IEEE ICBC'24 1570981757

Consensus-based Access Control

***Abstract*—Emerging systems (such as the Internet of Things (IoT) and Web3) and new security paradigms (such as the Zero Trust security paradigm) demand novel access control solutions. Traditional access control mechanisms assume a single, centralized, trusted entity responsible for defining, deciding, and enforcing access control policies. However, this paradigm does not meet the evolving demands of modern systems. In this paper, we propose a decentralized, fine-grained access control solution, exploiting Distributed Ledger Technologies (DLTs) that allows many, mutually untrusted entities, to make collaborative access control decisions. We present two different approaches, one based on smart contracts and another that leverages the consensus mechanism of Hyperledger Fabric, a popular permissioned blockchain. For both approaches, we design mechanisms for integrating external Authorization Servers (ASes) in an efficient and secure way. Lastly, we develop proof-of-concept implementations of our designs, we demonstrate their feasibility, and we evaluate their performance and security properties.**

***Index Terms*—Collaborative systems, Consensus, Access control, Distributed Ledger Technologies, Blockchains, Smart Contracts**

I. Introduction

Emerging systems and applications depart from the traditional centralized host-centric paradigm and allow multiple stakeholders to share their resources in a decentralized manner, as well as to jointly collaborate on the same resources. Examples of such systems are IoT data-sharing applications, collaborative editing tools, decentralized storage services, and many others. In such collaborative systems, resource may be owned by one entity, stored by another entity, and might refer to other entities. Existing access control mechanisms and models, which often *rely on a single trusted entity* for defining, deciding, and enforcing access decisions, are not well suited for these cases [1], [2], since they are too *rigid* and *structured*, and *they do not allow and support collaborative definition of access policies*. But even when it comes to traditional systems, a centralized approach for access control is deemed insecure by emerging security paradigms, such as the Zero Trust paradigm. For this reason, new paradigms that enable multiple *Authorization Servers (ASes)* to jointly manage access control policies and make access control decisions are required. However, this collaborative form of access control becomes challenging if these entities are not mutually trusted. In that case, consensus protocols must be securely applied, and efficient transparency, traceability, and auditability mechanisms are required.

Responding to these challenges, we explore how Distributed Ledger Technologies (DLTs) can be used for building secure and reliable collaborative access control solutions. DLT is an emerging and revolutionary technology with many benefits and intriguing properties, which can contribute significantly in this direction. DLTs have even been envisioned as an enabler of new security mechanisms that can be used for building secure and trusted access control systems [3].

In this paper, we design an application-independent access control solution that allows multiple *ASes* to cooperate and make access control decisions. Initially, we study the case where all these ASes belong to the same administrative domain (e.g., a company wishing to apply the Zero Trust paradigm). Then, we extend our design to consider cases, where each AS may belong to a different administrative domain but all ASes evaluate the same access control policy (e.g., collaboration platforms that data access requests are approved by all collaborating entities using the same policy). Finally, we further extend the latter design to consider cases, where each AS makes access control decisions based on a "local" policy and the final decision is made by applying a *consensus* protocol (e.g., a collaboration platform where users must prove they have "approval" to perform an action by at least n other collaborating entities).

Our solution addresses the main challenges of the collaborative access control [2], [4], it eliminates the need for reliance on a trusted centralized service, and it facilitates co-operation among entities that are not mutually trusted. In order to achieve our goal, we leverage Hyperledger Fabric, a permissioned, private blockchain system. We model administrative domains as Fabric *organizations* and we allow each organization to maintain its own external AS, which is accessed over HTTPS. Our Fabric-based implementation, takes measures to decrease the communication overhead towards ASes, as well as to "isolate" servers, so that they can be accessed only by the Fabric *peers* of the corresponding organization. Overall, with this paper we are making the following contributions:

- We propose a consensus-based access control solution tailored to collaborative systems based on Hyperledger Fabric.
- We design, implement, and compare two distinct approaches for implementing consensus-based access control decision
  - An approach where consensus rules are encoded in a *smart contract*.
  - An approach where consensus rules are integrated in the consensus mechanism of Hyperledger Fabric
- We develop a proof-of-concept implementation of a data sharing system and we evaluate the performance and security properties of both approaches.

The remainder of this paper is organized as follows. In Section II, we present some background information about Hyperledger Fabric. In Section III, we introduce our consensus-based

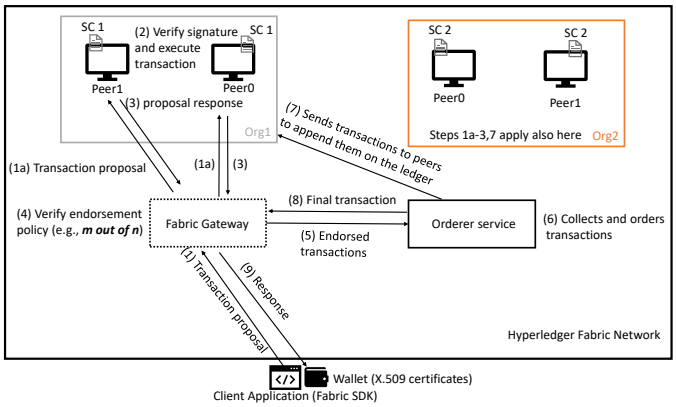

Fig. 1. Transaction flow in a Hyperledger Fabric blockchain network.

