Decentralized Authorization using Hyperledger Fabric

Muthukur Venkata Akhil Vasishta
Indian Institute of Technology Kharagpur
India
akhilvasishtamuthukur@iitkgp.ac.in

Balaji Palanisamy
University of Pittsburgh
USA
bpalan@pitt.edu

Shamik Sural
Indian Institute of Technology Kharagpur
India
shamik@cse.iitkgp.ac.in

***Abstract*—Access control in multi-user web service applications is an important problem. Access control methods ensure that only authorised users have access to data and resources by deciding whether to give access to a resource or not by verifying credentials of the entity requesting access. In several real-world situations, these credentials may not be available within a single authority and it may require verification through multiple authorities. In this paper, we propose a decentralized framework using Hyperledger Fabric and Attribute Based Access Control (ABAC) to verify access requests through multiple authorities in a decentralized manner. Hyperledger Fabric enables transparent authorization and immutability of credentials and ensures a secure operation between authorities. Each authority can safely store the credentials in their own Hyperleger Fabric network and our framework performs access requests by automating the end-to-end process of authorization across multiple authorities. The framework can be extended to perform access requests across any number of authorities and is applicable to any attribute based access control use case. We implemented a prototype of the framework using Hyperledger Fabric and the results validate the performance of the proposed approach.**

***Index Terms*—Decentralized Authorization, Computer Systems, Attribute Based Access Control Model (ABAC), Hyperledger Fabric, Blockchain.**

I. Introduction

Multi-user web service applications generally have a large number of users and not every user is entitled to access all of the information. Access control is crucial in determining who has access to which resources. To ensure that only authorised users have access to data or resources, various access control methods have been developed over the last few decades. These include RBAC (Role Based Access Control) [1] model, IBAC (Identity Based Access Control) [2] model and LBAC (Lattice Based Access Control) [3] model. ABAC (Attribute Based Access Control) [4] model provides significant flexibility in the formulation of access rules. Unlike other access control methods such as IBAC, by including subject and object properties in the access rule instead of subject and object itself, ABAC considerably reduces the size of the access policy. Due to this feature, ABAC subsumes all of the existing access control models [5].

The ABAC model evaluates access rules against the attributes of entities (subject, object, environment) relevant to a request to regulate access to the objects. Attributes are the characteristics of the entities, namely the subject, the object, and the environment. In ABAC terminology, data or resource is referred to as object and access requesting entity for a resource is referred to as subject. Each entity has a set of well-defined attributes, each of which can accept one or more different values. For example, a subject *Nancy* can have the value *Student* for subject attribute *Designation* and the value *ECE* for subject attribute *Department*. Similarly, an object *File* can have the value *Journal Paper* for object attribute *Filetype* and the value *1990* for object attribute *yearOfPublication*. The administrator or owner of objects creates access policy (set of access rules) for access to the objects using attributes of entities. Each rule specifies which subjects are authorized access to which objects in terms of attribute-value pairs. For example, an access rule can be defined to allow all students from ECE department to view journal Papers written in 1990. Here, viewing the file refers to the operation. We can also include other environment aspects (e.g., location, time) in the access rule. For instance, we can define at what time the subject may request to view the file. Every access request in ABAC is made up of three components: the subject making the request, the object being requested, and the environment in which the request is made.

Existing research in ABAC has developed new algorithms for constructing the rule set from current accesses described using the components of traditional access control models and for evaluating access requests in an efficient way. Existing centralized implementation of cloud-based ABAC becomes a soft target and involves a single point of attack. On the other hand, decentralized implementation using blockchain can maintain immutability of data and if any repudiation of data occurs, it can be traced and verified on the blockchain.

In several real-world scenarios, attributes are not always available with a single authority but available in several authorities issuing attributes. For example, a patient arriving at the hospital may request treatment through his/her health insurance. The hospital will need to check the State ID of the patient to verify the identity and check the health insurance information to determine if the insurance is applicable or not. Here, the hospital may not directly have the attributes of the patients on the State ID and insurance card. Instead, the hospital will request the State ID issuing authority to verify the patient's identity and the hospital will request his or her health insurance company to verify the insurance information.

Once the attributes are verified, the hospital can authorize the patient's treatment using the insurance.

