Attribute-Based Access Control Policy Review in Permissioned Blockchain

Sherifdeen Lawal(✉) and Ram Krishnan(✉)

University of Texas at San Antonio, San Antonio, TX 78249, USA
{sherifdeen.lawal,ram.krishnan}@utsa.edu

Abstract. Permissioned blockchain is of a great deal to enterprise uses cases. There is a need to support access control policy review for legal and security reasons in some use cases. Specifying and maintaining a complex access policy for a permissioned blockchain may be well managed using attributes. The ABAC policy approaches offer a solution to a peculiar set of challenges for distributed system access control, like the blockchain. There are studies on leveraging Smart Contracts in implementing blockchain-based ABAC policy. However, most of these contributions implement an Attribute-Based Access Control policy expressed in a logical format. We proposed the ABAC enumerated policy format as an access control mechanism for the permissioned blockchain, Hyperledger Fabric network. We also proposed an algorithm for a set of policy review problems and implemented the algorithm for a blockchain-based policy specification.

Keywords: Attribute based access control · Policy review · Authorization · Revocation · Policy machine · Authorization graph

1 Introduction

The two general classes of a blockchain network are permissionless (public) and pemissioned (private) blockchain, from the context of network entity identity. The participants in a public blockchain network are anonymous. A federated or centralized authority grants access to the participants of a permissioned blockchain. The permission blockchain serves enterprise use cases where there is a need to verify the identity of their customers, such as financial transaction that requires the Know-Your-Customer (KYC) and Anti-Money Laundering (AML) regulations. This work aims at the study of revocation and authorization policy review in the Hyperledger Fabric blockchain network. Hyperledger Fabric is a pluggable, modularized, and open-source architecture for commercial-grade permissioned distributed ledger technology (DLT) platform.

The blockchain is not an exemption to the particular set of challenges for distributed system access control. It requires a unique set of concepts and considerations different from traditional systems. An important requirement is that distributed applications on multiple coordinated systems have permission to access

R. Krishnan et al. (Eds.): SKM 2021, CCIS 1549, pp. 97–109, 2022.
https://doi.org/10.1007/978-3-030-97532-6_6

the data for processing and controlling the access to the distributed processes and data from their local users. The Attribute-Based Access Control (ABAC) offers a solution to the discussed problem of access policy on the blockchain network. It uses dynamic attribute values for privilege assignment in a distributed system that requires federated or autonomous control. Applications in Hyperledger Fabric interact with the blockchain network by submitting a request to operate on the ledger. A logic-based ABAC policy controls the access of these applications on the Fabric network.

The access control model of a blockchain network is critical. Nevertheless, the ability to create and modify policy specifications without unintended consequence is of equal importance. The policy review in ABAC models that express policy using logic-based formula has the NP-complete time complexity. For instance, the evaluations for a given user attribute to access a particular resource. The work in [10] shows the lack of scalability by the logic-based ABAC model like the XACML in policy evaluation. An empirical study by Mell et al. demonstrates that the enumerated-based policy ABAC model of the NIST Next Generation Access Control (NGAC) is scalable [11].

This work proposes a modularized ABAC architecture of the Policy Machine as an on-chain mechanism to control access to the blockchain ledger. The Policy Machine is the foundation for the NIST Next Generation Access Control (NGAC) [8,9]. This work implemented the Policy Machine standard architectural components on the Hyperledger Fabric network. We applied our proposed algorithm to the authorization and revocation policy review problem of the Policy Machine. Through a set of Smart Contracts (chaincode), our implementation stores access policy information to the blockchain ledger. The protected resource ledger is a different ledger from the access policy information ledger. A low-level Hyperledger Fabric API enables the communication between the Smart Contracts deployed for the two types of blockchain ledgers.

In summary, the contributions of this paper include:

1. the NIST NGAC system architecture implemented for a blockchain network.
2. a proposed algorithm for the policy review of revocation and constrained authorization in a blockchain-based Policy Machine system.

