EAP-based Authentication Service for CoAP

draft-ietf-ace-wg-coap-eap-03

Rafael Marín-López, University of Murcia
Dan García-Carrillo, University of Oviedo

IETF 111 meeting, July 29th, 2021
CoAP-EAP – Updates Summary of 03 version

• Added /.well-known/ for both entities
• Changed URI to conform to HATEOAS
• Added Error handling section
• Added Cryptosuite negotiation
• Elaborated process of key derivation
• IANA considerations
CoAP-EAP – Added /.well-known/ for both entities

• Added well-known URI /.well-known/a (To be assigned by IANA)
• Set in both entities, IoT device and Controller

1. NON POST /.well-known/a [NON, MID=50, Token, Options(No-response)]

2. POST /.well-known/a [CON, MID=0x20, Token, Payload(EAP Req/Id)]

3. ACK [MID= 0x20, Token, 2.01 Created, Location-Path(a/x), Payload(EAP Rep/Id)]
CoAP-EAP – Ordering guarantee following HATEOAS

• In the first ACK the server can choose the value of the URI as it pleases.

**Example**

• The server can create a new resource with structure
  • /a/x
    • x -> Value representing the current step in the authentication process
    • Could be a completely different value /randomValue

• The CoAP engine will take care of handling retransmissions, duplicate detection, sending error for non-existing resources, etc.
CoAP-EAP – Error handling

Possible Issues - How to manage out of place POST /well-known/a?

From EAP authenticator to peer

WITH or WITHOUT ONGOING Authentication

- Send a CoAP Reset message. The IoT device did not send the starting message.

```
1  NON POST /well-known/a [NON, MID=50, Token, Options(No-response)]
2  POST /well-known/a [CON, MID=0x20, Token, Payload(EAP Req/Id)]
3  ACK [MID=0x20, Token, 2.01 Created, Location-Path(a/x), Payload(EAP Rep/Id)]
4  POST /well-known/a [CON, MID=0x20, Token, Payload(EAP Req/Id)]
5  RST [MID=0x20, Token, 0.00]
```
CoAP-EAP – Error handling

*From EAP peer to Authenticator*

- WITH ONGOING Authentication
  - OMIT since the message is NON Confirmable with No-Response Option

- WITH no ONGOING Authentication
  - If arrives to the CoAP-EAP application in the Controller, tries to start.
  - Being out of place, the IoT did not purposely send this message, sends Reset.

```
IoT Device/CoAP Server

1. NON POST /well-known/a [NON, MID=50, Token, Options(No-response)]

2. POST /well-known/a [CON, MID=0x20, Token, Payload(EAP Req/Id)]

3. RST [MID= 0x20, Token, 0.00]

Controller/CoAP Client
```
CoAP-EAP – Cyphersuite negotiation

• How to manage the Cyphersuite negotiation within the existing exchange (Not adding more messages)
  • New Option
    • Not our first choice because
      • All CoAP implementations should be updated
      • It will only be used for CoAP-EAP, it is not something useful in other CoAP application
  • **Embedding the cyphersuite negotiation in the CoAP payload**
    • A cleaner option as we do not modify existing implementations
    • Only need a defined structure to parse.

• The cyphersuite negotiation is embedded into the key derivation to bind them and prevent a downgrading attack.
CoAP-EAP – Cryptosuite negotiation

Example of disposition of the CoAP Payload

Exchange with the cryptosuite negotiation

NON POST ./well-known/a [NON, MID=50, Token, Options(No-response)]

POST ./well-known/a [CON, MID=0x20, Token, Payload(EAP Req/Id || CBORArray [0, 1, 2])]  

ACK [MID= 0x20, Token, 2.01 Created, Location-Path(a/x), Payload(EAP Resp/Id || CBORArray [0])]  

IoT Device/CoAP Server

Controller/CoAP Client
CoAP-EAP – Key derivation

• Master Secret = KDF(MSK, CSO | "OSCORE MASTER SECRET", length)
• Master Salt = KDF(MSK, CSO | "OSCORE MASTER SALT", length)
• Recipient ID = KDF(MSK, "OSCORE RECIPIENT ID", length)
• Sender ID = KDF(MSK, "OSCORE SENDER ID", length)

Where:
• KDF is the HKDF-Expand function from (HMAC)-based key derivation function (HKDF) defined in [RFC5869]
• MSK is the Master Session Key derived from the EAP method
• CSO is the concatenated content of the Cyphersuite negotiation. If empty the null-string is used.
• labels are specific for each derivation
• Length is the max length of the output key material. Each one as a specific maximum length specified by OSCORE.

Cyphersuites compatible with OSCORE

<table>
<thead>
<tr>
<th>AEAD</th>
<th>HASH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-CCM-16-64-128</td>
<td>SHA-256</td>
</tr>
<tr>
<td>A128GCM</td>
<td>SHA-256</td>
</tr>
<tr>
<td>A256GCM</td>
<td>SHA-384</td>
</tr>
</tbody>
</table>
CoAP-EAP – Current state

1. NON POST /.well-known/a [NON, MID=50, Token, Options(No-response)]
2. POST /.well-known/a [CON, MID=0x20, Token, Payload(EAP Req/Id || Cyphersuite)]
3. ACK [MID=0x20, Token, 2.01 Created, Location-Path(a/x), Payload(EAP Resp/Id || Cyphersuite)]
4. POST /a/x [CON, MID=0x32, Token, Payload(EAP-X-Req 1)]
5. ACK [MID=0x32, Token, 2.01 Created, Location-Path(a/y), Payload(EAP-X-Resp 1)]
6. POST /a/y [CON, MID=3, Token, Payload(EAP-X-Req n)]
7. ACK [MID=43, Token, 2.01 Created, Location-Path(a/z), Payload(EAP-X-Resp n)]
8. POST /a/z [CON, MID=4, Token, Option(OSCORE) Payload(EAP Success || Session Lifetime)]
9. ACK [MID=43, Token, 2.01 Created, Options(OSCORE)]
IANA considerations

• Assignment of EAP lower layer identifier

• Assignment of the URI /.well-known/a
THANK YOU