Blockchain-based IoT-Cloud Authorization and Delegation

Nachiket Tapas*, Giovanni Merlino*, Francesco Longo*
*Università degli Studi di Messina, Italy
{ntapas,gmerlino,flongo}@unime.it

***Abstract*—In a Smart City scenario, the authors envisioned an IoT-Cloud framework for the management of boards and resources scattered over a geographic area. It can also become a tool to let device owners contribute freely to the infrastructure. In comparison to datacenter-oriented Cloud middleware, the administrator and the owner of the infrastructure are not one and the same. This translates into the requirement to support delegation-enabled authorization. In this paper, the authors investigate an authorization and delegation model for the IoT-Cloud based on blockchain technology. In particular, the scheme is implemented in the form of smart contracts over the Ethereum platform. Indeed, this approach represents an enhancement, over a function previously designed in a centralized fashion, by enabling the user to audit authorization operations and inspect how access control is actually performed, without blindly trusting the Cloud as a proxy for access to resources.**

***Keywords*-Smart Cities, IoT, Blockchain, Cloud, Smart Contracts, Ethereum, access control, authorization, delegation**

I. Introduction

Cities are complex ecosystems where people, objects, buildings, vehicles, natural elements interact in ways that are often difficult to analyze and understand. Both social and technological issues merge thus making cities a fertile application domain for different sciences and technologies. The concept of Smart City has become pervasive in multidisciplinary research fields, ranging from architecture and urban-planning to information and communication technologies (ICT), as a new paradigm to manage and organize the city life.

From the ICT point of view, a Smart City infrastructure can be viewed as a multitude of heterogeneous network-enabled cyber-physical "things" providing sensing and actuation facilities, such as traffic sensors, security cameras, traffic lights as well as citizens' smartphones. Such huge amount of objects usually belongs to different owners and administrative domains. Recently, Internet of Things (IoT) has gained attention as a technological trend aiming at providing methods and mechanisms for the interconnection and communication of such smart objects while Cloud computing has been taken into consideration as the paradigm of reference for managing them elastically, on-demand, and "as-a-service", after (possibly) applying abstraction and virtualization techniques.

The heterogeneity in the ownership of Smart City objects represents an interesting problem to be investigated, mainly related to the management of access control, authorization, and delegation of IoT resources. As an example, let us take into consideration the #SmartME project [1]. #SmartME is a crowd-funded initiative aiming at morphing Messina into a Smart City. The main goal is to disseminate IoT resources throughout the territory of the Messina municipality thus creating a ubiquitous sensing and actuation infrastructure and a virtual laboratory to which multiple tenants can contribute with their own resources and on top of which they can develop applications and services for research, business, and administrative activities. Several tenants have been already identified and yet more to come. The Messina municipality is the main #SmartME tenant providing several IoT resources hosting sensing and actuation subsystems monitoring the territory from several points of view, e.g., weather stations, soil monitoring stations to prevent hydrogeological instability, water monitoring stations for the two salt-water lakes situated in the Ganzirri locality. The University of Messina also provided several resources in the form of Arduino YUN-based IoT nodes hosting a set of sensors to monitor temperature, pressure, humidity, quality of air, and other quantities. Finally, private citizens have been encouraged in contributing with their own private resources by involving, e.g., meteorology enthusiasts and community of makers. Such a complex and dynamic environment calls for authorization and delegation models and mechanisms. In fact, while the Messina municipality and the University of Messina provided their resources almost permanently, private citizens are more prone to contribute their resources only on a temporary basis whenever under demand, delegating other stakeholders of the system (mainly small and medium enterprises developing innovative applications and services) to the use of such resources for prototyping and testing activities.

In a recent work [2], we took into consideration such a challenge and extended our IoT-Cloud framework, Stack4Things [3], with an access control, authorization, and delegation model trying to overcome the limitations of existing solutions based on access control protocols and policy frameworks such as XACML [4] and CCAAC [5] which may result in heavy-weight implementations for resource constrained nodes, such as typical sensor/actuator-hosting boards and can, therefore, be considered unfeasible. However, the proposed solution implicitly require the end user to trust the Stack4Things framework and its (hidden) behaviour. In this paper, we introduce the use of blockchains to let the user do without trusting the IoT-Cloud framework, at least for what regards authorization.

