IoT Group Membership Management using Decentralized Identifiers and Verifiable Credentials

Nikos Fotiou and Vasilios A. Siris and George Xylomenos and George C. Polyzos
Mobile Multimedia Laboratory, Department of Informatics
School of Information Sciences and Technology
Athens University of Economics and Business
Patision 76, Athens 10434, Greece
E-mail: {fotiou, vsiris, xgeorge, polyzos}@aueb.gr

***Abstract*—Many IoT use cases can benefit from group communication, where a user requests an IoT resource and this request can be handled by multiple IoT devices, each of which may respond back to the user. IoT group communication involves one-to-many requests and many-to-one responses and this creates security challenges. In this paper we focus on the *provenance* challenge, i.e., how a user can determine whether or not a response has been received by an *authorized* device. We provide an effective and flexible solution for securing IoT group communication using CoAP, where a CoAP client sends a request to a CoAP group and receives multiple responses by many IoT devices, acting as CoAP servers. We design a solution that allows CoAP servers to digitally sign their responses in a way that clients can verify that a response has been generated by an authorized member of the CoAP group. In order to achieve our goal we leverage Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). In particular, we consider that each group is identified by a DID, and each group member has received a VC that allows it to participate in that group. The only information a client needs to know is the DID of the group, which is learned using DNSSEC. Our solution allows group members to rotate their signing keys, it achieves group member revocation, and it has minimal communication and computational overhead**

***Index Terms*—CoAP, Group management, DNSSEC.**

I. Introduction

Traditional Internet-based systems involve one-to-one communication, where a *client* request a resource from a *server* (e.g., a web browser requests a web page from an HTTP server). IoT systems on the other hand involve uses cases where one-to-many and many-to-one communication patterns can be the norm and not an exceptional case. For example, mass actuation (e.g., turn on the lights of a smart city), or location-dependent queries (e.g., get the temperature as measured by all sensors in a smart building), or subscription to alerts (e.g., subscribe for a fire alert from any fire sensor in a building). In this paper, we are considering an IoT system where multiple IoT devices or gateways can provide the same "type" of information (e.g., temperature measurements) and all similar information items can be accessed through the same "channel", e.g., a user can collect all temperature measurements through that channel: as a channel abstraction we are considering URIs used by the Constrained Application Protocol (CoAP) group communication [1]. Using CoAP group communication, IoT *endpoints* can become members of *groups* that can be accessed by CoAP clients using multicast IP. In this context new security challenges arise. Our paper focus on the *provenance* verification security challenge, i.e., how CoAP clients can verify that a response has been received by an IoT device *authorized* to participate in the corresponding CoAP group. In order to achieve our goal, we leverage Decentralized Identifiers (DIDs) [2] and Verifiable Credentials (VCs) [3].

A DID is a new form of identification under standardization by W3C. A DID is a URI that can be resolved to cryptographic material that can be used for authenticating the corresponding DID holder. Similarly a VC is a standardized method for asserting attributes of a *subject* using a machine readable encoding. In the proposed solution, a DID is used for authenticating IoT endpoints as well as for protecting CoAP groups. Similarly, VCs are used for asserting CoAP groups membership for IoT endpoints. From a high level perspective, the proposed system operates as follows. Each CoAP group is associated with a DID and the corresponding DID document includes public key(s) used for signing group *membership* VCs for IoT device owners. VCs map the DIDs of IoT device owners to groups in which their devices are allowed to participate. Then, each IoT device owner configures their devices with a signing key, an appropriate DID document and the corresponding membership VC. Using these, IoT endpoints can sign CoAP responses by generating a signature that can be verified using information included in the configured DID document. With our solution we make the following contributions:

- Given the DID of a group, any entity can verify that a signed item has been produced by an authorized member of that group.
- The signing keys of the IoT devices can be rotated without requiring any communication with the group administrator, or other group members, or clients.
- IoT endpoints can be easily added to or removed from a CoAP group.
- IoT endpoints have only to implement legacy digital signature algorithms in order to support our solution.
- Breached singing keys can be easily detected.

The remainder of this paper is organized as follows. We introduce CoAP group communication, DIDs, and VCs and we discuss related work in Section 2. In Section 3, we detail our design. In Section 4, we present the implementation and the evaluation of our solution. Finally, we conclude our paper and we discuss future work items in Section 5.

