ACE Working Group
Internet-Draft
Intended status: Experimental
Expires: April 11, 2015

R. Sanchez
R. Marin
D. Garcia
University of Murcia
October 8, 2014

EAP-based Authentication Service for CoAP
draft-marin-ace-wg-coap-eap-01

Abstract

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

- Authenticate two CoAP endpoints.
- Bootstrap key material to protect CoAP messages exchanged between them.

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
2. General Architecture 3
3. General Flow Operation 3
3.1. EAP in CoAP with AUTH option 4
3.2. EAP in CoAP with DTLS 7
4. Key Derivation for protecting CoAP messages 9
4.1. Deriving COAP_AUTH_KEY 10
4.2. Deriving DTLS_PSK 10
5. Generating AUTH option 11
6. Implementation 12
7. Future Work: CoAP Relay 13
8. Use Case Scenario 13
9. Acknowledgments 15
10. Security Considerations 15
11. IANA Considerations 15
12. References 15
12.1. Normative References 15
12.2. Informative References 16
Authors' Addresses 17

1. Introduction

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

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

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

- To define a new AUTH option that includes an authentication tag generated with the cryptographic material exported during an EAP authentication. This new option is used to protect the integrity of the CoAP message that carries the AUTH option. (NOTE: Encryption will be considered in the future).

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

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

1.1. Requirements Language

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

2. General Architecture

The Figure 1 shows the architecture defined in this document. Basically a node acting as the EAP peer wants to be authenticated by using EAP. At the time of writing this document, we have considered a model where the EAP peer will act as CoAP server for this service and the EAP authenticator will act as CoAP client and may interact with a backend AAA infrastructure. Nevertheless, a model where the EAP peer act as CoAP client and the EAP authenticator as CoAP server will be also analyzed in the future.

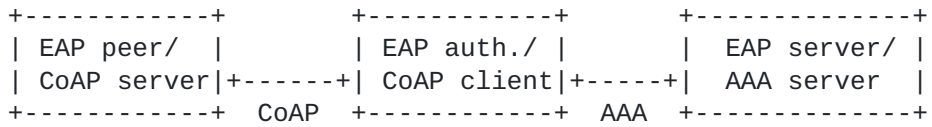


Figure 1: CoAP EAP Architecture

3. General Flow Operation

The authentication service uses the CoAP protocol as transport layer for EAP. CoAP becomes an EAP lower-layer (in EAP terminology). In general, it is assumed that, since the EAP authenticator may need to implement an AAA client to interact with the AAA infrastructure, this endpoint will have more resources. We describe two different sequence flow. First, it is shown in Figure 2 where the AUTH option is used at the end of EAP authentication. Second diagram (see Figure 3) shows the flow when DTLS is used to protect CoAP messages

at the end of the EAP authentication. As an example, both diagrams show the usage of the EAP-PSK method [RFC4764] as authentication mechanism. (NOTE: any EAP method which is able to export cryptographic material should be valid).

3.1. EAP in CoAP with AUTH option

If the EAP peer discovers the presence of the EAP authenticator and wants to start the authentication, it can send a Confirmable "GET /auth" request to the node (Step 0). If the EAP authenticator implements the authentication service will return a 2.05 in a Acknowledgment with a piggy-backed response (Step 0'). Immediately after that, the EAP authenticator will start the authentication service. It is worth noting that the EAP authenticator may decide to start the authentication without waiting for a "GET /auth" message.

