Self-Sovereign Identities for Smart Devices

Stephan Penner[1], Thomas Wieland[1], Marquart Franz[2]
[1]Hochschule Coburg, Friedrich-Streib-Straße 2, 96450 Coburg, Germany
[2]Siemens AG, Technology, Otto-Hahn-Ring 6, 81739 München, Germany

Abstract: *Current research in identity management is focusing on decentralized trust establishment for distributed identities. One of these decentralized trust models is Self-Sovereign Identities (SSI). With SSI each entity should be able to independently present and manage provable information about itself as well as request and review evidence from other entities. Using a distributed blockchain, information for verifying the authenticity of this evidence can be obtained from any other entity. This concept can be used not only for people, but also for authentication and authorization during the life cycle of devices in the Internet of Things (IoT). This paper presents an SSI-based concept for authentication and authorization of IoT devices among each other, intended to contribute to the change in trust on the internet. The SSI methodology employing a blockchain offers the possibility to establish mutual trust and proof of ownership without relying on any third party. The paper describes the concept, offers a reference implementation, and gives a discussion of the approach.*

1. Introduction

Communication and interaction over digital channels often require that the entities involved are able to authorize themselves mutually before exchanging information or committing transactions. Authenticity and the confirmation of certain abilities become more and more important, for human users as well as for IoT devices.

In the physical world certificates are one example to provide this confirmation for certain information, like an academic degree or the skill to maintain certain machines. In numerous countries, these certificates are also used for the authentication and authorization of an entity, like a driver's license. In this context, trust is established by trusting the author and signatures of the certificates and testing if the certificate is still valid. Then the checking entity can decide to trust this information or not. If the verifier trusts the issuer of the credential, the information stored in the certificate can also be trusted. This model is well known and has been used in various forms for decades in our connected world [1], [2].

In the digital world, a similar model is widely used to establish trust between entities and to check the authenticity of information. For this purpose, a third party is used as an anchor of trust, which stores and manages all relevant information and makes it available to others, for example in the form of digitally signed certificates. This anchor is intended to ensure the integrity and authenticity of this data to establish trust between the entities [2], [3], [4], [5], [6], [7].

What is already economically crucial in e-commerce or financial services systems, is even more important for cyber-physical systems like power plants or trains, where errors could endanger the safety of many people. Especially for the devices in the Internet of Things (IoT) [8] that are becoming more and more popular in these facilities a robust IoT system architecture is required. For this, it is necessary, among other things, that information and attributes of IoT devices can be securely managed, maintained, received and tracked across company boundaries throughout their lifetime [9], [10], [11], [12]. One approach to ensure such a system is to use a life cycle view on IoT devices [3], [4], [5], [11], [12], [13]. During this lifecycle, IoT devices interact with each other. For this interaction a system for authentication and authorization is required. Today most of such systems are using a central trust anchor, holding the data of system participants. Each entity that uses this trust anchor is forced to trust it in order not to leak, misuse or change any sensitive data. Protecting own data from such a misuse is hard because the data is stored in the data processing system with trust anchors [3], [4], [5], and [14].

For solving this trust issue, research and development currently focusses on concepts to eliminate centrally oriented trust anchors for the interaction between entities, including IoT devices. One promising approach utilizes blockchain architectures, leveraging the concept of self-sovereign identities (SSI for short) [3], [9], [15], [16]. With the help of SSI, entities can establish trust between each other on their own using digitally signed certificates, eliminating the need of a central trust anchor. Data can be stored and managed by the entities themselves.

Those previously mentioned certificates could also be used for unequivocal and verifiable proofs of attributes of IoT devices in their life cycle. While stored and managed by the devices, those certificates give a device the possibility to provide evidence for authentication and authorization without involving a third party. Any devices can be programmed in such a way that trust to an entity for particular actions during various scenarios in its lifetime is only given under certain conditions, depending on information in a digital certificate.

In this contribution we present a concept including a reference implementation and evaluation of a system that

can establish an authentication and authorization mechanism for the management and tracking of the life cycle of a cyber-physical system, consisting of several different IoT devices from different manufacturers, using SSI, without the need for a central storage of private data. Other issues of the security of IoT systems, which could theoretically be addressed by SSI, are not discussed here. This paper is structured as follows: First, background is provided about IoT, cyber-physical systems, project context, self-sovereign identities, and the blockchain systems Hyperledger Indy and Aries. Next, the proposed concept will be introduced including a view on related works by other authors. Afterwards an implementation of this concept, using Hyperledger Indy and Hyperledger Aries together with the Python web framework Flask, will be discussed and evaluated. At the end, the main aspects will be summarized, and possible future work will be presented.