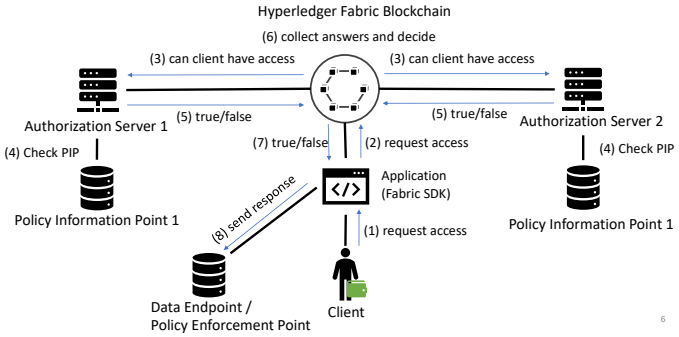

Fig. 2. Overview of the system's architecture.

access control model backed by DLTs and the two approaches, while in Section IV, we present the implementation and the evaluation of the proposed solutions. In Section V, we present related work in the area. Finally, Section VI, concludes the paper and presents some future research directions.

II. Background – Hyperledger Fabric

A popular implementation of a private blockchain is Hyperledger Fabric (for simplicity, we will refer to it as Fabric) [5]. A Fabric network consists of many organizations that come together to form the blockchain network. Every organization includes its own Certificate Authority (CA) that dispenses X.509 certificates (identities) to its members. In Fabric, the membership to the network is controlled, by a special node called Membership Service Provider (MSP). The MSP implementation follows the PKI model. The MSP identifies which CAs are authorized to issue valid certificates and maps the trusted CAs to organizations. Fabric, like other popular blockchains, supports the execution of smart contracts, written in any general-purpose language, such as JavaScript. *Smart contracts in Fabric, in addition to other popular blockchains can communicate directly with the "outside world", i.e., they can send requests, call APIs, etc.*

Fabric introduces a new model for transactions, called *execute-order-validate*, in addition to the existing *order-execute* model. Therefore, the transaction flow (depicted in Fig. 1) slightly changes. Initially, the transaction is executed by all the appropriate (endorsing) peers, then, it is validated against an *endorsement policy*, and finally, it is ordered in a block based on the consensus protocol, before it is committed to the ledger. In the ordering phase, the results are gathered by the orderer(s), which create and broadcast the block to the network. The orderers are orchestrated through a Kafka-based consensus algorithm. The endorsement policy [6] defines the peers that must execute the transaction and the required combination of responses, e.g., "*n out of m*" peers should execute the transaction and produce the same output in order for the transaction to be considered valid. Endorsement policies are defined *per smart contract*. The endorsement policy is validated twice, during the transaction flow. The Fabric gateway, which is usually a peer in the blockchain network, when it collects all the responses from all the appropriate peers, it validates if the endorsement policy is satisfied (shown also in Fig. 1). Then, when the orderer broadcasts the block to the peers to append it on the ledger, the peers are checking the endorsement policy of each transaction included in the block, again (this is not show in the figure).

In any blockchain, after the deployment of a smart contract, all the participating peers must have the same state and the same source code of the smart contract. However, a new feature introduced in Fabric v2.0[1], allows smart contracts not to be identical across the members of the network. Organizations can slightly modify a smart contract, e.g., to perform different validations in the interest of their organization. Nevertheless, a transaction will be validated and committed to the ledger, only if the required number of organizations endorse the transaction with matching results, as it happens in any other case.

III. Consensus-based access control

In this section, we present the design of the proposed consensus-based access control solution and we detail the components of our system, as well as, their interactions, also illustrated in Fig. 2. The goal of the proposed system is to allow authorized users (clients) to get access to a resources, hosted in a particular endpoint identified by a URL (the endpoint is oblivious to our solution). Clients request access to the resource through a client application that interacts with the blockchain network and the smart contracts. To approve a resource access request, many ASes, *owned by one or more organizations* depending on the scenario, must cooperate to decide whether clients can be granted access or not, based on collaboratively defined access control policies. The ASes are also identified by URLs, e.g., https://as1.company1.

From a high level perspective, our system involves the following functions.

Set up. Our system assumes a setup phase during which the Fabric network is created, the blockchain and its parameters are configured, and the smart contracts are developed and deployed on the blockchain. Before deploying the smart contracts, organizations must agree on the endorsement policy

[1]https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatsnew.html

for the smart contracts. In this phase, the certificates and the keys for the members of the network (peers, orderer, etc.) are created. Furthermore, each organization should configure its ASes with access controll policies. Policies are in the form of $pub\_key_{client_x} \rightarrow allow/deny$ and are stored in a Policy Information Point (PIP). As we discuss in the following, all ASes may be configured with the same policy, or each organization may configure its ASes with "local" access control policies, e.g., AS_1 has a policy that dictates that the client $pub\_key_{client_x}$ can access $data$, but AS_2 is configured with a policy dictating that $pub\_key_{client_x}$ cannot access $data$. Finally, in the setup phase, the client obtains a X.509 certificate, issued by a CA that participates to the Fabric network, in order to be able to interact with the system.