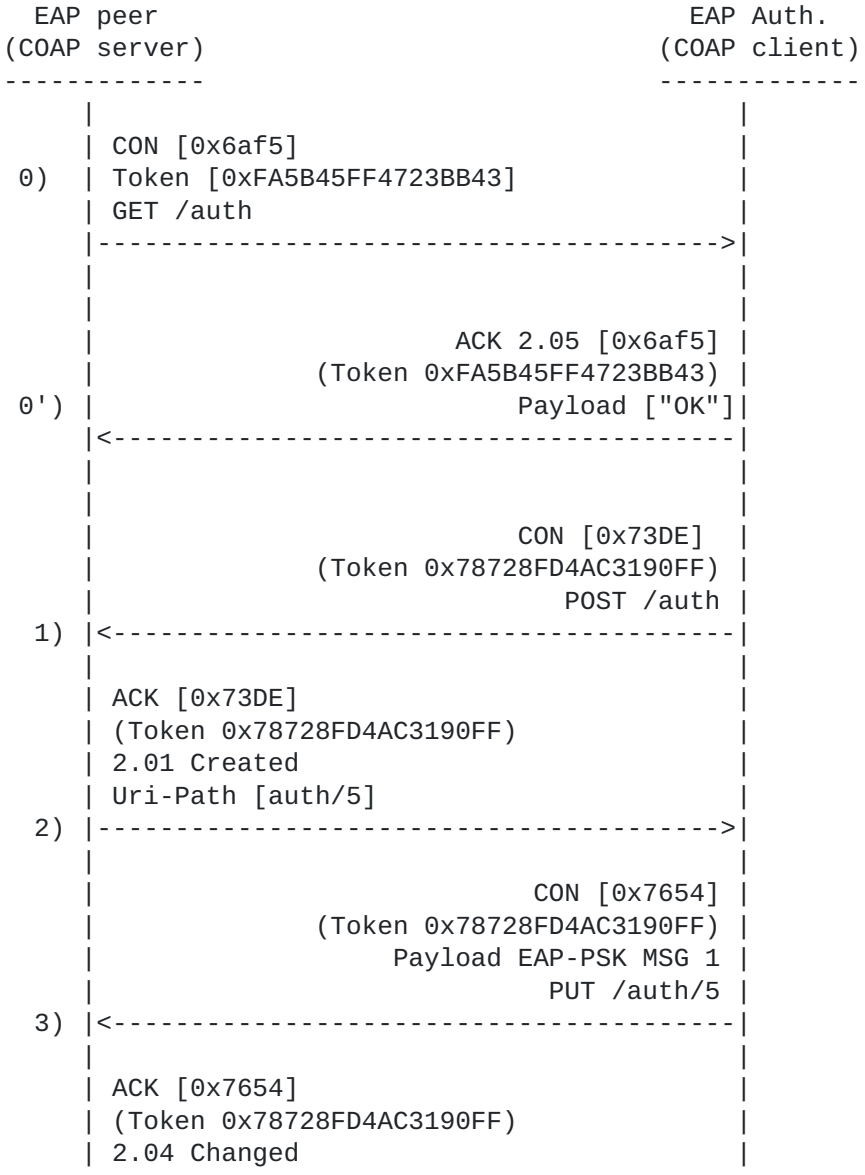
In any case, to perform the authentication service, the CoAP client (EAP authenticator) sends a Confirmable "POST /auth" request to the CoAP Server (Step 1). POST message indicates to the CoAP server the creation of a resource for the EAP-based authentication service. The CoAP server assigns a resource and answers with an Acknowledgment with the piggy-backed resource identifier (Uri-Path) (Step 2). It is assumed that the CoAP server will only have an ongoing authentication and will not process simultaneous EAP authentications in parallel to save resources. Moreover if after a period of time (TBD) no further message is received from the CoAP client, the CoAP server will free the created state. In this moment, the CoAP server has started a resource for the EAP authentication, whose resource identifier value will be used together with the Token option value to relate all the EAP conversation between both CoAP endpoints.

From now on, the EAP authenticator and the EAP peer will exchange EAP packets transported in the CoAP message payload (Steps 3,4,5,6,7). The EAP authenticator will use PUT method to send EAP requests to the EAP peer. The EAP peer will use a Piggy-backed response in the Acknowledgement message to carry the EAP response. At the end of the message exchanges, if everything has gone well, the EAP authenticator is able to send an EAP Success message and both CoAP endpoints will share a Master Session Key (MSK) ([RFC5295])

If the new defined AUTH option is used, an authentication tag is generated with a new key named COAP_AUTH_KEY, derived from the MSK. The Acknowledgment message from the CoAP server will also include an AUTH option so that the CoAP client can verify that the CoAP server obtained the MSK. This is shown in Steps 7 and 8. From that point any exchange (for example, Steps 9 and 10) between both CoAP endpoints are protected with the AUTH option. Finally, the CoAP client MAY send a Confirmable DELETE request to remove all the state

related with the authentication service in the CoAP server (Steps 11 and 12). The CoAP server may decide to remove the state after period of time in case not receiving a DELETE request. This may be easier if the EAP authenticator sends a session lifetime option (TBD) in the Step 7 (where the EAP Success is sent).