The rest of the paper is organized as follows. In Section II

978-1-5386-4705-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SMARTCOMP.2018.00038

there is a description of the background about the middleware, the technologies involved, and the motivations. Then, Section III is where the design, notable implementation details, and some experimental results are shown. In Section IV a selection of related work can be found. Finally, in Section V we draw some conclusions and outline future work.

II. Background

In this section, we first show the architecture of the Stack4Things framework, illustrating some details about how the already existing authorization and delegation mechanisms used to work. Then, we provide some background about blockchain technologies, with specific reference to the Ethereum platform [6]. Finally, we list the main motivations that prompted us in undertaking the present work.

A. Stack4Things authorization and delegation

The overall architecture of Stack4Things is defined in [3]. It highlights interactions and communication facilities between end users, the Cloud, and (possibly mobile) sensor- and actuator-hosting IoT nodes.

On the IoT node side, the *Stack4Things lightning-rod* represents the point of contact with the Cloud infrastructure allowing the end users to manage node-hosted resources even when nodes are behind a NAT or a strict firewall. This is ensured by WebSocket-based tunneling and WAMP-based[1] messaging between the Stack4Things lightning-rod and its Cloud counterpart, namely the *Stack4Things IoTronic service*.

The Stack4Things IoTronic is designed as an OpenStack service, providing end users with the possibility to manage one or more nodes, remotely. The main goals of IoTronic lie in extending the OpenStack architecture towards the management of mobile/embedded system-hosted sensing and actuation resources.

In Stack4Things, authentication mechanisms are implemented by leveraging the OpenStack Keystone subsystem. Keystone is an implementation of the OpenStack Identity service, providing authentication and high-level authorization mechanisms through the use of credentials and authentication tokens. To authorize an incoming request, Keystone validates a set of credentials supplied by the issuing user. Initially, the credentials are represented by a username and a password. As soon as such credentials are validated by Keystone, an authentication token is released that can be exploited by the user in subsequent requests.

OpenStack ensures API protection by exploiting the role-based access control (RBAC) model. Each token issued by Keystone includes a list of roles for the user. When the user calls a service, that service interprets the set of user roles and determines which operations or resources each role grants access to. However, delegation mechanisms, i.e., temporarily granting a user permissions to perform an API call on a specific subset of the resources available as a whole and then revoking them, could be quite hard to implement in terms of Keystone APIs, due to the requirement to dynamically associate API calls to roles, a feature which requires a number of non-trivial modifications to policy (JSON) files. For such a reason, we devised for Stack4Things its own authorization and delegation mechanisms to access resources without resorting to the standard OpenStack approach.

[1] See http://wamp.ws.

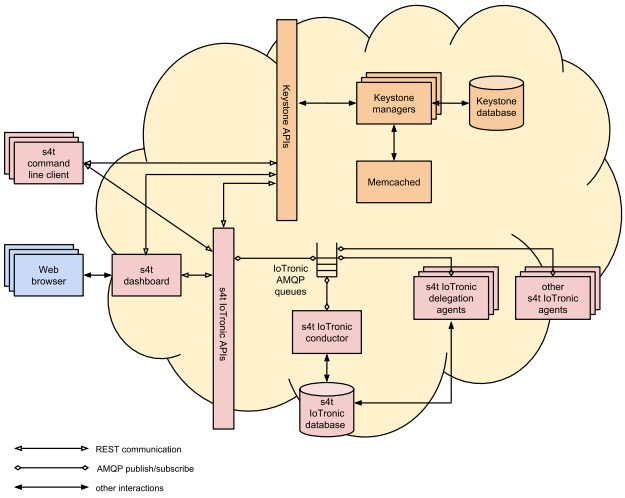

Fig. 1. Cloud-side Stack4Things architecture with focus on authentication, authorization, and delegation mechanisms.