Access request. After acquiring the certificate, a client can request access to a resources, stored in an endpoint. The request, which is essentially a transaction to the blockchain, is initiated via a client application that utilizes the Fabric client SDK and is configured with the X.509 client's certificate. The transaction includes the client's certificate along with the client's digital signature, which is generated using the client's private key. The request ends up on the smart contract deployed on the Fabric blockchain. Following, the smart contract verifies the client's signature using client's public key, extracted from her certificate. Then, the smart contract includes client's public key to the request and forwards it to the appropriate ASes, requesting access for the client with that particular public key.

Access decision. Each AS checks the client's public key and responds to the request, based on its "local" access control policies (two different organizations (or ASes) might have different "local" access control policies for the same client). To evaluate the requests, the ASes are communicating with their PIPs and send back to the smart contract their decision. Following, the blockchain collects the replies from all ASes and based on the access control policies, defined collaboratively by all involved organization, responds with the final decision (the exact flow of this step is presented in the next subsections).

Access enforcement. In this final phase, the client application receives the definitive response and forwards it along with the client public key to the Policy Enforcement Point (PEP). The PEP, which is configured with a certificate, queries the ledger, and gets the access decision for that particular public key. If the access response is true, then the client is granted access to the resource otherwise the PEP sends back to the client application an appropriate error message.

Our system shares the main components, protocols, and access control flow common to any access control systems, but introduces key enhancements for collaborative systems. We incorporate two distinct PDPs; one, involving the ASes, is responsible for decisions based on "local" access control policies, and the other, utilizing the blockchain and the smart contracts, takes the results and decides based on collaboratively defined access control policies. The PEP in our system is incorporated in the resources endpoint, which is a single trusted entity, as in any other legacy access control system.

A. Usage scenarios

In this paper, we consider three distinct scenarios that vary in complexity. These usage scenarios have been carefully chosen to encompass a spectrum from simpler, more straightforward scenarios to progressively more sophisticated ones. By exploring these diverse scenarios, we aim to comprehensively analyze the applicability, security, and performance of our proposed designs across a range of real-world contexts. In the first (baseline) scenario, we consider the case, where a protected resource is owned only by single organization. Namely, only one entity is responsible for defining, deciding, and enforcing the access control policies and requests. However, we assume that the organization owns multiple ASes that perform the same checks. All of the ASes should individually respond to the access requests and then the final response is determined based on these individual responses. *This approach is used to enhance the resilience of the system against AS compromises.* The second scenario is similar to the baseline scenario, with the difference that instead of having only one organization with multiple ASes, we have multiple organizations owning multiple ASes. However, as in the baseline scenario, all the ASes from all organizations perform the exact same checks. *This approach can be used for use cases, where the organizations do not fully trust each other.* The final scenario that we consider includes multiple organizations owning ASes, but instead of performing the same checks each AS group performs different checks. For instance, the ASes of one organization verify if the requester is a student and the ASes of another organization if her age is over 20. *This scenario can be used for attribute-based access control, where each organization provides and verifies different attributes.*

Next, we present the design of our solutions and we detail the exact flow of the access control processes considering the aforementioned scenarios.

B. Smart contract-based approach

Our first solution utilizes smart contracts to enable collaborative access control. In our design, we consider two types of smart contracts, the Authorization Smart Contract (ASC), which receives the access request from the client, calculates the final decision, and responds back to the client the access decision, and the Request Smart Contract (RSC), which sends requests to ASes. The design of our solution is presented in Fig. 3. The solution works as follows.

The client, after acquiring her certificate, sends an access request through the client application to the deployed ASC. The smart contract receives the request and forwards the request to RSC. Following, the RSC extracts the public key of the user and forwards the request to all participating ASes. To do so, the smart contract should include all the URLs of all ASes. The smart contracts are deployed on all peers participating in the Fabric network. We should note here that in Fabric, every peer participating in the network that has deployed the smart contract, executes the transactions,

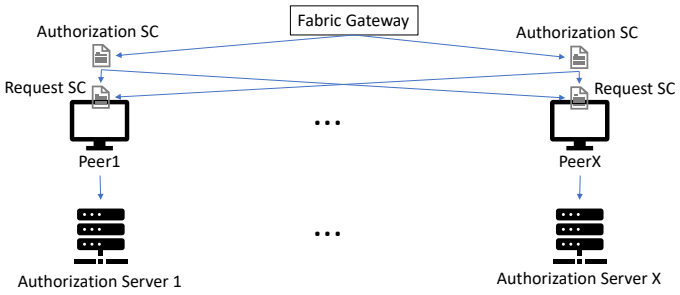

Fig. 3. Design of the smart contract-based solution.