2. Background

2.1 Trust Anchors

The Internet of Things, or IoT for short, is a network of devices with embedded microcontrollers collecting information about their physical environment using sensors [8]. Within this network, those devices can communicate with each other or with other systems. A typical property of an IoT device is its limitation of resources like computing capacity, RAM, bandwidth, or available energy.

This work has been conducted in the context of a project focusing on cyber-physical systems in trains [17], [18]. We define a cyber-physical system as an entity consisting of one or several IoT devices as computational units collecting information about its environment and forwarding this information via wired or wireless networks. An example for such a system is a train, consisting of multiple actuators, sensors, and subsystems realized with IoT devices.

Currently a central anchor of trust is often used for authentication and authorization in a cyber-physical system. Humans and devices rely on this anchor to authorize activities during the lifecycle of IoT devices - in manufacturing, in trains as well as in various other scenarios. It is very important in such cyber-physical systems to provide evidence that an information has not been altered.

Such a central instance must be trusted unconditionally. The trust anchor is necessary for the authentication and authorization of all actions in the system holding all necessary information about the connected devices. Technically, authentication is usually realized by asymmetric cryptography, offering public keys in X.509 certificates, signed by a trust authority [19], [13, p. 5]. In many use cases the trust anchor additionally provides signed information for the mutual interaction of the IoT devices, like the affiliation of the owner of a public key. This third party is also necessary for authentication and authorization between the entities. In complex environments like manufacturing sites or trains it is very hard to select the right entity that should hold the role of such a trust anchor. The IoT devices in this system have been manufactured by different vendors. Those are seeking not to reveal anything about their devices. Determining who should provide the trust anchor may easily result in long discussions and strict regulations. Even during operation, no one can be really sure that the trust anchor does not misuse its role.

2.2 Self-Sovereign Identity (SSI)

The goal of SSI is to eliminate the need for a trust anchor and give entities the opportunity to regain control over their data. In addition, entities should be able to decide on their own, if a piece of information should be trusted, whom to show selected information and whom to issue a certificate under certain conditions. In an SSI system the check for authenticity of information provided as a certificate is not carried out by a third party, but by the entities themselves, using information in a distributed, publicly available data storage [3], [5], [20], and [21]. Furthermore, no information associated with the entity should be made publicly available without the permission of the entity in question.

The identification of entities should also take place without a third party. When receiving information as well as when checking and presenting it to other entities, only two parties at most should be involved in these processes [2], [3], [14], and [22]. Attempts to realize SSI systems are using the W3C standards Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) together with the Blockchain technology [2], [3], [14], and [22].

A DID is a pseudonymized and portable representative of an identity, not disclosing any information about the entity behind it. It is associated with a DID document which provides publicly available information. This information can be used to contact the entity behind the DID. Furthermore, it contains public keys owned by that identity for the establishment of a secure communication channel. This DID with the associated DID document can be securely made persistent on a distributed data storage like a blockchain [3], [20], [23], and [24].

Verifiable Credentials represent the digital counterpart to signed certificates from the physical world. A VC can be used to issue information about a device or a person. Using the issuer's signature, each entity can check by whom a Verifiable Credential was issued and whether its contents are authentic. For this purpose, Verifiable Credentials are signed by the issuer and a corresponding entry is written to a distributed data storage like a blockchain. On this storage the entry itself however contains no information about the contents of the VC or the receiver of it [3], [20], [24], and [25]. So there are three roles involved (according to [24]): the issuer issues the credential and hands it to the holder, often on his request. The holder keeps all his credentials in a wallet.

The verifier requests proof from the holder, e.g., about authenticity. The holder presents the proof using its signature which the verifier in turn may check. All these actions are based on DIDs that are stored in a public blockchain or another DID network.

2.3 Hyperledger Indy and Hyperledger Aries

A possible option for such a blockchain is a combination of the Hyperledger Indy and Aries [26], [27]. Both are open-source projects founded by the Linux Foundation to establish SSI systems providing the necessary infrastructure. Hyperledger Indy provides tools and libraries to establish the infrastructure for an SSI system. It provides a blockchain solution to store DIDs and information to verify VCs. It also provides an implementation of the W3C standards DID and VCs. It offers a digital identity wallet called Indy-wallet to store those credentials and DIDs [20], [26].

The provided data storage and infrastructure in Hyperledger Indy is a distributed and public permissioned blockchain. Each entity can read its contents, while writing to it is only allowed for entities with the roles trustee, steward, or endorser. In context of this paper, those roles will be summarized as the role “endorser” or “issuer”. The blockchain provided by Indy is not controlled by one entity but distributed across the so-called validator pool. Each member of this pool holds a copy of the blockchain. To write a new entry into the blockchain one member executes a write request from an endorser and broadcasts its result to the other members. After each member has received a certain number of identical answers, related to a write request, the system finds a consensus and this result will be written to each copy of the blockchain [26], [28], and [29].

