Capabilities-based access control for IoT devices using Verifiable Credentials

Nikos Fotiou, Vasilios A. Siris, George C. Polyzos
Mobile Multimedia Laboratory
Department of Informatics
Athens University of Economics and Business, Greece
{fotiou,vsiris,polyzos}@aueb.gr

Yki Kortesniemi, Dmitrij Lagutin
Department of Communications and Networking
School of Electrical Engineering
Aalto University, Finland
{yki.kortesniemi,dmitrij.lagutin}@aalto.fi

***Abstract*—Capabilities-based access control is a promising paradigm that can handle the particularities of IoT systems. Nevertheless, existing systems are not interoperable and they have limitations, such as lack of proof of possession, inefficient revocation mechanisms, and reliance on trusted third parties. In this paper we overcome these limitations by designing and implementing a system that leverages Verifiable Credentials (VCs) to encode the access rights. Our solution specifies protocols for requesting and using VCs that can be mapped to OAuth 2.0, includes an efficient and privacy preserving proof of possession mechanism, and it supports revocation. We implement and evaluate our solution and we show that it can be directly used even by constrained devices.**

***Index Terms*—Decentralized Identifiers, OAuth 2.0, Proof-of-Possession, Internet of Things**

I. Introduction

Access control for the Internet of Things (IoT) is challenging as IoT devices are often constrained (memory-, power-, and computation-wise) and they may have limited or even no network connectivity, but they are still expected to handle a heterogeneous, dynamic set of users [1]. At the same time, user privacy is not always protected as many access control solutions require the user to reveal their identity to the devices they use even when that is not necessary for the safe use of the device.

Many IoT systems apply custom, inefficient, coarse-grain access control mechanisms, resulting in unauthorized access to devices [2]. Additionally, traditional Role/Attribute-based Access Control Systems impose intolerable management overhead since IoT devices have to be continuously configured with up-to-date Access Control Lists (ACLs). In order to overcome this problem, many systems deploy and rely on trusted intermediate entities, which are responsible for maintaining the ACLs and take access control decisions on behalf of the IoT devices. However, this creates availability issues, e.g., in the recent Facebook outage, Facebook's employees couldn't use the smart locks in Facebook facilities.[1] Capabilities-based access control (CapBAC) is a promising paradigm that can address these issues since it does not require the IoT devices to maintain ACLs; instead user *capabilities* are encoded in tokens, which can be validated by the device itself. Whilst most of the existing CapBAC solutions define their own encoding schemas for the capabilities, our solution relies on Verifiable Credentials [3], which is currently a W3C Recommendation.[2] A *Verifiable Credential (VC)* provides a cryptographically secure and machine-verifiable means for expressing digital and real-world credentials in the cyber world. In contrast to commonly used *identity certificates* (e.g., X.509 certificates) that link the user's identity to a public key and then only provide a binary identification, i.e., either the whole identity of the subject is revealed, or nothing is revealed, VCs (as *authorization certificates*) can be used to reveal only the required attributes of a subject [4]. In particular, the user's real identity does not always have to be revealed, as anonymous or even ephemeral identifiers can be utilized to protect user privacy. In our solution, VCs are used as *self-descriptive tokens* that include the *capabilities* of a user over the IoT device; then, IoT devices can easily enforce access control simply by verifying the validity of the VC.

The VC data model, which is currently a W3C recommendation, allows the definition of various VC *types* for different uses. Each type defines the attributes that a VC should contain, thus facilitating interoperability within that use case. Our solution leverages this property and defines a new VC type for device access control: any security system or IoT device can easily integrate our approach by supporting our VC type.

Many existing solutions treat (capabilities) tokens as mere "bearer" tokens, where anyone having access to the token can utilise it — this can pose problems if the token leaks to unintended users. A more secure approach is to issue credentials, which specify the intended user's (subject's) identifier: then the credential can only be used by proving control of that identifier. Our solution leverages *Decentralized Identifiers* (DID) [5] for authenticating both the VC *issuers* and *subjects*. Our prototype implementation utilizes the *did:key* [6] method that not only allows VCs to be bound to a DID controlled by the user, but also allows users to easily create a different DID for each target IoT device and even replace them after each use, thus avoiding user tracking and correlation. Additionally, the did:key method does not require the device to be online when the credential is used, thus supporting a wide range of

[1]https://www.businessinsider.com/facebook-employees-no-access-conference-rooms-because-of-outage-2021-10

[2]https://www.w3.org/standards/types#REC

use cases. Nevertheless, our design is DID method agnostic hence, depending on the capabilities of the IoT device and the use case, other DID methods (including methods that require online connection for the device) can also be used.