The reminder of this paper is structured as follows. In Sect. 2, we touch on related work on this subject. Section 3 provides overview of the Hyperledger Fabric and Policy Machine framework. Section 4 details on policy review problem in Policy Machine. Section 5 describes the policy review algorithm to revoke and grant (with constraint) access. An implementation of policy machine in Hyperledger Fabric and evaluation of our policy review algorithm is in Sect. 6, and Sect. 7 concludes this work.

2 Related Work

In this section, we touch on previous research contributions that implement ABAC on a blockchain network. We also discuss the few contributions to ABAC

policy reviews in general. There is a handful of work on blockchain as an Attribute-Based Access Control system for different domains. Pinno et al. [1], Ding et al. [3], and Dukkipati et al. [4] study the implementation of blockchain-based ABAC in IoT systems. Zhang and Posland studied the blockchain authorization approach for Electronic Medical Records (EMRs) [5]. A granular authorization scheme for blocks and attribute values query was at the core of their research. Also, they lower the computational overhead for access decisions by eliminating the use of Public Key Infrastructure (PKI).

Few research works are out there on blockchain attribute-based access control for the general-purpose use case [2,6,7]. Previous studies [2,7] utilized the XACML to express access policies. We applied the Policy Machine, the NIST implementation of the attribute-based access control framework, an open-source project. The only generic implementation of attribute-based access control on Hyperledger Fabric blockchain network deployed ABAC components as smart contracts to control access to an off-chain system [2]. In contrast, we implemented Smart Contracts for access control to the blockchain ledger. This work includes the capability for review of authorization and revocation.

Mell et al. improve the efficiency of the existing functions that answer users' capability and object access entry queries [11]. Their contribution reduced the computational overhead of capability and access entry queries using an optimized graph search algorithm. We proposed an algorithm for the policy review questions not addressed by the NIST Policy Machine specification or any previous research work.

The analysis of ABAC policies through the category-based metamodel [12] addresses a similar set of policy review questions in the NIST Policy Machine specification. The policy review algorithm we proposed answers question not covered by the NIST Policy Machine or any previous research work.

3 Background

3.1 Hyperledger Fabric

Hyperledger Fabric core building blocks are the distributed ledger, different types of nodes, chaincode, channel, and Membership Service Provider (MSP).

Hyperledger Fabric blockchain ledgers are deployed on peer nodes to store assets. An asset is a representation of valuable items digitally stored on the blockchain network. Participants on the blockchain network can trade (transfer) assets. Hyperledger Fabric ledger has two components. The first component is the blockchain ledger that is an immutable sequence of transaction blocks. The second component is the state database that contains the current value of the key-value pairs created, modified, or deleted by transaction requests in the blockchain network. Blockchain transaction occurs when a client application invokes the programmable business logic (smart contract/chaincode) to read or write from the ledger.

The Hyperledger Fabric has three types of nodes - client, peer, and orderer nodes. The client node has an application that provides an interface for users to

invoke smart contracts by sending a transaction proposal to a peer node. The peer nodes are where the shared ledger resides and their installed chaincode mediated end-user read/write operations to the distributed ledger. The orderer nodes perform the ordering of transactions on a first-come-first-serve basis for the blockchain network. It distributes the ordered blocks to peer nodes. Hyperledger Fabric allows the integration of other implementation of the orderer service apart from the out-of-the-box Kafka and Raft varieties.

A smart contract is a code packaged as a chaincode in Hyperledger Fabric. It manages access and modifications to a set of key-value pairs in the state database when invoked by client applications external to the blockchain network. A channel is an isolated overlay of the blockchain network on the Hyperledger Fabric network that provides data privacy and confidentiality. Each channel has a ledger shared across the peers on the channel, and only participants authenticated to the specific channel can transact on such channel. The Membership Service Provider (MSP) governs the validity of credentials for a group of participants on the network. Transaction authentication and validation respectively by client and peer node requires identity credentials. There's a need to install chaincode on a channel before end-user invocation to read and write to the ledger through an application or client node CLI.

