Group OSCORE Profile of the Authentication and Authorization for Constrained Environments Framework

draft-ietf-ace-group-oscore-profile-02

Marco Tiloca, RISE
Rikard Höglund, RISE
Francesca Palombini, Ericsson

IETF 120 meeting – Vancouver – July 22nd, 2024

Recap

- **Group OSCORE profile of ACE**
  - Enables access control for accessing resources at group members
  - Group OSCORE [1] used as security protocol between C and RSs
  - The group joining must separately happen first (as defined in [2])
  - Access Token is bound to the already existing Group OSCORE Security Context and to the authentication credential of the Client
- **Properties**
  - Proof-of-Possession of the Client private key
    - Achieved when verifying a first Group OSCORE request from the Client
    - Both the group mode and pairwise mode of Group OSCORE are covered
  - Proof-of-Group-Membership achieved for the exact Client
  - Mutual authentication, when completing a first Group OSCORE exchange

[1] https://datatracker.ietf.org/doc/draft-ietf-core-oscore-groupcomm/
[2] https://datatracker.ietf.org/doc/draft-ietf-ace-key-groupcomm-oscore/

Applicability and Features

- **For application scenarios relying on group communication**
  - A Client wants to access resources at multiple Resource Servers
  - Secure communication by using a shared set of keying material
  - Aims to enforce access control within the group, for resources at servers
- **Separation between group membership and access control**
  - Being a legitimate group member does not naturally imply access rights
  - The following two concepts are separate:
    - access to the secure group communication channel (through membership)
    - access control to the resource space provided by servers in the group - This draft
- **Follows the Zero-Trust paradigm [3]**
  - Focus on resource protection
  - Trust is never granted implicitly, but must be continually evaluated
  - Access control enforcement must be as granular as possible

[3] NIST-800-207: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

Main Updates in v -02 (1/5)

CBOR diagnostic notation in examples

- **Revised to use the construct e'' (*) for Importing values from a CDDL model**
  - e'SOME_NAME' is replaced by the value assigned to SOME_NAME in the CCDL model in Appendix B
  - For example, {e'context_id_param': h'abcd0000', e'salt_input_param': h'00} stands for {71: h'abcd0000', 72: h'00}

- **Use registered CBOR abbreviations**
  - Together with comments
  - "exp" : "1360289224", ⟹ / exp / 4 : 2035353600,

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: application/ace+cbor
Payload:
{
  / audience /        5 : "tempSensor4711",
  / scope /           9 : "read",
    e'context_id_param' : h'abcd0000',
    e'salt_input_param' : h'00',
  e'client_cred_verify' : h'c5a6...f100' / elided for brevity /,
  / req_cnf /         4 : {
    e'kccs' : {
      / sub / 2 : "42-50-31-FF-EF-37-32-39",
      / cnf / 8 : {
        / COSE_Key / 1 : {
          / kty /  1 : 2 / EC2 /,
          / crv / -1 : 1 / P-256 /,
          / x /   -2 : h'd7cc072de2205bdc1537a543d53c60a6
                         acb62eccd890c7fa27c9e354089bbe13',
          / y /   -3 : h'f95e1d4b851a2cc80fff87d8e23f22af
                         b725d535e515d020731e79a3b4e47120'
        }
      }
    }
  }
}
```

```
; ACE Profiles
coap_group_oscore = 5

; OAuth Parameters CBOR Mappings
context_id_param = 71
salt_input_param = 72
client_cred_verify = 73
client_cred_verify_mac = 74