On the contrary, if DTLS is used (see Figure 3), a DTLS_PSK is derived from the MSK. Moreover, exchanges between both CoAP endpoints are protected with DTLS from that point.



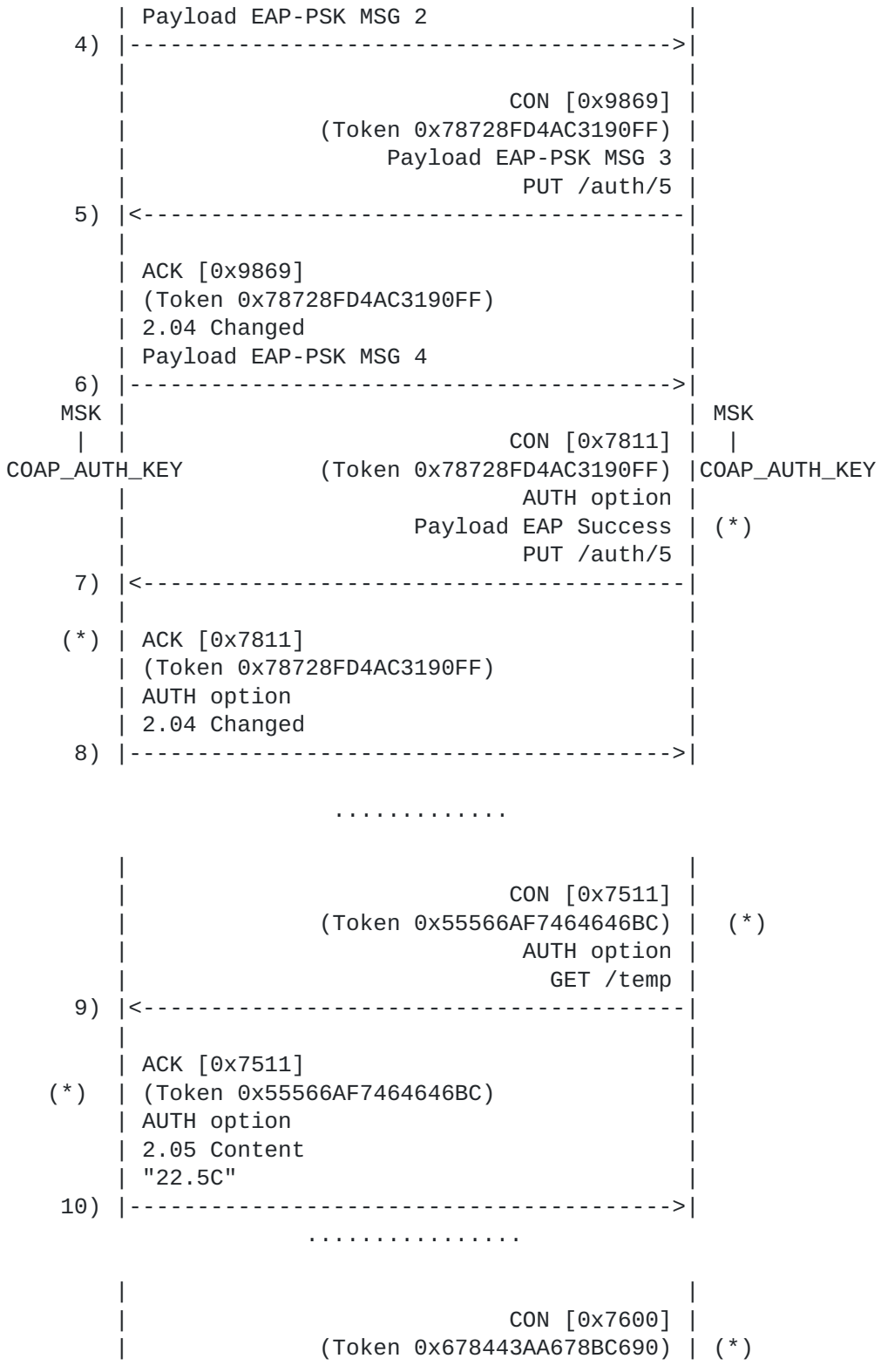
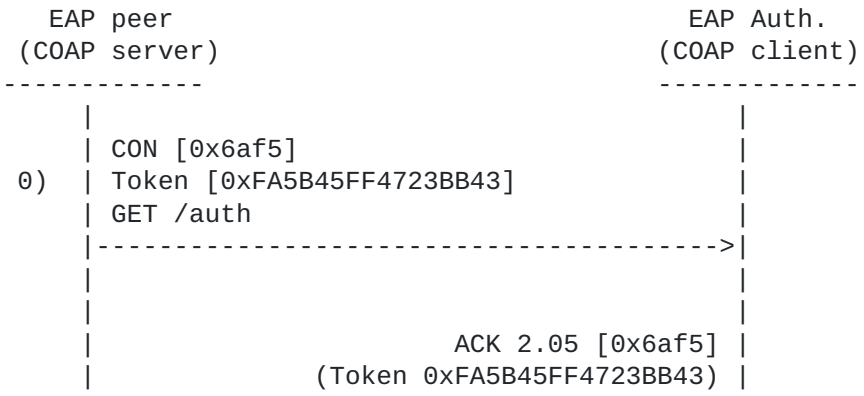
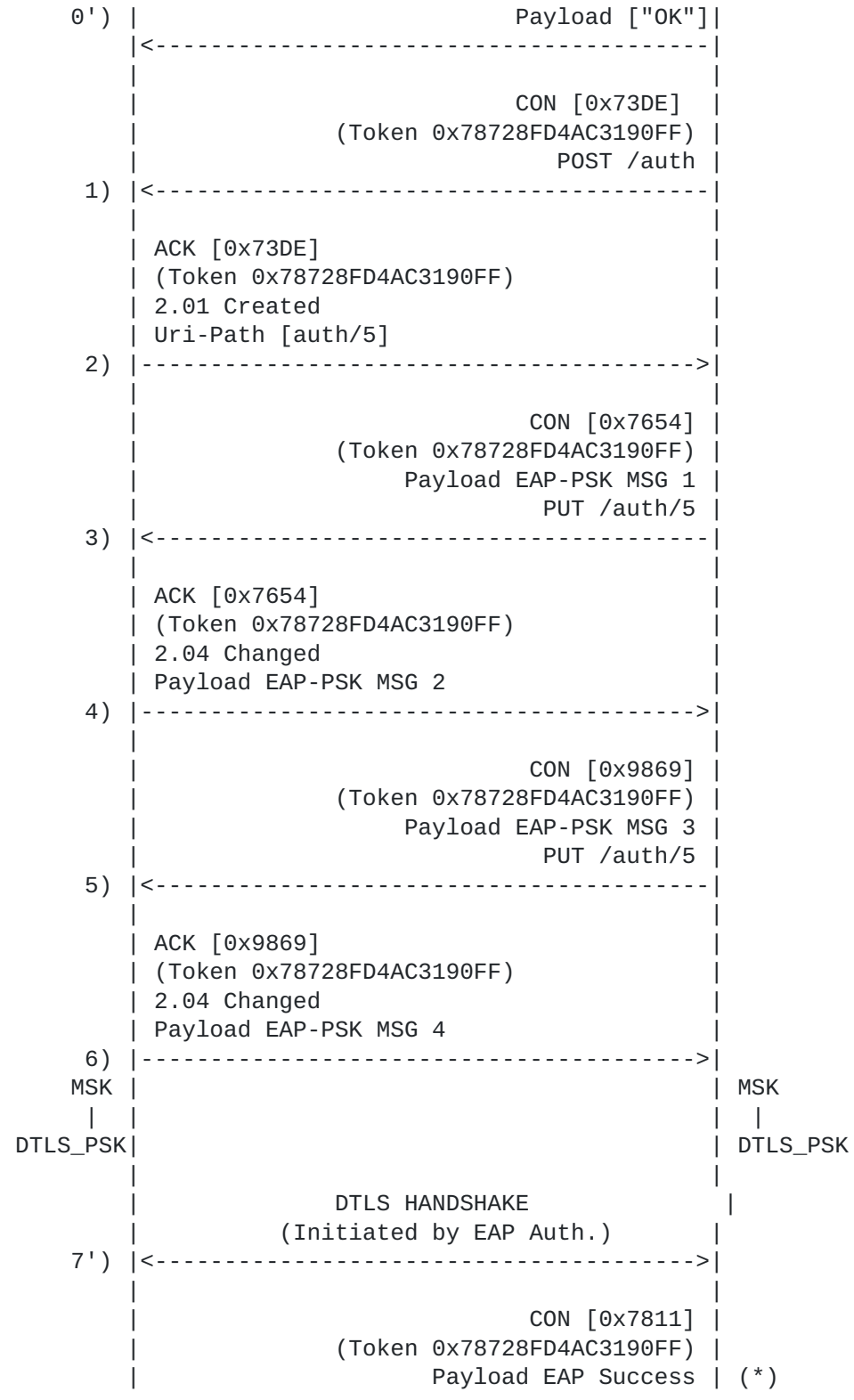