3.2 Policy Machine Basic Elements and Relations

Policy specification in Policy Machine has an annotated Directed Acyclic Graph (DAG) representation. The node of the Policy Machine authorization graph is the of Policy Elements (PE). The policy elements are the finite sets of Users (U), Objects (O), User Attributes (UA), Object Attributes (OA), and Policy Classes (PC). An assignment relations in Fig. 1 are unlabeled DAG edges between the ordered pair of a user to user attribute node, object to object attribute node, an attribute to an attribute of the same type, and user or object attribute node to the policy class node. Any outward unlabeled edge (assignment relation) from a source policy element must terminate at a policy class of an authorization graph.

An association relation in Fig. 1 is an annotated edge between user attributes and user attributes or object attributes. For example, the association edge (*Group Head*, $aars_i$, *Retail & Foreign Serv*) specifies that individuals with a sequence of assignments to the *Group Head* can execute the actions enabled through administrative access right set $aars_j$ on *Retail & Foreign Serv* and the policy element *Retail & Foreign Serv* contains. The association relation is partitioned into two as administrative (a-association) and resource (r-association) association.

An association is a relation represented by labeled (annotated) downward-arcing edge from a user attribute node to an attribute (user attribute or object attribute node). For example, in Fig. 1, the association triple (*Group Head*, $aars_i$, *Retail & Foreign Serv*) implies that a user who has a path to *Group Head* is authorized to perform operations enabled by $aars_i$ on *Retail & Foreign Serv* and policy element that has a sequence of assignment relation to *Retail & Foreign Serv*. An association grants access through a set of resource access rights

(i.e., r-association in the legend) or a set of administrative access rights (i.e., a-association in the legend). The policy elements and the relations constitute the authorization graph.

4 Policy Review Problem in Policy Machine

The Policy Machine authorization graph can grant access by creating assignment and/or association relations. Likewise, the deletion of assignment and/or association relations may revoke access. Given the hierarchical structure of the Policy Machine, for a lot of scenarios, we can grant or deny access in various ways. However, a subset of the possible ways of allowing or denying access may contradict another policy. Also, some of the approaches of granting or denying access may have unintended authorization or revocation. The proposed algorithm of this paper generates a comprehensive list with the combination of relations to delete or create for revoking or authorizing access requests, respectively. The algorithm result provides the Policy Machine administrator guidance on the approach for access authorization or revocation.

We demonstrated in the coming example how the number of approaches to grant access explodes and how utilizing constraints can limit the authorization approaches.

Example: Figure 1 shows the authorization graph for a financial institution with the policy class called *BankOp Access*. The task 'trans-T' requires two related ordered administrative operations with the permissions granted through access rights $aars_q$ and $aars_p$ on *Wire Trans Serv* and *ATM & POS Serv*, respectively. Two employees (*Alice* and *Bob*) of the financial institution each have a different subset of authorities granted to *Cathy* to complete the task 'trans-T'.

In another task, 'T-1' an officer in this financial institution with the attributes *ATM Custodian* and *Trans Serv Supervisor* needs to assign a member of the *Backup Officer* role to *ATM Custodian* for the completion of the task 'T-1'. *Cathy* has no permission to assign a *Backup Officer* to the *ATM Custodian* role in the current transition state of the authorization graph of Fig. 1. Let's assume the employees (i.e., *Jane* and *Paul*) in this example with permissions enabled by the administrative access right set, $aars_i$, can authorize *Cathy*'s requested access. Here are approaches that will allow the assignment of a *Backup Officer* to the *ATM Custodian* role by *Cathy*:

1. *Creating association:* Using an association only and assuming a label (access right) $aars_k$ grants the permission *Cathy* is seeking, an association relation from *ATM Custodian*, or *Trans Serv Supervisor*, or *Op Officers* to *Backup Officer*, or *Op Officers* will authorize *Cathy*'s request. There are six possible relations to allow *Cathy*'s request using association relation.
2. *Creating user attribute assignment:* Users with permissions from the access right set $aars_i$ can create an assignment to authorize *Cathy*'s request. This assignment is from user attribute nodes that are descendants of the user