While the VC data model specifies the format of a VC and the entities involved in the VC lifecycle, it does not specify any protocol for *issuing* or using VCs. Our system defines such protocols that can be mapped to standard OAuth 2.0 authorization flows, thus facilitating easy integration with legacy systems.

Finally, many existing solutions do not provide token revocation; our solution specifies a lightweight, privacy-preserving VC revocation mechanism that can be used even with disconnected IoT devices.

In summary, the contributions of our work are the following:

- We design an interoperable access control solution that can be easily integrated into existing systems.
- We show that our solution is lightweight and it has intriguing security and privacy features.
- We provide a lightweight, privacy preserving revocation mechanism and we consider various deployment strategies.
- We add support for DIDs without binding our solution to a specific DID method, enabling a wide range of applications.
- We implement a prototype of our solution and evaluate its performance with constrained devices.

The remainder of this paper is organized as follows. In Section 2 we introduce VCs and DIDs and we discuss related work. In Section 3 we give an overview of our solution and we detail its design in Section 4. We present the implementation and evaluation of our solution in Section 5. Finally, we discuss various aspects of our solution in Section 6 and we conclude our paper in Section 7.

II. Background and related work

This section first introduces the verifiable credentials and decentralized identifiers, and then covers the previous work on capability-based access control for IoT devices.

A. Verifiable credentials

A Verifiable Credential (VC) [3] allows an *issuer* to assert some *attributes* of a *subject*. A VC includes information about the issuer, the subject, the asserted attributes, as well as possible constrains (e.g., expiration date). Then, holders of a VC can prove to a *verifier* that they possess a VC with certain characteristics. To facilitate interoperability, the VC data model allows different VC *types* that defines the attributes a VC should include. A VC type is defined in a *context*, which is a URL. An example of VC type is "UniversityDegreeCredential" defined in the context "https://www.w3.org/2018/credentials/examples/v1". As it can be seen (by visiting the context URL) this credential type includes attributes such as "alumniOf", "degree", "college", and so forth. Our solution leverages this property and defines a new VC type for device access control: any security system or IoT device can easily integrate our approach by supporting our VC type.

B. Decentralized Identifiers

W3C defines Decentralized identifiers (DID) as "a new type of identifier for verifiable, self-sovereign digital identity. DIDs are fully under the control of the DID subject (the identity owner), independent from any centralized registry, identity provider, or certificate authority." [5] Additional information associated with the DID can be stored in a *DID Document*, which describes the subject of a DID, including mechanisms, such as public keys, that the DID subjects use to prove their ownership of the DID. The process of retrieving a DID Document for a DID is called *DID resolution*. Depending on the DID method, some DIDs are resolved using a domain-specific distributed ledger (e.g., the Sovrin DID [7]), some are resolved using trusted Web server (e.g., the did:web DID [8]), and some are exchanged directly between the parties involved (e.g., the Peer DID [9]). The *did:key* [6] method utilized in our prototype uses public keys, encoded in a particular format as DIDs. Hence, did:key does not require any resolution (as the DID contains the public key), thus making it particularly suitable for constrained offline IoT devices.

C. Related work

Our solution is motivated by the broad body of recent work that highlights the need for re-visiting access control in the IoT [10]–[13]. These papers postulate that access control in IoT systems cannot be handled using legacy mechanisms, since IoT systems usually involve many users, require complex rules that take into consideration the physical world, and consider more advanced user relationships and access rights.

Many existing solutions propose capabilities-based access control (CapBAC) system for the IoT. Gusmeroli et al. [14] design a system where capabilities are encoded in signed XML documents using a system-specific schema. Additionally, a revocation and authentication mechanisms are discussed, but their actual design is left deployment specific. Similarly, Hernadez-Ramos et al. [15] propose a CapBAC system where capabilities are represented as signed JSON objects using a system-specific format, and af Heurlin [16] proposed a solution using SPKI authorization certificates, but neither solution supports revocation. Heracles [17] is a CapBAC system for the industrial IoT. Heracles uses a custom token format, which includes a unique user identifier. This identifier however is not used for proving token possession. Moreover, all tokens of a user include the same identifier, hence the user can be tracked. Our solution provides proofs of possession by using dynamic user identifiers, hence preventing user tracking.