Payload EAP-PSK MSG 2
4)
CON [0x9869]
(Token 0x78728FD4AC3190FF)
Payload EAP-PSK MSG 3
PUT /auth/5
5)
ACK [0x9869]
(Token 0x78728FD4AC3190FF)
2.04 Changed
Payload EAP-PSK MSG 4
6)
MSK
MSK
CON [0x7811]
COAP_AUTH_KEY
(Token 0x78728FD4AC3190FF)
COAP_AUTH_KEY
AUTH option
Payload EAP Success
(*)
PUT /auth/5
7)
(*)
ACK [0x7811]
(Token 0x78728FD4AC3190FF)
AUTH option
2.04 Changed
8)
.....
CON [0x7511]
(Token 0x55566AF7464646BC)
(*)
AUTH option
GET /temp
9)
ACK [0x7511]
(*)
(Token 0x55566AF7464646BC)
AUTH option
2.05 Content
"22.5C"
10)
.....
CON [0x7600]
(Token 0x678443AA678BC690)
(*)

Figure 2: EAP over CoAP with AUTH option

3.2. EAP in CoAP with DTLS

Other possibility at our disposal is to do a DTLS handshake after the MSKs generation and continue the communication between endpoints using CoAP through DTLS as we can see at Figure 3. The Steps 0-6 are the same as the case with AUTH option, however, before continuing with Steps 7 and 8, the EAP authenticator starts the DTLS handshake with the EAP peer (Step 7'). To establish a DTLS Security Association, a key named DTLS-PSK is derived from MSK (see Section 4). In this case the CoAP client can start DTLS before sending the last message containing the EAP Success. Once DTLS is established, any posterior CoAP exchange is protected. Thus, new AUTH option is not needed. A successful DTLS negotiation confirms the possession of DTLS_PSK that, in turn, corroborates that both entities participated in the EAP authentication.




0')
Payload ["OK"]
1)
CON [0x73DE]
(Token 0x78728FD4AC3190FF)
POST /auth
ACK [0x73DE]
(Token 0x78728FD4AC3190FF)
2.01 Created
Uri-Path [auth/5]
2)
CON [0x7654]
(Token 0x78728FD4AC3190FF)
Payload EAP-PSK MSG 1
PUT /auth/5
3)
ACK [0x7654]
(Token 0x78728FD4AC3190FF)
2.04 Changed
Payload EAP-PSK MSG 2
4)
CON [0x9869]
(Token 0x78728FD4AC3190FF)
Payload EAP-PSK MSG 3
PUT /auth/5
5)
ACK [0x9869]
(Token 0x78728FD4AC3190FF)
2.04 Changed
Payload EAP-PSK MSG 4
6)
MSK
DTLS_PSK
MSK
DTLS_PSK
DTLS HANDSHAKE
(Initiated by EAP Auth.)
7')
CON [0x7811]
(Token 0x78728FD4AC3190FF)
Payload EAP Success
(*)