Only publicly available information will be stored on a Hyperledger Indy blockchain. The blockchain consists of information like DIDs, DID documents and the used public key as well as algorithms etc. to verify an issued digital credential [26], [28], and [29].

Hyperledger Aries emphasis lies on how entities using a Hyperledger Indy SSI system can interact and communicate with each other in a peer-to-peer and secure way. To achieve this, it provides several protocols like *DID Communication*, *Issue Credential, Connection* and *Basic Message* [29], [27]. Other open-source projects, like Aries Cloud Agent Python (ACA-PY), are establishing some kind of framework, implementing the defined protocols in Hyperledger Aries. In particular, Aries Cloud Agent Python (ACA-PY) is recommended to be used by the Aries group [29].

ACA-PY provides a so-called *agent* as a webservice, offering a REST interface to enable the usage of the Aries protocols. The agent needs a *controller* for managing the agent, which must be implemented individually for each use case. The controller tells the agent what to do, for example, to issue a new VC, establish a new DID communication connection using the connection protocol, or to send messages with the basic message protocol. The agent also informs its controller via webhooks about events like receiving a new VC from an issuer.

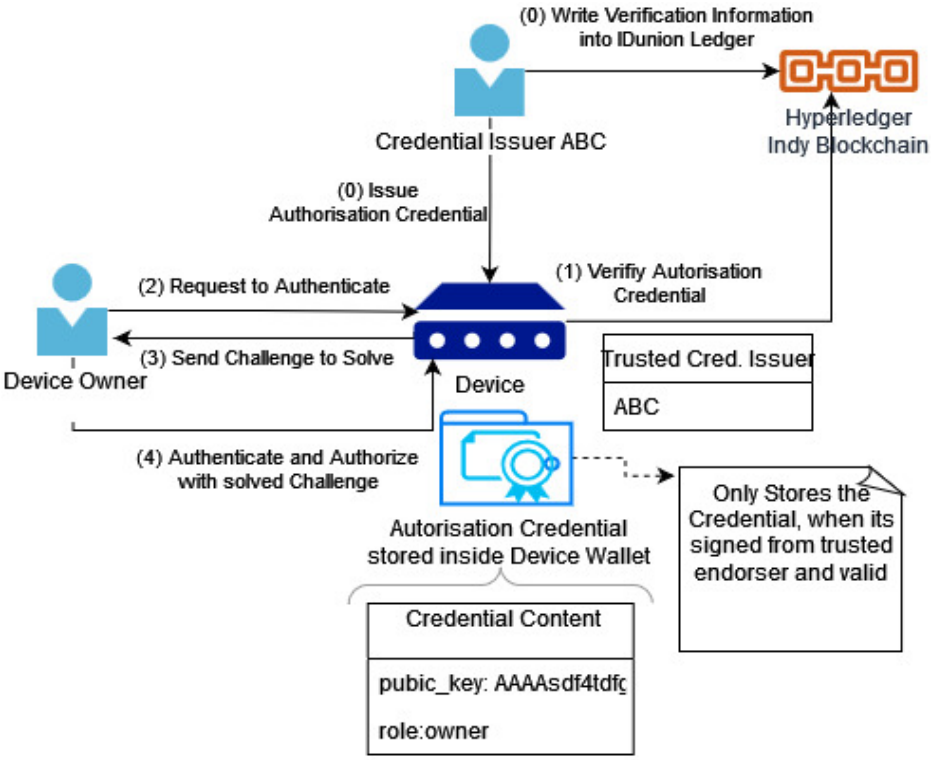

Fig. 1: Overview of the authentication and authorization concept Idea

One major protocol defined by Hyperledger Aries is *DID communication*. With it, two entities can encrypt their messages independent of the used transport protocol like HTTP or MQTT. To ensure this, both entities must exchange their DID and DID document, containing a public key. This can be done by using the connection protocol of Aries. To transmit encrypted messages or verifiable credentials the protocols *basic message* and *issue credential* can be used.

3. Conceptual Approach

We propose to make use of *DID Communication* and basic message to establish an authentication and authorization system using Verifiable Credentials and JSON Web Tokens (JWT). For the communication between entities, we defined message protocol called *Commands*. With this message protocol, entities may transfer tasks or solutions with *basic message* to each other, such as asking for authentication. Before that, a connection between the entities must be established using, for example, the *connection* protocol.

Using *Commands*, entities can transmit asymmetric encrypted requests and answers independent of any message transport protocol like HTTP or MQTT. This is realized by the Aries protocol *DID communication*. In this work, the *Commands* message protocol was used for the message exchange during the authentication and authorization process between two entities. This design was chosen to rely only on the security mechanism provided by *DID Communication*.

