP-EAP once, as a bootstrapping phase, to establish a security association with C. Moreover, any other entity, which wants to join and establish communications with nodes under C's domain must also do the same. By using EAP, we can have the flexibility of having different types of credentials. For instance, if we have a device that is not battery dependent, and not very constrained, we could use a heavier authentication method. With varied IoT devices and networks we might need to resort to more lightweight authentication methods (e.g., EAP-NOOB[I-D.ietf-emu-eap-noob], EAP-AKA'[RFC5448], EAP-PSK[RFC4764], EAP-EDHOC[I-D.ingles-eap-edhoc], etc.) being able to adapt to different types of devices according to organization policies or devices capabilities.

A.1. Example 1: CoAP-EAP in ACE

In ACE, the process of Client registration and provisioning of credentials to the client is not specified. The process of Client registration and provisioning can be achieved using CoAP-EAP. Once the process of authentication with EAP is completed, fresh key material is shared between the IoT device and the Controller. In this instance, the Controller and the Authorization Server (AS) of ACE can be co-located.

Next, we exemplify how CoAP-EAP can be used to perform the Client registration in a general way, to allow two IoT devices (A and B) to communicate and interact after a successful client registration.

Node A wants to communicate with node B (e.g. to activate a light switch). The overall process is divided into three phases. Let's start with node A. In the first phase, the node A (EAP peer) does not yet belong to Controller C's domain. Then, it communicates with C (EAP authenticator) and authenticates with CoAP-EAP, which, optionally, communicates with the AAA server to complete the authentication process. If the authentication is successful, a fresh MSK is shared between C and node A. This key material allows node A to establish a security association with the C. Some authorization information may be also provided in this step. In case EAP is used in standalone mode, the AS itself having information about the devices can be the entity providing said authorization information. If authentication and authorization are correct, node A is enrolled in controller C's domain for a period of time. In particular, [RFC5247] recommends 8 hours, though the the entity providing the authorization information can establish this 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 controller C for authorization to access node B and obtain all the required information to do that securely (e.g. keys, tokens, authorization information, etc.). This phase does NOT require the usage of CoAP-EAP. The details of this phase are out of the scope of this document, and the ACE framework is 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 required to join the controller C's domain. Once it is performed with success, the communications are local to the controller C's domain and there is no need to perform a new EAP authentication as long as the key material is still valid. When the keys are about to expire, the IoT device can engage in a reauthentication as explained in Section 3.3.2, to renew the key material.

A.2. Example 2: Multi-domain with AAA infrastructures

We assume we have a device (A) of the domain acme.org, which uses a specific kind of credential (e.g., AKA) and intends to join the um.es domain. This user does not belong to this domain, for which first it performs a client registration using CoAP-EAP. For this, it interacts with the controller's domain acting as EAP authenticator, which in turn communicates with a AAA infrastructure (acting as AAA client). Through the local AAA server to communicate with the home AAA server to complete the authentication and integrate the device as a trustworthy entity into the domain of controller C. In this scenario, the AS under the role of the Controller receives the key material from the AAA infrastructure

A.3. Example 3: Single domain with AAA infrastructure

A University Campus, we have several Faculty buildings and each one has its own criteria or policies in place to manage IoT devices under an AS. All buildings belong to the same domain (e.g., um.es). All these buildings are managed with a AAA infrastructure. A new device (A) with credentials from the domain (e.g., um.es) will be able to perform the device registration with a Controller (C) of any building as long as they are managed by the same general domain.

A.4. Example 4: Single domain without AAA infrastructure

In another case, without a AAA infrastructure, we have a Controller that has co-located the EAP server and using EAP standalone mode we can manage all the devices within the same domain locally. Client registration of a node (A) with Controller (C) can also be performed in the same manner.

A.5. Other use cases

A.5.1. CoAP-EAP for network access control

One of the first steps for an IoT device life-cycle is to perform the authentication to gain access to the network. To do so, the device first has to be authenticated and granted authorization to gain access to the network. Additionally, security parameters such as credentials can be derived from the authentication process allowing the trustworthy operation of the IoT device in a particular network by joining the security domain. By using EAP, we are able to achieve this with flexibility and scalability, because of the different EAP methods available and the ability to rely on AAA infrastructures if needed to support multi-domain scenarios, which is a key feature when the IoT devices deployed under the same security domain, belong to different organizations. Given that EAP is also used for network access control, we can adapt this service for other technologies. For instance, to provide network access control to very constrained technologies (e.g., LoRa network). Authors in [lo-coap-eap] provide an study of a minimal version of CoAP-EAP for LPWAN networks with interesting results. In this specific case, we could leverage the compression by SCHC for CoAP [RFC8824].

A.5.2. CoAP-EAP for service authentication

It is not uncommon that the infrastructure where the device is deployed and the services of the IoT device are managed by different organizations. Therefore, in addition to the authentication for network access control, we have to consider the possibility of a secondary authentication to access different services. This process of authentication, for example, will provide with the necessary key material to establish a secure channel and interact with the entity in charge of granting access to different services. In 5G, for example, consider a primary and secondary authentication using EAP [TS133.501].

Authors' Addresses

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
30100 Murcia
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
33203 Gijon Asturias
Spain

Email: garciadan@uniovi.es