Blockchains can be used to provide a secure means to store and verify attributes. As attributes are issued by different authorities, each attribute issuing authority will store the attributes in their own private blockchain network. As a result, we require a system or framework that can handle access requests by communicating with all the attribute issuing authorities. In our work, we design an ABAC-based framework using Hyperledger Fabric [6], which serves as a basis for building modular applications and solutions. Its modular and adaptable architecture caters to a wide range of industrys applications and its consensus method allows for scalability while maintaining anonymity. Key challenges in designing the proposed ABAC framework includes efficient storage of attributes on the Hyperledger Fabric, dividing composite requests into subrequests so that each authority is only responsible for verifying the attributes that it has issued and splitting global access rules that are present in the general ABAC model into subrules so that each authority is only responsible for verifying the part of the global rule involving the attributes that it has issued. We designed and implemented techniques for these features through chaincodes that interact with the Hyperledger Fabric blockchain network. We prototyped the proposed framework using an institute-library ABAC use case and the results validate the efficacy of the proposed approach.

II. Related Work

The distributed nature of blockchain tackles the concerns related to single point of failure in traditional centralized systems. Consensus mechanisms in the blockchain protocol ensures that only valid transactions are recorded on the blockchain. Initially, blockchains were used to implement cryptocurrencies and financial transactions. The notion of smart contract enabled the creation of diverse applications in various fields including healthcare [7] [8], IoT [9] [10], and supply chain management [11] [12] [13]. Because of the immutability, durability, auditability, and dependability features, blockchains provide significant potential to supplement traditional access control solutions. Furthermore, by using smart contracts, we can monitor and enforce access permissions under complex conditions. These features have motivated researchers to consider blockchain as an infrastructure for access control systems. [14] gives a deep insight on the state of the art blockchain-based access control solutions.

Recently the idea of using blockchain to store the attributes of ABAC is being explored and various blockchain-based implementations of ABAC have been proposed in recent years. [15] proposed an implementation of ABAC using extensible access control markup language (XACML) to define policies. This implementation used Bitcoin blockchain to store arbitrary data by which users can transparently view access control policies on resources. In [16], instead of simple transactions, smart contracts were used to impose access control regulations and implementations of XACML policies were deployed on the Ethereum platform to create a proof of concept.

ABAC policy relating the individual subjects and objects is implemented using an Ethereum-based attribute-based access control system [17], in which a smart contract is launched for each subject-object pair to record the attributes and thus implement an ABAC policy. When a subject requests access to an object, the smart contract that regulates the subject and the object receives a transaction.

Most of the existing implementations were based on the idea that a single organization is using ABAC to restrict access to its own resources. In many real world scenarios, the attributes of a subject are not issued and maintained by a single organization, but by multiple organizations. Some access control implementations have considered this aspect. Using Ethereum blockchain and Solidity smart contracts, in [18], a platform for role-based access control is designed to be used by many organisations. It uses a smart contract to initialise the roles and a challenge-response protocol to verify role ownership and user verification. [19] proposed an implementation of ABAC for decentralized authorization across multiple entities. In this implementation, each organization has their own smart contract to add attributes but the access rule checking of all attributes is done by the object organization only. This is not always feasible because it is not possible for a single organization to have information about all the attributes of subjects. It results in a need to divide global rules and access requests into sub rules and sub requests. Also, this implementation uses Ethereum to store the attributes for all the organizations. As a result, attributes issued by one organization will be entirely visible to other organizations and may create data privacy issues.

III. System Design

The proposed framework includes an *Object Authority* which maintains objects and multiple *Subject Attribute Issuing Authorities (AIAs)* since subject attributes are maintained by more than one authority. Each authority (either Object Authority or AIA) comprises of four main components namely (i) Application (ii) API-Server (iii) Chaincode and (iv) Hyperledger Fabric Network (HF) and CouchDB. The basic operations involved in a general ABAC model are (i) Adding subjects and objects, (ii) Adding rules and (iii) Processing access requests. Next, we explain how these basic operations are carried out in multiple substeps for a multi-organization setup using our framework. The workflow of these basic operations in our framework is depicted in Figure 1.

A. Adding Subjects and Objects

To add objects in the Object Authority, *Object Administrator* will enter the values (V_1,V_2) of the object attributes (OA_1,OA_2) of the object O_1 into the *Object Application* as shown in Figure 2. OI_1 is the identifier of the object O_1 in the Object Authority. *Object Application* then sends a request to the *Object Server* to call *add_object* function in the *Object Chaincode*. This function adds object in the *Object CouchDB* database and also the logs of adding the object will be stored as transactions in the *Object Hyperledger Fabric* network. If

