Workgroup: ACE Working Group
Internet-Draft: draft-ietf-ace-wg-coap-eap-06
Published: 7 December 2021
Intended Status: Standards Track
Expires: 10 June 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 main goals is to authenticate a CoAP-enabled IoT device (EAP peer) that intends to join a security domain managed by a 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.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies an authentication service (application) 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 is an application that allows authenticating two CoAP endpoints by using EAP, and to establish an Object Security for Constrained RESTful Environments (OSCORE) security association between them.

More specifically, this document specifies how CoAP can be used as a constrained, link-layer independent, reliable 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] for this specification. This way, CoAP messages are protected after the authentication. After CoAP-EAP's operation, an OSCORE security association is established between endpoints of the service. Using the keying material derived from the authentication, other security associations could be generated. Appendix A shows how to establish a (D)TLS security association using the keying material from the 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][RFC5247] 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 acting as 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.

The protocol stack is described in Figure 2. CoAP-EAP is an application built on top of CoAP. On top of the application, there is an EAP state machine that can run any EAP method. For this specification, the EAP method MUST be able to derive keying material. CoAP-EAP also relies on CoAP reliability mechanisms in CoAP to transport EAP: CoAP over UDP with Confirmable messages ([RFC7252]) or CoAP over TCP, TLS and Websocket, which is specified in [RFC8323].

```
+----------+        +---------------+            +----------+
| EAP peer |        |      EAP      |            |   AAA/   |
|   peer   |<------>| authenticator |<---------->|EAP Server|
+----------+  CoAP  +---------------+     AAA    +----------+
                                      (Optional)
<---(SCOPE OF THIS DOCUMENT)---->
```

Figure 1: CoAP-EAP Architecture

```
+-------------------------------+
|        EAP State Machine      |
+-------------------------------+ \
|     Application(CoAP-EAP)     |  | This Document
+-------------------------------+ /
| Request/Responses/Signaling   | RFC 7252 / RFC 8323
+-------------------------------+
|    Message / Message Framing  | RFC 7252 / RFC 8323
+-------------------------------+
|Unreliable / Reliable Transport| RFC 7252 / RFC 8323
+-------------------------------+
```

Figure 2: CoAP-EAP Stack

3. CoAP-EAP Operation

Since CoAP-EAP uses reliable delivery in CoAP ([RFC7252], [RFC8323]), EAP retransmission time is set to infinite as mentioned in [RFC3748]. To keep ordering guarantee, CoAP-EAP uses Hypermedia as the Engine of Application State (HATEOAS). Each step during the EAP authentication is represented as a new resource in the EAP peer (CoAP server). The previous resource is removed once the new resource is created indicating the resource that will process the next expected step of the EAP authentication.

An EAP method that does not export keying material MUST NOT be used. One of the benefits of using EAP is that we can choose over a large variety of authentication methods. Although for IoT, where we can find very constrained links (e.g., limited bandwidth) and devices with limited capabilities, EAP methods that do not require many exchanges, with short messages, and that use cryptographic algorithms that are manageable by constrained devices are preferable.

In CoAP-EAP, the IoT device (EAP peer/CoAP server) will only have one authentication session with a specific Controller (EAP authenticator/ CoAP client) and it will not process any other EAP authentication in parallel (with the same Controller). That is, a single ongoing EAP authentication is permitted for the same IoT device and same Controller. Moreover, EAP is a lock-step protocol ([RFC3748]). The benefits of the EAP framework in IoT are highlighted in [eap-framework].

To access the authentication service, this document defines the well-known URI "/.well-known/coap-eap" (to be assigned by IANA). This URI is referring to the authentication service that is present in the Controller so that IoT device can start the service.

3.1. Discovery

Prior to the CoAP-EAP exchange takes place, the IoT device needs to discovers the Controller or the entity that will enable the exchange between the IoT and the Controller (e.g., an intermediary such as a proxy).

The discovery process is out of the scope of this document. This document provides the specification using the mechanisms provided by CoAP to discover the Controller for CoAP-EAP.

The CoAP-EAP application is designated by the well-known URI "coap-eap" for the trigger message (Step 0). The CoAP-EAP service can be discovered by asking directly about the services offered. This information can be also available in the resource directory [I-D.ietf-core-resource-directory].