II. Background and related work

A. CoAP group communication

CoAP [4] is a lightweight protocol, designed to be the "HTTP of the IoT." CoAP resources are identified by a URI scheme, similar to HTTP URIs, and the CoAP interaction model is similar to the client-server model of HTTP. Therefore, IoT endpoints act as CoAP "servers", exposing one or more CoAP URIs that can be accessed by CoAP "clients" using a suitable CoAP "method".

CoAP group communication is a CoAP extension that allows CoAP clients to retrieve (or set) resources from a group of CoAP servers e.g., retrieve the temperature measurements from all sensors of a building, turn on and off all the lights of a smart city and so forth. With CoAP group communication, a request to a CoAP URI is received by all group members. An approach for realizing CoAP group communication is by using IP multicast (section 2 of [1]). With this approach CoAP servers belonging to the same group join an IP multicast address, and CoAP clients learn the IP multicast address of a group using DNS resolution. Then, CoAP clients can send CoAP requests to an IP multicast address, and receive the corresponding response(s) using unicast.

The RFC-recommended approach for realizing CoAP group communication is the following: CoAP group URIs are associated with an IP multicast group, all CoAP servers join the appropriate IP multicast groups, and DNS servers map group URIs to the corresponding IP multicast address.

B. Decentralized Identifiers

Decentralized Identifiers (DIDs) is a new identification system under standardization by W3C. The goal of DIDs is to enable individuals and organizations to generate their own identifiers using systems they trust [2]. In a DID architecture a the Decentralized Identifier (DID) is associated with a DID *document*. A DID document includes, among other things, public keys (or "pointers" to public keys) that can be used as *verification methods*, e.g., for authenticating the DID "owner", or for verifying digital signatures generated by the DID owner. DID documents are usually maintained by a DID *registry*, e.g., a web server or even a blockchain system. Registries are responsible for implementing proper security and access control mechanisms. Registries allow 3^{rd} parties to securely resolve DID documents. Our system uses the did:self DID method and leverages DNS servers and the DNSSEC protocol to implement a DID document registry.

C. Verifiable Credentials

A Verifiable Credential (VC) [3] allows an *issuer* to assert some *attributes* about an entity referred to as the VC *subject*. A VC includes information about the issuer, the subject, the asserted attributes, as well as possible constrains (e.g., expiration date). To facilitate interoperability, the VC data model allows different VC *types* that defines the attributes a VC should include. Our system creates a new VC type named *membership* and that includes a list of *groups* the VC subject is allowed to participate.

D. Related work

The use of DIDs and VCs has been explored in the context of the IoT by some research efforts. Ansay et al. [5] are using DIDs and VCs to provide secure firmware/software update on IoT devices. In that system DIDs and VCs are used for authenticating software providers to IoT devices. Our system on other hand uses DIDs and VCs for authenticating IoT devices to clients, as well as to prove CoAP group membership. Lorenzo et al. [6] leverage DIDs to provide IoT device authentication for the Modbus protocol. Their approach leverages a DID method which is based on Hyperledger Fabric blockchain. Terzi et al. [7] use DIDs and VCs to express access rights related to vehicles, as well as to implement delegation of these rights. Similar to [6] they rely on a blockchain for implementing a DID registry. Fan et al. [8] also utilize blockchain (and smart contracts) to implement a DID registry for DIDs used by IoT devices and manufacturers. Our solution does not require any external entity (such as a blockchain) for retrieving DID documents, instead all required information is stored in a DNS server and it is resolved as part of the CoAP group URI DNS resolution. Diego et al. [9] study the business aspects of DIDs in the context of "IoT as a service" platforms. Their work is orthogonal to our approach which proposes a security solution for the IoT using DIDs and VCs.

In our previous work we use DIDs to protect the routing layer of a Next-Generation Internet architecture from poisoning attacks [10], as well as for providing authenticity for content items stored in the Inter-Planetary File System (IPFS) [11]. In those systems, all DID documents are integrated in the protected resources and these resources are identified by a DID (i.e., a public key). In our system we leverage DNSSEC to store some DID documents. This has the following advantages: (i) protected resources' identifiers(i.e., CoAP group URIs) can be of arbitrary form (including human readable and memorable names), (ii) every time a DID is revoked the corresponding protected resource does not have to be renamed.