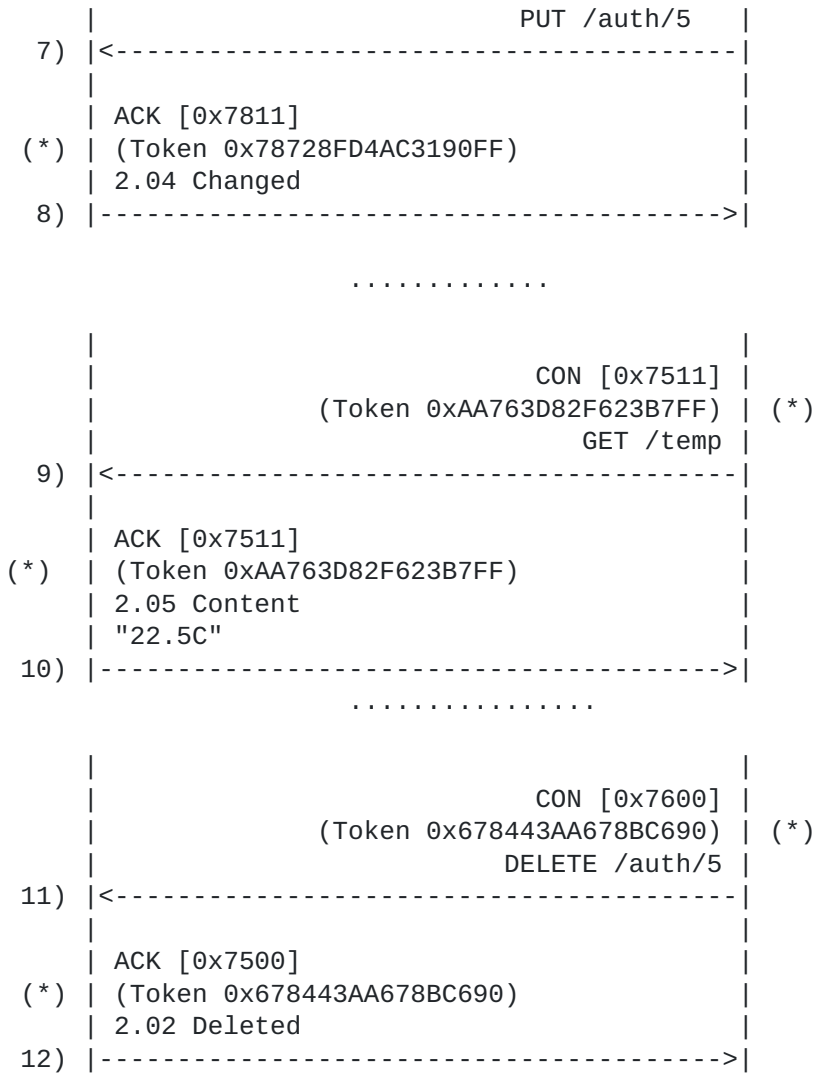


Figure 3: EAP over CoAP with DTLS

4. Key Derivation for protecting CoAP messages

As a result of a successful EAP authentication, both CoAP server and CoAP client share a Master Key Session (MSK). The assumption is that MSK is a fresh key so any derived key from the MSK will be also fresh. We have considered the derivation of either COAP_AUTH_KEY or DTLS_PSK.

4.1. Deriving COAP_AUTH_KEY

A key COAP_AUTH_KEY is derived from the MSK to generate the authentication tag included in the AUTH option. COAP_AUTH_KEY is derived by using AES-CMAC-PRF-128 [RFC4615], which, in turn, uses AES-CMAC-128 [RFC4493]. In this case, rest of CoAP exchanges between both entities can be protected with integrity (NOTE: encryption will be considered in the future) with AUTH option without the need of using DTLS. Thus, all CoAP messages MUST include AUTH option from that point. (NOTE: We understand that this would not be the standard way of protecting CoAP but instead a new way of protecting CoAP messages).

COAP_AUTH_KEY is a 16-byte length key which is computed in the following way:

COAP_AUTH_KEY = AES-CMAC-PRF-128(MSK, "IETF COAP AUTH" || Token Option value, 64, length("IETF COAP AUTH" || Token Option value))

where:

- The AES-CMAC-PRF-128 is defined in [RFC4615]. This function uses AES-CMAC-128 as building block.
- The MSK exported by the EAP method.
- "IETF COAP AUTH" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it). This value is concatenated with the value of the Token Option value.
- 64 is the length of the MSK.
- length("IETF COAP AUTH" || Token Option value) is the length of the label "IETF COAP AUTH" (14 bytes) plus the Token Option value.

4.2. Deriving DTLS_PSK

In the second alternative, a DTLS_PSK is derived from the MSK between both CoAP endpoints. So far, DTLS_PSK will have also 16 byte length and it will derived as follows:

DTLS_PSK = AES-CMAC-PRF-128(MSK, "IETF COAP DTLS" || Token Option value, 64, length("IETF COAP DTLS" || Token Option value)). This value is concatenated with the value of the Token Option value.

where:

- MSK is exported by the EAP method.