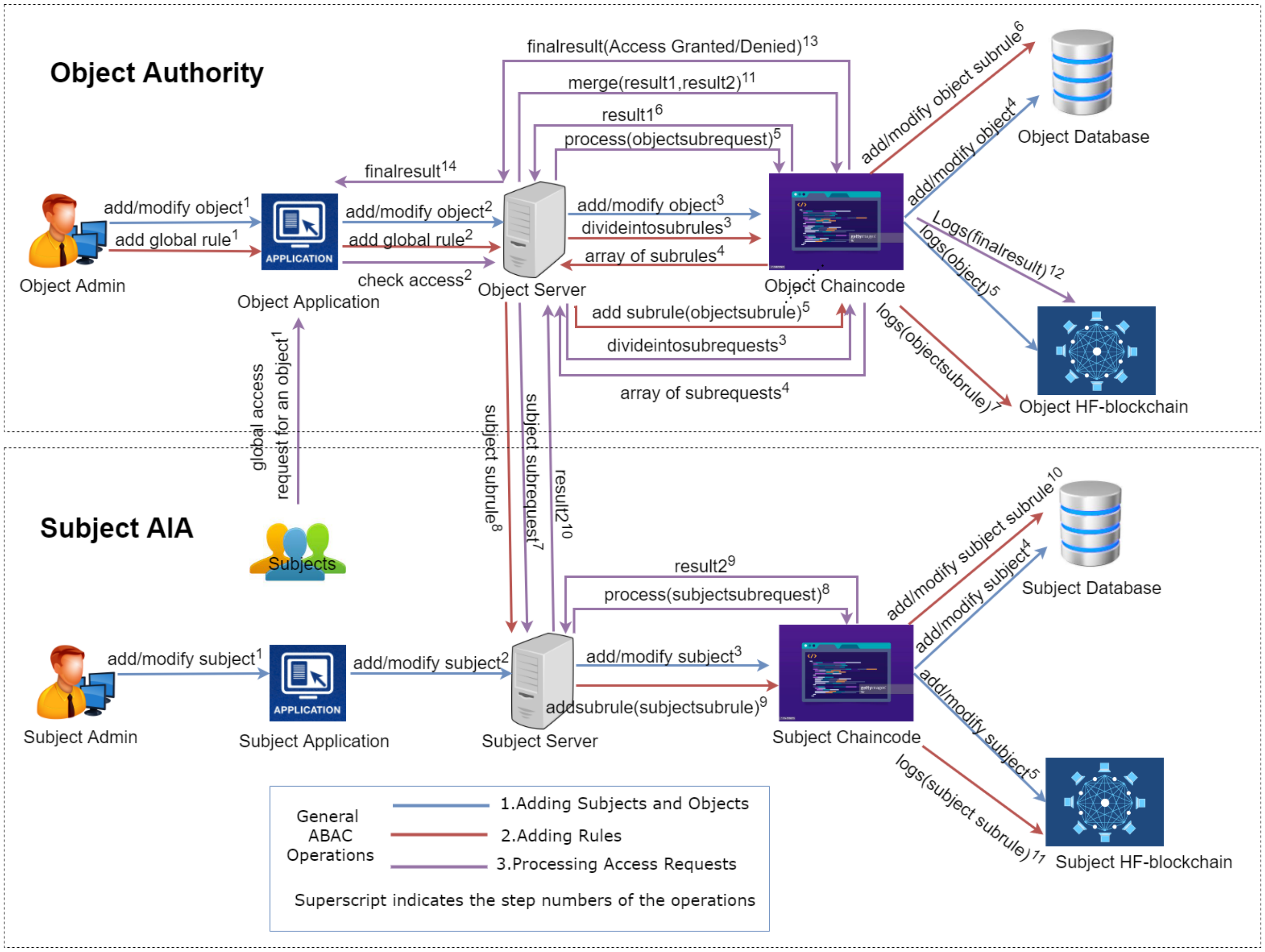


Fig. 1. System Architecture

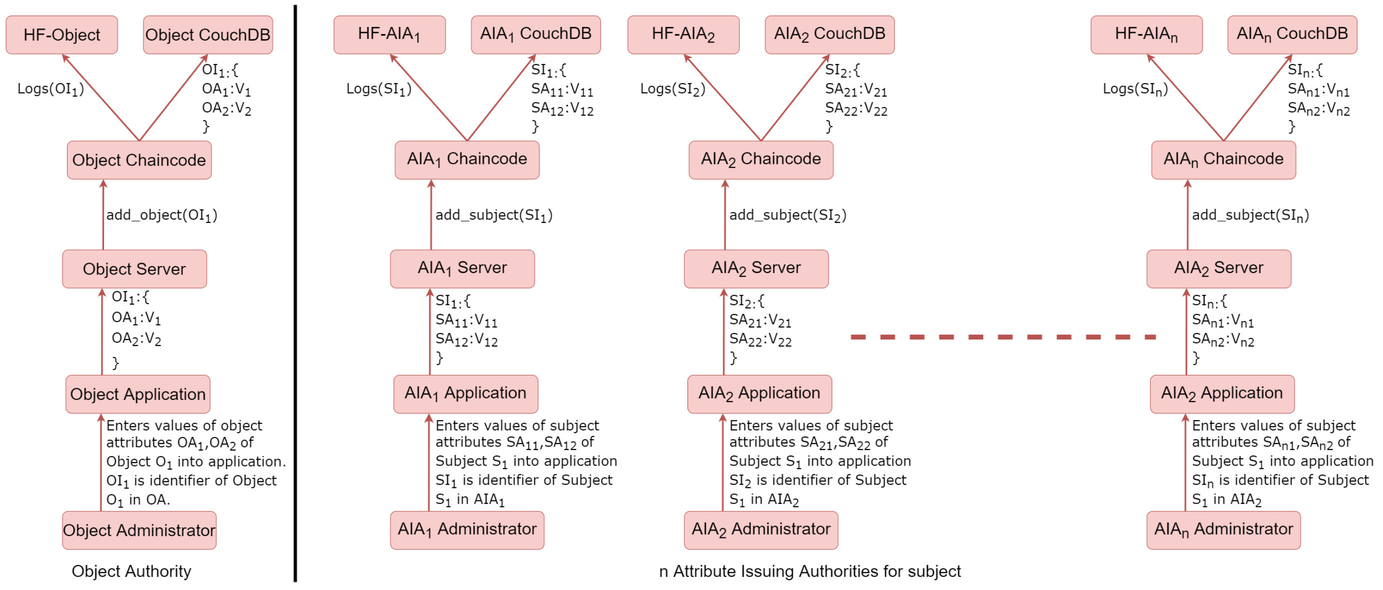
someone tries to modify the attributes of the object at a later point of time, one can verify them with the immutable entries of these logs stored in the blockchain.

Similar workflow is followed by all AIA's to add subjects as depicted in Figure 2. Instead of *add_object* function in Object Authority's chaincode, all AIA's chaincodes would be having *add_subject* function. For example, information related to the State ID, Insurance card are subject attributes and these are issued by different Authorities. Each of these authorities will be having their own independent Hyperledger Fabric networks, Chaincodes, Applications, API-Servers. SI_1, SI_2, ..., SI_n are the identifiers of subject S_1 in AIA_1, AIA_2, ..., AIA_n authorities. SA_{11}, SA_{12} are subject attibutes issued by AIA_1, similarly SA_{21}, SA_{22} are subject attibutes issued by AIA_2 and so on. V_{11},V_{12} are values of attributes (SA_{11}, SA_{12}) for subject S_1 similarly for all attributes.