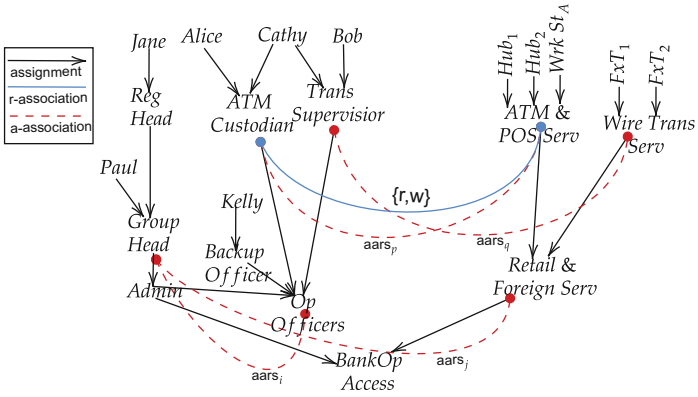

Fig. 1. Policy machine authorization graph

node, *Cathy*, to the ancestor user attribute nodes of *Group Head* and *Group Head*. For this approach of authorization, the user attribute assignments that conform with the DAG definition of the authorization graph are the relations from *ATM Custodian* or *Trans Serv Supervisor* to *Group Head*or *Regional Head*.

3. *Creating user assignment:* Any user with the permissions from the access right set $aars_i$ can use the user assignment operation to create an assignment of the *Cathy* user node to either of the user attribute nodes *Group Head* or *Regional Head*.

This illusive example above considers only single operation approaches to authorize *Cathy*'s request, to keep it simple. Overall, there are twelve different approaches of creating assignment or association relation to allow *Cathy* to complete task 'T-1'. The only caveat is that only two of these twelves ways of granting access to *Cathy* do not violate the constraint on the task 'trans-T'. Authorization of *Cathy*'s request through the approaches enumerated in (1) and (2) leaves the room for *Alice* and *Bob* to collude on the sensitive task 'trans-T'.

In addition, this example is by no means a comparison of the more complicated issues in an enterprise scenario. Note that the structure of the Policy Machine authorization graph permits granting access using any non-redundant combination of the three operations. For instance, using the preceding example, permissions enabled by the access right set $aars_i$ allows creating two assignment operations to authorize *Cathy*'s request. The sequence of operation may be a user assignment of the user node *Cathy* to user attribute node *Backup Officer* and the *Backup Officer* to a user attribute node granted the permissions of the access right set $aars_i$.

Observation
A Principal Authority (PA) is a mandatory preexisting user in the Policy Machine, also called the root user. The PA is responsible for the creation and control of the Policy Machine policies in their entirety. S/he fundamentally holds the universal authority to perform all the actions within the Policy Machine framework.

Apart from the permissions for creating and deleting policy classes and attribute assignments to policy classes, the PA may delegate access rights to a domain or sub-ordinate administrators. Note that the deployed system represented by the authorization graph of Fig. 1 does not include the Principle Authority. The Principal Authority of the authorization graph created the policy elements and relations as shown in the figure. Authorities delegated to the Group Head user attribute will suffice for creating new policy elements and assignment and association relations.

5 Policy Review Algorithm

We now provide our graph algorithm to answer these two questions on a given request (*user*, *op*, *resource*).

1. If a *user* is allowed to perform *op* on *resource*, what are the approaches to deny the *user* access to perform *op* on the protected *resource*?
2. If a *user* is not authorized to perform *op* operation on a *resource*, What are the approaches to grant the *op* on protected *resource* to the *user*?

5.1 Derived Functions

To generate approaches to revoke or authorize a given request in a policy graph, we utilize the following derived functions in creating groups of attributes in the preceding subsection. A combination of elements from these groups of attributes enables the creation of relation(s) as an approach to authorizing a denied access. Similarly, the deletion of the relation(s) created through elements of attribute groups is an approach to revoke authorized access.