Figure 1 shows the Cloud-side Stack4Things architecture with specific focus on authentication, authorization, and delegation mechanisms. While authentication is implemented by fully relying on Keystone, a specific IoTronic agent, namely the *Stack4Things IoTronic delegation agent*, deals with authorization and delegation duties. Whenever an API call is issued by a user on a specific IoT-node and after the authentication token is validated by interacting with Keystone, the Stack4Things IoTronic APIs contact the *Stack4Things IoTronic delegation agent* in order for it to check if the user is granted access to the specific operation on that specific IoT-node or not. If authorization is granted, the Stack4Things IoTronic APIs deliver the request to the *Stack4Things IoTronic conductor* which triggers all the steps that are needed to fulfill it, involving other Stack4Things IoTronic agents if required.

Fig. 2 depicts the entity-relationship model for authentication and delegation mechanisms in Stack4Things. The *Node* entity represents the set of IoT nodes managed by the IoTronic service while the *User* entity represents the set of users that are somehow allowed to interact with IoT nodes. The *Operation* entity models the set of API operations that needs to be authorized on specific IoT nodes. The *Role* entity represents the set of IoTronic roles that are currently defined in the system.

Finally, the *Delegation* entity contains the list of roles assigned to users for each IoT nodes. Delegations can be distinguished in two main categories. First level delegations represent roles that are assigned to users by default or by an administrator, e.g., the role IoT-node-owner is automatically

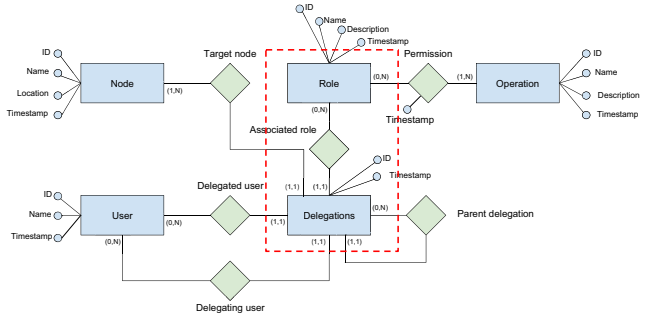

Fig. 2. The entity-relationship model for the authorization and delegation mechanisms in Stack4Things.

assigned to the user that creates a node in the system, representing the situation in which a user contributes a node to the system and therefore has all the permissions on that node. Lower level delegations represent roles that are temporally assigned to a user by another user that is already granted with that role, or with a role that includes all the considered permissions. In this sense, they represent an operation of trust from the delegating user to the delegated one. Each lower level delegation contains the information about the parent delegation from which it derives. In this way, the IoTronic implements a multi-level delegation system in which if a delegation is revoked all its child delegations are revoked (cascade revocation). More details about the legacy mechanisms that have been enhanced in this work can be found in [2].

The red box, in Figure 2, highlights the portion of the data model that has been implemented on-chain while migrating toward blockchain-based mechanisms.

B. Blockchain, Ethereum, and Smart Contracts

In 2008, Satoshi Nakamoto [7] proposed a cryptocurrency named Bitcoin in a completely decentralized environment. Blockchain is the enabling technology behind Bitcoin. A Blockchain is a public distributed ledger that stores all transactions ever executed in the network. It is a sequence of blocks in which each block is linked to the previous block via its cryptographic hash. The design of the Blockchain is such that it resists any modification to data. Once the data has been recorded in a block on the chain it cannot be altered without changing all subsequent blocks. This immutable nature of blockchain leads to an open, distributed ledger capable of efficiently recording transactions between couples of entities and allows public verification. A peer-to-peer network is responsible for managing the distributed ledger and a specific protocol is enforced for validating new blocks. Once the network confirms the validity of a block, the block is added to the chain permanently. The Blockchain can easily be proposed as a generalized framework for managing assets in a distributed way in a read-only manner. Blockchain can be particularly useful in situations requiring maintenance of unmodifiable and verifiable audit trail.