B. Adding Rules

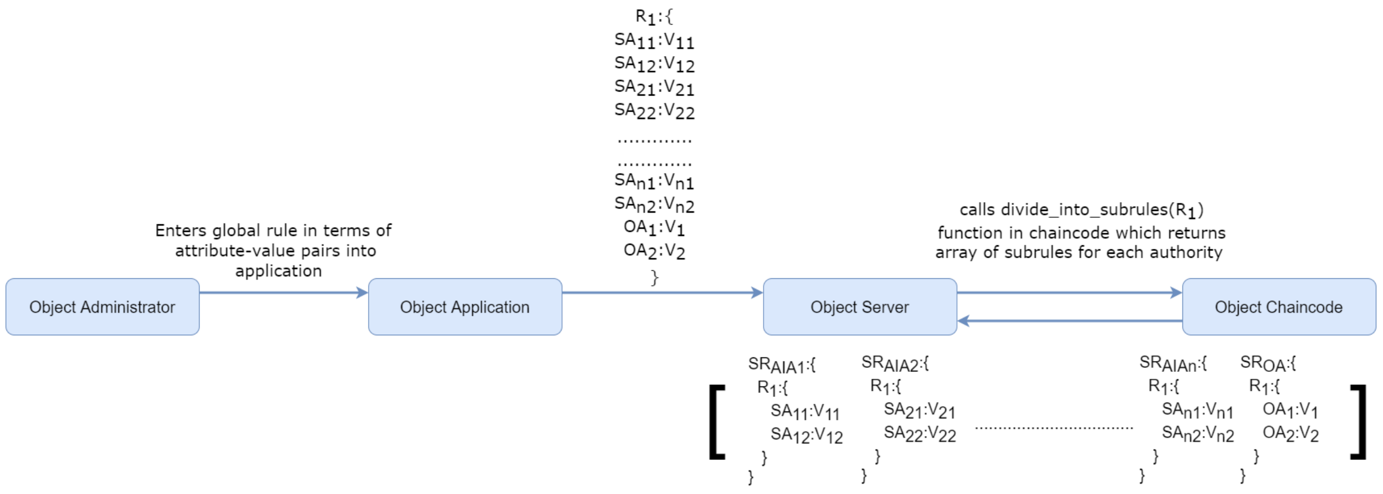
1) Splitting Global Rule into SubRules: Being the owner of objects, *Object Administrator* designs rules describing which attributes should be possessed by subjects to gain access to objects with certain attributes. A rule in ABAC can be described as conjunction of attribute value pairs of subject and object. As shown in Figure 3, *Object Administrator* will enter the global rules (R_1) into the *Object Application* which are a conjunction of all the subject and object attribute value pairs. *Object Server* will call *divide_into_subrules* function in the *Object Chaincode*. This function separates the global rule ($R1$) into subrules (SR_{AIA_1}, SR_{AIA_2},...,SR_{AIA_n},SR_{OA}) for each authority based on information of which attributes are verified by which authority, so that each authority is only responsible for validating attributes issued by themselves. This information is provided in the form of configuration files.