Related work leverages DIDs and CapBAC for adding access control in the IoT, by introducing intermediate entities, which are responsible for taking access control decisions on behalf of the IoT devices. These can be trusted network entities (as in [18], [19]) or even a blockchain-based smart contract [20]. In our solution, we remove the need for such an

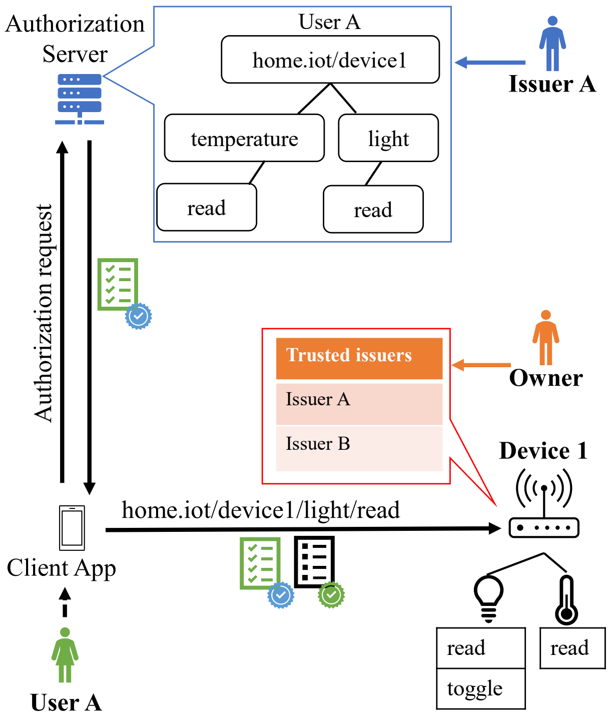

Fig. 1. Solution overview

intermediate entity by allowing even constrained IoT devices to enforce access control on-device.

Our system leverages VCs for expressing capabilities because VCs are well understood techniques being standardized. Additionally, supporting a specific VC type is straightforward, hence interoperability can be supported with low effort. Related approaches that can be used instead of VCs in a system similar to ours are Macaroons [21], Authorization Capabilities for Linked Data (ZCAP-LD) [22], and capabilities as defined by the WAVE framework [23].

III. Solution overview

As illustrated in Fig. 1, our solution considers an IoT system composed of the following entities: IoT *devices* managed by their *owners*, *users* wishing to interact with devices using a *client application*, and authorization *issuers* that grant users access to the devices through an *authorization server*. Each IoT device provides *resources* (e.g., in Fig. 1 the device provides a temperature sensor and a light switch) and each resource allows *operations* (e.g., *read* temperature, *read* switch status, *toggle* switch status). For confidentiality, authorization servers and devices can be accessed using a secure communication protocol (e.g., our implementation uses HTTPS).

In simple use cases, e.g., a smart home, an owner and an issuer can be the same entity, but there can be cases where these are two distinct entities, i.e. the owner can *delegate* the authorization decisions to a separate issuer. Suppose a situation where an office space containing IoT devices is leased to some company; in this case the owner is the building owner, while the tenant is a trusted issuer that can issue credentials for users, which are office workers or visitors. Naturally, there can be multiple trusted issuers for each owner (e.g. multiple tenants in an office building).

From a high-level perspective these entities interact with each other as follows. Issuers configure their authorization servers with the *capabilities* each user should have and IoT devices are configured with lists of *trusted issuers*. A user wishing to perform an operation on a IoT device, authenticates themselves to an authorization server and requests "authorization"; authorization is granted in the form of a *Verifiable Credential* (VC) that includes the operations that the user can perform, as well as a user controlled, ephemeral DID. With the newly acquired VC, the user sends a request to the IoT device together with a *proof* that that the user indeed controls the DID included in the VC. The IoT device verifies the validity of both the VC and the proof, and if the VC permits the requested operation, the IoT device executes it.

Our system achieves the following:

- The VC generation process involves no communication with the IoT device allowing the device to remain offline.
- IoT devices do not maintain any user-related information, only a list of trusted issuers and the required VC type.
- Users' access rights can be easily modified/removed.
- User tracking can be prevented even when IoT devices collude with anonymous and ephemeral subject DIDs.

IV. System design

We now detail the design of our system, present the *CapabilitiesCredential* VC type, and specify protocols for requesting and using a VC in our system.