To authorize transmitted commands an authentication is needed first. After success, the IoT device can decide whether to execute the command or to decline it. For this, it will check the role of the authenticated entity. Information about which entity has which role is saved by means of a certificate in the IoT device. Therefore, a VC called Authorization Credential, or AuthS-VC for short, was defined. It contains the role and public key of an entity and is stored on an IoT device. Fig. 1 gives an overview of the described concept.

The device will accept this VC if it has originated from a credential issuer it trusts. If the device has an owner configured, it will ask this entity whether to accept or decline the VC instead.

The IoT device has a role-based access system that allows the requested execution of certain commands, like changing the owner, only to entities with certain roles. To authenticate and authorize an entity for such a command, a JWT holding the role of the entity is used combined with a challenge-response mechanism.

To authenticate, the requesting entity presents its public key to the device. The device is then able to check whether the presented key is known or unknown by retrieving VCs. If known, the device can find the corresponding VC with the provided public key, containing the role of the requesting entity. Then, the device uses its own secret key to create a JWT. After the JWT has been created, a challenge is constructed by encrypting the JWT using the public key of the requesting entity found in the VC. If the entity can decrypt this JWT with its secret private key, it has proven its authenticity. By attaching the JWT in future requests, the entity can authorize.

While the public key and role of this entity is saved as a VC, the device can check at any time on its own if this information was somehow altered. Fig. 2 shows the described flow.

Storing information about its owner on a IoT device provides full control over this data by the device itself. Misuse of this information by others is prevented. By holding this information as a Verifiable Credential, the device can even use this information in other domains, because

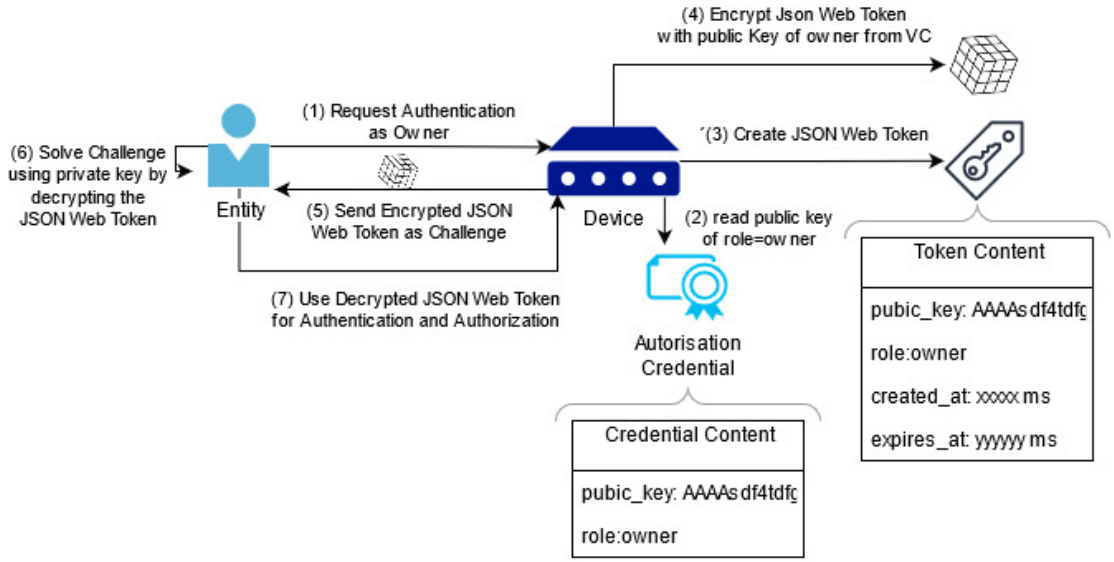

Fig 2: Proposed authentication and authorization system. Messages are delivered using the Aries *basic message* protocol.

there is no dependency to a trust anchor. Only a connection to the Hyperledger Indy blockchain, holding the corresponding verification information to this VC, is needed when the device wants to check its own VCs even in another domain.

New roles are easily defined. An entity with the role *maintainer* for example could be configured by issuing a corresponding VC to the device. By defining a public key to the role, the VC would identify an entity with the role maintainer. Corresponding access rights to certain commands for this role would be defined in the device itself. Because the VC is verifiable, this information can even stay valid if the device changes its domain or is transferred to a new owner. Therefore, the same authentication and authorization system is appliable over the entire lifecycle of an IoT device across company boundaries.

4. Related Work

4.1 RBAC-SC

Cruz et al. propose a distributed role based access control system called RBAC-SC using blockchain technology [16]. The goal is to use a role of an entity, like a student at University X, across company boundaries.