- $\texttt{tail} : \texttt{ASSOCIATION} \longrightarrow \texttt{UA}$: is a function that maps an edge, association relation, $(ua_i, ars_j, at_k) \in \texttt{ASSOCIATION}$ to the (user attribute) node $ua_i \in \texttt{UA}$ it originates.
- $\texttt{head} : \texttt{ASSOCIATION} \longrightarrow \texttt{AT}$: is a function that maps an edge, association relation, $(ua_i, ars_j, at_k) \in \texttt{ASSOCIATION}$ to the (user/object attribute) node $at_k \in \texttt{AT}$ it terminates. Where $\texttt{AT} = \texttt{UA} \cup \texttt{OA}$
- $\texttt{anc}: \texttt{PE} \longrightarrow 2^{\texttt{PE}}$: is the mapping from a policy element to the set of policy elements that is an ancestor to the policy element.
- $\texttt{des}: \texttt{PE} \longrightarrow 2^{\texttt{PE}}$: is the mapping from a policy element to the set of policy elements that is a descendant to the policy element.
- $\texttt{PE}_{i_{\texttt{func}}} = \{\texttt{node} \mid (\exists pe_j \in \texttt{PE}_i)[\texttt{node} \in \texttt{func}(pe_j)]\}$: is the set of all policy elements returned by func for the set $\texttt{PE}_i$, where func is anc or des.

5.2 Groups of Attribute Enabling Authorization and Revocation

The following defined sets of (user and object) attribute groups form the basis of our algorithm for the policy review of access authorization and revocation. We derived the attribute groups considering the resource a user wants to perform an action on is of type user or object. When the resource in question is a user or user attribute, the following user attribute groups create relations that authorize and revoke access requests.

- $UA_1 = \{\texttt{ua} \mid \texttt{ua} = \texttt{tail}((ua_i, ars_j, at_k)) \vee \texttt{ua} \in \texttt{anc}(\texttt{tail}((ua_i, ars_j, at_k)))\}$
- $UA_2 = \{\texttt{ua} \mid \texttt{ua} \in \texttt{anc}(\texttt{head}((ua_i, ars_j, at_k))) \wedge \texttt{ua} \in \texttt{des}(user)\}$
- $UA_3 = \{\texttt{ua} \mid \texttt{ua} \in \texttt{anc}(\texttt{head}((ua_i, ars_j, at_k)))\ ,\ \texttt{ua} \notin UA_{2_{\text{anc}}},\ \texttt{ua} \notin UA_2,\ \texttt{ua} \notin UA_{1_{\text{des}}},\ \texttt{ua} \notin UA_1\}$
- $UA_4 = \{\texttt{ua} \mid \texttt{ua} \in \texttt{des}(user) \wedge \texttt{des}(resource)\ \}$

Where (ua_i, ars_j, at_k) is an association relation for authorizing *user* to operate on *resource*

Assuming we want to grant or deny access to an object or object attribute. Combining the sets UA_1, UA_2, UA_3, above and the following object attribute groups enable the creation of relations that authorize or revoke access.

- $OA_1 = \{oa \mid oa \in \texttt{anc}(\texttt{head}((ua_p, ars_q, ao_r))),\ oa \notin \texttt{des}(resource),\ oa \neq resource\}$
- $OA_2 = \{oa \mid (oa \in \texttt{des}(resource) \wedge oa \notin OA_{1_{\text{des}}}) \vee oa = resource\}$
- $OA_3 = \{oa \mid (oa \in \texttt{anc}(\texttt{head}((ua_p, ars_q, ao_r))),\ oa \in \texttt{des}(resource),\ oa \in OA_{2_{\text{des}}}\ ) \vee oa = \texttt{head}((ua_p, ars_q, ao_r))\ \}$

This scenario requires two association relations. The association (ua_i, ars_j, at_k) grants authority to create or delete the relation(s) from attribute(s) of the user to whom we want to authorize/deny access. The second association (ua_p, ars_q, ao_r) allows the creation or deletion of relation(s) to the requested resource (object or object attribute).

5.3 Revocation and Constrained Authorization Methodology

Our policy review algorithm generates approaches to revoke and authorize access to Policy Machine protected resource. The relation(s) created or removed amongst group attributes (authorization/revocation enablers) provides approaches to allow and revoke access.