A. Definitions

In the rest of this paper we are using the following notation. We denote by Pub_X and DID_X the public key and DID of an entity X respectively. Moreover each device is identified by a URL denoted by URL_{device} and each resource has a *name* which is unique in the context of an IoT device; hence, a resource is uniquely identified by the pair $(URL_{device}, resource\ name)$. Let $R_{name} = [O_a, O_b, .., O_x]$ be a resource named $name$ that can be accessed using operations $O_a, O_b, .., O_x$. We denote by $\vec{R}_{URL_{device}}$ the set of resources R of an IoT device URL_{device}. Similarly, let $C_{name} = [O_a, O_b, .., O_x]$ be the operations a user is allowed to invoke on a resource R_{name}, i.e., the *capabilities* of the user over R_{name}. We denote by $\vec{C}_{URL_{device}}$ the set of capabilities the user has over all resources of an IoT device URL_{device}. By definition, $C_{name} \subseteq R_{name}$ and $\vec{C}_{URL_{device}} \subseteq \vec{R}_{URL_{device}}$. This relationship is illustrated in Fig. 2, where the first graph represents the set $\vec{R}$ on an IoT device and the second graph represents the capabilities set $\vec{C}$ of a user for the same device. Finally, we denote by S_{user} the "account" information (e.g., a username and a password) that a user uses for authenticating to an authorization server when requesting a credential.

B. *The* CapabilitiesCredential *VC type*

In our system we define a new VC *type* named *CapabilitiesCredential*. A VC of this type includes a property named *capabilities* that expresses the resource operations that a VC subject can invoke. This property includes pairs of resource names and allowed operations, e.g., the subject of the VC included in the third column of Fig. 2 is allowed to invoke the "read" operation of the "temperature" and "light" resources. This credential type is defined in the *context* https://mm.aueb.gr/contexts/capabilities/v1.

VCs in our system are encoded as JSON Web Tokens (JWT) [24] which include the following claims:[3]

- **iss**: The DID of the issuer.
- **sub**: The DID of the VC subject.
- **aud**: The URL of the target IoT device
- **nbf**: A timestamp before which the VC is not valid.
- **exp**: A timestamp indicating VC's expiration time.
- **vc**: This composite claim describes the actual capabilities granted and it includes the following properties:
  - **context**: The context of the credential.
  - **type**: The type of the credential (i.e., *CapabilitiesCredential*).
  - **capabilities**: The capabilities property.

Finally, each VC is embedded in a JSON Web Signature (JWS) [25] that can be verified using the public key.

C. Operations

We now detail the operations of our system.

1) Setup: During this phase users register with an issuer and agree on a secret S_{user} that can be used for authenticating them. Furthermore, each IoT device is configured with a list of DID_{issuer} identifiers, which belong to trusted issuers. Finally, the authorization server of the issuer is configured with the capabilities sets $\vec{C}$ of their users.

2) Credential request: Prior to requesting a credential, users generate a new DID $DID_{subject}$, which will be used as the DID of the VC subject for this particular device. Then they send to the issuer their secret S_{user}, the generated DID, and the URL of the device they want to access. The issuer authenticates the user, constructs the appropriate VC, and sends it to the user. Therefore, the following exchange of messages takes place:

$$U \rightarrow I : S_{user}, URL_{device}, DID_{subject}$$

$$I \rightarrow U : VC\{URL_{device}, DID_{subject}, capabilities\}$$

3) Access request: A user can request a device to perform an operation on a particular resource by providing a suitable VC, as well as a suitable *proof of possession*. The proof demonstrates that the user knows the private key corresponding to the VC subject, and therefore, the provided VC has been issued to this particular user. This proof is a simple digital signature over the request parameters and a nonce used for preventing replay attacks. Upon receiving the user request, the device then performs the following verifications:

1) It extracts DID_{issuer} from the received VC and it examines if the issuer is trusted.
2) It verifies that the VC is valid (i.e. current time is between *nbf* and *exp*).
3) It verifies that the VC is of type *CapabilitiesCredential*
4) It extracts the *capabilities* property of the *vc* claim and it verifies that the entry for the requested resource name includes the requested operation.
5) It resolves the public key that corresponds to DID_{issuer} and verifies the VC signature.
6) It resolves the public key that corresponds to $DID_{subject}$ and validates the provided proof.