Implementation Notes: On the methods on how the IPv6 address of the Controller or intermediary entity can be discovered, there can be different methods depending on the specific deployment. For example, on a 6LoWPAN network, the Border Router will typically act as the Controller hence, after receiving the Router Advertisement (RA) messages from the Border Router, the IoT device may engage on the CoAP-EAP exchange. Different protocols can be used to discover the IP of the Controller. Examples of such protocols are Multicast DNS (mDNS) [RFC6762] or DHCPv6 [RFC8415].

3.2. Flow of operation (OSCORE establishment)

Figure 3 shows the general flow of operation for CoAP-EAP to authenticate using EAP and establish an OSCORE security context. The flow does not show a specific EAP method. Instead, we represent the chosen EAP method by using a generic name (EAP-X). The flow assumes that the IoT device knows the Controller implements the CoAP-EAP service. The specific mechanism of discovery is out-of-scope of this document. Some comments about Controller discovery is in Section 3.1.

The steps for the operation happens as follows:

* Step 0. The IoT device MUST start the authentication process by sending a "POST /.well-known/coap-eap" request (trigger message). This message carries the 'No-Response' [RFC7967] CoAP option to avoid waiting for a response that is not needed. This message is the only instance where the Controller acts as a CoAP server and the IoT device as a CoAP client. The message also includes a URI in the payload of the message to indicate to what resource (e.g. '/a/x') the Controller MUST send the first message with the EAP authentication. The name of the resource is selected by the CoAP server as it pleases. After this, the exchange continues with the Controller as a CoAP client and the IoT device as a CoAP server.

* Step 1. The Controller sends a "POST" message to the resource indicated by the IoT device in the Step 0 (e.g., '/a/x'). The payload in this message contains the first EAP message (EAP Request/Identity), the Recipient ID of the Controller (RID-C) for OSCORE and it MAY contain a CBOR array containing a list with the cipher suites (CS) for OSCORE. If the cipher suite is not included the default cipher suite for OSCORE is used. The details of the cipher suite negotiation are discussed in Section 5.1.

* Step 2. The IoT device processes the POST message by passing the EAP request (EAP-Req/Id) to the EAP peer state machine, which returns an EAP response (EAP Resp/Id); it assigns a new resource to the ongoing authentication process (e.g., '/a/y'), and deletes the previous one ('/a/x'). It finally sends a '2.01 Created' response with the new resource identifier in the Location-Path (and/or Location-Query) options for the next step; the EAP response, the Recipient ID of the IoT device (RID-I) and the selected cipher suite for OSCORE are in the payload. In this step, the IoT device MAY create a OSCORE security context (see Section 5.2). The required key, the Master Session Key (MSK), will be available once the EAP authentication is successful in step 7.

*Step 3-6. From now on, the Controller and the IoT device will exchange EAP packets related to the EAP method (EAP-X), transported in the CoAP message payload. The Controller will use the POST method to send EAP requests to the IoT device. The IoT device will use a response to carry the EAP response in the payload. EAP requests and responses are represented in Figure 3 using the nomenclature (EAP-X-Req and EAP-X-Resp, respectively. When a POST message arrives (e.g, '/a/x') carrying an EAP request message, if processed correctly by the EAP peer state machine, returns an EAP Response. Along with each EAP Response, a new resource is created (e.g, '/a/z') for processing the next EAP request and the ongoing resource (e.g., '/a/y') is erased. This way ordering guarantee is achieved. Finally, EAP response is sent in the payload of a CoAP 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 a (4.00 Bad Request). There is a discussion about error handling in Section 3.5.

*Step 7. When the EAP authentication ends with success, the Controller obtains the Master Session Key (MSK) exported by the EAP method, an EAP Success message and some authorization information (i.e. session lifetime) [RFC5247]. The Controller creates the OSCORE security context using the MSK and Sender ID and Recipient ID exchanged in Step 1 and 2. The establishment of the OSCORE Security Context is defined in Section 5.2. Then, the Controller sends the POST message protected with OSCORE for key confirmation including the EAP Success. The Controller MAY also send a Session Lifetime, in seconds, which is represented with an unsigned integer in a CBOR object (see Section 4. If this Session Lifetime is not sent, the IoT device assumes a default value of 8 hours as RECOMMENDED in [RFC5247]. The verification of the received OSCORE protected "POST" message using RID-I (Recipient ID of the IoT device) sent in Step 2 is considered by the IoT device as an alternate indication of success ([RFC3748]). The EAP peer state machine in the IoT device interprets the alternate indication of success similarly the arrival of an EAP Success and returns the MSK, which is used for the OSCORE security context in the IoT device to process the protected POST message received from the Controller.