As input, the algorithm takes a request (*user*, *op*, *resource*), a graph associated with the request, and a record (*authmode*) with fields of key-value pair. Firstly, if there is an association relation (policy) that grants the user the authority to perform op on the resource, the algorithm generates approaches to revoke the access. Otherwise, it produces possible relation(s) that allow the user access to perform op on the resource. The key-value pairs from the input record allow a policy administrator to specify modes of authorization. The algorithm can generate all possible approaches with/without constraint to authorize a request. A key

Table 1. Scope of authorization/revocation on attribute groups

Attribute groups	Pattern of relations(s)	Authorization effect
UA_1	Assignment: to	No effect
UA_2	Assignment: from Association: from	Access granted or inherited
UA_3	Assignment: from and to Association: from	Access granted or inherited
UA_4	Association: to	No effect
OA_1	Association: to	No effect
OA_2	Association: to Assignment: from	No effect Access entry granted
OA_3	Association: to	No effect

isGeneric with a boolean value of true generates all approaches without restriction, while the value of false produces constrained authorization approaches. When *isGeneric* is true the two other key-value pair in the record becomes null. Another key is the *denySet*, and the value is a set that authorization granted or inherited by its elements is constrained. Its value is a user attribute set.

Let's examine the scope of access granted through the revocation/ authorization enablers attribute sets. While authorizing a request, access is granted or inherited by some attribute groups. The table summarizes the pattern of relation(s) created using these attribute groups and the change in capability or access entry of these attributes after an authorization.

The column pattern of relation created (i.e., <relation/edge type> : <direction>) in the table describe the type of edge(s) we can create from or to elements of a given access enabling attribute set. As an example when the resource is a user or user attribute, a possible approach to authorize access is creating an edge (assignment) from $\texttt{ua}_2$ to $\texttt{ua}_1$ or creating an assignment from $\texttt{ua}_2$ to $\texttt{ua}_3$ and creating an association from $\texttt{ua}_3$ to $\texttt{ua}_4$, where $\texttt{ua}_i \in UA_i$. The third column of the table signifies the change in capability or access entry of a user or an object attribute respectively. Authorizing a request elevate the capability of the user attributes UA_2 and UA_3, and access entry of object attribute OA_3. A policy administrator can constrain the authorization of a request through these attributes with elevated capability or access entry.

For example, if the key *denySet* has a value UA_2, the algorithm excludes all relations(s) that authorize access through the user attribute set UA_2. An attribute set with elevated capability or access entry that is not the value of the *denySet* key is also constrained through the third key *limitto*. The value for the key *limitto* specifies the number of elements used to generate approaches to authorize access. Its value is a user attribute set if the resource is a user or user attribute and an object attribute set for an object or object attribute resource.

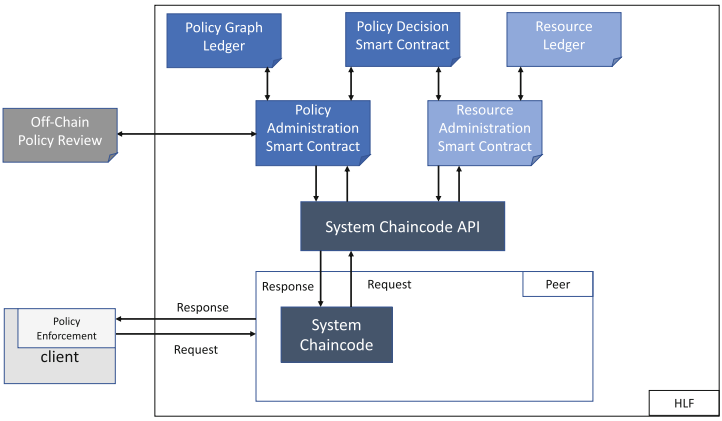

Fig. 2. Blockchain access control system architecture for policy machine.

Assuming request = (user, op, resource), Graph = (PE, ASSIGN, ASSOCIATION), *authmode* = {*isGeneric* : false, *denySet* : UA_2, *limitto* : 1} are input parameters for the algorithm, the output is a set of constrained approaches of authorizing *user* request. It excludes authorization approaches using elements in the user attribute set UA_2. The value of *limitto* permits creating authorization approaches using one element of UA_3 or OA_3 if the resource is a user or an object type, respectively.