- "IETF COAP DTLS" is the ASCII code representation of the non-NULL terminated string (excluding the double quotes around it).
- 64 is the length of the MSK.
- length("IETF COAP DTLS" || Token Option value) is the length of the label "IETF COAP DTLS" (14 bytes) plus the Token Option Value.

As mentioned in [RFC4279], a PSK identity is needed. We are considering the usage of the Token Option value chosen during the EAP authentication as identity. In any case, this still needs further investigation.

5. Generating AUTH option

A new AUTH option is defined in this document for authentication purposes. Following guidelines in [I-D.ietf-core-coap] this option is:

1. Format opaque (sequence of bytes).
2. Elective.
3. Unsafe to Forward.
4. No cacheable.

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

1. Elective (C = 0)
2. Unsafe to Forward (0)
3. NoCacheKey (111)

The number of the AUTH option will fit this pattern: xxx11100

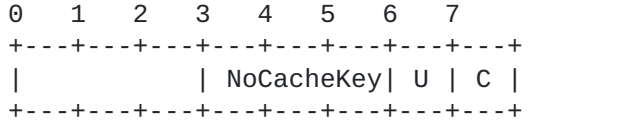


Figure 4: Auth Option Number Mask

First available option number is 01011100 (92).

The resultant AUTH option is:

No.	C	U	N	R	Name	Format	Length	Default
92			x		AUTH	opaque	128	(none)

Figure 5: AUTH Option

C, U, N and R columns indicate the properties, Critical, UnSafe, NoCacheKey and Repeatable, respectively.

To generate the value of the AUTH option, we use AES-CMAC-128 as authentication algorithm. Thus, the AUTH option content will have an authentication tag of 16 bytes.

AUTH Option value = AES-CMAC-128(COAP_AUTH_KEY, MSG, MSG_LENGTH)

where:

- COAP_AUTH_KEY is the key derived in the CoAP Security Association process.
- MSG is the CoAP message including AUTH option filled with zeros.
- MSG_LENGTH. Length of the CoAP message.

After applying AES-CMAC-128 function, the AUTH option value will be set in the AUTH option replacing the zeros.

6. Implementation

At the time of writing this document, we have developed a proof-of-concept based on libcoap ([libCoAP]) in PC platform and started the development of a simulation with COOJA network simulator for Contiki ([Contiki]).

So far, we have implemented an authentication tag by using AES-CMAC-128. However this authentication tag has been included in the payload of two final messages after sending the EAP Success. The implementation of the AUTH option will come soon. Moreover, we have used AES-CMAC-128 for COAP_AUTH_KEY. Since this function does not allow a key longer than 16 bytes, we have used the most significative 16 bytes of the MSK as input key. Since AES-CMAC-PRF-128 eliminates this limitation, we will implement this version instead.

We are using (for the PC version) libeap in wpa-supplicant and hostapd open source software ([hostapd]) to implement the EAP stack and, in particular, the EAP-PSK method.

7. Future Work: CoAP Relay

Architecture explained in Figure 1 assumes that EAP peer can communicate with the EAP authenticator. In certain scenarios, EAP authenticator may not be reachable from the EAP peer if the EAP authenticator is placed several hops-away. In this scenario, described in Figure 6, we are considering the usage a new service that assists the authentication. This service acts as a relay of CoAP messages between the EAP peer and EAP authenticator. We have called the entity in charge of performing this service CoAP relay. The strategy is similar to the one described in PANA Relay ([RFC6345]) or DTLS Relay ([I-D.kumar-dice-dtls-relay]). Unlike CoAP proxy, the CoAP relay is not intended to keep any state (stateless behaviour) and the EAP peer is not assumed to be aware of the presence of the CoAP relay. In any case, this part needs further investigation since CoAP already provides the concept of CoAP proxy and, particular, CoAP-to-CoAP proxy that might be used instead.

