Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT

Oscar Novo

***Abstract*—The Internet of Things (IoT) is stepping out of its infancy into full maturity and establishing itself as a part of the future Internet. One of the technical challenges of having billions of devices deployed worldwide is the ability to manage them. Although access management technologies exist in IoT, they are based on centralized models which introduce a new variety of technical limitations to manage them globally. In this paper, we propose a new architecture for arbitrating roles and permissions in IoT. The new architecture is a fully distributed access control system for IoT based on blockchain technology. The architecture is backed by a proof of concept implementation and evaluated in realistic IoT scenarios. The results show that the blockchain technology could be used as access management technology in specific scalable IoT scenarios.**

***Index Terms*—Access control, blockchain, Internet of Things (IoT), smart contracts.**

I. Introduction

With a predicted 18 billion devices by 2022 [1], Internet of Things (IoT) has become a technology with large influence across many vertical markets. It is foreseen that many IoT services will provide global reach across millions of simple and sometimes tiny devices. Besides that, the constrained capabilities of many IoT devices, as well as the current access control systems based on centralized and hierarchical structures, create new challenges in the IoT domain.

Centralized access control systems—otherwise known as the client/server paradigm—were designed to meet the needs of traditional human–machine oriented Internet scenarios where devices are within the same trust domain, which usually requires centralized access management. However, some IoT scenarios are much more dynamic than the traditional scenarios in which IoT devices may be mobile [2] and belong to various management communities during their lifetime. On the other hand, IoT devices can be managed by several managers at the same time. Moreover, many IoT devices and constrained managers will be too limited [3] in terms of CPU, memory, and battery resources to be able to operate properly using the current systems. Henceforth, new ways of approaching the problem are needed.

In this paper, we present a new architecture for managing IoT devices. The architecture provides a decentralized access control system connected to geographically distributed sensor networks. The solution is based on blockchain technology whereas the access control policies are enforced by it. By adopting blockchain, this solution eliminates centralized access management. On the contrary, a single centralized access control server might become a bottleneck when access control queries and updates are frequent.

In contrast to other centralized system proposals [4], our approach brings the following advantages to access control in IoT.

1) *Mobility:* The architecture can be used in isolated administrative systems or domains. Thus, every administrative domain has its own freedom to manage the IoT devices while the access control policies are still enforced by the rules in the blockchain.
2) *Accessibility:* In some IoT systems, the constrained managers may use sleeping patterns that make it infeasible to constantly access them directly. This solution makes the access control rules available at any time. In addition, failures in some administrative servers do not ruin access to the information; all access control information is distributed.
3) *Concurrency:* A constrained device can have multiple managers at the same time, and all of them can access or modify the access control policies concurrently.
4) *Lightweight*: The IoT devices do not need any modification to adopt our solution. Besides, the communication between the managers and IoT devices happens through the blockchain network enabling cross platform communication.
5) *Scalability:* A constrained manager can still handle multiple IoT devices using our solution due to the fact that the IoT devices do not access the access control information directly from the managers. Furthermore, our solution supports numerous IoT devices connected through different constrained networks to a single blockchain.
6) *Transparency:* The system hides the location of the IoT devices and how a resource is accessed.

In particular, this paper contributes to the design of a new decentralized access control architecture for IoT using blockchain technology. Our approach differs from other solutions in the way that it applies a specific design to avoid integrating blockchain technology into IoT devices. This increases the usability of our solution in a vast number of IoT scenarios with limited capabilities. As opposed to other solutions, the design operates in a single smart contract,

Manuscript received November 7, 2017; revised January 8, 2018 and February 9, 2018; accepted March 1, 2018. Date of publication March 5, 2018; date of current version April 10, 2018.

The author is with NomadicLab, Ericsson Research, 02420 Helsinki, Finland (e-mail: oscar.novo_at_ericsson.com).

Digital Object Identifier 10.1109/JIOT.2018.2812239

2327-4662 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

simplifying the whole process in the blockchain network and reducing the communication overhead between the nodes. Additionally, the access control information is provided to the IoT devices in real time. In summary, the adoption of blockchain technology in our approach has been specifically designed to better handle scalability and to achieve better results than current solutions in lightweight IoT scenarios.

The remainder of this paper is structured as follows. Sections II and III describe the architecture and the implementation, respectively. Section IV describes the set-up used to evaluate our architecture and presents the results of the measurements. Section V analyzes the security of our system. Section VI presents the related access control technologies in IoT, and Section VII concludes this paper.

A. Blockchain

Bitcoin's public ledger—the blockchain—was first introduced in 2009 by Nakamoto [5]. Bitcoin was the first widely used implementation of peer-to-peer trustless electronic cash. Thenceforth, many other forms of electronic cash (call cryptocurrencies) have been created using similar structures. At the same time, different applications using blockchain have been developed over the years to implement other scenarios beyond cryptocurrencies. New concepts, such as smart contracts and smart properties, have entered the scene. Smart contracts [6] are computer protocols that facilitate, verify, or enforce the negotiation or performance of a contract. They provide the ability to directly track and execute complex agreements between parties without human interaction. On the other hand, smart properties are agreements whose ownership is controlled via the blockchain, using contracts.

The potential uses of blockchain technology go beyond Bitcoin. Blockchain technology has the following properties.

1) *Decentralized Control:* A decentralized scheme in which no central authority dictates the rules.
2) *Data Transparency and Auditability:* A full copy of every transaction ever executed in the system is stored in the blockchain and is public to all the peers.
3) *Distribute Information:* Every network node keeps a copy of the blockchain to avoid having a centralized authority privately keep all that information.
4) *Decentralized Consensus:* The transactions are validated by all the nodes of a network instead of a central entity. This breaks with the paradigm of centralized consensus.
5) *Secure:* The blockchain is tamper-proof and cannot be manipulated by malicious actors.

Those are few of the major strengths of blockchain technology. The secure, decentralized, and autonomous capabilities of the blockchain make it an ideal component to become a fundamental element of IoT solutions.

B. Blockchain Technology

The blockchain [5] is a distributed database that does not need a central authority and eliminates the need for third party verification. A blockchain contains a set of blocks, and every block contains a hash of the previous block, creating a chain of blocks from the *genesis block* to the current block. A *genesis block* is the first block in a blockchain. The genesis block is almost always hardcoded into the software. It is a special case in that it does not reference a previous block. For any block on the blockchain, there is only one path to the *genesis block*. Coming from the *genesis block*, however, there can be *forks*. *Forks* are generated when two blocks are created just a few seconds apart. When that happens, the latest block in the longest valid chain is always chosen. The longest valid chain is calculated based on the combined difficulty of that chain, not the number of blocks. The blocks in shorter chains are considered invalid chains and are often called *orphan* blocks.

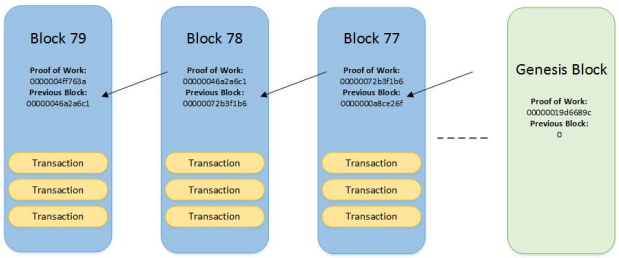


Fig. 1. Blockchain structure.