2) Adding Subrules in individual Hyperledger Fabric Networks: Once *Object Server* gets back the subrules for all authorities, it will send the subrule for each authority to the appropriate servers. Each authority server will then call the function *add_subrule* in their chaincodes which will add the subrule in their respective couchDB database and the logs of adding subrule will be stored in the respective Hyperledger



HF-Object
Object CouchDB
Logs(OI1)
OI1:{ OA1:V1 OA2:V2 }
Object Chaincode
add_object(OI1)
Object Server
OI1:{ OA1:V1 OA2:V2 }
Object Application
Enters values of object attributes OA1,OA2 of Object O1 into application. OI1 is identifier of Object O1 in OA.
Object Administrator
Object Authority
HF-AIA1
AIA1 CouchDB
Logs(SI1)
SI1:{ SA11:V11 SA12:V12 }
AIA1 Chaincode
add_subject(SI1)
AIA1 Server
AIA1 Application
Enters values of subject attributes SA11,SA12 of Subject S1 into application SI1 is identifier of Subject S1 in AIA1
AIA1 Administrator
HF-AIA2
AIA2 CouchDB
Logs(SI2)
SI2:{ SA21:V21 SA22:V22 }
AIA2 Chaincode
add_subject(SI2)
AIA2 Server
AIA2 Application
Enters values of subject attributes SA21,SA22 of Subject S1 into application SI2 is identifier of Subject S1 in AIA2
AIA2 Administrator
HF-AIAn
AIAn CouchDB
Logs(SIn)
SIn:{ SAn1:Vn1 SAn2:Vn2 }
AIAn Chaincode
add_subject(SIn)
AIA2 Server
AIA2 Application
Enters values of subject attributes SAn1,SAn2 of Subject S1 into application SIn is identifier of Subject S1 in AIA2
AIAn Administrator
n Attribute Issuing Authorities for subject

Fig. 2. Issuing Attributes to Entities (Subjects/Objects)



R1:{ SA11:V11 SA12:V12 SA21:V21 SA22:V22 .... SAn1:Vn1 SAn2:Vn2 OA1:V1 OA2:V2 }
Enters global rule in terms of attribute-value pairs into application
calls divide_into_subrules(R1) function in chaincode which returns array of subrules for each authority
Object Administrator
Object Application
Object Server
Object Chaincode
SRAIA1:{ R1:{ SA11:V11 SA12:V12 } }
SRAIA2:{ R1:{ SA21:V21 SA22:V22 } }
SRAIAn:{ R1:{ SAn1:Vn1 SAn2:Vn2 } }
SROA:{ R1:{ OA1:V1 OA2:V2 } }

Fig. 3. Splitting a Global Rule into Sub Rules

Fabric network.

C. Processing Access Requests

1) Splitting Global Access Request into Sub Access Requests: When a subject requests access for an object, he/she will present his/her identifiers in each of the subject Attribute Issuing Authorities (AIA's) and the identifier of the object for which access is being requested. As shown in Figure 4, subject (S_1) enters these identifiers (SI_1, SI_2, ...,SI_n, OI_1) as an access request (AR_1) into the *Object Application*. *Object Server* will call the function *divide_into_subreq* in *Object Chaincode*. This function divides the global access request into subrequests (SAR_{AIA_1}, SAR_{AIA_2}, ..., SAR_{AIA_n}, SAR_{OA}) based on the information of which identifier is being verified by which authority. Again, the server receives this information in the form of configuration files.