Ethereum is a decentralized generalized transaction ledger [6]. The key elements of Ethereum are Blockchain as a backbone, a Turing-complete language, and an unlimited publicly verifiable storage. It can be viewed as a transaction-based state machine where each valid transaction causes the state machine to move to the next state, thus, forming a chain of states. Bitcoin is the most popular cryptocurrency to date but it lacks support for a programming environment to leverage the decentralized environment for other applications. The Turing-complete language of Ethereum supports the creation of decentralized applications, called DApps, that work over a peer-to-peer network. Ethereum handles internal state and computation via Ethereum Virtual Machine (EVM). Each node on the peer-to-peer network runs the EVM which executes transactions and code based on global consensus. Ethereum encourages building and deployment of DApps. Thus, any centralized service can be decentralized using Ethereum. A Decentralized Autonomous Organizations (DAO), which is fully decentralized, autonomous organization, can be built using Ethereum. Another important feature of Ethereum is Smart Contract. Ethereum implements DApps through smart contracts running on top of the EVM.

In 1994, Nick Szabo - a legal scholar, and cryptographer - realized that a decentralized ledger could be used for smart contracts[2]. Smart Contracts are a formal generalization of a transaction based state-machine. Also known as self-executing contracts, Blockchain contracts, or digital contracts, smart contracts enables the exchange of cryptocurrency, or any tangible value in a fair, conflict-free way while eliminating any third party. Smart contracts can be characterized by three properties namely autonomy, decentralization, and auto-sufficiency. Autonomy implies once the contract is on the Blockchain, no interaction is required from the initiator of the contract. Smart contracts are decentralized in nature as they exist on the Blockchain. Auto-sufficiency points towards the automatic execution of contracts based on certain conditions. Smart contracts particularly solve the problem of a trusted third party in any system. The smart contract triggers automatically on certain programming conditions and is publicly verifiable on the Blockchain, it can eliminate the need for a trusted entity. Smart contracts can find application in various scenarios e.g., real estate law, financial transactions, legal processes, crowdfunding, insurance premiums.

C. Motivations

Accordingly to the mechanisms described in Section II-A, the IoTronic service is supposed to be fully trusted by end users that expect it to manage IoT resources and provide access to them in a fair way. However, malicious administrators and/or unfair internal policies could affect IoTronic supposed behavior by, e.g., forcing it to deny access to specific operations to users with full rights. Moving authentication and delegation mechanisms to the blockchain allows moving part of the trust to the deployed smart contracts and it can

[2]See http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

be considered as a first step in the migration toward a fully decentralized version of the Stack4Things framework.

The aim of the present work is to leverage blockchain technologies with the following motivations:

- *Trustness*: Ethereum smart contracts are deployed and run on top of all the nodes in the network and consensus is reached each time a smart contract is invoked. Moreover, provided that the user is able to check that the bytecode that is deployed on the blockchain is actually coming from the claimed source code (this is actually possible and similar services are even provided online by, e.g., the Etherscan Contract Verification tool[3]), he/she can reasonably be sure that access control is performed the way it is documented.
- *Auditing*: However, this is not enough to reach a complete trustiness in the system because the user should also be sure that the Stack4Things IoTronic delegation agent is actually interacting with the smart contracts to check for user permissions. Thus, another reason for the use of blockchain is the possibility to implement a public auditing system. Thanks to blockchain (and specifically via Ethereum events and logs), each operation can be logged on the blockchain and the end user is able to check why access to a certain operation/resource has been granted or denied.

Note that, in the present work, privacy is not taken into consideration, as a requirement. In fact, maintaining the public auditing system and guaranteeing user privacy at the same time could be complex. Future work will be devoted to investigating this aspect.

III. Blockchain-based Authorization and Delegation

In this section, we first give some details about how the implementation of the blockchain-based authorization and delegation mechanisms in Stack4Things has been conducted. Then, we show how the exploited smart contracts have been designed with specific reference to the data structures and their interfaces. Finally, we provide some experimental results that give a first idea about the performance of the overall system.