Blocks have a set of transactions. A transaction is a transfer of values between different entities that are broadcast to the network and collected into the blocks. All transactions are visible in the blockchain. The transactions are mined into a block by the so called *pool miners* or *solo miners*. The *pool miners* technique is a mining approach where multiple devices called *miners* contribute to the generation of a block. *Pool miners* or *solo miners* are entities that add transaction records into the blockchain. That process is called *mining*. Mining is intentionally designed to be resource-intensive and difficult.

Individual blocks must contain a proof of work (PoW) [7] to be considered valid in the blockchain. The PoW is verified by other *miners* each time they receive a block. The primary purpose of mining is to allow the nodes in a system to reach a secure, tamper-resistant consensus. Mining is also the mechanism used to introduce new cryptocurrency (e.g., Bitcoins) into the system. The miners are paid a transaction fee as well as a determined amount of newly created coins when they validate a block. This method serves the purpose of disseminating new coins in a decentralized manner as well as providing security to the system. The system automatically adapts to the total mining power of the network keeping it constant to a specific amount of time (e.g., 10 min in Bitcoin). The difficulty target of the PoW is also adjusted after every certain amount of blocks (e.g., 2016 blocks in Bitcoin) based on the network performance. A transaction takes time to reach all the nodes in the network, and the delay ensures that all the transactions are verified by all the nodes in the network, to prevent the so called *double spending* problem. *Double spending* is the result of using some cryptocurrency more than once at the same time.

Consensus is a fundamental problem in distributed systems that requires two or more agents to mutually agree on a given value needed for computational purposes. Some of these agents may be unreliable, and therefore the consensus process needs to be reliant. Blockchains can use various consensus

algorithms. Some of them include PoW, proof of stake (PoS), proof of storage [8], proof of burn, or proof of capacity [9] among others.

The PoW of every block guarantees a specific level of difficulty to generate a new block, and the decentralized consensus enforces the validity of every block in the blockchain. If there is *consensus* to accept a new block, the new block will be added into the blockchain and all the *miners* will have to start mining using that block as a reference. Each block is computationally impractical to modify once it has been added into the blockchain because the whole blockchain would also have to be regenerated.

PoS [10] is a proposed alternative to PoW. PoS build on the notion that only those holding assets in the system may participate in the consensus process growing the blockchain. While the PoW method forces miners to repeatedly run expensive hashing algorithms to validate transactions, PoS asks users to prove ownership of a certain amount of currency (i.e., their stake in the currency).

C. Blockchain Implementations

Blockchain technology can be used in a myriad of ways rather than just as a digital currency system, e.g., using blockchains as the underlying technology to build software. This section describes what we believe are some of the popular blockchain systems and their salient features. The following systems mainly focus on building software applications on top of blockchain technology.

1) Bitcoin: Bitcoin [5] was the first blockchain to be conceptualized and implemented, and it is a cryptocurrency that serves as a digital financial asset. Bitcoin uses public key cryptography, peer-to-peer networking, and PoW to make transactions and verify them. The Bitcoin system is programmed so that a new block is created once every 10 min. If a fork is not a part of the *longest computationally* chain, it becomes a stale block.

It is worthy to note that there are no balances in Bitcoin, or rather there are unspent transaction outputs (UTXO) in the blockchain. Whenever some bitcoins are received, they are recorded as UTXO. Thus, sending someone a bitcoin actually means creating a UTXO corresponding to the receiver's address. A transaction output typically consists of two fields, namely the amount and a locking script. The locking script sets out conditions that need to be fulfilled in order to spend the UTXO. A *satoshi* is the smallest denomination of the amount that can be sent.

2) Ethereum: Ethereum [10] was designed in 2013 by a Bitcoin developer V. Buterin, who wanted to build a platform to facilitate the development of decentralized applications on top of the blockchain. Ethereum has its own cryptocurrency called *ether* and an internal currency to pay for computations and transaction fees called *gas*. The decentralized applications can be programmed with a built-in Turing complete language called *Solidity*.[1] A Turing complete language refers to a programming language that can solve any computational problem if enough time and space are provided.

Ethereum uses PoW as its consensus mechanism, but it is soon switching to PoS. The basic build of the PoW algorithms that Ethereum currently uses is a memory-hard hashing algorithm called Dagger-Hashimoto. The block creation time is significantly lower than in many other systems and amounts to approximately 12 s. Since a lower block creation time leads to a higher rate of stale blocks, the system uses GHOST protocol [11] to consider the *heaviest computational* chain as the main blockchain. The heaviest chain in this case includes the stale blocks as well.

3) Rootstock: Rootstock,[2] a new open source platform, is very similar to Ethereum in terms of creating smart contracts on a Turing complete smart platform, except that it utilizes the Bitcoin ecosystem to do so. The advantages to this platform are that it exists as a Bitcoin *sidechain* and is backward compatible with the Ethereum virtual machine. This means that all Ethereum contracts can easily run on Rootstock. Their biggest advantage, however, is the fact that they can be merged-mined with Bitcoin, thereby making Rootstock as secure. A *sidechain* is a separate blockchain whose assets can be transferred to and from the main blockchain, i.e., the Bitcoin blockchain in this context.

4) Hyperledger: Hyperledger[3] is a project hosted by the Linux Foundation as a cross industry collaborative project. The system was designed with the enterprise architecture in mind with customizable networking rules that help different consensus protocols operate. It borrows the UTXO and script-based logic from Bitcoins, as described in Section I-C1, and uses practical Byzantine fault tolerant (PBFT) [12] consensus protocol instead of the PoW algorithm. PBFT is known to process thousands of requests per second with a latency increase of less than a millisecond.

II. Overview of the Decentralized Access Control System in IoT

The architecture proposed in this paper describes a new decentralized access management system where access control information is stored and distributed using blockchain technology.

All the entities will be part of blockchain technology except for IoT devices and *management hub* nodes. Nodes in a blockchain network must include a copy of the blockchain. The blockchain can be considerably large in size and will keep increasing over time. The majority of IoT devices will not be able to store blockchain information due to their constrained nature. Consequently, our architecture does not include IoT devices in the blockchain and, alternatively, defines a new node called *management hub* that requests access control information from the blockchain on behalf of the IoT devices.

In addition to that, the solution involves a single smart contract that defines all the operations allowed in the access control system. That contract is unique and cannot be deleted from the system. Entities called *managers* interact with the

[1] [Online]. Available: https://solidity.readthedocs.io/

[2] [Online]. Available: http://www.rsk.co

[3] [Online]. Available: https://www.hyperledger.org

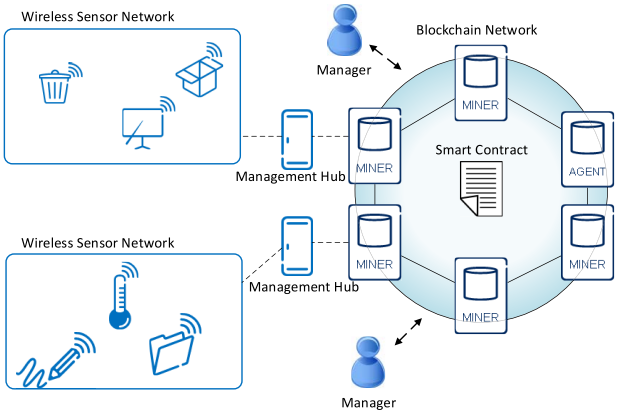


Fig. 2. Decentralized access control system.

smart contract in order to define the access control policy of the system.