An instance that wants to assign roles to entities defines a smart contract on the Ethereum blockchain. For example, if a university wants to assign its students the role of X student, the smart contract is used and assigns this role to an address on the blockchain. This address is owned by an entity.

Later, this address will be used to check whether the entity has the role of X student. Before that, the smart contract is checked. Next, a challenge is used to evaluate whether the entity has access to the address in question to which the student role of university X has been assigned. To do this, the entity must sign information from the public address block with its private key. The associated public key is also stored on the Ethereum blockchain.

4.2 SSIBAC

The core idea of SSIBAC from Belchior et al. in [15] is to map Verifiable Credentials presented by entities to access rules. To use a resource, an entity first sends a request to the system. The system checks the defined rules for the requested access, in which the necessary attributes or roles are specified. On this basis, the requesting entity is checked by means of a challenge, by using verifiable credentials to generate a verifiable presentation (VP) that proves the required attributes or role of the requesting entity in a VC. If the entity responds with a VP, this is first validated and passed on to an access control engine, if the VP is valid. The engine then uses the information from the VP to calculate whether the access request is to be granted or denied. The authors used Hyperledger Indy for the necessary infrastructure of their system.

5. Reference Implementation

To implement the system proposed above, we used Hyperledger Indy as SSI infrastructure and Hyperledger Aries framework to implement our process. As mentioned before, we defined a message protocol called *Commands* to exchange commands like an authentication and authorization request. Also, we defined a Verifiable Credential called AuthS-VC. Its schema is written to a blockchain based on Hyperledger Indy. To exchange messages with *basic message,* structured using the *Commands* protocol, a controller was implemented with Python web framework Flask [30]. The controller receives notifications as webhooks from the agent. The agent informs the controller, when, for example, a new basic message has been received. The controller also contains an implementation of the *Commands* protocol to send and interpret messages using it. It also possesses the logic to get an AuthS-VC using a public key and to create and send a JWT challenge over the *Commands* protocol. To achieve this, the controller uses the ACA-PY agents REST interface to send basic messages.

The system consists of two major components: a web portal that sends a request with the *Commands* message protocol and a device that receives and processes such a request. To be able to authenticate, the device holds an AuthS-VC about the owner in its wallet. Fig. 3 shows the issued AuthS-VC with role owner in the wallet of the device. The *cred_def_id* field provides the information who has signed this VC and where to look at in the blockchain to verify the authenticity of this VC.


```
"referent": "51e3aad8-e227-4c5e-8040-ba7831226a78",
"attrs": {
  "endpoint": "http://localhost:9803",
  "public_key": "AAAAC3NzaC1lZDI1NTE5AAAAIFE9xj2JxMJDyzk0
  "role": "owner"
},
"schema_id": "PfUQbbAm8mSp2zvGoquabn:2:Autorisation:0.1",
"cred_def_id": "Y5q6MYZrDnZ6Q3BnoYZduk:3:CL:2294:0.157719
```


Fig. 3: Owner stored as VC in the wallet of the device

Using the basic message protocol, the web portal sends a *Commands* formed request to authenticate to the device with its public key. Fig. 4 shows the authentication request sent by the web portal to a device using basic message and *Commands.*

```
BEGIN;
AUTH:public_key-= AAAAC3NzaC1lZDI1NTE5AAAAI
FE9xj2JxMJDyzk0yW9rbMEmWyRvdNos6xjA/Y2XYBqy;
END
```

Fig. 4. Authentication request from the web portal, sent to the device. This request is transmitted using the Aries basic message protocol and formatted using the Commands protocol. This string is the content of the basic message to be sent using ACA-PY.

The device checks if the presented public key is saved as an AuthS-VC. If so, a JWT is created using the information public key and role inside the VC. Subsequently, the JWT

is encrypted using the public key and sent back to the web portal as challenge. This is shown in Fig. 5.

```
BEGIN;
RESP_AUTH:
     challenge-= <string with encrypted JWT>,
     role-= <role of the requester found in the VC>
END
```

Fig. 5: Response from the device, after it receives an AUTH request with a known public key.

With the JWT, the web portal can authenticate itself to the device, in this case as its owner. The JWT is also used as a proof of authorization for certain actions. In case the presented public key is not saved in an AuthS-VC, the device responds with an error message shown in Fig. 6.

```
BEGIN;
ERROR:msg-= Auth Rejected. Public key
AAAAC3NzaC1lZDI1NTE5AAAAIFE9xj2JxMJDyzk0yW9rbMEmWyRvd-
Nos6xjA/Y2XYBqy is unknown!;
END
```

Fig. 6: Error response when the provided public key is unknown.

6. Evaluation

For evaluation, we defined a function called *Change Owner* that changes the owner of a device. If an entity could authenticate as the owner, this function can be executed. To be able to transfer the ownership to a new entity, the device holds two AuthS-VCs. They represent the current owner and the new owner.