D. Revocation

Our revocation mechanism is based on the system described in [26]; a similar approach for VC revocation is followed by a recent W3C draft [27]. In order to support revocation, an issuer maintains a revocation list that covers all *not expired* VCs it has issued. This list is a simple bitstring and each credential is associated with a position in the list (each revocable VC includes a property named "revocationListIndex" that specifies the position in the revocation list). Revoking a VC then means setting the bit corresponding to the VC to 1. Since the list includes only non-expired VCs, its size is tolerable for most use cases. For example, an issuer that issues on average 100 VCs per day with lifetime equal to one month, would only need 30×100 bits to store its revocation list.

Upon receiving a request that includes a non-expired VC that support this revocation mechanism, the IoT device verifies the status of the VC by examining the value of the bit of the corresponding revocation list. When an entity requests from an issuer a revocation list, the issuer encodes it in a JWT, it adds a timestamp, and it signs it. Depending on the capabilities of the IoT device, various design choices can be considered for retrieving the revocation list.

1) Devices with full connectivity: The device fetches the list from a URL specified in the credential itself. Since the revocation list concerns many VCs, it can be cached by the device for an "acceptable" period of time (e.g. the list may contain a validity period during which no new lists will be issued).

2) Devices with limited connectivity: The device (or its owner) can periodically pull the latest version of the revocation lists of all trusted issuers and store it in a location accessible by the device.

3) Devices with no connectivity: In that case, the user is responsible for retrieving the signed revocation list from the issuer and including it in the access request. The device must verify the signature of the JWT, as well as its "freshness".

V. Implementation and Evaluation

We have implemented a prototype of the solution and taken measurements to evaluate it's performance.

[3]Our system does not use the (optional) VC *id* property, hence, the JWT-encoding of a VC does not include the 'jti' claim.

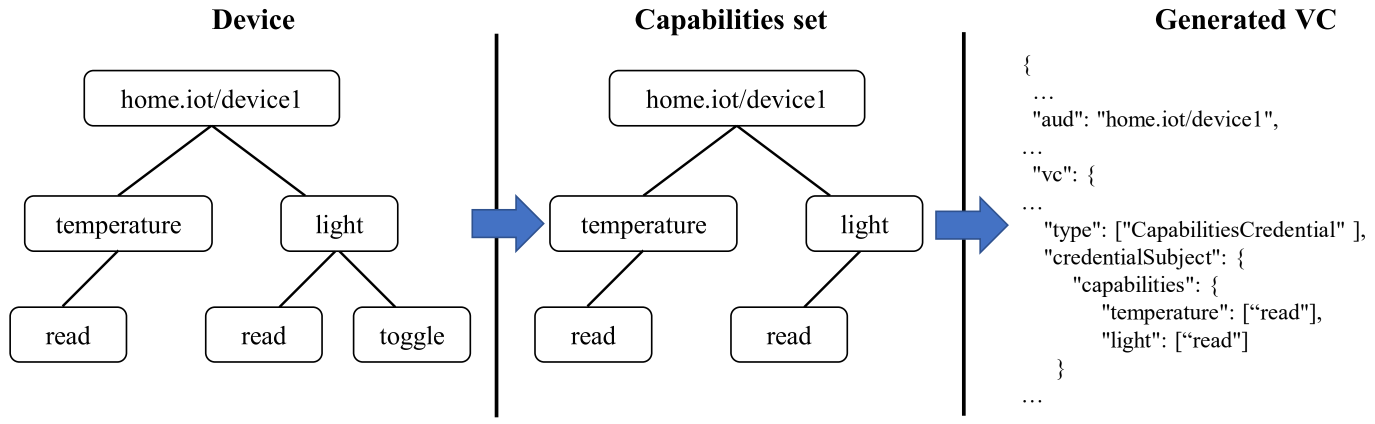

Fig. 2. Mapping access rights to a VC

A. Implementation

Our solution consists of an issuer, a client and a device that interact using HTTPS. Our issuer[4] is a .net core web application that implements OAuth 2.0 specification. *Credential request* is implemented as an OAuth 2.0 authorization flow using the "client credentials" grant (section 4.4 of [28]). Similarly, for the IoT device *access request* we rely on OAuth2.0 and the "Demonstrating of Proof-of-Possession at the Application Layer" (DPoP) draft [29]. Hence, an access request includes the corresponding VC in the *Authorization* header of the generated HTTP request and the appropriate proof in the *dpop* header. Finally, we have implemented a Python 3-based verifier[5] as well as a verifier implemented for IoT devices using Arduino IDE and Arduino Cryptographic Library.[6]

B. Performance evaluation

We have measured the VC issuing processes in a desktop PC equipped with an Intel i5 5540 CPU and 8GB RAM, running Windows 10, and we have measured access request verification (which includes the verification of a VC and a proof) in Raspberry Pi 2 Model B Rev 1.1 with a 900MHz quad-core ARM Cortex-A7 CPU and 1GB RAM, running Raspberry Pi OS, and an Espressif ESP32 WROOM-32 IoT device (240MHz dual-core Xtensa LX7 CPU). We are using EdDSA and ES256 signature algorithms for JWS.