Section II-A explains the different components of the architecture in more detail. Section II-B defines the interfaces. Section II-C describes the interactions of those components while Section II-D defines limitations and possible solutions of our architecture.

A. System Architecture

The architecture of our system is depicted in Fig. 2. The architecture can be divided into six different components.

1) Wireless sensor networks.
2) Managers.
3) Agent node.
4) Smart contract.
5) Blockchain network.
6) Management hubs.

1) Wireless Sensor Networks: A wireless sensor network is a communication network that allows constrained connectivity in applications with limited power and light requirements. Further, the IoT devices belonging to the wireless sensor network are limited in their computational power, memory, and/or energy availability.

IoT devices do not belong to the blockchain network. Consequently, one of the requirements of our architecture is that all the devices will have to be uniquely identified globally in the blockchain network. Public key generators can provide a feasible solution for the problem producing acceptably large and unique random numbers. Typically, using the existent IoT cryptographic technologies would automatically create a public key for every device. Hence, enforcing encryption connections will ensure unique identifiers. In fact, current IoT communication protocols such as CoAP [13] already support secure channels through DTLS [14].

2) Managers: A *manager* is an entity responsible for managing the access control permissions of a set of IoT devices. Normally, *managers* are considered lightweight nodes in our system. Lightweight nodes do not store the blockchain information or verify the blockchain's transactions as the *miner* nodes do. As a result, constrained devices can also become *managers* in our system without representing an impediment to their hardware limitations. In addition, *managers* using our approach do not need to be constantly connected to the blockchain network, which helps to decrease the usage of their hardware resources.

Any entity can be registered as a *manager*. However, the devices registered as IoT *devices* have to register under a manager's control. That is done to avoid managers from registering to devices under their control without the permission of the devices. In addition, all registered IoT devices in the system have to belong to at least one registered manager. Otherwise, nobody would be able to manage that device. A registered IoT device can belong to multiple managers at the same time.

After registration of the IoT device under the *manager*'s control, the *managers* can define specific access control permissions for them.

3) Agent Node: The *agent node* is a specific blockchain node in our architecture responsible to deploy the only smart contract in our system. The *agent node* is the owner of the smart contract during the lifetime of the access control system. Once the smart contract is accepted into the blockchain network, the *agent node* receives an address that identifies the smart contract inside the blockchain network. In order to interact with the smart contract, all the nodes in the blockchain network need to know that smart contract's address.

4) Smart Contract: The access management system described in this paper is governed by the operations defined in a single smart contract. This smart contract is unique and cannot be deleted from the system. Hence, all the operations allowed in the access management system are defined in the smart contract and are triggered by blockchain transactions. Once an operation is triggered through a transaction, the miners will keep the information of the transaction globally accessible. The smart contract and its operations are also globally accessible.

In addition to that, it has to be taken into consideration that *managers* are the only entities with the ability to interact with the smart contract in order to define new policies in the system.

5) Blockchain Network: The blockchain network in our architecture is a private blockchain for the sake of simplicity. We chose a private blockchain since all the elements of the prototype are more dimensioned, providing us more reliable results when evaluating the system. However, in a real scenario, a public blockchain should be used to facilitate the adoption of the solution.

Private blockchains are those that can be read by anyone but only written by private nodes. The *miners* in the network help keep the network secure and stable by approving transactions and keeping copies of the blockchain. Nodes can use the blockchain interface to store and globally access the access control policy of specific devices. The information is fully decentralized and tamper-proof.

6) Management Hubs: As mentioned before, IoT devices do not belong to the blockchain network. The majority of IoT devices are very constrained in terms of CPU, memory, and battery. Those limitations restrict IoT devices to be part of the blockchain network. Being part of the blockchain network

implies keeping a copy of the blockchain locally and a track of the network transactions. Even though there are lightweight solutions that do not keep the entire blockchain information locally and rely on other nodes [15], all those lighter solutions are still too heavy for the majority of IoT devices. In consequence, we opted to use a node called *management hub*. The *management hub* is an interface that translates the information encoded in CoAP messages by the IoT devices into JSON-RPC messages understandable by the blockchain nodes. The *management hub* is connected directly with a blockchain node, for instance, a *miner*. Multiple sensor networks can be connected to a *management hub* node and multiple *management hub* nodes can be connected to the same blockchain node. IoT devices will only be able to request access information from the blockchain using the *management hub*.

Management hub nodes cannot be constrained devices. Such devices need high performance characteristics to be able to serve as many simultaneous requests as possible from the IoT devices.

In the simplest case where authentication is not needed, any IoT device will be able to connect to any *management hub* directly and access the blockchain network. However, in many situations an access control is needed. In such a case, the IoT devices will only be able to connect to some specific management hubs. After an IoT device is added into the system, the *manager* node of that device will have to inform the specific *management hub* node about the credentials of that device, as well as, inform the device about the location of the *management hub* node.

B. System Interfaces

This section gives a more detailed explanation of the operations defined in the single smart contract of the system and the interface used by the *management hub* nodes to query the policy from the blockchain.

1) Smart Contract: Given that I is the set of the public keys $I(m)$ of each *manager* m, G is the set of the public keys $G(s)$ of each IoT *device* s and P is the set of *policies* P, where $p^{s\to s'_r}$ refers to the permissions that the IoT device with public key s has over the resource r of the IoT device with public key s'

$$I = \{I(m_1), I(m_2), \ldots, I(m_n)\}$$
$$G = \{G(s_1), G(s_2), \ldots, G(s_n)\}$$
$$P = \left\{p_1^{s\to s'_r}, p_2^{s\to s'_r}, \ldots, p_n^{s\to s'_r}\right\}.$$

The information below describes the operations defined in our smart contract.

1) *RegisterManager*(I_m); $I' \leftarrow I \bigcup I_m$ for any device m.
2) *RegisterDevice*(I_m, G_s); $G' \leftarrow G \bigcup G_s$ iff device m is the manager of device s.
3) *AddManagertoDevice*(I_m, G_s); $G' \leftarrow G \bigcup G_s$ iff device m is the manager of device s.
4) *RemoveManagerfromDevice*(I_m, G_s); $G' \leftarrow G \bigcap G_s$ iff device m is the manager of device s.
5) *AddAccessControl*(I_m, G'_s, G_s, r, p); $P' \leftarrow P \bigcup p^{s\to s'_r}$ iff device m is the manager of device s'.
6) *DeregisterManager*(I_m); $I' \leftarrow I \bigcap I_m$.
7) *DeregisterDevice*(G_s); $G' \leftarrow G \bigcap G_s$.
8) *RevokePermission*$(p^{s\to s'_r})$; $P' \leftarrow P \bigcap p^{s\to s'_r}$.
9) *QueryManager*(I_m) returns the tuple (m, S), where $S = \{s \mid s\exists G : m \text{ manages } s\}$, *if* $I_m \in I$.
10) *QueryPermission*$(G_{s'})$ returns the set $\{q|q\exists Gs \wedge p^{s\to s'_q} \in P\}$.