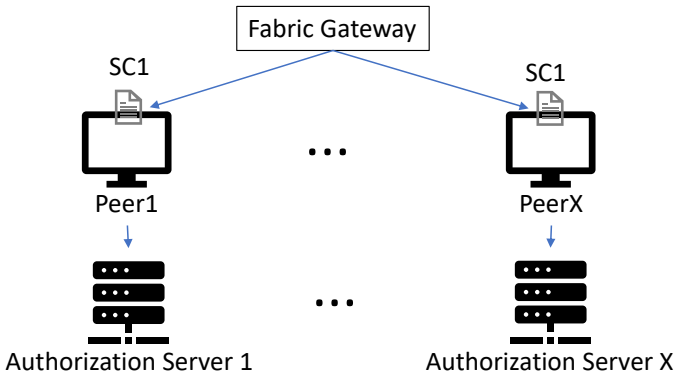

Fig. 4. Design of the consensus-based approach.

meaning that all of them will send requests to all ASes. For instance, if there are 1000 peers and everyone has to execute the transaction, then all these peers will send a request to all ASes. Each AS will eventually end up receiving 1000 requests for the same access request. It is clear that this can lead to a Denial of Service (DoS) for the ASes. So, to tackle this challenge and to allow the peers request different ASes, we modify the DNS configuration of each individual participating peer, by modifying its hosts file. In particular, in our peer customization process, for each peer, the hosts file is adjusted to associate the URL (which is included in the smart contract in the form of http://as.company) with the IP address of a distinct AS. Thus, every peer will request a different AS, if the peer-to-AS ratio is one to one (as shown in Fig. 3). If there are more peers than ASes, then $\frac{\#peers}{\#ASes}$ peers will request each AS, and so on. Due to this customization process, there is no need for including all the URLs of all the ASes within the smart contract. Thus, we include only one generic URL, e.g., http://as.company1. In this manner, we also manage to successfully secure the location information of the ASes, by excluding their URLs from the smart contracts.

When an AS receives the access request from the smart contract, it checks the client's public key and responds accordingly. Following, the RSC receives the responses from the ASes and sends these responses back to the ASC. Then, the smart contract aggregates the responses from individual ASes and determines the final response based on the collaboratively defined access policies. Various strategies can be employed for calculating the final response, depending on the use case. Examples of such strategies include deny-overrides, majority voting, or a threshold scheme. In the third scenario, where each group of ASes check different attributes, the collaboratively defined access control policy can be for example, all requirements should be meet or two out of three of the following requirements should be meet to grant access. Example requirements are if the requester is a student, she is over 18 years old, and she lives in Athens, Greece. Finally, the client application receives the response from the blockchain network and its subsequent actions are contingent upon the response (as we show above in the access enforcement process).

C. Consensus-based approach

With the previous design, we achieve to decentralize the PDP and enable transparency and auditability regarding the access decision and how it is achieved, by making the smart contracts responsible for this process, since the smart contracts are deployed and run in all the involved peers and their execution output is immutably written in the ledger. However, this design might be inappropriate for some use cases. For instance, in the second and third scenarios, where there are many organizations, an organization might not want other organizations to know its individual access responses. To address that, in this solution, we take advantage of some of the features of Fabric, as well as its consensus mechanism to create a more elaborate design.

As shown in Fig. 4, instead of having two types of smart contracts, we have only one smart contract. This smart contract receives the request from the client application, extracts the public key of the user and forwards the request to the appropriate AS. As in the previous solution, the peers are modified to send the request on a distinct AS. Following, the flow remains the same, the ASes receive the request, they checked their "local" access control policies, and based on them, they respond. Then, the smart contract receives the response and sends it back.

Now, in contrast to the preceding design, where a smart contract is responsible for collecting the individual responses, aggregating them, and deciding the final access response, the current design exhibits a different flow. The *endorsement policy*, which has been agreed among the involved organizations during the setup phase, plays also the role of the collaboratively defined access control policy. For instance, in a scenario that involves four organizations, and in order for a client to get access to data three out of four organizations should permit it, then the endorsement policy would be *OutOf(3, 'Org1MSP.member', 'Org2MSP.member', 'Org3MSP.member', 'Org4MSP.member')*. As we mentioned above, for an endorsement policy to be validated, the peers that are dictated by the endorsement policy should execute the transaction and produce the same results. Therefore, in a scenario, where there are two organizations, each one owning

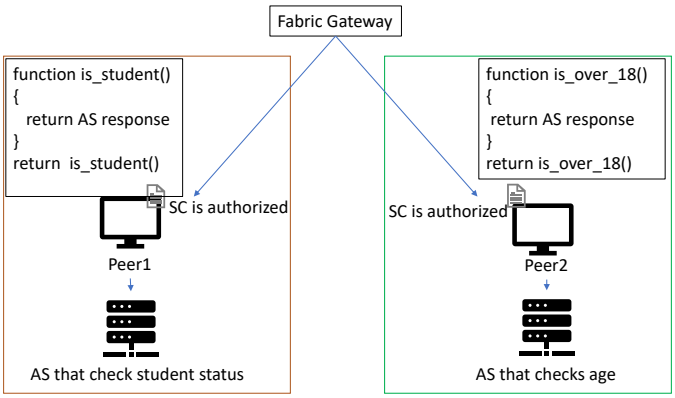

Fig. 5. Alternative to DNS configuration.

one peer, and both of them should give positive responses, then, to get access, the members of the organizations should execute the transaction, send requests on the appropriate ASes, and then respond all $true$. If one of them responds with $false$ and another with $true$, then during the endorsement, the transaction will not be validated and not be appended on the ledger, since the answers are not the same. The Fabric Gateway will send back to the client application an error message and the client will not get access to data. If both ASes respond with true or false, then the transaction will be validated and written in the ledger and the Fabric gateway will return the result. If the result is true, the access is permitted, otherwise, the access is denied.