III. Design

A. The did:self method

DID specifications allow DID *methods* implementors to decide the information that will be included in the DID documents of their method, as well as how DID documents will be resolved. Our system uses the *did:self* DID method [1] that does not restrict the type of information that can be included in a DID document neither imposes any particular registry type. In particular, owners of did:self DIDs are responsible for disseminating their DID documents by themselves, e.g., by directly transmitting them to interested parties, or by storing them in a Web server: did:self assures that a DID document is correct even if is retrieved over an unsecured channel. Another salient feature of did:self is that it permits multiple valid DID documents for a specific DID to co-exist: in our system we can take advantage of this feature to allow each IoT device

[1] did:self specifications can be found at https://github.com/mmlab-aueb/did-self

to be configured with a different DID document of the same did:self DID (that of the device owner).

A did:self-based DID is a base64url [12] encoded Ed22519 public key [13] prefixed with the string "did:self:".

A *DID document* in did:self may include any of the properties defined by the DID specifications, and it is encoded using JSON. However, and as we detail in Section IV, in this paper we experiment with the "Concise Binary Object Representation" (CBOR) [14] which results in smaller representations. It should be noted that CBOR representation are compliant to DID specifications. A DID document in our system includes the following properties:

- `id`: The DID which the document concerns.
- `verificationMethod`: A list of public keys expressed using the "JsonWebKey2020" notation [15]. Each key in the list is identified by an *id*.
- `authentication`: A list of public keys, or public key identifiers that can be used to authenticate the DID holder.
- `assertion`: A list of public keys, or public key identifiers that can be used to verify digital signatures of VCs.

The private key that corresponds to an *assertion* key is used in our system for signing issued VCs, hence the corresponding public key is used for verifying these signatures. Similarly, the private key that corresponds to an *authentication* key is used for singing CoAP messages, hence the corresponding public key is used for authenticating message senders. Each public key included in the `verificationMethod` property is identified by a unique `id`: there cannot be two keys with same `id` for the same did:self DID even if these keys are defined in different DID document. In our system we take advantage of this property in order to achieve efficient authentication key rotation, as well as for detecting breaches of the private key used by an authentication method. These two security properties are achieved by following a "use the most recent key" principle. In particular, given two keys with the same `id` the one included in the older DID document will be considered invalid and it wil be discarded.

Each DID document in did:self is associated with a *proof* which is a compact encoded JSON Web Signature (JWS) [16]. However, in this paper we use CBOR Object Signing and Encryption (COSE) [17] instead. The payload of the proof is a CBOR object that includes the following claims:

- `jti`: The DID the proof refers to.
- `iat`: The date and time of the proof's generation.
- `exp`: An expiration time.
- `s256`: The base64url encoded hash of the DID document, calculated using SHA-256.

The signature of the proof is generated using the private key that corresponds to the did:self DID and the *Edwards-curve Digital Signature Algorithm* (EdDSA) Given a did:self DID, a DID document, and the document proof, any entity can trivially verify the binding between the DID and the document by executing the following steps:

1) Verify that the DID is equal to the `jti` claim of the proof.
2) Verify that the digest of the DID document is equal to the `s256` claim of the proof.

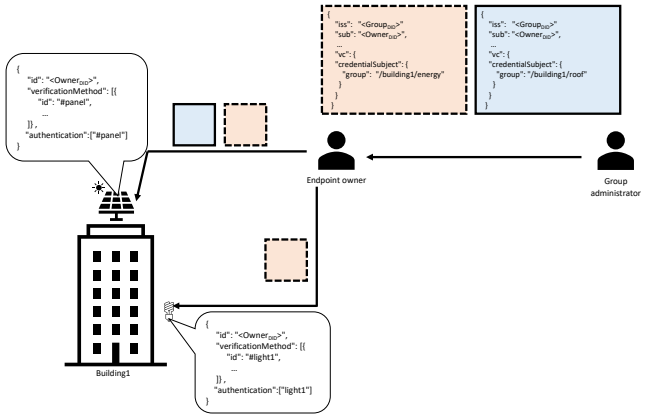

Fig. 1. Endpoint onboarding example.