As the information above suggests, the *managers* and IoT *device* entities will be identified in the system by their public keys. Furthermore, the resources will be identified by their names; the permissions that allow or prevent others from viewing, modifying or executing them can be defined through the *AddAccessControl* operation. A resource in our context is any item that can be obtained from an IoT device.

2) Management Hub: A *management hub* can request information of any IoT device freely and obtain the result almost instantly from the blockchain node. On the other hand, the query operation is not stored in the blockchain since the *management hub* does not use a transaction to fetch the information from the blockchain. The information is fetched directly from the blockchain store in the blockchain node. Querying information from the blockchain does not incur any fee or delay.

Given that S is the set of IoT devices and $s, u \in S$, R the set of resources and $e_s \in R$, and P the set of permission types where $\{r, w, x\} \in P$ and r, w and x specify the read, write, and execute access, respectively, a blockchain node will query the blockchain with the information provided by the *management hub* as follows:

$$\text{Allow}(u, e_s, x) = \begin{cases} 1, & \text{if } x(e_s) \in u \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

C. System Interactions

This section explains the different interactions between the different components of our architecture. As shown in Fig. 3, the interactions can be divided into four different phases.

1) Setting up the management blockchain network.
2) Registering the managers and IoT devices into the system.
3) Defining the policy for those aforementioned components.
4) Discovering the policy.

It is worth mentioning that there are two more types of interactions which are not defined in Fig. 3 due to its similarity to the previous phases but are explained below. Those interactions are the modification of the access control policies in the system after registration and the modification of a device's manager after registration.

1) Network Set-Up: During this phase, the access management system is created in the blockchain network. Upon the creation of the blockchain network, the *agent node* deploys the smart contract into the blockchain network. This single smart contract defines all the operations of the access control management system. Once the smart contract is accepted into the blockchain network, the *agent node* receives the address of the smart contract. The address is used to identify the smart contract in the access management system and other components of the blockchain network need the smart contract's

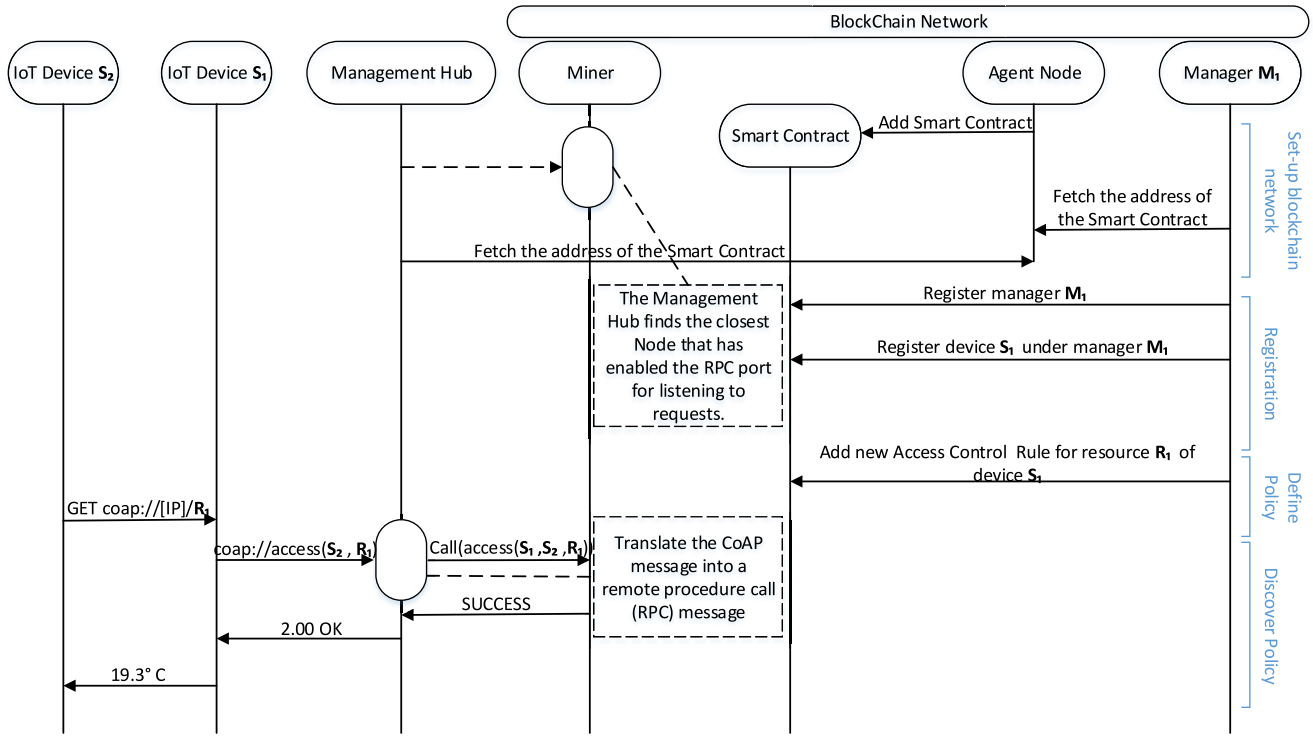


Fig. 3. Network set-up, registration, definition, and discovery of the policy.

address to interact with it. For instance, all the *managers* in the system will interact with this single smart contract to register as managers or modify the control access rules of the IoT devices.

Fig. 3 shows how a *manager* and a *management hub* discover the address, querying the *agent* node. Typically, that would be one possibility to obtain the information. However, in our implementation, that information is hard-coded into those components for simplicity.

The *management hub* will connect with the nearest available node in the blockchain network, a *miner* node in Fig. 3. That *miner* hosts a personal copy of the blockchain. In addition, it enables the RPC port for listening for requests and lets *management hubs* connect to it. *Management hubs* also have to have a way to find the available nodes next to them. That information could be obtained from a centralized system in Internet, but in our particular implementation, is set manually in every *management hub*.

2) Registration: Any blockchain node in the access management system can be registered as a *manager*. In order for a blockchain node to register itself as a *manager*, it needs to know the address of the smart contract. Once it obtains that information, it can register itself sending a transaction to the function *RegisterManager* defined in the smart contract. Thereafter, the *manager* will receive the address of its registration once the transaction is successfully accepted into the blockchain. That address will identify the *manager* in the access management system.

Manager nodes can also register IoT devices under a manager's control. There is no limitation of the number of managers an IoT device can have. Thus, an IoT device can have several managers at any time. As in the previous case, the *manager* will receive an address of the registered device that will be used to identify the device into the access management system. IoT devices should be able to verify the registration under a manager before the operation is accepted in the blockchain. Otherwise, any *manager* could register any device under its control. For the sake of simplicity, our implementation eludes that verification. That assumption makes our system substantially insecure in case of a malicious *manager*, but our goal was to prove the feasibility of the architecture rather than the security.

3) Management Modification: As explained before, every IoT device has to belong to at least one *manager*. In addition, our system supports a multiple number of managers controlling the same device. There are multiple ways in our system to transfer the management control from one manager to another or to add or delete several managers from the system.

In our prototype, we choose one of the simplest options in which every *manager* node in the system can remove itself as a manager of the devices it controls. On the contrary, *managers* cannot delete other managers from the system. The system will always let a *manager* to remove itself from an IoT device as long as the IoT device is under the control of at least another *manager* node. Otherwise, the smart contract will not allow the operation and will be canceled.