*Step 8. If the EAP authentication and the verification of the OSCORE protected "POST" in Step 7 is successful, then the IoT Device answers with an OSCORE protected '2.04 Changed'. From this point on, the communication with the last resource (e.g. '/a/w') MUST be protected with OSCORE. If allowed by application policy, same OSCORE security context MAY be used to protect communication to other resources between the same endpoints.

```
               IoT device                              Controller
             -------------                            ------------
                 |  POST /.well-known/coap-eap             |
              0) |  No-Response                            |
                 |  Payload("/a/x")                        |
                 |---------------------------------------->|
                 |                               POST /a/x |
                 |          Payload(EAP Req/Id||CS||RID-C) |
              1) |<----------------------------------------|
                 | 2.01 Created Location-Path [/a/y]       |
                 | Payload(EAP Resp/Id||CS||RID-I)         |
              2) |---------------------------------------->|
                 |                               POST /a/y |
                 |                     Payload(EAP-X Req)  |
              3) |<----------------------------------------|
                 | 2.01 Created Location-Path [/a/z]       |
                 | Payload(EAP-X Resp)                     |
              4) |---------------------------------------->|
                                    ....
                 |                             POST /a/q   |
                 |                     Payload(EAP-X Req)  |
              5) |<----------------------------------------|
                 | 2.01 Created Location-Path [/a/w]       |
                 | Payload (EAP-X Resp)                    |
              6) |---------------------------------------->|
                 |                                         |  MSK
                 |                               POST /a/w |   |
                 |                                  OSCORE |   V
                 | Payload (EAP Success||*Session-Lifetime)| OSCORE
         MSK  7) |<----------------------------------------| CONTEXT
          |      |                                         |
          V      | 2.04 Changed                            |
        OSCORE   | OSCORE                                  |
      CONTEXT 8 )|---------------------------------------->|

                   (*) Session-Lifetime is optional.
```

Figure 3: CoAP-EAP flow of operation with OSCORE

3.3. Reauthentication

When the CoAP-EAP state is close to expire, the IoT device MAY want to start a new authentication process (re-authentication) to renew the state. The main goal is to derive new and fresh keying material (MSK/EMSK) that, in turn, allows deriving a new OSCORE security context, increasing the protection against key leakage. The keying material MUST be renewed before the expiration of the Session-Lifetime. By default, the EAP Key Management Framework establishes a default value of 8 hours to refresh the keying material. Certain EAP methods such as EAP-NOOB [I-D.ietf-emu-eap-noob] or EAP-AKA' [RFC5448] provides fast reconnect for quicker re-authentication. The EAP re-authentication protocol (ERP) [RFC6696] MAY be also used for avoiding the repetition of the entire EAP exchange.

The message flow for the re-authentication will be the same as the one shown in Figure 3. Nevertheless, two different CoAP-EAP states will be active during the re-authentication: the current CoAP-EAP state and the new CoAP-EAP state, which will be created once the re-authentication has finished with success. Once the re-authentication is completed successfully, the current CoAP-EAP state is deleted and the new CoAP-EAP becomes the current one. If by any reason, the re-authentication fails to complete, the current CoAP-EAP state will be available until it expires, or it is renewed in another try of re-authentication.

If the re-authentication fails, it is up to the IoT device decide when to restart a re-authentication before the current EAP state expires.

3.4. Managing the State of the Service

The IoT device and the Controller keep a state during the CoAP-EAP negotiation. The CoAP-EAP state includes several important parts:

* A reference to an instance of the EAP (peer or authenticator/ server) state machine.

* The resource for the next message in the negotiation (e.g '/a/y')

* The MSK exported when the EAP authentication is successful. In particular, CoAP-EAP is able to access to the different variables by the EAP state machine (i.e. [RFC4137]).

* A reference to the OSCORE context.

Once created, the Controller MAY choose to delete it as described in Figure 4. On the other hand, the IoT device may need to renew the CoAP-EAP state because the key material is close to expire, as mentioned in Section 3.3.

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 4.