3) Verify that the proof has not expired.
4) Verify the signature of the proof using the *did:self* DID (recall that a *did:self* DID *is* a public key.)

B. System entities

Our solution considers IoT-based services, e.g., a smart building management system, where IoT *endpoints* can be grouped together. An IoT endpoint can be a an IoT device or gateway.

IoT endpoints are owned by an endpoint *owner*. Each IoT endpoint acts as a CoAP *server* that "hosts" a number of IoT resources. A CoAP *client* can access simultaneously "similar" resources using CoAP group communication.

A group is administrated by a *group administrator* and involves multiple IoT *endpoints*. Moreover, a group is associated with a did:self DID denoted by $Group_{DID}$ and a Fully Qualified Domain Name (FQDN). The DID document of a $Group_{DID}$ includes the *assertion* verification method: the public key defined in this method can be used for verifying VCs issued by the group administrator. A group may include endpoints from multiple owners Group administrators are responsible for controlling which owners can can be part of a group.

The FQDN of each group is associated with an IP multicast address: all CoAP servers are assumed to have joined the appropriate IP multicast groups. The semantics of a group FQDN are deployment specific, e.g., group FQDNs can be location specific such as "building1.floor1"; then a CoAP client may read the temperature measurements of all sensors deployed in the "first floor of building 1" using a CoAP group URI such as "coap://building1.floor1/temperature".

CoAP clients are pre-configured with the required group FQDNs (or they are configured with an appropriate discovery mechanisms). Finally, it is assumed that the DNS resolution process is secured (e.g., using DNSSEC).

C. Owner onboarding and endpoint management

Endpoint owners generate one or more did:self DIDs referred to as $Owner_{DID}$. Moreover, for each of their IoT endpoint, owners generate a public-private key pair and a DID

document that includes the public key in the *authentication* verification method; keys and documents are then installed in the IoT endpoint. Therefore, for each $Owner_{DID}$ there are as many DID documents as the IoT endpoints of the owner. IoT endpoints will use private keys for signing CoAP responses and the installed DID documents for authenticating themselves. Device owners update periodically the keys and the corresponding DID documents of their IoT endpoints. Our system follows the convention that the new keys will use the same key identifier as the one used by the keys being replaced.

A group administrator can authorize an endpoint owner to participate in a group simply by issuing a *membership* VC. In our system, VCs are encoded as CBOR objects and are signed by the group administrator using CBOR Object Signing and Encryption. A CBOR encoded VC in our system includes (among others) the following claims:

- `jti`: An issuer-specific VC *identifier*.
- `iss`: The $Group_{DID}$.
- `sub`: The $Owner_{DID}$.
- `iat`: A timestamp indicating the VC's issuance time.
- `exp`: A timestamp indicating the VC's expiration time.
- `vc`: The actual VC (see the following).

The `vc` property of a *membership* VC includes a claim, named *group*, that contains the FQDN of the group the endpoint owner is allowed to participate. An example of membership VC is presented in the following listing (in this example we are using JSON representation for clarity reason):

Listing 1. Example of membership VC

```
{
  ``jti": ``member1",
  ``iss": ``did:self:...",
  ``sub": ``did:self:...'',
  ``iat": 1650558962,
  ``exp": 1681663521,
  ``vc": {
    ``type":[``membership''],
    ``credentialSubject": {
      ``group": ``building1.floor1",
    }
  }
}
```

Owners will receive as many VCs as the number of groups they can participate. Owners finally, install the VC(s) to the corresponding IoT endpoints. Figure 1 provides an example of the onboarding process. In this example, an owner has configured two IoT endpoints with the appropriate DID documents. Then, the owner receives two VCs one for the group "/building1/energy" and another for the group "/building1/roof". Finally, the owner installs to each endpoint the appropriate VCs. As it can be observed, the roof panel is configured with two VCs which means it can participate to both groups.

D. CoAP request

In order for a client to send a CoAP request to a CoAP group it needs to know the IP multicast address of that group, as well as the $Group_{DID}$ and the corresponding DID document; $Group_{DID}$ is later used for verifying CoAP responses (see section III-E). A CoAP client learns the required information using DNS. The IP multicast address of the CoAP group is stored in a type *A* DNS record that maps the CoAP group FQDN to an IP multicast address. The DID document that corresponds to the $Group_{DID}$ is stored in a *TXT* DNS record of the group FQDN, which a similar approach use by "DNS-Based Authentication of Named Entities" [18].