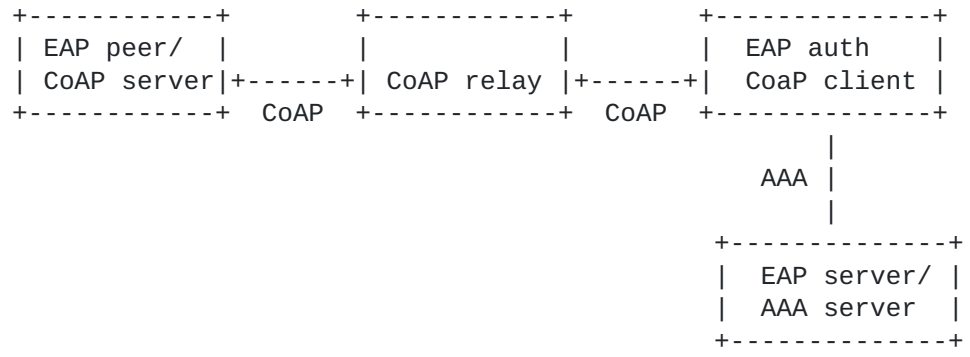


Figure 6: CoAP EAP Relay Architecture

Once the EAP peer has been authenticated, CoAP relay service should not be needed anymore for this EAP peer.

Development of this new service may modify the "Unsafe to Forward" flag of the AUTH option.

8. Use Case Scenario

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

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

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

- 1 controller (C). The controller manages a domain where nodes can be deployed. It can be considered a more powerful machine than the nodes, however it may have some constrained resources.

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

Any node wanting to join the domain managed by the controller, must perform and CoAP-EAP authentication with the controller C. This authentication, as depicted in Figure 6, may involve an external AAA server. This means that A and B, once deployed, will perform this CoAP-EAP once as a bootstrapping phase to establish a security association with the controller C. Moreover, any other entity (i.e. node D) , which wants to join and establish communications with nodes under the controller C's domain must also do the same.

One use case is the following. The node A wants to communicate with node B (e.g. to active a light switch). The overall process is divided in three phases. Let's start with node A. In the first phase, the node A (EAP peer) does not yet belong to the controller C's domain. Then, it communicates with controller C (EAP authenticator) and authenticates with CoAP-EAP, which, in turn, communicates with the AAA server to complete the authentication process. If the authentication is successful, key material is distributed to the controller C and derived by node A. This key material allows node A to establish a security association with controller C. Some authorization information may be also provided in this step. If authentication and authorization are correct, node A is enrolled in the controller C's domain during a period of time. In particular, [RFC5247] recommends 8 hours, though the AAA server can establish 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 the controller C for authorization to access node B and obtain all the required information to do that in a secure manner (e.g. keys, tokens, authorization information, etc.). It does not require the usage of CoAP-EAP. The details of this phase are out of scope of this document.

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

It is worth noting that first phase with CoAP-EAP is only required to join the controller C's domain. Once it is performed with success, the communications are local to the controller C's domain so there is no need to contact the external AAA server.

Another use case is the following. Node D wants to communicate with node A (e.g. to obtain a temperature measurement). To do that, first of all, node D must join the controller C's domain. To do that it performs a CoAP-EAP authentication and authorization with the controller C (first phase). If everything ends with success, the node D can request access to node A to C (second phase). Then if node D is authorized can access to node A (third phase). So, in the end, node D also implements CoAP-EAP as any other constrained node.

9. Acknowledgments

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

This work has been partly funded by European Commission through the FP7- SMARTIE-609062 EU Project.

10. Security Considerations

TBD.

11. IANA Considerations

This document has no actions for IANA.

12. References

12.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.kumar-dice-dtls-relay]
Kumar, S., Keoh, S., and O. Garcia-Morchon, "DTLS Relay for Constrained Environments", draft-kumar-dice-dtls-relay-00 (work in progress), October 2013.

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

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

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, June 2006.

[RFC4615] Song, J., Poovendran, R., Lee, J., and T. Iwata, "The Advanced Encryption Standard-Cipher-based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE)", RFC 4615, August 2006.

[RFC4764] Bersani, F. and H. Tschofenig, "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method", RFC 4764, January 2007.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.

[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, August 2008.

[RFC6345] Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", RFC 6345, August 2011.

12.2. Informative References

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

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

[libCoAP] "C-Implementation of CoAP", January 2013.

Authors' Addresses

Raul Sanchez-Sanchez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 92 81
Email: raul@um.es

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

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

Dan Garcia Carrillo
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 87 71
Email: dan.garcia@um.es