If the Controller deems necessary, the removal of the CoAP-EAP state from the IoT device before it expires, it can send a DELETE command in a request to the IoT device, referencing the last CoAP-EAP state resource given by the CoAP server, whose identifier will be the last one received (e.g., '/a/w' in Figure 3). This message is protected with the OSCORE security association to prevent forgery. Upon reception of this message, the CoAP server sends a response to the Controller with the Code '2.02 Deleted', which is also protected with the OSCORE security association. If a response from the IoT device does not arrive after EXCHANGE_LIFETIME the Controller will remove the state from its side.

```
          IoT device                               Controller
        -------------                            -------------
             |                                         |
             |                          DELETE /a/w    |
             |                               OSCORE    |
             |<----------------------------------------|
             |                                         |
             | 2.02 Deleted                            |
             | OSCORE                                  |
             |---------------------------------------->|
```

Figure 4: Deleting state

3.5. Error handling

This section elaborates how different errors are handled, from EAP authentication failure, a non-responding endpoint, lost messages or initial POST message arriving out of place.

3.5.1. EAP authentication failure

EAP authentication MAY fail for different situations (e.g. wrong credentials). The result is that the Controller will send an EAP failure because of the EAP authentication (Step 7 in Figure 3). In this case, the IoT device MUST send a response '4.01 Unauthorized' in Step 8. Therefore, Step 7 and Step 8 are not protected in this case because no MSK is exported and the OSCORE security context is not generated.

If the EAP authentication fails during the re-authentication and the Controller sends an EAP failure, the current CoAP-EAP state will be still usable until it expires.

3.5.2. Non-responding endpoint

If, by any reason, one of the entities becomes non-responding, the CoAP-EAP state SHOULD be kept only for a period of time before it is removed. The removal of the CoAP-EAP state in the Controller assumes that the IoT device will need to authenticate again. According to CoAP, EXCHANGE_LIFETIME considers the time it takes until a client stops expecting a response to a request. A timer is reset every time a message is sent. If EXCHANGE_LIFETIME has passed waiting for the next message, both entities will delete the CoAP-EAP state if the authentication process has not finished correctly.

3.5.3. Duplicated message with /.well-known/coap-eap

The reception of the trigger message in Step 0 containing /.well-known/coap-eap needs some additional considerations, as the resource is always available in the EAP authenticator.

If a trigger message (Step 0) arrives to the Controller during an ongoing authentication, the Controller MUST silently discard this trigger message.

If an old "POST /.well-known/coap-eap" (Step 0) arrives to the Controller and there is no authentication ongoing, the Controller may understand that a new authentication process is requested. Consequently, the Controller will start a new EAP authentication. However, the IoT device did not start any authentication and therefore, it has not selected any resource for the EAP authentication. Thus, IoT device sends a '4.04 Not found' in the response (Figure 5).

```
   IoT device                                  Controller
 -------------                               -------------
       |  *POST /.well-known/coap-eap               |
    0) |  , No-Response                             |
       |  Payload("/a/x")                           |
       |              ------------------------->|
       |                            POST /a/x  |
       |               Payload (EAP Req/Id||CS) |
    1) |<----------------------------------------|
       |                                        |
       | 4.04 Not found                         |
       |---------------------------------------->|
       * Old
```

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

3.6. Proxy operation in CoAP-EAP

The CoAP-EAP operation is intended to be compatible with the use of intermediary entities between the IoT device and the Controller, when direct communication is not possible. In this context, CoAP proxies can be used as enablers of the CoAP-EAP exchange.

This specification is limited to use standard CoAP [RFC7252] as well as standardized CoAP options [RFC8613]. It does not specify any addition in the form of CoAP options. This is expected to ease the integration of CoAP intermediaries in the CoAP-EAP exchange.

There is a consideration that needs to be considered, when using proxies in the CoAP-EAP, as the exchange contains a role-reversal process at the beginning of the exchange. In the first message, the IoT device acts as a CoAP client, and the Controller as the CoAP server. After that, remaining exchanges the roles are reversed, being the IoT device, the CoAP server and the Controller, the CoAP client.

4. CBOR Objects in CoAP-EAP

In the CoAP-EAP exchange, there is information that needs to be exchanged between the two entities. Examples of these are the cipher suites that need to be negotiated or authorization information (Session-lifetime). There may be also a need of extending the information that has to be exchanged in the future. This section specifies the CBOR [RFC8949] data structure to exchange information between the IoT device and the Controller in the CoAP payload.