CoAP clients construct their request and send it to the appropriate IP multicast address. Requests include a *token*, which is then used to match the received responses (the token format and usage are defined in section 5.3.1 of [4]).

E. CoAP response

Each CoAP response in our system includes three CoAP *options* (i.e., a structure akin to HTTP headers, see section 3.1 of [4] for more details). These options, defined by our solution, are: *membership*, *endpoint*, and *attestation*.

The *membership* option includes the appropriate membership VC (and the corresponding proof), which is installed in the IoT device during the onboarding process. The *endpoint* option includes the DID document (and the corresponding proof) of $Owner_{DID}$. Finally, the *attestation* option is a COSE signature. The payload of that signature is a structure that includes the following fields:

- **URI**: The CoAP URI of the requested resource.
- **token**: The token included in the CoAP request.
- **s256**: The hash of the CoAP response payload, calculated using SHA-256.

The digital signature for that option is generated using the private key that corresponds to the authentication key defined in the DID document included in the *endpoint* option.

F. CoAP response verification

Upon receiving a CoAP response a CoAP client executes the following steps:

1) Initially, it extracts the *membership* VC included in the *membership* option and verifies its proof using the appropriate *assertion* key defined in the DID document of the group *administrator*. If the proof includes an expiration time, the client verifies that this time has not passed. If all verifications are successful the client validates the *groups* claim includes the requested group and that the *sub* claim includes the $Owner_{DID}$ of the endpoint that responded.
2) The client extracts the DID document of $Owner_{DID}$ included in the *endpoint* option and verifies its proof. Then it retrieves from that DID document the public key which has been used as the authentication key.
3) The client verifies that the *attestation* option includes the correct values for the URI, token, and sha-256 fields. Then it verifies the digital signature of that option using the extracted authentication key.

The first step of this process verifies that the received response has been generated by an endpoint authorized by the

group administrator to provide responses for that particular CoAP URI. The next steps of this process verify the integrity and the authenticity of the received response.

G. Membership cancellation

Membership to a group can be cancelled either at the endpoint owner level or at the endpoint device level. In the former case the corresponding *membership* VC is *revoked* and all IoT endpoints of the affected provider are removed from the group. In the latter case the corresponding DID document is deleted from the IoT endpoint and the affected endpoint is removed from the group (since it cannot generate anymore valid singed responses).

1) Membership VC revocation: As discussed in section III-A a *membership* VC has an expiration time. Therefore, a trivial approach for removing an endpoint owner from a group is to not update the corresponding *membership* VC. Of course, on the other hand, this creates a security-performance tradeoff.

An alternative solution is to use the revocation scheme described in [19] for verifying the status of a *membership* VC. Based on this scheme, the group administrator maintains a revocation list that concerns all non-expired VCs it has issued. This list is a bitstring and each VC is associated with a position in the list. Revoking a VC means setting the bit corresponding to the VC to 1. Furthermore, each issued VC includes a field named "revocationListIndex" that specifies the position of the VC in the revocation list. Therefore, given a revocation list and the revocationListIndex of a non-expired VC a client can learn the status of that VC.

This revocation scheme implies that clients can access the revocation list. This problem is solved in [19] by requiring from issuers (i.e., the group administrators in our system) to "publish" the revocation list under a URL and include this URL in the VCs. In our solution we follow an alternative approach: since the size of a revocation list will be small, we include it in a *TXT* DNS record of the group FQDN with name "RevocationList". Therefore, clients can easily retrieve it with the DNS resolutions that take place before sending a CoAP message (described in section III-D).

IV. Implementation and evaluation

A. Performance evaluation

We have implemented our DID related operations of our system in Python3 using the cbor2 library[2] and the cose library[3].

The size of a DID document that includes a public key is 242 bytes and the size of the corresponding proof is 124 bytes. The size of a membership VC is 127 bytes and the size of a corresponding proof is 112 bytes.