On the other hand, only the *manager* nodes that control an IoT device can register other *managers* under the control of that device.

One of the advantages of our solution is that transferring the management control of an IoT device is a simple process due to the fact that all the operations in the system are defined

and enforced using a single smart contract and *managers* do not need to interact with each other. The *managers* only need to know the device's address and the blockchain address of the smart contract to be able to modify the management relationships.

4) Policy Definition: Managers can define access control rules for the resources of their IoT devices. The permissions can be defined in many ways. However, the permissions in our implementation list the devices entitled to access a particular resource. Thenceforth, *managers* not only need to know the address of the devices under their control but also the address of the devices authorized to access their IoT devices. *Managers* can enforce the policy creating a transaction toward the smart contract with all that information.

5) Policy Modification: Similar to the *policy definition*, *managers* can modify and delete policies at any time. The method is similar to the method described in the *policy definition*. If a *manager* adds an existing policy using the *AddAccessControl* operation, that policy gets modified automatically.

6) Policy Discovery: When device S_2 in Fig. 3 wishes to access a resource hosted by device S_1, S_2 sends a CoAP message requesting the resource information of S_1. S_1 consequently, can request the access control information of S_2 through the *management hub*. Before an IoT device can connect to the closest *management hub*, the device first needs to discover the hub's IP address. There can be several mechanisms for discovering the closest *management hub* node but the method used in our implementation assumes a default location for every device.

The *management hub* then translates the device's message into an RPC message [16] and sends it to the *miner* in the blockchain network attached to it. The operation queries the information from the blockchain stored in the *miner*. Essentially, that means that the operation is not a transaction and is not stored in the blockchain. As a result, the operation is processed immediately and does not incur any fee. Once the *miner* informs about the access policy of S_1 to the *management hub*, the *management hub* translates the answer back to S_1. S_1 acts accordingly depending on the information received by the *management hub*. Fig. 3 shows a successful answer, and therefore, S_1 sends the information of the resource to S_2.

D. System Limitations

The adoption of the blockchain technology in our solution improves the way IoT devices can be managed. However, the blockchain technology has some technicalities that could limit the solution proposed in this paper. This section explains those technicalities and describes methods to overcome them.

1) Cryptocurrency Fees: Cryptocurrency fees are a fundamental part of blockchain-based computing platforms. All the transactions include a fee, and miners are awarded with certain amount of cryptocurrency money if they successfully manage to include one of their mined blocks into the blockchain. In some public ledgers there is a minimum fee amount required for a transaction to be accepted. That is a method used by some systems to avoid unwanted spam transactions.

In our architecture, IoT devices do not belong to the blockchain network themselves. The *management hub* nodes translate the messages from the devices into RPC messages and forward them to the blockchain network. In fact, querying information from the blockchain does not incur any fee and, therefore, the *management hub* can request information freely from any device. However, only the *manager* nodes will be able to create transactions on behalf of the devices and will also have to pay the transaction fees on behalf of them.

2) Processing Time: It is a fact that transactions take time to get accepted into the blockchain. As of writing this paper, Bitcoin's [5] transactions can take up to 10 min and 12 s in Ethereum [10]. However, as stated before, the *management hub* nodes do not need to use transactions to request the information from the blockchain network. The *management hub* can query the information immediately from the blockchain's node attached to it. Thus, the *management hub* can provide the information to the devices in real time. However, the transactions created by the *manager* nodes might incur long delays. In some situations, it might be inadequate to wait for such time. For instance, in a situation where a *manager* node grants or denies access to a particular resource in a device. An unauthorized attacker could gain access to the restricted information before the revoked operation by the *manager* node is spread and accepted by the majority of the miners.

One possible solution to overcome that disadvantage is the introduction of an expiration date parameter in the operations of the smart contract. This way, the policy rules can expire automatically after a certain time. Since time is not really specified in the blockchain, the expiration time can instead be translated as the number of accepted blocks into the blockchain. Therefore, a certain rule can expire after a specific number of mined blocks. This solution does not solve the case in which a rule in a smart contract has to be revoked immediately before its expiration time. Those specific cases are still very hard to solve using the blockchain. The best way would be to include higher transaction fees for the revocation operations to minimize the time spent adding the revocation operation into the blockchain network.

III. Implementation

We developed a proof of concept (PoC) implementation of the decentralized access control system in order to test and evaluate it. The following section provides additional details about our implementation, in particular regarding the IoT devices, *management hub*, and the blockchain network.

A. Blockchain Network

The chosen blockchain technology for our PoC implementation was Ethereum [10]. We developed our prototype in our own private Ethereum network under a generic *genesis block*. We chose a private blockchain since all the elements of the prototype are more dimensioned providing us more accurate results than a public blockchain when evaluating the system. However, the goal of this implementation is to deploy it in public blockchains in real scenarios.

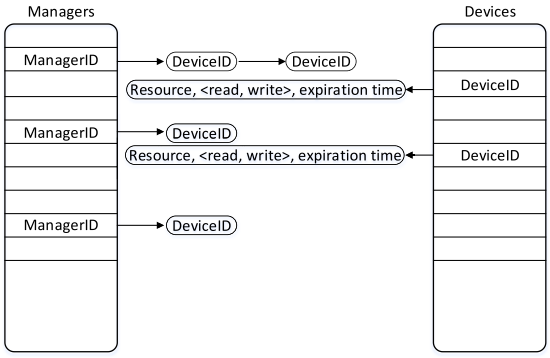


Fig. 4. Data structure in the smart contract.

Ethereum is a programmatic platform that includes a Turing complete scripting language called *Solidity*[4] that can be utilized to build, deploy and implement smart contracts. These contracts have no restrictions in terms of size and are stored in the blockchain.

There are two types of accounts in Ethereum. One called *externally owned* account, meaning it is controlled by private keys and the *contract* accounts controlled by contract code; when the contract account receives a message, the code is executed. All the *managers* in our architecture are externally owned accounts while the smart contract is deployed under a contract account.

In addition, Ethereum supports the sort of operations that query the information of the blockchain without requiring any transaction fee since they do not need to be mined. The operations are called *Call*, and they invoke a function in a smart contract. Such operations are used in the *management hubs* to query the information from the access control system.

1) Smart Contract: The single smart contract in our system was implemented using the *Solidity* programming language. The constrained device information, manager information, and the access control policy details are stored in two different data structures in the smart contract as shown in Fig. 4. The data structure used to store the information is called *mapping*. Mapping structures are similar to hash-tables where the values are initialized from the start with all possible keys. Mapping facilitates combining different data types to form a unified data type.

2) Management Hub: The *management hub* is a JavaScript interface that helps the IoT devices to connect with the blockchain network. The interface uses the *Web3* JavaScript API to communicate with the Ethereum nodes through RPC calls and a CoAP JavaScript library called *node-coap*[5] to connect with the IoT devices.