A. Development process and design choices

The implementation of the Ethereum-based authentication and delegation mechanisms in Stack4Things has been conducted following an agile development process. We selected Solidity[4] as our reference language, due to its similarity with Javascript and the better support that the Ethereum community provides with respect to other languages, e.g., Serpent. As a first step, we designed and implemented all the mechanisms within a single smart contract, with complete functionality. In this phase, we used the Remix IDE[5] that acts as a Solidity compiler and allows to test simple transactions for checking correctness and removing bugs. In order to tune smart contracts' code size to satisfy gas requirements, we designed two separate contracts. Role.sol contract captures the association of each role with the corresponding allowed operations, while a second contract, named Delegation.sol, represents the relationship between users, resources, and roles. Composition design strategy is chosen over inheritance with the goal of reducing coupling between the two smart contracts, as both the paradigms are supported by Solidity. Such a decision presents a second advantage: inheritance would lead to a single contract being actually deployed to the blockchain, with corresponding increase in gas consumption.

For code development in this second phase, we used the Truffle framework[6]. Truffle is a more powerful development environment compared to Remix as it not only acts as a Solidity compiler but also allows testing smart contracts in a flexible way, supporting different testing environments. We initially tested the implemented smart contracts using the testRPC and the Ganache tools[7] that locally emulate a complete blockchain on top of the development host. On the client side, we used Javascript bindings of the Web3 libraries to interact with the contracts and to simulate use cases. Once the system was working as expected, we migrated the contracts on the Ethereum public testnets (i.e. Ropsten, and Rinkeby) - each of which exhibits specific characteristics and implements different consensus algorithms - to observe real world performance of our system. Performance parameters have been recorded to present a comparative study of our experiments.

B. Data structures and interfaces

Role.sol smart contract is designed to store the relationship between roles and corresponding allowed operations. Thus, a structure data type, named`AllowedOperations`, is used to group all authorized `operations`. Solidity facilitates efficient data retrieval via mapping consisting of key-value pair. A second member of the structure is an unsigned integer (named `index`) whose purpose will be clarified later. A mapping variable (named `roles`) is defined with role identifiers (in the form of bytes32 datatype) as keys pointing to the corresponding `AllowedOperations` structures. `Index` variable, with variable size array (named `chkRole`), maintains the existence of a particular role. A code snippet for the described data structures is provided below.

```
struct AllowedOperations {
       bytes32[] operations;
       uint index;
}

mapping (bytes32 => AllowedOperations) roles;

bytes32[] public chkRole;
```

Role.sol contract contains several public functions acting as an interface for the above-reported data structure. Transaction-

[3]See https://etherscan.io/verifyContract.
[4]See http://solidity.readthedocs.io/.
[5]See https://remix.ethereum.org.
[6]See http://truffleframework.com/.
[7]See http://truffleframework.com/ganache/].

generating functions modify the internal state of the contract, and thus, need to wait for mining and consume gas, e.g., `addRole` (to add a new role). Other functions, e.g., `getRoles` (to obtain the IDs of all the roles in the system) are declared as view functions which do not alter the internal state of the contract and, thus, do not consume gas. Prototypes of the `addRole` and `getRoles` functions are provided in the following code snippet as an example.

```
function addRole(bytes32 _roleID, bytes32[]
    _operationID) public returns (bool)

function getRoles() public view returns (bytes32[])
```

The contract contains a view function to check if a specific operation is allowed by a specific role. It will be used for access authorization of a user.

In the Delegation.sol contract, a `DelegationInfo` structure is used to store information about a delegation, i.e., an identifier of the delegating user, an identifier of the beneficiary user, an identifier of the IoT resource, and an identifier of the assigned role.

```
struct DelegationInfo {
       bytes32 delegatingUserID;
       bytes32 beneficiaryUserID;
       bytes32 resourceID;
       bytes32 roleID;
       uint index;
}

mapping (bytes32 => DelegationInfo) delegInfo;

bytes32[] public chkDeleg;
```

Similarly to Role.sol contract, we introduce an index to track the existence of a delegation. As a key for such a mapping, we explored the use of keccak256 hash (shorthand notation for the KECCAK-256 variant of SHA-3) that allows obtaining a single key from a combination of: beneficiary user identifier, IoT resource identifier, and a role identifier. Checking for the existence of a delegation is similar to what has been done in the Role.sol contract, except for the computation of the keccak256 hash. Here, an important assumption that needs to be mentioned is that the identifier of the beneficiary user, the identifier of the IoT resource, and the identifier of the role are considered as primary attributes (i.e., equivalent to a primary key, combined). On the other hand, the identifier of the delegating user is considered optional as it is absolutely possible to have two different roles being delegated to the same user on the same resource by the same delegating user. Given that some default roles are assigned to the user that creates an IoT node in the system, for the sake of simplicity, we considered the IoT_Admin to be the delegating user if not specified otherwise.