Next, is the specification of the CBOR Object to exchange information in CoAP-EAP

```
CoAP-EAP_Info = {
    ?  1 : array,                      ; cipher suite
    ?  2 : bstr,                       ; RID-C
    ?  3 : bstr,                       ; RID-I
    ?  4 : uint                        ; Session-Lifetime
}
```

Figure 6: CBOR data structure for CoAP-EAP

The parameters contain the following information:

1. cipher suite: It contains a array with the list of the proposed or selected CBOR algorithms for OSCORE. If the field is carried over a request, the meaning is the proposed cipher suite, if it is carried over a response, corresponds to the response.

2. RID-I: It contains the Recipient ID of the IoT device. The Controller uses this value as Sender ID for its OSCORE Sender Context. The IoT device uses this value as Recipient ID for its Recipient Context.

3. RID-C: It contains the Recipient ID of the Controller. The IoT device uses this value as Sender ID for its OSCORE Sender

Context. The Controller uses this value as Recipient ID for its Recipient Context.

4. Session-Lifetime: Contains the time the session is valid in seconds.

The indexes from 65000 to 65535 are reserved for experimentation.

5. Cipher suite negotiation and key derivation

5.1. Cipher suite negotiation

OSCORE runs after the EAP authentication, using the cipher suite selected in the cipher suite negotiation (Step 1 and 2). To negotiate the cipher suite, CoAP-EAP follows a simple approach: the Controller sends a list, in decreasing order or preference, with the identifiers of the supported cipher suites (Step 1). In the response to that message (Step 2), the IoT device sends a response with the choice.

This list is included in the payload after the EAP message with a CBOR array that contains the cipher suites. 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 cipher suite negotiation is shown in Figure 8, where can be appreciated the disposition of both EAP-Request/Identity and EAP-Response/Identity, followed by the CBOR object defined in Section 4, containing in the cipher suite field the CBOR array for the cipher suite negotiation.

```
+-----+-----------+-------+------++-------------+
|Code |Identifier |Length | Data ||cipher suites|
+-----+-----------+-------+------++-------------+
          EAP Packet                 CBOR Object
```

Figure 7: cipher suites are in the CoAP payload

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

Figure 8: cipher suite negotiation

In case there is no CBOR array stating the cipher suites, the default cipher suites are applied. If the Controller sends a restricted list of cipher suites that is willing to accept it MUST include the default value 0 since it is mandatory to implement. The IoT device will have at least that option available.

The cipher suite requirements are inherited from the ones established by OSCORE. By default, the HKDF algorithm is SHA-256 and the AEAD algorithm is AES-CCM-16-64-128. Both are mandatory to implement. The other cipher suites supported and negotiated in the cipher suite negotiation are the following:

0. AES-CCM-16-64-128, SHA-256 (default)

1. A128GCM, SHA-256

2. A256GCM, SHA-384

3. ChaCha20/Poly1305, SHA-256

4. ChaCha20/Poly1305, SHAKE256

This specification uses the (HMAC)-based key derivation function (HKDF) defined in [RFC5869] to derive the necessary key material. Since the key derivation process uses the MSK, which is considered fresh key material, we will use the HKDF-Expand function, which we will shorten here as KDF.

5.2. Deriving the OSCORE Security Context

The derivation of the security context for OSCORE allows securing the communication between the IoT device and the Controller once the MSK has been exported providing, confidentiality, integrity, key confirmation (Step 7 and 8) and detecting a downgrading attack.

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

Master Secret = KDF(MSK, CS | "COAP-EAP OSCORE MASTER SECRET", length)

where:

* The algorithms for OSCORE are agreed in the cipher suite negotiation.

* The MSK exported by the EAP method. Discussion about the use of the MSK for the key derivation is done in Section 7.

* CS is the concatenation of the content of the cipher suite negotiation, that is, the list of cipher suites sent by the Controller (Step 1) the selected option by the IoT device (Step 2). If any of the messages did not contain the CBOR array (default algorithms), the null string is used.

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

* CS and "COAP-EAP 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, CS | "OSCORE MASTER SALT", length)

where:

* The algorithms are agreed in the cipher suite negotiation.