B. IoT Devices

Devices are implemented using the LibCoAP library.[6] LibCoAP is a *C* implementation of CoAP. It can support transport layer security utilizing the tinydtls[7] framework. The

[4][Online]. Available: https://solidity.readthedocs.io/
[5][Online]. Available: https://github.com/mcollina/node-coap
[6][Online]. Available: https://libcoap.net/
[7][Online]. Available: https://projects.eclipse.org/projects/iot.tinydtls

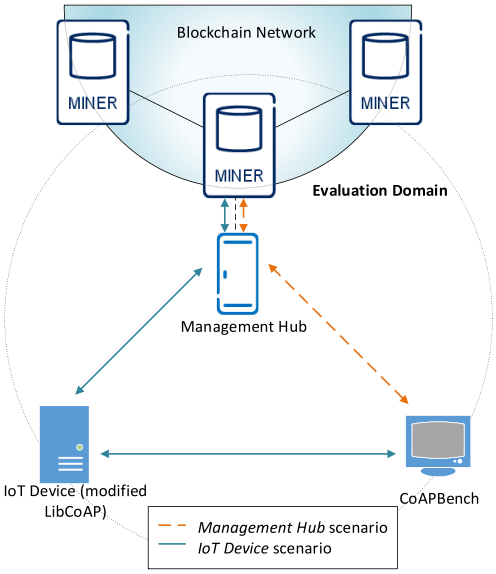


Fig. 5. Evaluation domain.

LibCoAP code was modified to automatically generate a public/private key per device. The key is 20-bytes long and used to uniquely identify the devices in the management system. The library implements a CoAP client and a CoAP server. The CoAP server is intended to listen to CoAP requests and respond to them while the CoAP client creates the CoAP request messages.

IV. Evaluation

Ethereum [10] is one of the most popular blockchain-based distributed computing platforms. Consequently, literature already provides extensive analysis and benchmarking[8,9] of the platform and its clients[10] in terms of performance and scalability capabilities.

For this reason, this section intentionally ignores the evaluation of the Ethereum network and targets the new components introduced in our architecture which are not part of the Ethereum network as such: the *management hub* and the IoT devices. In this section, we evaluate how the introduction of the *management hub* nodes in the blockchain system affect the overall delay of the architecture. Along these lines, we evaluate if the integration of the DTLS library as well as the connection of the IoT devices with a *management hub* node would be a technical limitation for the approach presented in this paper.

A. Experiment Setup

The experiments were done on an Ubuntu-16.04 desktop with Intel Core i7-950@3.07 GHz. We used Docker[11] and an image called *vertigo/ethereum*[12], which is derived from the

[8][Online]. Available: https://github.com/ethereum/wiki/wiki/Benchmarks
[9][Online]. Available: https://github.com/ethereum/tests
[10][Online]. Available: https://blog.ethcore.io/performance-analysis/
[11][Online]. Available: https://www.docker.com
[12][Online]. Available: https://github.com/vertigobr/ethereum

golang implementation image *client-go* of the Ethereum protocol *ethereum/client-go*. *Vertigo* has been slightly modified to make it simpler to run a private Ethereum network. The IoT devices ran our modified version of the LibCoAP library on the same machine.

To dimension our experiments, we use a benchmark tool called CoAPBench[13], which uses Californium[14] as the CoAP implementation baseline. CoAPBench is a tool that resembles ApacheBench and uses virtual clients to meet the defined concurrency factor. CoAPBench sends confirmable requests and waits for the response before issuing the next request. If messages are lost, the client times out after 10 s and records the loss in a separate counter. CoAPBench is, to the best of our knowledge, the only benchmark tool available for CoAP. However, the tool was too limited for our testing purposes and we decided to extend it with new functionality. After our modifications, CoAPBench was able to send POST, PUT, and DELETE messages and allow specifying a payload in the request messages.

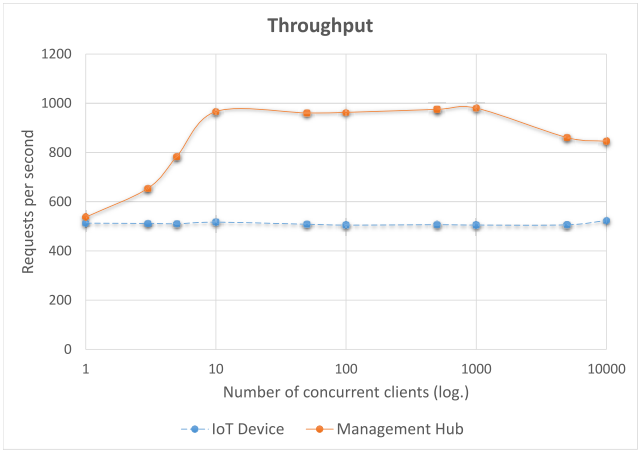
B. Performance

For this experiment, we evaluate how the introduction of the *management hub* node in the blockchain system affects the latency of the access control operations in the system as depicted in Fig. 5.

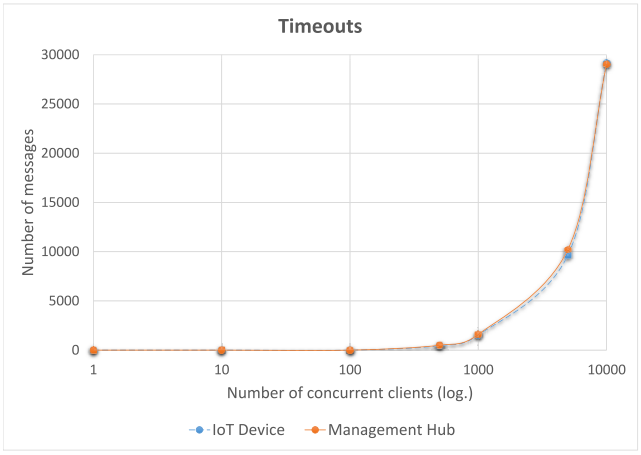
First, we evaluate the performance of the *management hub* independently, that is, a set of virtual IoT devices from the CoAPBench tool connect directly with the *management hub*. In this scenario, we configure the virtual clients to request the access information of a certain IoT device to a resource in another IoT device. Once the *management hub* receives the request, it fetches the information from the blockchain network through an RPC call and returns the response to the virtual client.

In our second scenario, we evaluate the performance of an IoT device connected to a *management hub* node, that is, a set of virtual IoT devices from the CoAPBench tool request the resource information of another IoT device which in turn is connected to a *management hub*. All the IoT devices in this scenario integrate with the DTLS library too. This scenario evaluates the performance since an IoT device requests the information of a resource from another IoT device until it obtains that information. In this scenario, a set of virtual IoT devices from the CoAPBench tool request the resource information from a single IoT device called IoT *device* in Fig. 5. The IoT *device* has a modified version of the LibCoAP library including a DTLS library and acts as a CoAP Server. The IoT *device* requests access permission from the *management hub* before providing the resource information to any virtual IoT device from the CoAPBench. The *management hub*, simultaneously, requests the information from the blockchain network. Once the authorization is obtained from the blockchain network and proxied to the IoT *device* through the *management hub*, the IoT *device* grants or denies the information of that resource accordingly.

[13][Online]. Available: https://github.com/eclipse/californium.tools
[14][Online]. Available: https://eclipse.org/californium/



(a)



(b)

Fig. 6. (a) Throughput performed independently in a *management hub* and in an IoT device requesting the information from a *management hub*. (b) Number of timeout request messages of the tests performed in (a).