During the change owner process, the device will connect to the new owner and test if he is able to authenticate and authorize. After all steps have succeeded, the device deletes the VC corresponding to the current owner, leaving only the new owner's VC in the wallet. After that, the previous owner is not able to gain access anymore. All previous JWTs, which are still valid, become automatically useless, because the device always checks if the public key in the JWT corresponds to a public key stored as a VC.

To execute *Change Owner,* the owner has first to obtain a JWT. This is done by setting up a connection to the device using the connection protocol of Hyperledger Aries. After a connection has been established, a connection ID is returned. This can then be used to send an authentication request as basic message with the *Commands* protocol to the device.

For this, we implemented a function called *Ask for new Access Token* shown in Fig. 7 for the web portal.

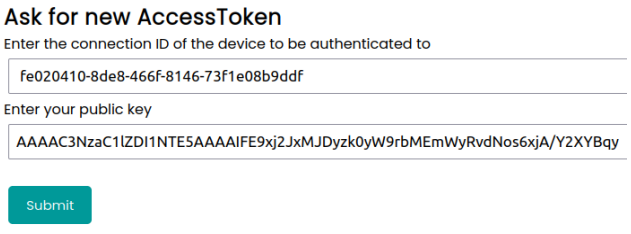

Fig. 7: Ask for new access token function on the web portal

By presenting the connection ID and the own public key, the request to a specific device can be constructed by the application. After clicking the submit button, the software prepares an authentication request to the device. The request received from the device is shown in Fig. 8.

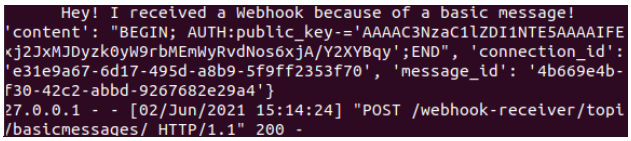

Fig. 2. The device controller receives an Auth request.

After the request, the device checks if the provided public key is known. If the key is stored in an AuthS-VC, it responds with an encrypted JWT. The web portal receives this challenge and tries to solve it by using its own private key. This is shown in Fig. 9.

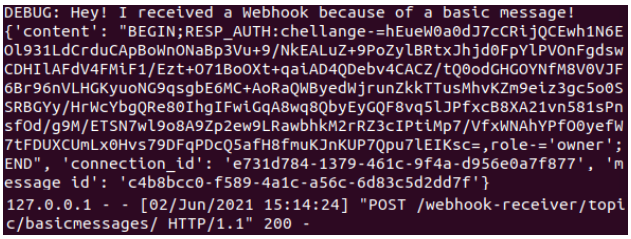

Fig. 9. Console output of the web portal controller, after receiving a challenge as a response to the previous AUTH request.

After the challenge has been mastered, the web portal obtains a new JWT and stores it. The user of the web portal will see this on his screen as shown in Fig. 10.

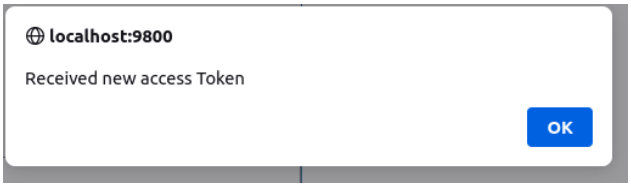

Fig. 10: Response of the web portal after successful authentication

Now, with the newly obtained JWT, the entity can authorize to let the device execute the process of changing its owner. For this, the user calls the "change owner" function and provides all necessary information for this process as shown in Fig. 11.

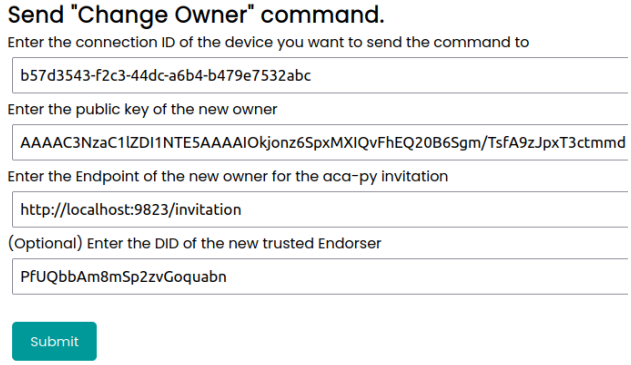

Fig. 11: Change Owner function on the web portal.

The controller of the web portal constructs a *Command* representation of this request by adding the obtained JWT for authentication and authorization.