Creation of new delegation alters blockchain state and, thus, generates a transaction. once mined, the changes are permanently stored on the blockchain. An access check function is used to verify if a user is granted a specific operation on a specific node. In addition to validating user access, the function logs the request on the chain by generating a transaction, thus, creating an audit trail for verification by the user.In this way, transaction-generating calls are recorded on the blockchain as state changes while view requests are stored as logs which can be later accessed from Web3 clients.

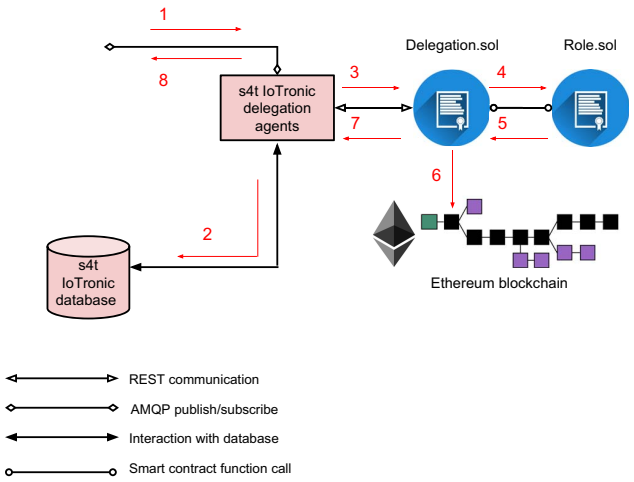

Fig. 3. Interaction between s4t IoTronic delegation agent and smart contracts on the Ethereum blockchain.

Fig. 4 represents the sequence of interactions that takes place when access to a specific operation on a specific IoT resource needs to be granted to a user. In step 1, the user requests for a particular operation on a specific IoT resource and the request is redirected to the S4T IoTronic delegation agent by the S4T IoTronic conductor. In step 2, the agent checks the S4T IoTronic database for validating the user request (e.g., the existence of the IoT resource is checked). In step 3, after successful validation, the S4T IoTronic delegation agent calls the Delegation.sol contract by using the Web3 client. A User identifier, an operation identifier, and IoT resource identifier are passed as parameters. In step 4, the Delegation.sol contract requests the roles granting access to the considered operation to the Role.sol contract. In step 5, such a list of roles is returned and the Delegation.sol contract checks if the user is associated with one of them for the considered IoT resource. In step 6, the Delegation.sol contract logs the result in the blockchain and finally, in step 7, it sends the result back to S4T IoTronic delegation agent that, in step 8, sends the result back to the S4T IoTronic conductor. Eventually, access to the user is granted or not depending on the result.

C. Preliminary evaluation: experimental results

In terms of a preliminary performance evaluation, we measured the average time required to add a delegation to the blockchain and average time required to retrieve the response for an access request across different simulators and public testnets. We used TestRPC and Ganache as simulators and Ropsten and Rinkeby as public testnets. Table I shows the average execution time, variance, and (95%) confidence interval of 1000 consecutive requests to add a delegation to the blockchain. Similarly, Table II shows the average

TABLE I
SIMULATION RESULTS FOR: ADDING DELEGATIONS

Response times [sec]	Simulators / Public Testnets			
	TestRPC	***Ganache***	***Ropsten***	***Rinkeby***
Average	0.8288 ± 0.0253	0.0929 ± 0.0026	18.9269 ± 1.1801	40.0096 ± 1.0690
Variance	0.1667	0.0018	362.5130	297.4485