The results in Table I show that issuing a credential takes less than 0.1 ms and the verification performance of the solution is sufficient even on a cheap constrained devices such as ESP32. Despite the fact that the Arduino Cryptography Library is optimized for 8-bit microprocessors (ESP32 is a 32-bit), the verification of both the VC and DPoP proofs, including two signature verifications, still only takes 160ms, allowing real-time usage of the devices in many use cases.

[4]https://github.com/mmlab-aueb/vc-issuer
[5]https://github.com/mmlab-aueb/py-verifier
[6]https://gitlab.com/h2020-iot-ngin/enhancing_iot_cybersecurity_and_data_privacy/privacy-preserving-self-sovereign-identities

TABLE I
EXECUTION TIME OF OUR SYSTEM OPERATIONS MEASURED IN MS.

Operation	**Execution time in ms**
VC issue using EdDSA/ES256 on PC	< 0.1
Request verification using EdDSA on RPi	10.01
Request verification using ES256 on RPi	19.3
Request verification using EdDSA on ESP32	160

Since VCs and proofs are transmitted in HTTP headers they are encoded using base64. The base64 encoding of a VC that includes two resources and two operations (like the VC included in Fig.2) is 656 bytes. Similarly, the base64 encoding of the proof is 440 bytes.

C. Security evaluation

Our system achieves the following security properties:

Distributed access control management. User and access control policy management is implemented independently of the IoT device (and its owner), since granting or revoking an access right does not involve any communication with the device. Also, each issuer is allowed to use its internal policy for deciding who can access which IoT resource and these policies can be kept secret.

Distributed IoT device management. Using our system, it is very simple to add or remove an issuer. New issuers can be added by appending their DID_{issuer} in the corresponding IoT device configuration file; similarly, an issuer can be removed by deleting the corresponding entry. We recognize that this requires remote access to the configuration of a device, which may not be possible/desirable in some use cases: an alternative solution based on chain of trusts is discussed in section VI. Futhermore, since VCs are structured using a type, it is easy for an entity to become an issuer. Additionally, many working groups are working on standardized types of VCs, which makes adding new issuers to a system even easier.

Attack surface reduction. In our solution the amount of verifications an IoT device needs to perform is less compared to a system that relies on Access Control Lists (ACLs), which are also inflexible, do not scale well, and are difficult to use and upgrade [30]. In our system, a device only has to verify the validity and the possession of the VC included in a resource access request. Furthermore, devices are not required to store any additional secret information to implement our protocols, nor do they have to maintain user accounts.

Enhanced privacy. In our system, user privacy is protected against "curious" device owners. Users can rotate the public key of their client application whenever they want (up to single-use identities) and request a new VC; this way they can prevent tracking [19]. Similarly a user is allowed to use as many client applications as they want. Finally, if a user requests access for multiple devices during an *Credential request*, they should receive one VC per device. Although it is possible to create a VC for multiple devices this is not recommended since: (i) such a VC would reveal to a verifier unnecessary information about the VC Subject, and (ii) revoking access to a single IoT device would result in the whole VC being revoked.

Resilience to attacks. Our system is resilient to many types of attacks. Since the VCs are bound to a user's DID, our system is not affected by attackers-in-the-middle that intercept the communication between a client application and a device. These attackers, even if they have access to the encryption key used in the HTTPS session, can neither modify the transmitted VCs without being detected, nor re-use the captured VCs to their own purposes. Similarly, our system utilizes different DIDs for accessing each IoT device, hence, even if the private key that corresponds to a DID is breached, the captured VCs can only be used for that specific purpose. An attacker that has access to the account information that a user uses for authenticating to an authorization server (i.e., S_{user}) can only use them for requesting new VCs and it cannot affect the already issued VCs. Finally, an attacker that has access to the private key of an issuer cannot affect VCs of other issuers, nor can they issue VCs that give access to devices that have not been configured with DID_{issuer}.