; CBOR Web Token (CWT) Claims
context_id_claim = 51
salt_input_claim = 52

; CWT Confirmation Methods
kccs = 14
```

(*) Proposed in *draft-ietf-cbor-edn-e-ref*

Main Updates in v -02 (2/5)

- **Renamed parameter & claim**
  - Was *contextId_input* ⟹ *context_id*
  - More accurate and consistent with the definitions in Section 3.1.1 and 3.2.2
- **Revised example of CWT Claims Set Using CBOR Encoding (Figure 6)**
  - Consistent with the CBOR abbreviations from the CDDL model
  - Meaningful values used for 'iat' and 'exp'
- **Fixes in the IANA considerations**
  - Fixed name of registry columns
  - Change controller: s/IESG/IETF
- **Nits and editorial improvements**

Main Updates in v -02 (3/5)

At the moment, this profile does not (and cannot) support the dynamic update of access rights for the Client, like other transport profiles of ACE do

Added initial, roadmap text about dynamic update of access rights

- **Sections 3.1.0 - C-to-AS Access Token Request**
  - Intended to also specify how C requests an Access Token that dynamically updates access rights
  - For now, an Editor's note pointing to Section 3.2.1 with the plan

- **Section 4.4 - Processing at the RS**
  - Intended to also specify the processing on the RS when receiving an Access Token that dynamically updates the access rights of C
  - For now, an Editor's note pointing to Section 3.2.1 with the plan

Main Updates in v -02 (4/5)

- **Section 3.2.1 - C-to-AS Access Token Response**
  - Intended to specify how the AS issues Access Tokens that dynamically update access rights

- **Current Editor's notes (content to use across Sections 3.1.0, 3.2.1, and 4.4)**
  - Processing of Access Token Request from C
    - To issue a first Access Token, or one for dynamically updating access rights
  - Usage of concepts (to be) defined in *draft-ietf-ace-workflow-and-params*
    - “Token series”: set of consecutive Access Tokens for the same C-RS (to be specialized here)
    - "token_series_id": identifier of a token series (to be used here, as parameter and token claim)
  - Rationale:
    - In the AS-to-C Response to the first Request: "token_series_id"
    - In the C-to-AS Request for updating access rights:
      - "token_series_id", while other parameters from the first request can be omitted
    - In the Access Tokens of the same token series, the claim "token_series_id" with the same value
      - The RS understands which Access Token to supersede

Main Updates in v -02 (5/5)

Storing multiple Access Tokens per PoP-Key on the RS

- From RFC9200: an RS is recommended to store only one Access Token per proof-of-possession (PoP) key
- New Section 4.6: when using this profile, an RS might have to deviate from that recommendation
  - A Client C with AUTH_CRED_C as PoP key...
  - … might obtain a Token T1 and a Token T2, both bound to AUTH_CRED_C
- Example situations:
  - The RS belongs to an audience AUD1 and a group-audience AUD2
    - T1 (T2) targets AUD1 (AUD2)
  - The RS is a member of the OSCORE groups G1 and G2; both group rely on the same format of public authentication credentials
    - T1 (T2) targets and reflects the membership of C in G1 (G2)
  - The RS uses this profile of ACE and a different one, also relying on asymmetric PoP keys