In our system, the following cryptographic operations are needed. For DID creation, an Ed22519 key pair, a DID document, and the corresponding proof have to be generated. Additionally, a group administrator has to generate a membership VC and an IoT device has to sign the "attestation" CoAP header. Finally, for the CoAP response verification, CoAP client has to verify the DID document, the membership VC and the attestation signature included in the corresponding CoAP headers. Table I shows the time required (in ms) to perform the cryptographic operations of our system, as measured in a Raspberry Pi 4 model B with 2 GB of RAM, as well as in a Espressif ESP32 WROOM-32 IoT device (240MHz dual-core Xtensa LX7 CPU).

It should be noted however that IoT devices are expected to perform only the attestation generation. For a memory efficient implementation of the required signing algorithms, as well as for a discussion related to their energy consumptions, interested readers are referred to [20].

B. Security properties

Assuming that the DNSSEC resolution processes is secured, our solution has the following security related properties:

The integrity of the CoAP response payload is protected. A digest of the CoAP response payload is recorded in the *attestation* option. Since the attestation option also includes the CoAP request token, an attacker cannot replace the payload using an old, valid one. On the other hand, an attacker can replace a payload (and the attestation) with the corresponding fields of a CoAP response generated by another endpoint but for the same request. In that case, the CoAP client will receive twice the same response, signed by the same key, hence the attack will be detected.

The authenticity of the CoAP response is protected. The authenticity of a CoAP response, i.e., the verification of the "binding" between the response payload and the requested CoAP URI, is achieved by including the CoAP URI in the attestation field. If the CoAP URI was not included in the attestation, an attacker could send a CoAP request to a different CoAP URI using the same token as a legitimate request, and then replace a legitimate response with the response it received to his request. Furthermore, the attestation is signed by an endpoint authorized by the publisher to make attestation for that particular URI. Hence, a malicious but authorized endpoint, cannot generate responses for URIs other than those for which it has been authorized.

Authorized endpoints can easily rotate keys. An endpoint can replace its authentication key with a new one by generating a new DID document; apart from that no more actions are required, e.g., the endpoint does not have to receive a new membership VC. In a certificate-based solution on the other hand, an endpoint would have to receive a new certificate from the group administrator.

1) Resilience to attacks: A did:self DID is associated with a private key which is used for signing the proofs of the corresponding DID documents. In our system we have two important did:self DIDs: the $Group_{DID}$ and the $Owner_{DID}$. If the private key that corresponds to $Group_{DID}$ is breached (or lost) then the $Group_{DID}$ must change. This means that the corresponding DNS record must be updated, all generated membership VCs must be revoked, and new membership VCs

[2]https://pypi.org/project/cbor2/
[3]https://pypi.org/project/cose/

TABLE I
CRYPTOGRAPHIC OPERATIONS AND THEIR OVERHEAD.

Operation	Time (ms) using RPi	Time (ms) using ESP32
Ed22519 pair generation	46	452
DID document and proof generation using COSE	2.7	293
Membership VC generation using COSE	2.7	293
Attestation generation using COSE	0.7	82
DID document verification	1.5	160
Membership VC verification	1.5	160
Attestation verification	1.5	160

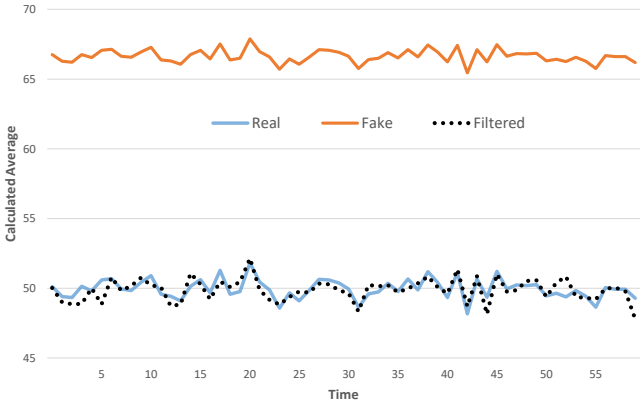

Fig. 2. Impact of an active attacker that generates fake measurements with valid signature. The blue line indicates that correct value that a device would have calculated if there was not an attack. The orange line indicates the estimated value if no defense is considered. The black dotted line indicates the estimated value if duplicate public keys are omitted

must be generated. If the private key that corresponds to an $Owner_{DID}$ is breached (or lost), then the corresponding endpoint owner must generate a new DID and receive new membership VCs. Old VCs must be revoked and the IoT devices must be re-configured with the new VCs and DID documents.