In both scenarios, the blockchain network grants all the requests to all the resources. In addition, the tests in both scenarios were performed with a different number of concurrent clients. Every test was measured five times, for 30 s each time, to calculate the average values and the number of concurrent clients ranged from 1 to 10 000.

Fig. 6(a) shows the results of both scenarios. Note that both Fig. 6(a) and (b) use a logarithmic scale. The first scenario (called *management hub* in the figure) shows how the throughput in the *management hub* increases from 500 requests per second until it achieves a steady throughput of 950 requests per second with ten concurrent clients reaching their maximum capacity. The performance slowly declines beyond that point. The decrease in the performance is directly associated with the number of timeout request messages as Fig. 6(b) shows.

Furthermore, the second scenario (called IoT device in the figure) achieves a steady throughput at 500 requests per second during all the tests. In this scenario, all the concurrent clients request the resource information from a single IoT device (shown in the Fig. 5 as IoT *device*) and, as a result, the latency between the *management hub* and this single IoT client limits the whole performance of this scenario. For this

TABLE I
STRIDE CLASSIFICATION

	S	T	R	I	D	E
Management Hub	X	X	X	X	X	
Manager	X					
IoT Device	X					

reason, the performance is exactly the same with one concurrent client in both scenarios. In general, in both scenarios, the limiting factor was the latency to fetch the access control information from the blockchain network.

As a baseline, our solution suffers the overhead of waiting for the blockchain network to issue the access control information. That waiting negatively affects the performance of our system. Nevertheless, the *management hub* performs best with up to 1000 concurrent clients without excessive timeouts at 900 requests per second. We consider the performance of the *management hub* to be acceptable considering that a WSN can have several *management hub* nodes attached to it. The performance of the IoT *device scenario* in terms of scalability is overall acceptable if we consider that the IoT devices are constrained in their nature and their hardware capabilities will be a much bigger limiting factor than the software in many cases.

V. SECURITY ANALYSIS

Security is of paramount significance in any management system and our system should not be an exception. Even though the design of our system aims to facilitate the access control of resources in constrained scenarios, the solution should provide a satisfying level of security. In this section, we identify the main possible threads in our architecture and provide solutions to guarantee the best level of security.

To identify the threats in our system, we use the STRIDE model [17] by asking whether one or more of the thread types apply. STRIDE classifies the threats into six categories, and its acronym derives from them: 1) spoofing; 2) tampering; 3) repudiation; 4) information disclosure; 5) denial of service; and 6) elevation of privileges.

As shown in Table I, each of the elements of our architecture is susceptible to a set of threats.

As a result, even though the blockchain technology provides a certain level of security such as integrity and reliability of the data, IoT devices are not part of the blockchain network and have to rely their access control decisions on the *management hub* nodes. A malicious *management hub* could spoof (impersonate a *management hub*), tamper (modify the access control information sent to the IoT devices), repudiate (claiming to have not performed an action), DoS (degrade the information sent to an IoT device), or disclose unauthorized information of the IoT devices. Signed certificates could solve that issue. The *management hub* nodes could get signed certificates from a certificate authority and the IoT devices could verify the authenticity of the *management hub* nodes through them.

Moreover, since the queries from the IoT devices to the *management hub* nodes are not transactions on the blockchain for obvious performance benefits, the blockchain loses the ability to verify which access control rules are implemented properly by the *management hub* nodes. That information could be stored locally in every management domain where the IoT devices reside. Since the IoT devices could be part of different management domains during their lifetimes, that information would be spread among many nodes making it difficult to audit and track it. For critical access control systems, the blockchain could force the *management hub* nodes to use transactions instead of queries. This solution incurs a performance penalty but could increase security in the system in some particular cases.

On the other hand, the discovery of the closest *management hub* node in a network should be reliable, as should the discovery of the address of the smart contract by the *managers* and the *management hub* nodes. That information could be queried from the *agent node* or stored privately on a trusted network accessible storage.

Furthermore, once an IoT device registers in the system for the first time, a malicious *manager* in the blockchain could claim control of that device. However, an IoT device should verify the registration under a *manager* before the operation is accepted in the blockchain. Otherwise, any *manager* could register any device under its control. A similar threat could occur when a malicious IoT device impersonates another device. In this case, the communication between the devices is done through DTLS which prevents spoofing as well as eavesdropping and tampering.

VI. RELATED WORK

Developing solutions for the IoT requires collaboration among different technologies. A common theme in this paper is the combination of blockchain, access control and IoT. This section explores the extent of those technologies in IoT by looking at previous research carried out for the topic.

A. Blockchain and Internet of Things

Conoscenti *et al*. [18] conducted a systematic literature review on the blockchain for the IoT. The survey described several papers that manage data collected by IoT devices. As an example, Wilson and Ateniese [19] described a system to verify the identity of the data and Zyskind *et al.* [20] described a method to preserve the data ownership of the IoT devices. Unlike this paper, none of the papers in the survey propose an architecture where managers can manage the entire lifecycle access policies of the IoT devices regardless of their location or provenance.

To the best of our knowledge, the only previous work related to our solution is [21], which describes a cryptocurrency blockchain-based access control framework called *FairAccess*. However, there are several differences between the work in [21] and ours. First, this paper focuses on creating a single smart contract to define the policy rules of the management system. The access control policies are defined creating transactions toward that smart contract. In contrast, the work in [21] creates a different smart contract for the access control policy of every resource-requester pair. Second, Ouaddah *et al.* [21]

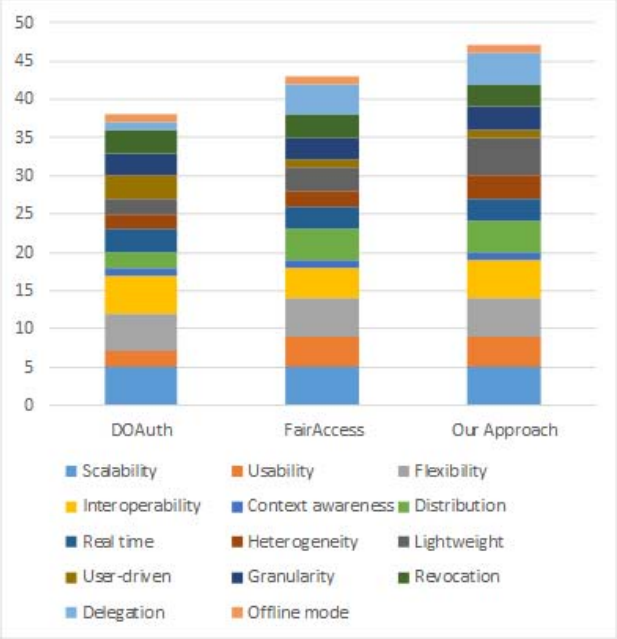


Fig. 7. Quantitative evaluation of the decentralized access control solutions proposed in [22].

included the IoT devices in the blockchain. We focus on a wider number of IoT devices that do not have the capabilities to run the blockchain technology in their systems. Third, our system provides the access control information to the devices in real time.

B. Access Control and IoT

Ouaddah *et al.* [22] provided an extensive review of different access control solutions in IoT. The survey identifies the current access control mechanism used in IoT and argues that commonly used Internet protocols cannot, in every case, be applied to constrained environments.