Related to this property, our system provides segmentation of secret information. In particular, user account information (S_{user}), which is a very sensitive resource, is used only for requesting VCs. Therefore, we can imagine scenarios where a user requests a VC using their well-protected corporate laptop and using multiple safeguards, and then storing the received VC in their mobile phone: even if the mobile phone is stolen, user account information is not jeopardized. Similarly, a user may use multiple clients and adjust the security properties of a VC accordingly. For example, a user wishing to use a VC from their "travel" laptop during a particular period, may request a VC that includes the corresponding "not used before" and "expires after" fields. Finally, users may rotate the secret information that corresponds to their accounts without affecting the already issued VCs.

VI. Discussion

Our solution defines the authorized issuers for each device by listing them in the device's configuration file. A key limitation of this approach is that any change to list of authorized issuers requires accessing the configuration file. Often, such files cannot be accessed over the network but require physically visiting the device, which can be inconvenient or even impossible at times. A more flexible approach is to only list the device owner's identity in the configuration file and then *delegate* the right to issue credentials to the issuers using suitable VCs. In this case, the client would have to present this issuer-VC along with the above discussed user-VC and proof to form a complete chain of trust from the issuer to the access request.

Also, our implementation uses the did:key method, which simply encodes a public key as a DID. However, if the IoT device can support DID resolution, other, more advanced DID methods can be used, resulting in some security related gains. For example, our Python-based verifier also supports the did:web method. In did:web, the DID is a URL which (when resolved over HTTPS) results in a DID document that includes the public key of the DID owner. The advantage of this method is that DID owners can update their DID documents without having to modify the corresponding DIDs. Therefore, a user can rotate keys without having to receive a new VC.

Finally, it is possible to construct Verifiable Credentials that support *selective disclosure*, i.e. revealing only the selected attributes of a credential. In some use cases this would allow even better privacy as only the required information would have to be revealed to the device.

VII. Conclusions

We presented an access control solution for the IoT, which leverages Verifiable Credentials (VCs). Our system has the advantages of capabilities-based access control systems, adding at the same type support for proof of possession and revocation. Our system is interoperable and can be easily integrated in legacy systems since its core operations are implemented by following existing standards. Moreover, our system supports Decentralized Identifiers, it is lightweight, and it has intriguing security and privacy features.

Future work in this area is focused on two areas. Firstly, our system can be extended to support delegation of VCs among users. Secondly, we investigate alternative signature schemes that will allow selective disclosure of the capabilities included in a VC. This will enhance the privacy of our system and will reduce attack surface by enforcing the principle of least privilege.

Acknowledgments

The work reported in this paper has been funded in part by European Union's Horizon 2020 research and innovation programme through subgrant *Enabling Zero Trust Architectures using Oauth2.0 and Verifiable Credentials (ZeroTrustVC)* of project *eSSIF-Lab*, under grant agreement No 871932, and project *IoT-NGIN* under grant agreement No 957246.

References

[1] S. Dramé-Maigné, M. Laurent, L. Castillo, and H. Ganem, "Centralized, distributed, and everything in between: Reviewing access control solutions for the iot," *ACM Comput. Surv.*, vol. 54, no. 7, sep 2021.

[2] S. Ravidas, A. Lekidis, F. Paci, and N. Zannone, "Access control in internet-of-things: A survey," *Journal of Network and Computer Applications*, vol. 144, pp. 79–101, 2019.

[3] Manu Sporny et al. (2019) Verifiable credentials data model 1.0. [Online]. Available: https://www.w3.org/TR/verifiable-claims-data-model/

[4] D. Bauer, D. M. Blough, and D. Cash, "Minimal information disclosure with efficiently verifiable credentials," in *Proceedings of the 4th ACM Workshop on Digital Identity Management*, ser. DIM '08. New York, NY, USA: ACM, 2008, pp. 15–24.

[5] D. Reed, M. Sporny, M. Sabadello, D. Longley, C. Allen, R. Grant, and M. Jonathan Holt, DO, "Decentralized identifiers (dids) v1.0," W3C, Working Draft, Jul. 2020, https://www.w3.org/TR/2020/WD-did-core-20200723/.

[6] D. Longley, D. Zagidulin, and M. Sporny. (2020) The did:key method. Https://w3c-ccg.github.io/did-method-key/.