D. Practical considerations

In order for each peer to communicate with different AS, despite the fact that the exact same smart contract is deployed on all peers, we modify the DNS configuration of each peer, by modifying their hosts file. In that way, every peer associates a different IP for the same URL, so each peer ends up requesting a different AS. However, this may be impractical for some use cases. We can avoid the modification of the DNS configuration of the peers by exploiting a newly introduced feature of Fabric, which allows the same smart contract not to be identical across the peers that is deployed, as we have mentioned above.

In particular, we can slightly modify the source code of the smart contract, and depending on the organization that is deployed, we configure it to include only the URL of the AS of its organization and only that. For instance, a smart contract that is deployed on the peers of organization1 includes only the URL of the ASes owned by that organization, while the same smart contract that is deployed on the peers of organization2 will include the URL of ASes owned by that particular organization, and so on.

As an illustration of this alternative solution, we present an example in Fig. 5 for the third considered scenario. In this case, we have two organizations that have to perform different verifications. The first organization has to verify if the client is student and the other if her age is over 18. Now, instead of having the same smart contract and modifying the DNS configuration of the peers, we modify the smart contract to perform different checks depending on the peer that is deployed. Following the protocols of the design, the client requests access through the client application and the access request ends up on the smart contract. Then, the smart contract sends an HTTP request to the appropriate AS. Each AS responds with true or false, based on its "local" access control policies. The smart contract receives the response from the AS and returns it, to another smart contract or to the Fabric gateway, depending on the design (first or second approach).

This solution proves applicable in cases, where modification of the DNS configuration of the peers may not be feasible. However, it is noteworthy that the URLs of the ASes are included in the smart contracts, in contrast to the alternative approach, where only a generic URL is stored within the smart contract.

IV. Implementation and Evaluation

A. Implementation

We developed a proof of concept implementation of our presented system, as well as the two collaborative access control solutions.[2] The emulated protected endpoint is an HTTP server that hosts shared data generated by IoT devices. The protected enpoint and the corresponding PEP were implemented using Node.js. The smart contracts used in our system and the ASes were implemented using Node.js and JavaScript. Furthermore, the client application, which is responsible for acquiring the certificate and interacting with the smart contracts and the PEP was implemented using JavaScript and it used the Fabric SDK. For our proof of concept implementation, the underlay Fabric network includes two peer organizations and one orderer organization, each one of them providing a single peer in the network. For the first scenario, where there is only one organization, our network consists of two peers. The ASes, we have in our proof of concept implementation, are two, as the peers in the network. All the components of the system (ASes, Fabric network, etc.) are hosted in a virtual box, having $6GB$ of ram and 2 processors.

B. Performance evaluation

To evaluate the performance of our system and its design, we conducted some experiments to find out the overhead of the blockchain and exam if it is bearable or not. To evaluate that, initially, we measured the time required to get/deny access to the shared data. In particular, we measured the time from the moment a client sends an access request, through the client application, to the smart contract, until she gets an access response. For comparison, we implemented an instance of our system without using blockchain. Namely, the client application just forwards the request directly to the appropriate ASes. In this case, the requests to each AS are made sequentially, and the URLs of all ASes are necessarily in the client application. We conducted this experiment 1000

[2]The source code of all the components of our system will be made available upon acceptance of this paper.

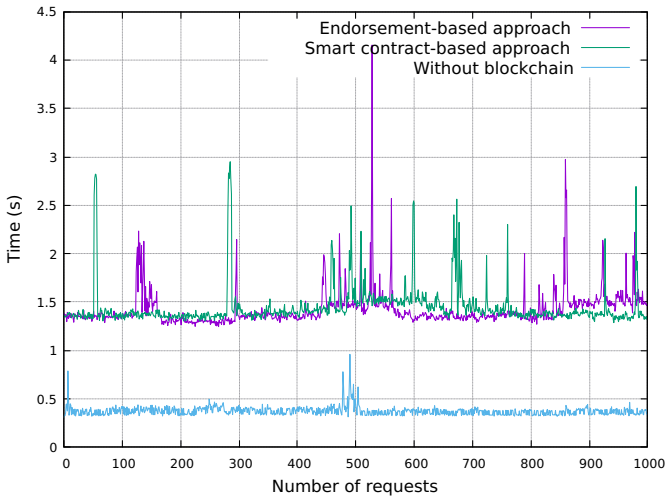

Fig. 6. Time required for requesting access and getting/denying access.

times. The results are presented in Fig. 6. The average time required for the endorsement-based design is $1.42s$, for smart contract-based design is $1.44s$, and for the baseline solution, where blockchain is not used, the average time is $0.37s$.

As we observe from the results, the overhead introduced by the blockchain is ≈ 1 second, which is tolerable in almost any case. As for the comparison of the two proposed approaches, we observe that the results are very similar, despite the fact that the smart contract-based approach involves two smart contracts and the ASes. This is happening, because to invoke a function of a smart contract from another smart contract, we are using the invokeChainode [3] function, which does not create a new transaction. In Fig. 6, we observe that there are several high spikes. This can happen for a variety of reasons. First of all, due to the ordering service of Fabric. In Fabric, transactions are sent to the orderer to be ordered into blocks. A block is created either after an adequate number of transactions has been collected, or after the expiration of a configurable timeout period. Thus, in some cases the orderer has to wait for the whole timeout period. Other cases that create the spikes can be network issues and congestion.