If the owner could authenticate and authorize correctly, the device responds to the WebPortal with the message that the *Change Owner* process has been completed. After that, the entity sending the *Change Owner* command is not able to receive a new JWT or use its old JWT to authenticate and authorize. Attempting to do this will result in an error message, informing that the provided public key is unknown.

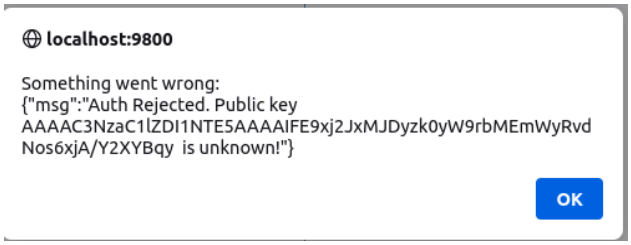

Fig. 12: Response, when old owner tries to authenticate itself to the device after Change Owner completed successfully

7. Conclusion

In this paper a possible solution has been introduced how to design and implement an authentication and authorization system, needed for the tracking and management of the lifecycle of IoT devices, using Verifiable Credentials and DIDs. A concept and an implementation of it could be provided and presented in a demonstration. The proposed system responds with an encrypted JWT as a challenge to an authentication request. The challenge will be constructed by only using public information from an entity stored as a VC on the device itself. The entity requesting an authentication can only solve the challenge when it possesses the right private key to the public key of an entity with a certain role.

Using a message format on top of Hyperledger Aries basic message protocol, the partners can communicate securely using asymmetric encryption for their messages. This is handled via DID communication. The only requirement is the ability to send and receive DID communication based basic messages. This can be achieved by using an ACA-PY agent.

Using Verifiable Credentials to store information, the proposed system becomes easily portable into other environments. In those, the VCs can be verified, when a connection to the corresponding Hyperledger Indy blockchain can be established. Those VCs can be used to configure any entity to a certain role even across company boundaries.

Future work is still necessary to analyze how to check if a trusted endorser or any credential issuer can be trusted to be allowed to issue certain credentials. For example, there should be research about how to verify that entity A is allowed to issue certificates of type X. One solution could be to ask for signed VCs proving this ability. Then of course the question pops up recursively, how to verify that the issuer of those VC is also allowed to do that. There should be research about this topic, concerning how to break this circle.

Acknowledgements

This work was supported by funding of German Federal Ministry for Digital and Transport (BMDV) and Federal Ministry for Economic Affairs and Climate Actions (BMWK) in the projects “RailChain” and “IDunion”.

References

[1] *IDunion - A Public Utility for Verification of Identity Data in Finance*. [Online Video]. Available: https://www.youtube.com/watch?v=CT0MrxRJXno

[2] Linux Foundation, “Introducing the Trust Over IP Foundation.” Accessed: Dec. 07, 2021. [Online]. Available: https://trustoverip.org/wp-content/uploads/2020/05/toip_introduction_050520.pdf

[3] G. Fedrecheski, J. M. Rabaey, L. C. P. Costa, P. C. Calcina Ccori, W. T. Pereira, and M. K. Zuffo, “Self-Sovereign Identity for IoT environments: A Perspective,” in *2020 Global Internet of Things Summit (GIoTS)*, Jun. 2020, pp. 1–6. doi: 10.1109/GIOTS49054.2020.9119664.

[4] B. Singhal, G. Dhameja, and P. S. Panda, *Beginning Blockchain: A Beginner’s guide to building Blockchain solutions*. Springer, 2018.

[5] Q. Stokkink and J. Pouwelse, “Deployment of a Blockchain-Based Self-Sovereign Identity,” *ArXiv180601926 Cs*, Jun. 2018, Accessed: Dec. 07, 2021. [Online]. Available: http://arxiv.org/abs/1806.01926

[6] N. Naik and P. Jenkins, “Self-Sovereign Identity Specifications: Govern Your Identity Through Your Digital Wallet using Blockchain Technology,” in *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, Aug. 2020, pp. 90–95. doi: 10.1109/MobileCloud48802.2020.00021.

[7] A. S. Wazan, R. Laborde, D. W. Chadwick, F. Barrere, and A. Benzekri, “How Can I Trust an X.509 Certificate? An Analysis of the Existing Trust Approaches,” in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, Dubai, Nov. 2016, pp. 531–534. doi: 10.1109/LCN.2016.85.

[8] F. Wortmann and K. Flüchter, “Internet of Things: Technology and Value Added,” *Bus. Inf. Syst. Eng.*, vol. 57, no. 3, pp. 221–224, Jun. 2015, doi: 10.1007/s12599-015-0383-3.

[9] S. Dramé-Maigné, M. Laurent, L. Castillo, and H. Ganem, “Augmented Chain of Ownership: Configuring IoT Devices with the Help of the Blockchain,” Singapore, Aug. 2018, vol. Part I, pp. 53–68. doi: 10.1007/978-3-030-01701-9_4.