2) Processing of Sub Access Request by all Authorities: Once *Object server* receives the subrequests that need to be processed by each authority, it will send the subrequests corresponding to each authority to their corresponding servers. The function *proc_subreq* in their chaincode will subsequently be called by each authority server. Subrequest processing is done by this function. All authorities process sub requests by retrieving attributes from their CouchDB database and determining whether or not the subrules stored in their HF networks give access to those attributes. Once each authority server receives the result of subrequest, they return the results back to *Object Server*.

3) Combining Sub Request Results from all Authorities: When *Object Server* receives all the subrequest results from all authorities, it will call the *Object Chaincode* function *proc_sub_results*. This function basically processes all sub

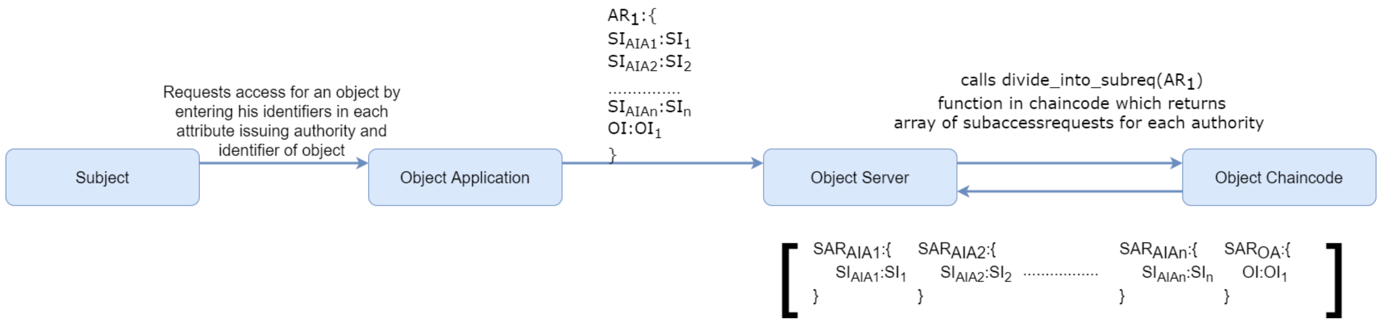

Requests access for an object by entering his identifiers in each attribute issuing authority and identifier of object
AR1:{ SIAIA1:SI1 SIAIA2:SI2 .... SIAIAn:SIn OI:OI1 }
calls divide_into_subreq(AR1) function in chaincode which returns array of subaccessrequests for each authority
Subject
Object Application
Object Server
Object Chaincode
[SARAIA1:{ SIAIA1:SI1 } SARAIA2:{ SIAIA2:SI2 } .... SARAIAn:{ SIAIAn:SIn } SAROA:{ OI:OI1 }]

Fig. 4. Dividing a Global Access Request into Sub Access Requests

TABLE I
EXECUTION TIME OF ALL FUNCTIONS

S.No	Operation	Time(secs)
1	Adding Attributes/SubRules to Blockchain	2.15
2	Splitting GlobalRules/Requests into SubRules/requests	0.028
3	Any Server to Server Interaction	0.25
4	Processing of Subrequests	0.056
5	Processing sub access results	0.011

request results and returns whether access is granted or not. If all sub request results grant access then access will be granted or else access will be denied. One must note that logs of sub request results and final result are all stored as transactions in the respective Hyperledger Fabric networks making this process immutable and safe.

IV. Implementation and Results

We have implemented three templates for automatically generating the chaincode and server application. The template files for the application, server, chaincode are written in HTML, Nodejs and Javascript respectively. To use our framework, one needs to create configuration files for each of the participating authority. There are 2 different configuration files and template files. One is generalized for any subject Attribute Issuing Authority(AIA) and the other is for Object Authority. A javascript code in our framework generates these configuration files in json format and they are loaded into template files to generate application code, server code and the chaincode for all authorites. Object Authority's configuration file contains information about attributes issued by each authority which will be used by Object Server to split global rules/requests into sub rules/requests.

We study the effectiveness of the framework by varying the following parameters namely (i) number of AIA's involved in the framework, (ii) number of attributes issued by each authority (AIA or Object Authority), (iii) number of rules designed by Object authority.

Table I shows the execution time of different operations involved in the framework with 3 AIA's, each issuing 50 attributes and 50 rules being designed by Object Authority.