The most risky component of our system is the private key of the IoT endpoints since IoT endpoints are usually exposed to attackers. A breached private key can be used for singing fake CoAP responses. In order to mitigate this attack we take advantage of the key `id` uniqueness property of did:self. In particular, in order for attackers to use a breached key, they must include in the generated responses a valid DID document. But since DID documents are signed by the endpoint owner, the attacker can only use the same DID document as the victim endpoint. This means that a CoAP client will receive two responses signed by the same key. The simplest strategy in that case is to ignore both these responses. In order to evaluate the effectiveness of this countermeasure we emulated an attack scenario. In particular we consider a group of 100 endpoints. These endpoints produce an integer measurement that follows a uniform distribution between 40 and 60 every 1 minute and a CoAP client is interested in learning the average value of these measurements. We assume that an attacker has breached the private keys of half of these devices and produces fake measurements with value equal to 100. We emulated a period of one hour, i.e., the CoAP client calculated 60 times the average value. Each average calculation is independent of the previous one and the client does not apply any heuristic for detecting abnormal values. Figure 2 shows the impact of this attack. The blue line shows that the average value of the measurements produced only by the legitimate devices. The yellow line shows the average value calculated by a "naive" client that takes into consideration all 150 received measurements (100 valid and 50 malicious). Finally the black dotted line shows the average value calculated by a client that filters out measurements that include duplicate keys. As it can be seen, the value calculated by the user that filters out the measurements with the same public key is very close to the real one, nevertheless there are differences due to the lower number of samples.

C. Comparison to other approaches

1) Registry-based DID methods: Being registry-less, did:self DID method allows a DID identifier to be mapped to multiple DID documents. Our system takes advantage of this property and allows endpoint owners to define a single DID and then use a different DID document for each IoT endpoint, each of which includes only a single key used by that particular endpoint. This makes management of IoT endpoints easier, since an endpoint owner can easily add or remove an IoT endpoint. On the other hand, if a registry-based approach was used (e.g., did:web) all public keys of all IoT endpoints should had been included in the (single) DID document, hence, adding or removing an IoT endpoint means that the corresponding DID document must be updated as well.

Additionally, since DID documents in did:self are "self-protected" (as they are signed by the DID owner) they can be included directly in CoAP responses. On the other hand, in a registry-based system DID documents must be resolved from a registry. This not only adds communication overhead (e.g., extra roundtrips for resolving the DID document) but it also creates new security threats since the registry must be trusted, secured, and available.

2) Lightweight certificates: Certificate-based solutions, such as the Simple Public Key Infrastructure (SPKI) [21], can be used as a mechanism alternative to VCs in our system. The advantage of VCs is that it is easier to extend their data model to include new features. For example, our system uses such data model extension to include information that can be used for determining the revocation status of a VC. Furthermore, VCs allow using DIDs for describing the VC subject and when a VC is combined with a DID method such as did:self, many entities can securely share the same VC. Using a certificate-based approach with a certificate that includes the public key

of the endpoint owner would require the endpoint owner to issue another certificate for each IoT endpoint. Therefore, if an owner manages x IoT endpoints, each of which participates in the same y groups, using our approach, the owner has to generate x public key pairs and copy y VCs to each IoT endpoint, whereas in a approach the uses certificates the owner would have to generate x public key pairs, sign $x*y$ certificates and install y certificates to each endpoint.

3) VC-less approach: An alternative approach for achieving the same functionality of our system is to not use VC at all and instead included the DIDs of endpoint owners to the DID documents of $Group_{DID}$. This has the advantage that CoAP responses become shorter (since they do not include a membership VC) and revocation becomes simpler (since the group owner has simply to update the corresponding DID document). On the other hand adding or removing a new endpoint owner requires modification of the corresponding DNS record, so depending on the frequency with which endpoint owners are added or removed to a group the VC-less approach may be a better or worse option. In any case, VCs allow to express more complex relationships than a DID document. For example there can be cases where an endpoint owner is allowed to participate in a group only during specific times: such advanced trust relationships can be easily expressed using VCs.

V. Conclusions and future work

In this paper, we presented a solution for securing group communications with an application to IoT CoAP-based group communication. Our solution leverages Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) to offer efficient group administration, increased security, and better key management. Our solution is lightweight with low computational and communication overhead, and it builds on well supported and widely used digital signature schemes.