* The MSK exported by the EAP method. Discussion about the use of the MSK for the key derivation is done in Section 7.

* CS is the concatenation of the content of the cipher suite 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).

* CS and "COAP-EAP OSCORE MASTER SECRET" are concatenated.

* length is the size of the output key material.

Since the MSK is used to derive the Master Key, the correct verification of the OSCORE protected request (Step 7) and response (Step 8) confirms the Controller and the IoT device have the same Master Secret, achieving key confirmation.

To prevent a downgrading attack, the content of the cipher suites negotiation (which we refer to here as CS) is embedded in the Master Secret derivation. If an attacker changes the value of the cipher suite negotiation, the result will be different OSCORE security contexts, that ends up with a failure in Step 7 and 8.

The Controller will use the Recipient ID of the IoT device (RID-I) as Sender ID for its OSCORE Sender Context. The IoT device will use this value as Recipient ID for its Recipient Context.

The IoT device will use the Recipient ID of the Controller (RID-C) as Sender ID for its OSCORE Sender Context. The Controller will use this value as Recipient ID for its Recipient Context.

6. Discussion

6.1. 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 but it can benefit for a reliable lower layer. In this sense, CoAP provides a reliability mechanism (e.g. 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/ TCP 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, [RFC3748] 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."

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

Regarding the impact that 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 generates large EAP messages. 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. For example, authors in [coap-eap] used EAP-PSK as an example, since it only involves 4 messages and their length can be less than 60 bytes. Moreover, it only uses symmetric cryptography.

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 EAP methods such as EAP-NOOB [I-D.ietf-emu-eap-noob] or EAP-EDHOC [I-D.ingles-eap-edhoc] that can benefit, as well as new ones that may be proposed in the future with IoT constraints in mind, from a CoAP-based EAP lower layer.

7. 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.

7.1. 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 case of AAA infrastructure is deployed.

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. RADIUS or 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.

7.2. 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 a new MSK is required, 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.

7.3. 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.

7.4. Additional Security Consideration

In the process of authentication, there is a possibility of an entity forging messages to generate denial of service (DoS) attacks on any of the entities involved. For instance, an attacker can forge multiple initial message to start an authentication (Step 0) with the Controller as if they were sent by different IoT devices. Consequently, the Controller will start an authentication per each message received in Step 0, sending the EAP Request/Id (Step 1).

To minimize the effects of this DoS attack, it is RECOMMENDED that the Controller limits the rate at which it processes incoming messages in Step 0 to provide robustness against denial of service (DoS) attacks. The details of rate limiting are outside the scope of this specification. Nevertheless, the rate of these messages are also limited by the bandwidth available between the IoT device and the Controller. This bandwidth will be specially limited in constrained links (e.g., LPWAN). Lastly, it is also RECOMMENDED to reduce at a minimum the state in the Controller at least until the EAP Response/Ids received by the Controller.

Other security-related concerns can be how to ensure that the IoT device joining the security domain can in fact trust the Controller. This issue is elaborated in the EAP Key Management Framework [RFC5247]. In particular, the IoT device 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.

8. IANA Considerations

Considerations for IANA regarding this document:

* Assignment of EAP lower layer identifier.

* Assignment of the URI /.well-known/coap-eap

* Assignment of the media type "application/coap-eap"

* Assignment of the content format "application/coap-eap"

* Assignment of the resource type (rt=) "core.coap-eap"

* Assignment of the numbers assigned for the cipher suite negotiation

* Assignment of the numbers assigned for the numbers of the CBOR object in CoAP-EAP

9. Acknowledgments

We would like to thank as the reviewers of this work: Carsten Bormann, Mohit Sethi, Benjamin Kaduk, Christian Amsuss, John Mattsson, Goran Selander, 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 valuable comments to Alexander Pelov and Laurent Toutain, especially for the potential optimizations of CoAP-EAP.

10. References

10.1. 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>.

[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>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[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>.

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

10.2. 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.

[eap-framework] Sethi, M. and T. Aura, "Secure Network Access Authentication for IoT Devices: EAP Framework vs. Individual Protocols - https://ieeexplore.ieee.org/document/9579387", October 2021.

[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-core-resource-directory] Amsüss, C., Shelby, Z., Koster, M., Bormann, C., and P. Van der Stok, "CoRE Resource Directory", Work in Progress, Internet-Draft, draft-ietf-core-resource-directory-28, 7 March 2021, <https://www.ietf.org/archive/id/draft-ietf-core-resource-directory-28.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.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.