Operations 1, 2, 3 take an almost constant amount of time irrespective of the variation in the 3 parameters, whereas the

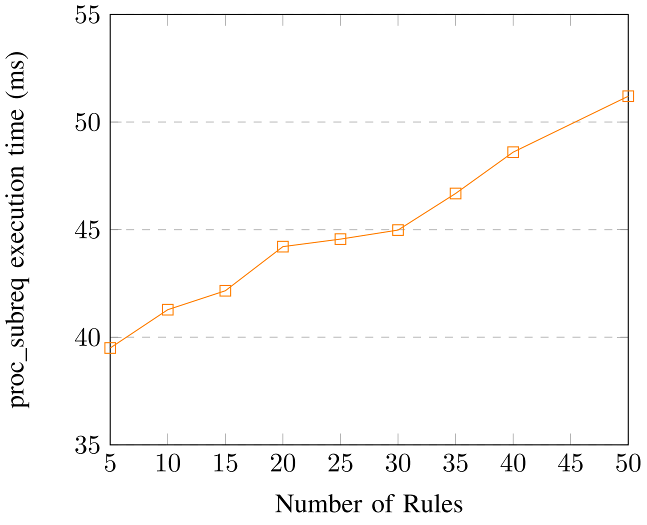


Fig. 5. SR Processing time Vs No. of Rules

time required to perform 4, 5 operations is observed to vary with the change in the parameters. The time it takes to process sub requests varies depending on the number of rules and attributes issued by that authority. The time it takes to process sub access results varies depending on the number of rules and AIAs involved.

Execution times of *proc_subreq* and *proc_sub_results* chaincode functions are plotted in Figure 5 and Figure 6 as the number of rules designed by Object Authority is varied from 0 and 50. During this process, we have fixed the number of AIA's to be 3 and the number of attributes to be issued by each authority to be 5. Execution time of sub request processing increases as number of rules designed by Object Authority is increased as more number of rules needs to be processed. As each sub access result is of the size of the number of rules, execution time of merging or processing sub access results also increases as number of rules designed by Object Authority is increased.

One has to note that object server sending subrequests to each authority's server is a sequential process i.e. object server will send subrequests and get subresults authority after authority i.e the object server will first send the subrequest that

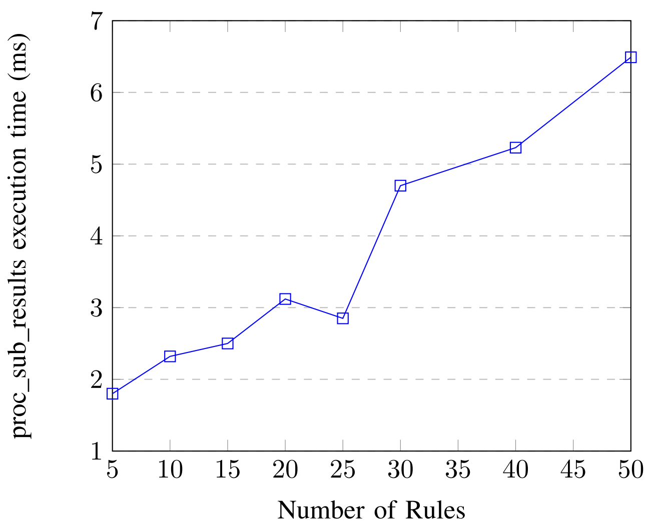


Fig. 6. Merging time Vs No. of Rules

needs to be processed by AIA_1 to AIA_1. AIA_1 will process subrequest and send the result back to object server. After that only, object server will send subrequest that needs to be processed by AIA_2 to AIA_2 and this process continues for all authorities. In case of subrules as well, once AIA_1 finished adding its subrule into its HF-network, then only object server will send the subrule to AIA_2.

V. Conclusion and Future Directions

In this paper, we proposed a decentralized framework using Hyperledger Fabric and Attribute Based Access Control to verify access requests from multiple authorities in a decentralized manner. Hyperledger Fabric enables transparent authorization and immutability of credentials and ensures a secure operation between authorities. Each authority can safely store the credentials in their own Hyperleger Fabric network. The framework is designed to support decentralized authorization across multiple attribute issuing authorities. We implemented a prototype of the framework using HyperLedger Fabric and the results validate the performance of the proposed approach. We plan to enhance this model by parallelizing all the server to server interactions to further reduce the access request time and the time taken for adding rules. We also plan to extend the framework to supporting multiple Object attribute issuing authorities.

Acknowledgments

Balaji Palanisamy acknowledges the partial support under Grant #2020071 from the US National Science Foundation (NSF) SaTC program. The work of Shamik Sural is partially supported by CISCO University Research Program Fund, Silicon Valley Community Foundation award number 2020-220329 (3696). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding sources.