Our solution uses the did:self method which is a DID method that does not impose particular technology for implementing the registry that maintains DID documents. Future work in this area includes the investigation of alternative DID methods.

Although we presented our solution in the context of IoT group communication, our goal was to provide a generic design that can be used in other group communication systems. For this reason, we did not consider security mechanisms related to CoAP, such as " Object Security for Constrained RESTful Environments (OSCORE)"; a realization of our system specifically for CoAP group communication would probably considered these solutions. Similarly, our solution assumed IP multicast as the enabler of group communication, nevertheless, related efforts have investigated alternative solutions such as Software-Defined Networking-based group communication, or even group communication based on the Information-Centric Networking (ICN) paradigm; our solution is agnostic to the group communication mechanism but we expect that there will be mutual benefits from an integrated approach.

References

[1] A. Rahman and E. Dijk, "Group communication for the constrained application protocol (CoAP)," IETF, RFC 7390, 2014.

[2] M. Sporny et al., "Decentralized identifiers (dids) v1.0," W3C, W3C Proposed Recommendation, 2021, https://www.w3.org/TR/did-core/.

[3] ——, "Verifiable credentials data model 1.0," W3C, W3C Recommendation, 2019, https://www.w3.org/TR/verifiable-claims-data-model/.

[4] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," IETF, RFC 7252, 2014.

[5] R. Ansey, J. Kempf, O. Berzin, C. Xi, and I. Sheikh, "Gnomon: Decentralized identifiers for securing 5g IoT device registration and software update," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–6.

[6] S. Figueroa-Lorenzo, J. Aorga Benito, and S. Arrizabalaga, "Modbus access control system based on ssi over hyperledger fabric blockchain," *Sensors*, vol. 21, no. 16, 2021.

[7] S. Terzi, C. Savvaidis, K. Votis, D. Tzovaras, and I. Stamelos, "Securing emission data of smart vehicles with blockchain and self-sovereign identities," in *2020 IEEE International Conference on Blockchain (Blockchain)*, 2020, pp. 462–469.

[8] X. Fan, Q. Chai, L. Xu, and D. Guo, "Diam-iot: A decentralized identity and access management framework for internet of things," in *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, ser. BSCI '20, New York, NY, USA, 2020, p. 186191.

[9] S. de Diego, C. Regueiro, and G. Maci-Fernndez, "Enabling identity for the iot-as-a-service business model," *IEEE Access*, vol. 9, pp. 159 965–159 975, 2021.

[10] N. Fotiou, Y. Thomas, V. A. Siris, G. Xylomenos, and G. C. Polyzos, "Securing named data networking routing using decentralized identifiers," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 2021, pp. 1–6.

[11] N. Fotiou, V. Siris, and G. Polyzos, "Enabling self-verifiable mutable content items in IPFS using Decentralized Identifiers," in *DI2F: Decentralising the Internet with IPFS and Filecoin, IFIP Networking 2021 workshop*, 2021.

[12] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," Internet Requests for Comments, IETF, RFC 4648, October 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4648.txt

[13] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of cryptographic engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[14] C. Bormann and P. Hoffman, "Concise Binary Object Representation (CBOR)," IETF, RFC 8949, 2020.

[15] W3C Credentials Community Group, "DDID Specification Registries," Working Group Note, 2021. [Online]. Available: https://www.w3.org/TR/did-spec-registries/

[16] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Signature (JWS)," Internet Requests for Comments, IETF, RFC 7515, May 2015.

[17] J. Schaad, "CBOR Object Signing and Encryption (COSE)," IETF, RFC 8152, 2017.

[18] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," IETF, RFC 6698, 2012.

[19] M. Sporny, D. Longley Eds., "Revocation list 2020," Draft Community Group Report, 2021. [Online]. Available: https://w3c-ccg.github.io/vc-status-rl-2020/

[20] Z. Liu, H. Seo, A. Castiglione, K.-K. R. Choo, and H. Kim, "Memory-efficient implementation of elliptic curve cryptography for the internet-of-things," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 521–529, 2018.

[21] L. af Heurlin, "Authorization certificate based access control in embedded environments," Master's thesis, Aalto University, Espoo, Finland, 2015.