TABLE II
SIMULATION RESULTS FOR: ACCESS AUTHORIZATION

Response times [msec]	Simulators / Public Testnets			
	TestRPC	***Ganache***	***Ropsten***	***Rinkeby***
Average	172.5450 ± 14.3827	59.4440 ± 1.4865	115.4050 ± 16.0350	122.6510 ± 15.2851
Variance	53847.7437	575.2221	66930.6076	60816.8741

execution time, variance, and confidence (95%) interval of 1000 consecutive access authorization requests. In order to prevent overwhelming of the simulator/testnets, we introduce a delay in processing of transactions. To eliminate any bias of the system, we recorded execution time for 1020 transactions and removed 10 initial and 10 final transactions. Access authorization requests take an order of magnitude less time when compared to requests for adding a delegation to the system. The response time for access authorization requests is minimal, which is compliant with our design goals. Compared to TestRPC, Ganache performs better, both in terms of adding a new delegation or validating access. When comparing public testnets, we find that the proposed design performs better on Ropsten testnet which is more closer to main Ethereum network. Future work will be devoted to analyzing this results, capturing useful insights for optimizing our implementation.

IV. RELATED WORK

Traditional access control models, like Role-Based Access Control (RBAC) model [8], depend on a trusted third-party authorization engine to grant access. This creates mistrust in the system. Also, such models are based on a centralized model, which does not overlap with the distributed architecture of the IoT.

This led us to explore blockchain technology as an access control mechanism, due to its distributed nature. A.Ouaddah et al. [9] used blockchain to store and audit access control policies. L.Chen and H.Reiser [10] store access rights for a resource in a blockchain and manage them via transactions.

MeDShare proposed by Xia, Qi, et al. [11], for controlling access to sensitive medical data, uses blockchain to store the history of operations performed on the data and smart contracts to enforce access control.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an enhancement to an IoT-Cloud solution in the form of a decentralized design for resource access authorization and delegation duties. The proposal features blockchain as a technological building block for this scheme and smart contracts as the main engine for trustless decentralization and independent audit of operations. Details about the design and implementation of the contracts have been included, as well as some preliminary results. Ongoing and future work is planned to deploy the smart contracts in the #SmartME ecosystem and other Stack4Things-powered testbeds, to experiment the validity of the solution at scale. The proposed solution will also be compared against promising approaches like OAuth2.

REFERENCES

[1] D. Bruneo, S. Distefano, F. Longo, and G. Merlino, "An IoT testbed for the Software Defined City vision: the #SmartMe project," in *2016 IEEE Int. Conf. on Smart Computing (SMARTCOMP)*, May 2016, pp. 1–6.

[2] D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito, "IoT-cloud authorization and delegation mechanisms for ubiquitous sensing and actuation," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 222–227.

[3] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4things: a sensing-and-actuation-as-a-service framework for iot and cloud integration," *Annals of Telecommunications*, vol. 72, no. 1, pp. 53–70, Feb 2017.

[4] L. Seitz, G. Selander, and C. Gehrmann, "Authorization framework for the internet-of-things," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th Int. Symp. and Workshops on*, 2013, pp. 1–6.

[5] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad, "Capability-based access control delegation model on the federated iot network," in *The 15th International Symposium on Wireless Personal Multimedia Communications*, Sept 2012, pp. 604–608.

[6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[8] D. Ferraiolo, R. Kuhn, and R. Sandhu, "Rbac standard rationale: Comments on "a critique of the ansi standard on role-based access control"," *IEEE Security Privacy*, vol. 5, no. 6, pp. 51–53, Nov 2007.

[9] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Towards a novel privacy-preserving access control model based on blockchain technology in iot," in *Europe and MENA Cooperation Advances in Information and Communication Technologies*. Springer, 2017, pp. 523–533.

[10] L. Y. Chen and H. P. Reiser, "Distributed applications and interoperable systems, 17th ifip wg 6.1 international conference, dais 2017, held as part of the 12th international federated conference on distributed computing techniques, discotec 2017, neuchtel, switzerland, june 1922, 2017." Springer, 2017.

[11] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "Medshare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14 757–14 767, 2017.