6 System Implementation and Evaluation

In this section, we discuss the implementation of the policy machine reference architecture in hyperledger fabric. We used Hyperledger Caliper and simulated policy graph to measure the performance of our policy review algorithm.

6.1 Blockchain Implementation of Policy Machine

Section 3 discussed the sets of policy elements and relations of the Policy Machine referenced in this work. The functional architectural components (see Fig. 2) of the Policy Machine we implemented as smart contracts in the Hyperledger Fabric network are the Policy Administration Point (PAP), Policy Decision Point (PDP), Resource Access Point (RAP). Also, the two databases of the Policy Machine standard architectural components, Policy Information Point and Resource Repository were implemented as the Policy Information Ledger and Resource Ledger respectively. The Policy Administration Smart Contract is an access mediator and manages the create, read, update and delete requests to the Policy Information Ledger. Administrative and resource users' requests are separately received by the Policy Administration and Resource Access Smart Contracts. The Policy Administration and Resource Access Smart Contracts

forwards intercepted access requests to the Policy Decision Smart Contract that has the logic for allowed and denied access requests.

The client modes are the Policy Enforcement Point (PEP) that imposed the access decision returned by the Policy Decision Smart Contract and responds to the user with the proper result. The Event Process Point (EPP) in the Policy Machine triggers obligations, which is outside the scope of this work. As documented in the Hyperledger Fabric developer guide, an iterative process like the policy review degrades the performance of the blockchain network. We implemented an interface for the policy review as an off-chain component.

We leverage the invokeChain Application Programming Interface (API) for the request and response between Smart Contracts (chaincode) for the different ledgers. Assuming the Smart Contracts deployed belong to the same channel, as an example. An application user receives access decisions through the Resource Access Smart Contract. This Smart Contract makes a local call through the invokeChaincode API to the Policy Decision Smart Contract for access request decisions. This implementation considers only chaincode invocation from another chaincode when on the same channel. Recall that the Policy Information Ledger preserves the abstract representation of the policy element and relations for the Resource Ledger. To maintain consistency between the two ledgers, the Policy Decision Smart Contract needs read and write access to the two ledgers. In a network configuration that the Policy Decision Smart Contract is on a different channel with the two ledgers, any (delete/create) modification request will not reflect in the blockchain ledgers.

6.2 Performance Evaluation

We present the details of our experiments carried out for the system evaluation. The experiments were in two steps, an on-chain that reads the Policy Information Ledger and an off-chain policy review analysis. An iterative process in the policy review algorithm will degrade the blockchain network performance if deployed to the network. We used a virtual machine configured with 2 CPUs, 10 GB memory, an Ubuntu Linux OS 16.04 LTS, and Hyperledger Fabric version 2.2. Our Fabric network for this experiment has a single Raft orderer node, two peer nodes on the same channel, and a LevelDB database.

We created a policy graph generator script that simulates the creation of policy elements to the Policy Information Ledger. The policy graph comprises a policy class, 300 user and object attributes, and 200 users and objects. The Hyperledger Caliper version 0.4.2 enables us to generate workloads for the read policy graph transaction into our configured Fabric network. Hyperledger Caliper is a performance benchmark framework that provides different blockchains a suite of performance evaluation outcomes. To test the performance of our algorithm another script reads the policy graph ledger, simulates requests for authorization and revocation, and sets values for authorization mode record. The graph in Fig. 3 shows the average latency for reading the policy graph using the Caliper. Also, on the same graph, the average response time to generate revocation and