[7] M. Lodder and D. Hardman, "Sovrin did method specification," W3C, Editor's Draft, Aug. 2020, https://sovrin-foundation.github.io/sovrin/spec/did-method-spec-template.html.

[8] M. P. et al., "did:web method specification," W3C, Editor's Draft, Dec. 2021, https://w3c-ccg.github.io/did-method-web/.

[9] D. H. et al., "Peer did method specification," DIF, Editor's Draft, Aug. 2020, https://identity.foundation/peer-did-method-spec/index.html.

[10] W. He, M. Golla, R. Padhi, J. Ofek, M. Dürmuth, E. Fernandes, and B. Ur, "Rethinking access control and authentication for the home internet of things (iot)," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, 2018, pp. 255–272.

[11] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: an analysis of IoT devices on home networks," in *28th USENIX Security Symposium USENIX Security 19)*, 2019, pp. 1169–1185.

[12] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the Internet of Things," in *26th USENIX Security Symposium USENIX Security 17)*, 2017, pp. 361–378.

[13] E. Zeng and F. Roesner, "Understanding and improving security and privacy in multi-user smart homes: a design exploration and in-home user study," in *28th USENIX Security Symposium USENIX Security 19)*, 2019, pp. 159–176.

[14] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the internet of things," *Mathematical and Computer Modelling*, vol. 58, no. 5, pp. 1189–1205, 2013.

[15] J. L. Hernández-Ramos, A. J. Jara, L. Marin, and A. F. Skarmeta, "Distributed capability-based access control for the internet of things," *Journal of Internet Services and Information Security (JISIS)*, vol. 3, no. 3/4, pp. 1–16, 2013.

[16] L. af Heurlin, "Authorization certificate based access control in embedded environments," 2015.

[22] C. L. Webber, M. Sporny Eds. (2020) Authorization capabilities for linked data. [Online]. Available: https://w3c-ccg.github.io/zcap-ld/

[17] Q. Zhou, M. Elbadry, F. Ye, and Y. Yang, "Heracles: Scalable, fine-grained access control for internet-of-things in enterprise environments," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 1772–1780.

[18] D. Lagutin, Y. Kortesniemi, N. Fotiou, and V. A. Siris, "Enabling decentralized identifiers and verifiable credentials for constrained IoT devices using OAuth-based delegation," in *Workshop on Decentralized IoT Systems and Security (DISS 2019), in conjunction with the NDSS Symposium 2019*, San Diego, CA, USA, 2019.

[19] Y. Kortesniemi, D. Lagutin, T. Elo, and N. Fotiou, "Improving the privacy of iot with decentralised identifiers (dids)," *Journal Comp. Netw. and Communic.*, vol. 2019, pp. 8 706 760:1–8 706 760:10, 2019. [Online]. Available: https://doi.org/10.1155/2019/8706760

[20] R. Xu, Y. Chen, E. Blasch, and G. Chen, "Blendcac: A blockchain-enabled decentralized capability-based access control for iots," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1027–1034.

[21] A. Birgisson, J. G. Politz, Úlfar Erlingsson, A. Taly, M. Vrable, and M. Lentczner, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," in *Network and Distributed System Security Symposium*, 2014.

[23] M. P. Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H.-S. Kim, D. E. Culler, and R. A. Popa, "WAVE: A Decentralized Authorization Framework with Transitive Delegation," in *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC'19. USA: USENIX Association, 2019, p. 1375–1392.

[24] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF, RFC 7519, 2015.

[25] ——, "JSON Web Signature (JWS)," Internet Requests for Comments, IETF, RFC 7515, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7515

[26] T. Smith, L. Dickinson, and K. Seamons, "Let's revoke: Scalable global certificate revocation," in *Network and Distributed System Security Symposium*, 2020.

[27] W. C. C. Group. (2020) Revocation list 2020. [Online]. Available: https://w3c-ccg.github.io/vc-status-rl-2020/

[28] D. Hardt (ed.), "The OAuth 2.0 authorization framework," IETF, RFC 6749, 2012.

[29] D. Fett et al., "OAuth 2.0 Demonstrating of Proof-of-Possession at the Application Layer (DPoP)," RFC draft, 2020. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-dpop/

[30] A. H. Karp, "Authorization-based access control for the services oriented architecture," in *Fourth International Conference on Creating, Connecting and Collaborating through Computing (C5'06)*, 2006, pp. 160–167.