References

[1] R. S. Sandhu, E. J. Coyne, H. L. Feinstein and C. E. Youman, ”Role-based access control models” in Computer, vol. 29, no. 2, pp. 38-47, Feb. 1996, doi: 10.1109/2.485845.

[2] B. B. Gupta and Megha Quamara, ”An identity based access control and mutual authentication framework for distributed cloud computing services in IoT environment using smart cards”, Procedia Computer Science, vol. 132, pp. 189-197, 2018, doi: 10.1016/j.procs.2018.05.185.

[3] R. S. Sandhu, ”Lattice-based access control models” in Computer, vol. 26, no. 11, pp. 9-19, Nov. 1993, doi: 10.1109/2.241422.

[4] Vincent C. Hu, David Ferraiolo, Rick Kuhn, A. Schnitzer, Kenneth Sandlin, R. Miller and Karen Scarfone, ”Guide to attribute based access control (ABAC) definition and considerations”, National Institute of Standards and Technology Special Publication, pp. 162-800, 2014.

[5] H. Shen and F. Hong, ”An Attribute-Based Access Control Model for Web Services”, 2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT’06), pp. 74-79, 2006, doi: 10.1109/PDCAT.2006.28.

[6] https://hyperledger-fabric.readthedocs.io/en/release-2.2/

[7] A. Azaria, A. Ekblaw, T. Vieira and A. Lippman, ”MedRec: Using Blockchain for Medical Data Access and Permission Management”, 2016 2nd International Conference on Open and Big Data (OBD), pp. 25-30, 2016, doi: 10.1109/OBD.2016.11.

[8] Gaby G. Dagher, Jordan Mohler, Matea Milojkovic and Praneeth Babu Marella, ”Ancile: Privacy-preserving Framework for Access Control and Interoperability of Electronic Health Records Using Blockchain Technology”, Sustainable Cities and Society, vol. 39, pp. 283-297, 2018, doi: 10.1016/j.scs.2018.02.014.

[9] O. J. A. Pinno, A. R. A. Gregio and L. C. E. De Bona, ”ControlChain: Blockchain as a Central Enabler for Access Control Authorizations in the IoT”, GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pp. 1-6, 2017, doi: 10.1109/GLOCOM.2017.8254521.

[10] M. Samaniego and R. Deters, ”Internet of Smart Things - IoST: Using Blockchain and CLIPS to Make Things Autonomous”, 2017 IEEE International Conference on Cognitive Computing (ICCC), pp. 9-16, 2017, doi: 10.1109/IEEE.ICCC.2017.9.

[11] T. Bocek, B. B. Rodrigues, T. Strasser and B. Stiller, ”Blockchains everywhere - a use-case of blockchains in the pharma supply-chain”, 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 772-777, 2017, doi: 10.23919/INM.2017.7987376.

[12] S. Chen, R. Shi, Z. Ren, J. Yan, Y. Shi and J. Zhang, ”A Blockchain-Based Supply Chain Quality Management Framework”, 2017 IEEE 14th International Conference on e-Business Engineering (ICEBE), pp. 172-176, 2017, doi: 10.1109/ICEBE.2017.34.

[13] Kari Korpela, Jukka Hallikas and Tomi Dahlberg, ”Digital Supply Chain Transformation toward Blockchain Integration”, Hawaii International Conference on System Sciences (HICSS), 2017, doi: 10.24251/HICSS.2017.506.

[14] Sara Rouhani and Ralph Deters, ”Blockchain based access control systems: State of the art and challenges”, 2019 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2019, Thessaloniki, Greece, ACM, pp. 423-428, 2019, doi:10.1145/3350546.3352561.

[15] Damiano Maesa, Paolo Mori and Laura Ricci, ”Blockchain Based Access Control”, IFIP International Conference on Distributed Applications and Interoperable Systems(DAIS), pp. 206-220, 2017, doi:10.1007/978-3-319-59665-5_15.

[16] D. Di Francesco Maesa, P. Mori and L. Ricci, ”Blockchain Based Access Control Services”, 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1379-1386, 2018, doi: 10.1109/Cybermatics_2018.2018.00237.

[17] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang and J. Wan, ”Smart Contract-Based Access Control for the Internet of Things” in IEEE Internet of Things Journal, vol. 6, no. 2, pp. 1594-1605, April 2019, doi: 10.1109/JIOT.2018.2847705.

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

[19] M. Varun, M. V. A. Vasishta , B. Palanisamy and S. Sural, ”Decentralized Authorization in Web Services Using Public Blockchain”, in Lee K., Zhang LJ. (eds) Blockchain – ICBC 2021. Lecture Notes in Computer Science, vol 12991. Springer, Cham, doi: 10.1007/978-3-030-96527-3_3