[10] M. A. Khan and K. Salah, “IoT security: Review, blockchain solutions, and open challenges,” *Future Gener. Comput. Syst.*, vol. 82, pp. 395–411, May 2018, doi: 10.1016/j.future.2017.11.022.

[11] L. F. Rahman, T. Ozcelebi, and J. Lukkien, “Understanding IoT Systems: A Life Cycle Approach,” *Procedia Comput. Sci.*, vol. 130, pp. 1057–1062, 2018, doi: 10.1016/j.procs.2018.04.148.

[12] N. Yousefnezhad, A. Malhi, and K. Främling, “Security in product lifecycle of IoT devices: A survey,” *J. Netw. Comput. Appl.*, vol. 171, p. 102779, Dec. 2020, doi: 10.1016/j.jnca.2020.102779.

[13] S. Fatima, S. Ahmad, and S. Siddiqui, “X. 509 and PGP Public Key Infrastructure methods: A critical review,” *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. Vol. 15, no. 5, pp. 55–59, 2015.

[14] G. Goodell and T. Aste, “A Decentralised Digital Identity Architecture,” *Front. Blockchain*, vol. 2, p. 17, Nov. 2019, doi: 10.3389/fbloc.2019.00017.

[15] R. Belchior, B. Putz, G. Pernul, M. Correia, A. Vasconcelos, and S. Guerreiro, “SSIBAC: Self-Sovereign Identity Based Access Control,” in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Dec. 2020, pp. 1935–1943. doi: 10.1109/TrustCom50675.2020.00264.

[16] J. P. Cruz, Y. Kaji, and N. Yanai, “RBAC-SC: Role-Based Access Control Using Smart Contract,” *IEEE Access*, vol. 6, pp. 12240–12251, 2018, doi: 10.1109/ACCESS.2018.2812844.

[17] J. Lee, B. Bagheri, and H.-A. Kao, “A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems,” *SME Manuf. Lett.*, vol. 3, Dec. 2014, doi: 10.1016/j.mfglet.2014.12.001.

[18] L. Monostori *et al.*, “Cyber-physical systems in manufacturing,” *CIRP Ann.*, vol. 65, no. 2, pp. 621–641, 2016, doi: 10.1016/j.cirp.2016.06.005.

[19] ITU-T Recommendation, “X.509 Information Technology - Open Systems Interconnection - The Directory: Authentication Framework.” ITU, Jun. 1997. [Online]. Available: https://www.itu.int/rec/T-REC-X.509

[20] P. C. Bartolomeu, E. Vieira, S. M. Hosseini, and J. Ferreira, “Self-Sovereign Identity: Use-cases, Technologies, and Challenges for Industrial IoT,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, pp. 1173–1180. doi: 10.1109/ETFA.2019.8869262.

[21] Y. Liu, Q. Lu, H.-Y. Paik, X. Xu, S. Chen, and L. Zhu, “Design Pattern as a Service for Blockchain-Based Self-Sovereign Identity,” *IEEE Softw.*, vol. 37, no. 5, pp. 30–36, Sep. 2020, doi: 10.1109/MS.2020.2992783.

[22] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, “In Search of Self-Sovereign Identity Leveraging Blockchain Technology,” *IEEE Access*, vol. 7, pp. 103059–103079, 2019, doi: 10.1109/ACCESS.2019.2931173.

[23] Sporny, Manu, Longley, Dave, Sabadello, Markus, Reed, Drummond, Steele, Orie, and Allen, Christopher, “Decentralized Identifiers (DIDs) v1.0 - Core architecture, data model, and representations,” W3C Proposed Recommendation, Aug. 2021. [Online]. Available: https://www.w3.org/TR/did-core/

[24] A. Preukschat and D. Reed, Self-sovereign identity: decentralized digital identity and verifiable credentials. Shelter Island: Manning, 2021.

[25] M. Sporny, D. Longley, and D. Chadwick, “Verifiable Credentials Data Model v1.1 - Expressing verifiable information on the Web,” Nov. 2021. [Online]. Available: https://www.w3.org/TR/vc-data-model/

[26] Hyperledger Revision, “Hyperledger Indy SDK.” [Online]. Available: https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/

[27] “Hyperledger Aries.” [Online]. Available: https://github.com/hyperledger/aries

[28] “Hyperledger Indy Node.” [Online]. Available: https://github.com/hyperledger/indy-node

[29] “Hyperledger Aries Cloudagent Python.” [Online]. Available: https://github.com/hyperledger/aries-cloudagent-pythonrabak

[30] “Flask documentation.” [Online]. Available: https://flask.palletsprojects.com/en/2.0.x/