    - T1 (T2) is issued to be used with this profile (the other profile)

Next Steps

- **Address comment from IANA on Section 11.3**
  - IANA review pointed out that we should add the entry "Original Specification" to the “OAuth Parameters CBOR Mappings” registry
- **Mention the possible use of the ACE alternative workflow**
  - Defined in *draft-ietf-ace-workflow-and-params*
- **Provide guidelines about a follow-up running of the OSCORE profile**
  - A client may wish to communicate to an RS using both OSCORE and Group OSCORE
  - E.g., first use the Group OSCORE profile of ACE; then use the OSCORE profile of ACE
- **Enable dynamic update of access rights**
  - Build on the roadmap in Section 3.2.1
- **Consider setups with multiple application groups and security groups**
  - A client might need an Access Token spanning multiple application/security groups
  - Currently the Access Token can only target one security group

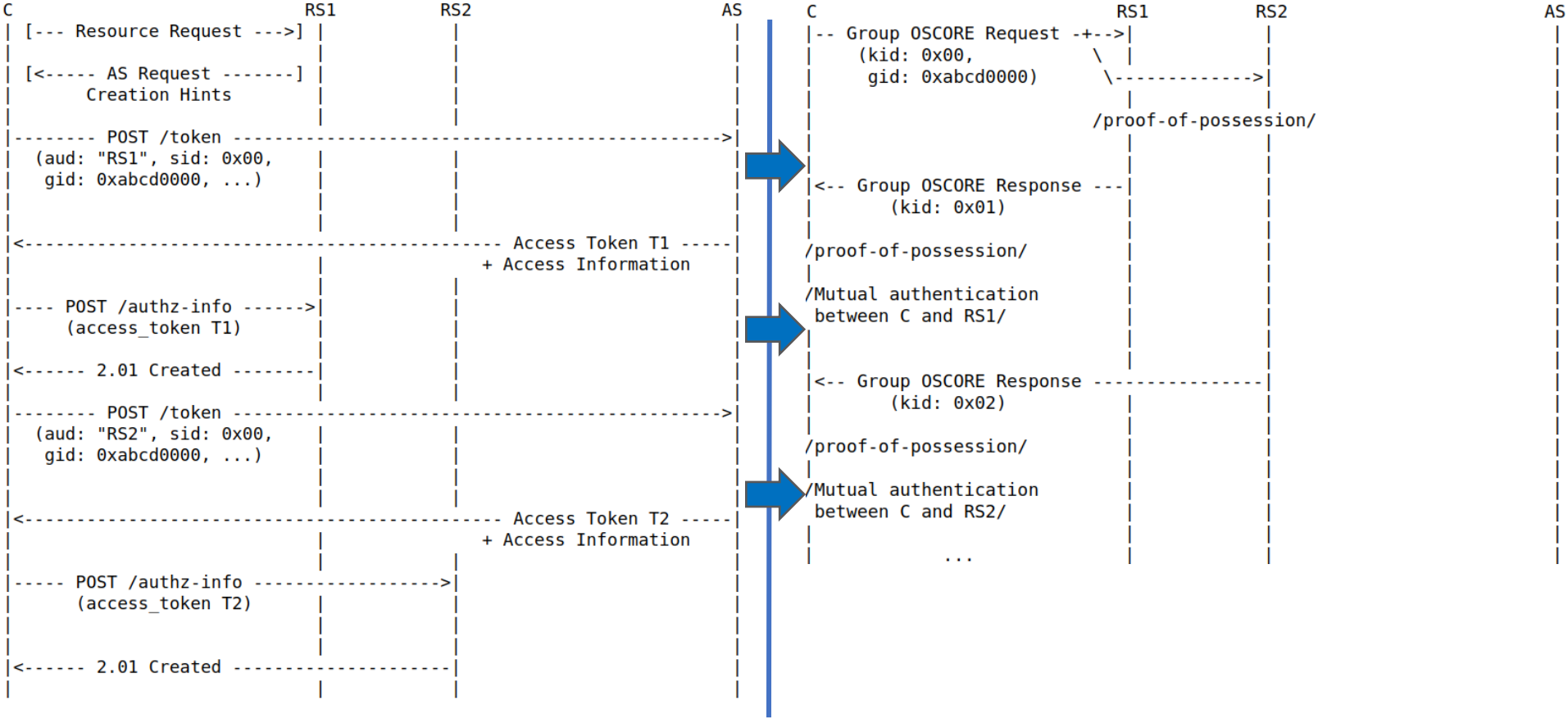
Thank you!

Comments/questions?

https://github.com/ace-wg/ace-group-oscore-profile

Backup

Overview - Protocol flow



Use cases for fine-grained control

- **Group of smart locks**
  - Some clients should only check the lock status (e.g. for a child's account)
  - Some clients can both check and change the lock status (e.g. for a parent's account)
  - The smart locks should be servers only, i.e. cannot lock/unlock each other

- **Building automation (BACnet; with classes of clients)**
  - Light switch (Class C1): issue only low-priority commands
  - Fire panel (Class C2): issue all commands, set/unset high-priority level
  - C1 cannot override C2 commands, until C2 relinquishes high-priority control
  - Goal 1: limit execution of high-priority commands to C2 clients only
  - Goal 2: prevent a compromised C1 client to lock-out normal control

How to accomplish this?

- **What about creating a security group for each different set of access rights?**
  - It scales poorly and is hard to manage
  - Change of access rights ==> Need to join a different group and to rekey groups
- **Better to do it using the ACE framework!**
  - Access to secure group communication with *draft-ietf-ace-key-groupcomm-oscore* [2] **OK**
    - Provisioning of keying material to communicate in the group with Group OSCORE [3]
  - Fine-grained access to the resource space of the RSs in the group with … … ???
- **Current transport profiles of ACE**
  - None of them cover secure group communication between C and RSs
  - None of them uses Group OSCORE as security protocol between C and RSs
- **The right transport profile is missing**

[2] https://datatracker.ietf.org/doc/draft-ietf-ace-key-groupcomm-oscore/
[3] https://datatracker.ietf.org/doc/draft-ietf-core-oscore-groupcomm/

Detailed message exchange (1/2)

- The C-to-AS Access Token Request includes also:
  - 'context_id': Group ID ('kid_context') of the OSCORE group
  - 'salt_input': Client Sender ID ('kid') in the OSCORE group
  - 'client_cred': Client's auth. credential in the OSCORE group
  - 'client_cred_verify': Client's proof-of-possession evidence

- Proof-of-possession evidence in 'client_cred_verify'
  - Computed with the private key in the OSCORE group

