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

draft-tiloca-ace-group-oscore-profile-11

Marco Tiloca, RISE
Rikard Höglund, RISE
Ludwig Seitz, Combitech
Francesca Palombini, Ericsson

IETF 117 meeting – San Francisco – July 24th, 2023
Motivation

› **Application scenarios relying on group communication**
  – A Client can access a resource shared by multiple Resource Servers
  – Secure communication can be achieved using a shared set of keying material
  – How to enforce access control within the group, for resources at servers?

› **For very simple use cases (e.g., basic lighting control)**
  – Straightforward and plain access control may be just fine
  – Joining the security group is enough to access resources
  – *Any* group member can do *anything* at *any* other group members’ resource

› **For more advanced use cases**
  – Different clients should have different access rights
  – A more fine-grained approach is necessary
Desired properties

› 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 control to the secure group communication channel (through membership)
    › access control to the resource space provided by servers in the group

› Zero-Trust paradigm [1]
  – Focus on resource protection
  – Trust is never granted implicitly, but must be continually evaluated
  – Access control enforcement must be as granular as possible

How to do that?

- 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

Contribution

› **New Group OSCORE profile of ACE**
  - Enables access control for accessing resources at group members
  - Group OSCORE used as security protocol between C and RSs
  - The group joining must separately happen first! (as defined in [2])
  - Access Token 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 for the exact Client
  - Mutual authentication, when completing a first Group OSCORE exchange

Overview - Protocol flow

[--- Resource Request --->]  
[<---- AS Request ------>]  
Creation Hints

POST /token  
(aud: "RS1", sid: 0x00, 
gid: 0x0abcd0000, ...)

<--- Group OSCORE Request --->  
(kid: 0x00, 
gid: 0x0abcd0000)

RS1
RS2
AS

Access Token T1  
+ Access Information

POST /authz-info  
(access_token T1)

2.01 Created

<--- Group OSCORE Response --->  
(kid: 0x01)

RS1
RS2
AS

/proof-of-possession/

Mutual authentication between C and RS1/

POST /token  
(aud: "RS2", sid: 0x00, 
gid: 0x0abcd0000, ...)

<--- Group OSCORE Response --->  
(kid: 0x02)

RS1
RS2
AS

/proof-of-possession/

Mutual authentication between C and RS2/

Access Token T2  
+ Access Information

POST /authz-info  
(access_token T2)

2.01 Created

Detailed content of exchanged messages available as backup slides
Main updates since version -04

› Removed the “Dual mode” (was a huge appendix)
  – It allowed for both OSCORE and Group OSCORE as security protocols
  – Current focus only on Group OSCORE - The document is greatly shortened and simplified

› Stressed that Access Tokens for group-audiences are seamlessly usable
  – The Access Token is intended to multiple RSs in the OSCORE group
  – The Client can efficiently post such Access Token to many or all of those RSs at once
  – Reminder: the Client obtains the authentication credentials of the RSs in the group from the Group Manager, not from the AS that is intentionally decoupled
Main updates since version -04

› Zero-trust adherence highlighted in the introduction
  – This profile is aligned and in harmony with the NIST Zero-Trust paradigm
  – Trust should not be granted implicitly, and must be continually evaluated

› Formats for public authentication credentials
  – The "req_cnf" specifies the Authentication Credential of the Client
  – The formats of authentication credentials used in Group OSCORE are usable here too:
    › CBOR Web Tokens (CWTs)
    › CWT Claims Sets (CCSs)
    › X.509 certificates
    › C509 certificates
  – The required CWT confirmation methods are defined in draft-ietf-ace-edhoc-oscore-profile
Summary

› **ACE profile for secure group communication**
  - Fine-grained access control *within* an OSCORE group
  - Now focused exclusively on Group OSCORE as security protocol

› **The core of the profile has been stable for a while**
  - Polished, clarified and simplified in the latest two versions

› We think this version is ready to consider for WG Adoption
Thank you!

Comments/questions?
Backup
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
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

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
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

Detailed message exchange (2/2)

Header: Created (Code=2.01)
Content-Type: "application/ace+cbor"
Payload:
{
  "access_token" : h’8343a1010aa2044c53 ...’
  (remainder of JWT omitted for brevity),
  "profile" : "coap_group_oscore",
  "expires_in" : 3600,
  "cntf" : {
    "oso" : {
      "alg" : "AES-CCM-16-64-128",
      "id" : h’01’,
      "ms" : h’f9af83836e353e7888e1426bd9466f’,
      "salt" : h’1122’,
      "contextId" : h’99’
    }
  }
}

Access Token Response

{ "aud" : "tempSensorInLivingRoom",
  "iat" : "1360189224",
  "exp" : "1360289224",
  "scope" : "temperature_g firwame_p",
  "cntf" : {
    "oso" : {
      "alg" : "AES-CCM-16-64-128",
      "id" : h’01’,
      "ms" : h’f9af83836e353e7888e1426bd9466f’,
      "salt" : h’1122’,
      "contextId" : h’99’
    }
  },
  "salt_input" : h’00’,
  "contextId_input" : h’abcd0000’,
  "client_cred" : {
    "OAuth_key" : {
      "kty" : EC2,
      "crv" : P-256,
      "x" : h’d7cc8072e22093d4c53a543d42e9a6ab62e2ccdd89007fa2c7c9e3540899be13’,
      "y" : h’f95e1d4b3812a2cc80faff87d8e23f22a6f7b25d535e153d0c731e79a3b4e7120’
    }
  }
}