To get more insights on the blockchain performance and its behavior in our designs, we utilize Hyperledger Caliper [7], which is a blockchain benchmark tool. Two important performance metrics that Caliper provides, other than latency, are throughput and scalability. Due to space limitations and since the two proposed designs exhibit similar performance, we present here only the results gathered for the endorsement-based approach. We test our proposed system using Caliper, under a variety of conditions, regarding the rate at which transactions (access requests) are sent to the blockchain. In particular, we utilize two different rates. The *fixed-load* rate, which maintains a backlog of transactions by modifying the Transactions Per Second (TPS). The second rate is the *linear rate* that increases the load intensity linearly, using a starting TPS value and a finishing one. For the *linear rate*, we set the starting TPS equal to 25 and the finishing TPS is 200 and 500 accordingly. With the first, we can test the system under realistic conditions and observe the actual throughput, while with the second, we explore the performance limits of the system. The results are shown in Table I.

Therefore, it is observed that our system does not introduce much overhead regarding the performance, since the time added by the blockchain can be considered negligible and it is tolerable. Furthermore, we observe that our system demonstrates robust throughput and scalability, even in non-realistic and intensive conditions. There are also some findings showing that Fabric can scale up to 20000 TPS [8]. Finally, Fabric does not introduce any monetary costs, as opposed to other popular blockchains, such as Ethereum. For all our experiments, we used the second scenario, where we have two organizations that perform the same checks. However, the performance of the system does not affected by the scenario, since the processes are the same in all of them.

C. Security analysis

Assumptions. We assume that client manages her X.509 certificates in a secure manner utilizing a secure storage mechanism, such as a secured wallet, so an attacker cannot stole them. Furthermore, we are not concerned with malicious services that can change the flow of the protocols. In particular, we assume that the client application and the PEP, which are centralized services, are trusted by anyone and they always operate correctly, e.g., if the final collaborative access decision is to grant permission, then the PEP cannot deny it. Lastly, we assume that the endpoint, where the shared data are stored, is adequately secured and always available.

Considered Attacks. Firstly, we consider the honest-but-curious attacker model for the organization's peers. This model does not want to create a malicious activity in the system, but in addition, the attacker follows honestly the protocol steps and at the same time aims to obtain information beyond their own outputs. In our designs, an honest-but-curious adversary can only learn the final access decision. It cannot learn the endpoints (URLs) of others organizations ASes. In the second design, he cannot even learn the individual responses of the other organization's ASes.

Furthermore, in our designs, the compromise of an AS does not compromise the client's access. In particular, if an adversary has compromised an AS, then he is not able to grant or deny access to the clients, he cannot prevent clients from granting access, and he cannot modify the given access, due to the blockchain's inherent properties. For an adversary to achieve the aforementioned, he has to control more than 50% of the peers or ASes. If that happens, then the adversary can gain full control of the blockchain and he can even alter it.

Additionally, our system demonstrates increased availability and robust resilience against DoS attacks. The PIPs are oblivious to the system, only the ASes know their URLs, thus they are secured from malicious users. Moreover, the URLs of the

[3]https://hyperledger.github.io/fabric-chaincode-node/release-2.2/api/fabric-shim.ChaincodeStub.html#invokeChaincode__anchor

Rate	Succ	Fail	Send rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
Fixed load	27990	0	46.7	0.69	0.02	0.13	46.7
Linear rate (max 200)	25359	6625	53.3	60	0.01	18.90	52.4
Linear rate (max 500)	24527	11256	59.7	60.05	0.01	18.69	59.5

TABLE I
CALIPER PERFORMANCE METRICS OF THE ENDORSEMENT-BASED APPROACH.

ASes and their IPs are not stored in the smart contract nor in the ledger. Even if an adversary compromises a peer, then he only learns the IP of one AS. Through these mechanisms, our system effectively mitigates the impact of malicious attempts to disrupt service availability.

D. Discussion

In the previous section, we presented two approaches for consensus-based access control tailored to collaborative systems. With our solutions, we allow semi-trusted and untrusted entities to collaborate for providing access to IoT shared data. With our first approach, we utilize smart contracts to completely decentralize the PDP, while with the second approach, we achieve to embed the access decision on the actual consensus mechanism of the Fabric blockchain, which is already trusted by all the involved entities and does not depend on a single trusted party. On the other hand, access enforcement, on both approaches, can be done efficiently and effectively by just querying the ledger, since the time required for reading a record from the ledger is negligible.

Paci et al. [1] in their survey on access control for community-centered collaborative systems, elicit the main requirements that the access control systems should have for the collaborative systems. These requirements are about the policy specification, governance, usability, and transparency. In our work, we are not concerned how the policies are defined, in the sense that we assume that each organization has its own policies and the collaborative access policies are defined based on the use case. For instance, in a bio data use case, the patient's individual decision may weight more than the hospital's access decision, etc. However, our proposed system meets all the other requirements. In particular, regarding governance, our system allows the collaborative administration of shared resources. Regarding the transparency requirement, our system is transparent to organizations and allow them to understand collaborative decisions and their effect. All organizations can audit the blockchain and see how the final decisions have been made. Finally, regarding usability, our system support users in the inspection and configuration of their access preferences during the setup phase, when the smart contract is developed or the endorsement policy is set.