```
Header: POST (Code=0.02)
Uri-Host: "as.example.com"
Uri-Path: "token"
Content-Format: "application/ace+cbor"
Payload:
{
  "audience" : "tempSensor4711",
  "scope" : "read",
  "context_id" : h'abcd0000',
  "salt_input" : h'00',
  "client_cred" : {
    "COSE_Key" : {
      "kty" : EC2,
      "crv" : P-256,
      "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
              27c9e354089bbe13',
      "y" : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020
              731e79a3b4e47120'
    }
  },
  "client_cred_verify" : h'...'
  (signature content omitted for brevity),
}
```

Access Token Request

- What is the proof-of-possession (PoP) input to compute the PoP evidence?
  - If **(D)TLS** is used between C and AS ==> an exporter value (Section 7.5 of RFC 8446)
  - If **OSCORE** is used between C and AS ==> PRK = HMAC-Hash(x1 | x2, IKM)
    - x1 = Context ID of the C-AS OSCORE Security Context ;
    - x2 = Sender ID of C in the C-AS OSCORE Security Context;
    - IKM = OSCORE Master Secret of the C-AS OSCORE Security Context

Detailed message exchange (2/2)

- The AS-to-C Access Token Response includes also:
  - Same OSCORE Security Context Object of the Access Token
- The Access Token includes also:
  - ‘salt_input’: Client Sender ID in the OSCORE group
  - ‘contextId_input’ : Group ID of the OSCORE group
  - ‘client_cred’: Client’s auth. credential in the OSCORE Group
- Token POST and response
  - RS checks the auth. credential of C with the Group Manager
  - RS stores {Token; Sender ID; Group ID; C's auth. credential}
  - Another group member cannot impersonate C

```
Header: Created (Code=2.01)
Content-Type: "application/ace+cbor"
Payload:
{
  "access_token" : h'8343a1010aa2044c53 ...'
   (remainder of CWT omitted for brevity),
  "profile" : "coap_group_oscore",
  "expires_in" : 3600,
  "cnf" : {
    "osc" : {
      "alg" : "AES-CCM-16-64-128",
      "id"  : h'01',
      "ms" : h'f9af838368e353e78888e1426bd94e6f',
      "salt" : h'1122',
      "contextId" : h'99'
    }
  }
}
```

Access Token Response

```
{
  "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" :  "temperature_g firmware_p",
  "cnf" : {
    "osc" : {
      "alg" : "AES-CCM-16-64-128",
      "id"  : h'01',
      "ms" : h'f9af838368e353e78888e1426bd94e6f',
      "salt" : h'1122',
      "contextId" : h'99'
    },
  "salt_input" : h'00',
  "contextId_input" : h'abcd0000',
  "client_cred" : {
    "COSE_Key" : {
      "kty" : EC2,
      "crv" : P-256,
      "x" : h'd7cc072de2205bdc1537a543d53c60a6acb62eccd890c7fa
            27c9e354089bbe13',
      "y" : h'f95e1d4b851a2cc80fff87d8e23f22afb725d535e515d020
            731e79a3b4e47120'
    }
  }
}
```

Access Token