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

[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>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

[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>.

[RFC8415]

Mrugalski, T., Siodelski, M., Volz, B., Yourtchenko, A., Richardson, M., Jiang, S., Lemon, T., and T. Winters, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 8415, DOI 10.17487/RFC8415, November 2018, <https://www.rfc-editor.org/info/rfc8415>.

[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. Flow of operation (DTLS establishment)

CoAP-EAP makes possible to derive a PSK for (D)TLS to allow PSK-based authentication between the IoT device and the Controller. In the instance of using (D)TLS to establish a security association, there is a limitation to the use of intermediaries between the IoT device and the Controller, as (D)TLS breaks the end-to-end communications when using intermediaries such as proxies.

```
       IoT device                                Controller
     -------------                             -------------
                             ...
           | 2.01 Created Location-Path [/a/w]        |
           | Payload (EAP-X Resp)                     |
         6)|---------------------------------------->|
           |                                         | MSK
           |          (D)TLS 1.3 Client Hello        |  |
   MSK  7) |<----------------------------------------|  V
    |      |                                         | DTLS_PSK
    V      |===============DTLS hanshake=============|
 DTLS_PSK  |                                         |
                               *...
                 (*) Protected with (D)TLS
```

Figure 9: CoAP-EAP flow of operation with DTLS

Figure 9 shows the last steps of the operation for CoAP-EAP when (D)TLS is used to protect the communication between the IoT device and the Controller using the keying material exported by the EAP authentication. The general flow is essentially the same as in the case of OSCORE, except that DTLS negotiation is established in Step 7). Once DTLS negotiation has finished successfully the IoT device is granted access to the domain. Step 7 MUST be interpreted by the IoT device as an alternate success indication, which will end up with the MSK and the DTLS_PSK derivation for the (D)TLS authentication based on PSK.

According to [RFC8446] the provision of the PSK out-of-band also requires the provision of the KDF hash algorithm and the PSK identity. To simplify the design in CoAP-EAP, the KDF hash algorithm can be included in the list of cipher suites exchange in Step 1 and Step 2 if DTLS wants to be used instead of OSCORE. For the same reason, the PSK identity is derived from (RID-C) (RID-I) as defined in Appendix A.2.

A.1. Cryptographic suite negotiation for DTLS

It is also possible to derive a pre-shared key for DTLS to establish a DLTS security association after a successful EAP authentication. Analogously to how the cipher suite is negotiated for OSCORE Section 5.1, the Controller sends a list, in decreasing order of preference, with the identifiers of the cipher suites supported (Step 1). In the response, the IoT device sends the choice.

This list is included in the payload after the EAP message with a CBOR array that contains the cipher suites. This CBOR array is enclosed as one of the elements of the CBOR Object used for transporting information in CoAP-EAP (See Section 4. An example of how the fields are arranged in the CoAP payload can be seen in Figure 7.

In case there is no CBOR array stating the cipher suites, the default cipher suites are applied. If the Controller sends a restricted list of cipher suites that is willing to accept it MUST include the default value 0 since it is mandatory to implement. The IoT device will have at least that option available.

The cipher suites are the following:

3. TLS_SHA256

4. TLS_SHA384

5. TLS_SHA512

A.2. Deriving DTLS PSK and identity

To enable DTLS after an EAP authentication using the key material generated, we define the Identity and the PSK for DTLS. The Identity in this case is generated by concatenating the exchanged Sender ID and the Recipient ID.

```
CoAP-EAP PSK Identity = RID-C || RID-I
```

It is also possible to derive a pre-shared key for DTLS [RFC6347], refereed to here as "DTLS PSK", from the MSK between both IoT device and Controller if required. The length of the DTLS PSK will depend on the cipher suite. To have keying material with sufficient length a key of 32 bytes is derived that can be later truncated if needed:

DTLS PSK = KDF(MSK, "CoAP-EAP DTLS PSK", length).

where:

* MSK is exported by the EAP method.
* "CoAP-EAP 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.

Appendix B. 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.

B.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-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 re-authentication as explained in Section 3.3, to renew the key material.

B.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

B.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.

B.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.

B.5. Other use cases

B.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].

B.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