Except for the above, our system has many desirable and usability properties. New organizations, new ASes, or new access control policies can easily be added. Also, existing access control policies can easily be removed or modified, by exploiting a feature of Fabric, which allows us to upgrade deployed smart contracts [4]. In addition, the client application does not need to be changed. Besides that, our system inherits all the properties of blockchains, which are reliability, transparency, robustness, auditability, and increased availability.

Our consensus-based access control solution can serve many use cases, since the different approaches address a wide range of requirements. However, in our second approach, the endorsement policy is used as the access control policy. That means that for all users and for all shared data, we have one collaboratively defined access control policy since there is one endorsement policy per smart contract. This might be inappropriate for some use cases and scenarios, but it can be addressed by deploying different smart contracts for different shared data. Another solution to that is to take advantage of a new feature of Fabric [6] that permits to set endorsement policies at the level of state variables of smart contracts. Another advantage is that the first design can easily be implemented with minor modifications in the public Ethereum blockchain [9]. This can serve other than the considered use cases and scenarios, where openness is a critical property. However, public Ethereum does not present good performance and introduces monetary costs, which may be not negligible.

Regarding revocation, our considered use case, which is IoT data sharing, assumes that when a client gets access to shared data, then she downloads these data and there is no need for requesting access for these data again. Therefore, there is no need for having a revocation mechanism. The same applies also for delegation. In this particular use case, there is no need for a client to delegate her access to another client. Thus, we do not include a delegation mechanism too.

V. RELATED WORK

Due to the development of IoT and Web 3.0, the last few years, there is an increasing number of collaborative systems, so, the need for securing them has also been increased. The first step towards that direction is through advanced and novel access control. In particular, many works [1], [2], [10] highlight the need for new access control models for collaborative systems, presenting the requirements that collaborative access control models and systems should meet and show the inefficiencies of conventional access control models when applied on collaborative systems. Due to the inefficiencies of traditional access control, many access control solutions tailored to collaborative multi-party systems have been proposed.

Shen et al. [11] were pioneers in introducing access control for collaborative environments. Their work, one of the earliest in this domain, propose the extension of the classic access matrix model, proposed by Lampson [12], to include collaboration rights, negative rights, automation, and inheritance-

[4]https://hyperledger-fabric.readthedocs.io/en/release-2.2/deploy_chaincode.html#upgrading-a-smart-contract

based specification. Damen et al. [13] propose a collaborative access control system that is based on RBAC and a policy language that is based on a hybrid logic. For every role, the entities that need to collaborate define authorization requirements. When an access request is made, the authorization requirements are evaluated individually and the results are combined. For the final decision, the authors consider two different strategies, permit-takes-precedence and deny-takes-precedence. Their evaluation shows that the introduced overhead is from $7ms$ to $263ms$. Carminati et al. [14] enhance topology-based access control to create collaborative access control for Online Social Networks (OSNs). They propose a policy language that allows defining collaborative access policies. Then, regarding the enforcement of the access decision, authors introduce a centralized trusted entity, which receives the request and asks the involved entities if they allow access or not.

There are some more recent works that propose access control for collaborative systems. In particular, Mahmudlu et al. [15] propose a data governance model for collaborative systems. They propose a framework that can express the relationships between stakeholders and data objects. To implement that, they use the access control policy language XACML. Additionally, the work of Sheikhalishahi et al. [16] aims at securing and protecting the collaborative policies to avoid leak of sensitive information, e.g., the relationships of co-owners with other parties. To achieve that, they use homomorphic encryption and secure function evaluation. The same problem is addressed by George et al. [17] for the use case of third-party Unmanned Aerial Vehicle (UAV) services. All these works present access control solutions for collaborative systems. However, they use complex policy languages, many of them add significant computational overhead, which may not be bearable for some use cases, and all of them are relying on centralized services. In our work, we use DLTs to create secure, usable, transparent, and decentralized consensus-based access control tailored to collaborative systems.

Blockchain technology has often been considered for access control systems and solutions in the last few years [3], [18] and especially for access control in the IoT domain [19], [20]. There are many research efforts that utilize DLTs, mainly Ethereum and Fabric, in various access control methods. In particular, some recent efforts [21], [22] utilize DLT to create and manage blockchain-based Access Control Tokens (ACTs). Other works implement various access control entities as smart contracts [23], [24], e.g., to store access control policies within a smart contract, i.e., a PIP, or to implement the PDP and the PEP as smart contracts. Furthermore, there are works that modify a blockchain to serve their needs, e.g., Ouaddah et al. [25] introduce new types of transactions in Bitcoin that are used to get, grant, and delegate access, while others create their own blockchain to use it for access control [26]. All these efforts highlight the advantages of using blockchains and smart contracts in access control solutions. In our work, we utilize the smart contracts to provide secure access control for collaborative systems. On the other hand, in the second design, we leverage the actual consensus mechanism of the Hyperledger Fabric blockchain for deciding on and enforcing access. To the best of our knowledge, there is not any other prior work that uses the consensus mechanism of a blockchain for access control processes.