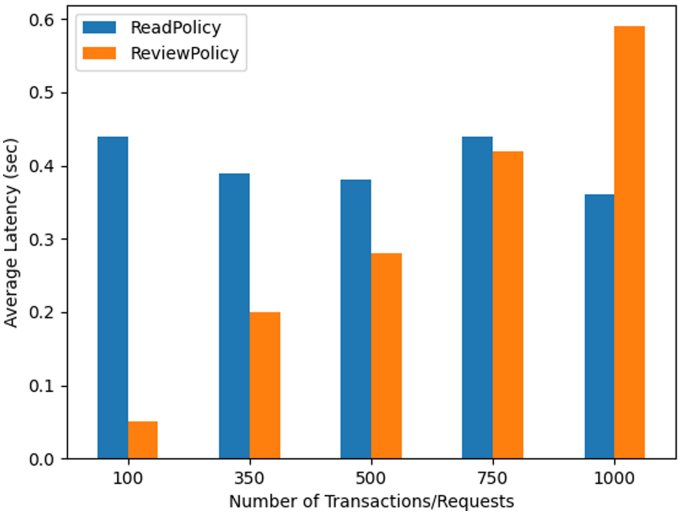

Fig. 3. Average latency for number of transactions and access requests

constrained authorization approaches for the request sizes are shown. The *policyRead* average latency varies in the range of 0.36 to 0.44 s for the number of transactions. The average response time of the *policyReview* increases as the number of requests for revocation and constrained authorization increases.

7 Conclusion

The Policy Machine is a promising alternative for logically expressed attribute-based policies with the prohibitive computational overhead in the policy review. It is feasible to perform policy reviews or queries using Policy Machine implemented access control for a permissioned blockchain. Apart from Policy Machine implemented in Hyperledger Fabric, we implemented our proposed algorithm that reviews authorization and revocation of access policy. Through the illustrative example, our proposed algorithm can help an administrator from granting access inadvertently. Our experimental results presented the evaluation of reading the Policy Information Ledger on the Hyperledger Fabric network and the response time for a policy review of various request sizes.

Acknowledgements. This work is partially supported by NSF grants HRD-1736209 and CNS-1553696.

References

1. Pinno, O., Gregio, A., De Bona, L.: ControlChain: a new stage on the IoT access control authorization. Concurr. Comput. **32**(12) (2020)

2. Rouhani, S., Belchior, R., Cruz, R., Deters, R.: Distributed attribute-based access control system using permissioned blockchain. World Wide Web (Bussum) **24**, 1617–1644 (2021)
3. Ding, S., Cao, J., Li, C., Fan, K., Li, H.: A novel attribute-based access control scheme using blockchain for IoT. IEEE Access **7**, 38431–38441 (2019)
4. Dukkipati, C., Zhang, Y., Cheng, L.: Decentralized, blockchain based access control framework for the heterogeneous Internet of Things. In: Proceedings of the Third ACM Workshop on Attribute-Based Access Control, pp. 6–69 (2018)
5. Zhang, X., Poslad, S.: Blockchain support for flexible queries with granular access control to electronic medicalrecords (EMR). IEEE International Conference on Communications (ICC) 2018, pp. 1–6 (2018)
6. Guo, H., Meamari, E., Shen, C.: Multi-authority attribute-based access control with smart contract. In: Proceedings of the 2019 International Conference on Blockchain Technology, pp. 6–11 (2019)
7. Di Francesco Maesa, D., Mori, P., Ricci, L.: A blockchain based approach for the definition of auditable access control systems. Comput. Secur. **84**, 93–119 (2019)
8. Ferraiolo, D., Gavrila, S., Jansen, W.: Policy machine: features, architecture, and specification. National Institute of Standards and Technology Internal Report 7987(2014)
9. Ferraiolo, D., Atluri, V., Gavrila, S.: The policy machine: a novel architecture and framework for access control policy specification and enforcement. J. Syst. Archit. **57**(4), 412–424 (2011)
10. Biswas, P., Sandhu, R., Krishnan, R.: Label-based access control: an ABAC model with enumerated authorization policy. In: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control. ACM Press (2016)
11. Mell, P. James, S., Harang, R., Gavrila, S.: Restricting insider access through efficient implementation of multi-policy access control systems. In: Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats (MIST 2016). ACM, New York, pp. 13–22 (2016)
12. Fernandez, M., Mackie, I., Thuraisingham, B.: Specification and analysis of ABAC policies via the category-based metamodel. In: Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (CODASPY 2019). Association for Computing Machinery, New York, pp. 173–184 (2019)