Based on its extensive literature review, Ouaddah *et al.* [22] identified and listed three decentralized authorization and access control solutions: 1) decentralized open authentication (DOAuth); 2) FairAccess [21], [23]; and 3) the autonomous decentralized peer-to-peer telemetry (*IBM Adept*) [15] framework. However, the same reference classifies the DOAuth-based access control solutions as a heavyweight protocol for IoT scenarios due to its communication and processing overheads. Then again, the *IBM Adept* solution provides a messaging and file sharing framework to build IoT applications but it does not yet implement an access control mechanism. *FairAccess*, to the best knowledge of the authors, has some similarities with our solution but, as explained in the previous section, our solution is much more broad focusing on devices with limited capabilities to support the blockchain in their systems. Fig. 7 shows a quantitative evaluation of the different access control solutions based on the evaluation method defined in [22]. The *IBM Adept* framework is not shown in the figure since it is not yet implementing any access control system. As the figure shows, our approach brought better results than the existing ones.

Further, Ouaddah *et al.* [22] overlooked the work done in the *Open Mobile Alliance* called LWM2M [24] and the work done by the IETF called CoAP management interface (CoMI) [25]. LWM2M and CoMI are protocols for IoT device management in a centralized manner. Unlike them, our implementation is decentralized and is not based on such protocols.

VII. Conclusion

In this paper, we address the scalability problem of managing access to billions of constrained devices in the IoT. Certainly, centralized access control systems lack the ability to deal with increased load efficiently. This paper introduces a new access management system that mitigates the issues associated with managing numerous constrained IoT devices. The solution is fully decentralized and based on blockchain technology. Since the majority of IoT devices are largely constrained to support blockchain technology directly, the IoT devices in our design do not belong to the blockchain network which makes easier the integration of the current IoT devices to adapt to our system.

The goal of this paper was to provide a generic, scalable, and easy-to-manage access control system for IoT and to implement a PoC prototype that proves our design. According to our implementation and evaluation, our solution scales well due to the fact that numerous constrained networks can be connected simultaneously to the blockchain network using specific nodes called *management hub* nodes. Additionally, the versatility of having different *management hub* nodes distributed around the whole blockchain network and connected in different ways to the constrained networks provides a considerably high flexibility to our solution. In general, our solution is able to adapt to various IoT scenarios confirming that blockchain technology can embrace IoT technology at its fullest.

Acknowledgment

The authors would like to thank N. Beijar and A. Ranjbar for their contribution to this paper.

The implementation work of this paper is partially based on a master's thesis titled "Using Blockchain Technology and Smart Contracts for Access Management in IoT Devices" written by R. Bagchi at Ericsson for the University of Helsinki.

References

[1] "Ericsson mobility report: On the pulse of the networked society," Ericsson, Stockholm, Sweden, Rep., Nov. 2017. [Online]. Available: https://www.ericsson.com/mobility-report

[2] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.

[3] X. Sun and N. Ansari, "Dynamic resource caching in the IoT application layer for smart cities," *IEEE Internet Things J.*, to be published.

[4] C. Li and L.-J. Zhang, "A blockchain based new secure multi-layer network model for Internet of Things," in *Proc. IEEE Int. Congr. Internet Things (ICIOT)*, Jun. 2017, pp. 33–41.

[5] S. Nakamoto. (2008). *Bitcoin: A Peer-To-Peer Electronic Cash System*. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[6] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[7] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure Information Networks*. Cham, Switzerland: Springer, 2015, pp. 258–272. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-0-387-35568-9_18.pdf

[8] K. He *et al.*, "DeyPoS: Deduplicatable dynamic proof of storage for multi-user environments," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3631–3645, Dec. 2016.
[9] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Proc. 35th Annu. Cryptol. Conf. Adv. Cryptol. II (CRYPTO)*, Santa Barbara, CA, USA, Aug. 2015, pp. 585–605, doi: 10.1007/978-3-662-48000-7_29.
[10] G. Wood. (2013). *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. [Online]. Available: http://gavwood.com/paper.pdf
[11] Y. Sompolinsky and A. Zohar. (2013). *Secured High-Rate Transaction Processing in Bitcoin*. [Online]. Available: https://eprint.iacr.org/2013/881.pdf
[12] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, "Byzantine fault tolerance, from theory to reality," in *Proc. Int. Conf. Comput. Safety Rel. Security (SAFECOMP)*, 2003, pp. 235–248. [Online]. Available: http://citeseer.ist.psu.edu/696238.html
[13] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," Internet Eng. Task Force, Fremont, CA, USA, RFC 7959, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7252.txt
[14] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," Internet Eng. Task Force, Fremont, CA, USA, RFC 7905, Jan. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6347.txt
[15] *The Autonomous Decentralized Peer-to-Peer Telemetry (ADEPT) System*, IBM, Armonk, NY, USA, 2015. [Online]. Available: https://www-935.ibm.com/services/multimedia/GBE03662USEN.pdf
[16] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *ACM Trans. Comput. Syst.*, vol. 2, no. 1, pp. 39–59, Feb. 1984. [Online]. Available: http://doi.acm.org/10.1145/2080.357392
[17] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover security design flaws using the STRIDE approach," *MSDN Mag.*, vol. 21, no. 11, Nov. 2006.
[18] M. Conoscenti, A. Vetrà, and J. C. D. Martin, "Blockchain for the Internet of Things: A systematic literature review," in *Proc. 13th Int. Symp. Internet Things Syst. Manag. Security (IOTSMS)*, 2016, pp. 1–6.
[19] D. Wilson and G. Ateniese, "From pretty good to great: Enhancing PGP using bitcoin and the blockchain," *CoRR*, vol. abs/1508.04868, pp. 368–375, Aug. 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1508.html#WilsonA15
[20] G. Zyskind, O. Nathan, and A. S. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Security Privacy Workshops (SPW)*, San Jose, CA, USA, 2015, pp. 180–184.
[21] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, *Towards a Novel Privacy-Preserving Access Control Model Based on Blockchain Technology in IoT*. Cham, Switzerland: Springer, 2017, pp. 523–533, doi: 10.1007/978-3-319-46568-5_53.
[22] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman, "Access control in the Internet of Things: Big challenges and new opportunities," *Comput. Netw.*, vol. 112, pp. 237–262, Jan. 2017, doi: 10.1016/j.comnet.2016.11.007.
[23] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Fairaccess: A new blockchain-based access control framework for the Internet of Things," *Security Commun. Netw.*, vol. 9, no. 18, pp. 5943–5964, 2016.
[24] *OMA Lightweight M2M Technical Specification*, Open Mobile Alliance, San Diego, CA, USA, 2017. [Online]. Available: http://openmobilealliance.org/iot/lightweight-m2m-lwm2m/
[25] P. V. der Stok, A. Bierman, M. Veillette, and A. Pelov, "CoAP management interface," Internet Eng. Task Force, Fremont, CA, USA, Internet-Draft, draft-ietf-core-comi-00, Jan. 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-comi-00

Oscar Novo received the M.Sc. degree in telecommunication software from Aalto University, Espoo, Finland, and the M.Sc. degree in computer science from the Universidad Politècnica de Catalunya, Barcelona, Spain. He is currently pursuing the Ph.D. degree in computer science at Aalto University.

He is currently with Ericsson Research, Helsinki, Finland. His current research interests include Internet of Things, blockchain, computer networking, multimedia, and computer architectures.