VI. Conclusions and Future Work

In this paper we proposed a decentralized, secure, consensus-based access control solution tailored to collaborative systems. Our solution utilizes DLTs, and in particular the Hyperledger Fabric permissioned blockchain. We presented two different approaches, one that utilizes smart contracts to allow the collaboration of many entities regarding access control and to decide the final access decision. This approach can be implemented in any blockchain, public or private that allows the execution of smart contracts. The second proposed approach takes advantage of the consensus mechanism of Hyperledger Fabric, i.e., access is decided based on the consensus mechanism of the blockchain itself. For both approaches, we proposed mechanisms for integrating external ASes, in an efficient and secure way. Our evaluation shows that the overhead introduced by our constructions is minimal and tolerable. Furthermore, our security analysis shows that our solutions are secure against a variety of attacks.

The use case considered in this paper is about IoT data sharing. In this particular use case, there is no need for continuous authorization, since, if the user gets access to data once, she does not (in general) need to repeat access. However, it would be very interesting to investigate how our solution can be adapted in use cases that require continuous authorization (e.g., for actuation), such as for Zero Trust architectures. Furthermore, we do not include in our system a revocation or a delegation mechanism, since they are not required by our use case. However, adding efficient and effective mechanisms regarding revocation and delegation would extend and strengthen the system as a whole. For this reason, we are currently experimenting with a variety of such mechanisms. Finally, most data produced by IoT devices are sensitive, so privacy must be ensured. Therefore, we plan to investigate how to exploit selective disclosure techniques to ensure privacy for the shared IoT data.

References

[1] F. Paci, A. Squicciarini, and N. Zannone, "Survey on Access Control for Community-Centered Collaborative Systems," *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018. [Online]. Available: https://doi.org/10.1145/3146025

[2] A. C. Squicciarini, S. M. Rajtmajer, and N. Zannone, "Multi-party access control: Requirements, state of the art and open challenges," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 49.

[3] Y. Liu, M. Qiu, J. Liu, and M. Liu, "Blockchain-Based Access Control Approaches," in *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2021.

[4] F. Paci, A. Squicciarini, and N. Zannone, "Survey on access control for community-centered collaborative systems," *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. Association for Computing Machinery, 2018.

[6] E. Androulaki, A. De Caro, M. Neugschwandtner, and A. Sorniotti, "Endorsement in Hyperledger Fabric," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019.

[7] Hyperledger Foundation. Hyperledger Caliper. [Online]. Available: https://www.hyperledger.org/projects/caliper

[8] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "FastFabric: Scaling hyperledger fabric to 20000 transactions per second," *International Journal of Network Management*, 2020.

[9] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[10] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong, "Access Control in Collaborative Systems," *ACM Comput. Surv.*, vol. 37, no. 1, mar 2005.

[11] H. Shen and P. Dewan, "Access Control for Collaborative Environments," in *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*, ser. CSCW '92. New York, NY, USA: Association for Computing Machinery, 1992.

[12] B. W. Lampson, "Protection," vol. 8, no. 1, p. 18–24, jan 1974. [Online]. Available: https://doi.org/10.1145/775265.775268

[13] S. Damen, J. den Hartog, and N. Zannone, "CollAC: Collaborative access control," in *2014 International Conference on Collaboration Technologies and Systems (CTS)*, 2014.

[14] B. Carminati and E. Ferrari, "Collaborative access control in on-line social networks," in *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2011.

[15] R. Mahmudlu, J. den Hartog, and N. Zannone, "Data Governance and Transparency for Collaborative Systems," in *Data and Applications Security and Privacy XXX*, S. Ranise and V. Swarup, Eds. Cham: Springer International Publishing, 2016, pp. 199–216.

[16] M. Sheikhalishahi, G. Tillem, Z. Erkin, and N. Zannone, "Privacy-Preserving Multi-Party Access Control," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, ser. WPES'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–13. [Online]. Available: https://doi.org/10.1145/3338498.3358643

[17] D. R. George, S. Sciancalepore, and N. Zannone, "Privacy-Preserving Multi-Party Access Control for Third-Party UAV Services," in *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 19–30. [Online]. Available: https://doi.org/10.1145/3589608.3593837

[18] S. Rouhani and R. Deters, "Blockchain Based Access Control Systems: State of the Art and Challenges," in *IEEE/WIC/ACM International Conference on Web Intelligence*, ser. WI '19. New York, NY, USA: Association for Computing Machinery, 2019.

[19] Y. Zhang, A. Memariani, and N. Bidikar, "A Review on Blockchain-based Access Control Models in IoT Applications," in *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, 2020.

[20] I. Riabi, H. K. B. Ayed, and L. A. Saidane, "A survey on Blockchain based access control for Internet of Things," in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2019.

[21] N. Fotiou, I. Pittaras, V. A. Siris, S. Voulgaris, and G. C. Polyzos, "Secure IoT Access at Scale Using Blockchains and Smart Contracts," in *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2019.

[22] G. Gan, E. Chen, Z. Zhou, and Y. Zhu, "Token-Based Access Control," *IEEE Access*, vol. 8, 2020.

[23] H. Liu, D. Han, and D. Li, "Fabric-iot: A Blockchain-Based Access Control System in IoT," *IEEE Access*, vol. 8, 2020.

[24] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart Contract-Based Access Control for the Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 2, 2019.

[25] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "FairAccess: a new Blockchain-based access control framework for the Internet of Things," *Security and Communication Networks*, vol. 9, no. 18, 2016.

[26] M. P. Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H.-S. Kim, D. E. Culler, and R. A. Popa, "WAVE: A Decentralized Authorization Framework with Transitive Delegation," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019.