Blockchain-Based Smart Contract Access Control System

Weiqi Dai*‡§, Chenlong Wang*†, Changze Cui*†, Hai Jin*†, Xinqiao Lv*†
* National Engineering Research Center for Big Data Technology and System, Cluster and Grid Computing Lab
Services Computing Technology and System Lab, Big Data Security Engineering Research Center
† School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P.R. China
‡ School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, 430074, P.R. China
§ Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen, 518057, P.R. China
Email: {wqdai, wangchenlong, changzecui, hjin, xqlv}@hust.edu.cn

***Abstract*—Research has shown that smart contracts have become a significant and promising technology. However, the lack of mature and stable security mechanism, such as access control, makes smart contracts quite vulnerable. To mitigate this problem, we present an attributed-based access control system based on blockchain by building a hierarchical management mechanism of contract administrator and contract owner. After combining blockchain inherent synchronization function with our designed system-level smart contract, all peers can easily get access control rules timely. Moreover, we modify the original smart contract finite state machine to prevent malicious modification on the rules through system contract. In our evaluation, the system only introduces 2%-5% additional cost with assured security on Hyperledger.**

***Index Terms*—blockchain, smart contract, access control, hyperledger**

I. Introduction

Blockchain is an epoch-making technology, providing trust even among unfamiliar people by maintaining a decentralized and transparent distributed ledger. Until now, there are two epochs in blockchain history. The first-generation named blockchain 1.0 is represented by Bitcoin, establishing a vital infrastructure for secure cryptocurrencies trading. The second-generation named blockchain 2.0 represented by Ethereum can satisfy various practical requirements by supporting Turing-complete smart contracts, which enriches the functionality of blockchain largely [1]. Besides Bitcoin and Ethereum, there are also many other blockchain systems and Hyperledger fabric is one of them. Hyperledger fabric is a fascinating system supporting smart contracts, which is programmed in Golang and executed in docker containers.

Although blockchain has strong potential to address tricky traditional problems [2], a series of security incidents reveal that blockchain still faces serious security risks and challenges [3]–[5]. Nicola Atzei et al. [6] provided a review of security vulnerabilities in Ethereum smart contract, including issues related to access control between contracts. Marino et al. [7] proposed a set of criteria to allow smart contracts to be changed or terminated. Bartoletti et al. [8] identified seven programming modes in smart contracts to reduce unnecessary expensive operations. Luu et al. [9] offered a static analysis tool to detect contract bugs and highlighted the importance of checking the callee's return value. In this paper, we will focus on fabric contract security. Specifically, when a smart contract calls another contract, the call will be considered successful if the caller does not handle the exception returned by the callee. Thus, all state changes including abnormal states in sub-call will be finally submitted to the blockchain. Moreover, Hyperledger does not record those calls between smart contracts, which can help attackers launch this attack without being traced. This situation will certainly bring contracts liable to be attacked.

To solve this problem, access control may be a good solution, which is usually used to regulate the access to critical or valuable resources by defining access control policies including the condition setting for access time and context. For attribute-based access control [10], the policy consists of attributes that describe the characteristics of the subject, resource, and environment involved in the access request. In addition, a third party is needed to make a decision about whether allowing access based on the policy, but it may be malicious. However, blockchain can address this problem, in which the authority can be granted to the resource owner or other peers in blockchain. Since any action is transparent and needs consensus, so they are unable to perpetrate. There are some works leveraging blockchain as a trusted third party. For example, Aafaf Ouaddah et al. use blockchain as a storage database for the RBAC strategy [11] and complete the evaluation of the policy evaluation by the smart contract. Azaria et al. linked patient IDs, contract addresses, data pointers, access rights, and provider IDs through three different contracts [12]. Patient record providers can modify the data, but this behavior needs the patient's authorization. Maesa et al. [13] divide blockchain transactions into rule creation and right transfer. They store the rules on blockchain in compressed code form.

This work is supported by National Natural Science Foundation of China under grant No.61602200, the Science andA Technology Program of Guangzhou under grant No.201902020016, the Shenzhen Fundamental Research Program under grant No.JCYJ20170413114215614, the Guangdong Provincial Science and Technology Plan Project under grant 2017B010124001, and the Guangdong Provincial Key R&D Plan Project under Grant No.2019B010139001.

Meanwhile, they use the transaction ID to link the two types of transactions and ensure that anyone can see the process of the access rights transfer.

Hence, in this paper, we propose a blockchain-based smart contract access control system, including inter-contract and intra-contract access control system to alleviate this issue. In general, the main contributions of this paper are as follows:

- ***Automatic synchronization of access rules.*** To maintain transparency, we propose a method to ensure access rules can also be distributed to all peers timely. Specifically, we store the rules in blockchain state and implement a system-level smart contract to provide rule management function and access control service.
- ***Secure execution guarantee for rules.*** The access control rules may be tampered maliciously. To address this threat, we design and realize new flow processes related to smart contracts, such as deployment, execution, constructing new messages in commands.
- ***Complete access control APIs.*** To make the system easier applied in contract development and trace the history of rule changes, we implement a series of APIs. For example, we add an interface to the chaincode's stub code for storing access control rules. What's more, we use message ID generated by the transaction to identify historical changes, which will record previous message ID, resulting in a chain of rights modification history.

The remainder is organized as follows. In section II, we present the architecture and design of our system. Section III discusses the implementation. Section IV shows our experiments and analysis. Finally, section V draws a conclusion and proposes future works.

II. Architecture and Design

A. System Overview

The proposed system consists of inter-contract access control system and intra-contract access control APIs. Inter-contract access control system consists of client and server as illustrated in Fig. 1. The client is responsible for generating commands, executing them, and helping deploy smart contracts, consisting of three modules — Crypto Module, Serialization Module and Command Construction Module. The server manages contract administrators and access control rules of contracts. Meanwhile, the server maintains a mapping between smart contracts and their owners, providing system services like access control check for inter-contract calls. The server includes five modules, namely, Initialization Module, Crypto Module, Serialization Module, Contract Administrator Module, and Access Control Module.

The intra-contract system is composed of Initiation Module, Subject Management Module, Object Rule Module, and Authentication Evaluation Module. Users can send requests to the smart contract for accessing objects. Intra-contract access control system will determine whether the request is valid according to users' attributes and access control rules of the object. The rule specifies the conditions to access the corresponding object, which can be a certain resource such as a file or an execution function of a smart contract.

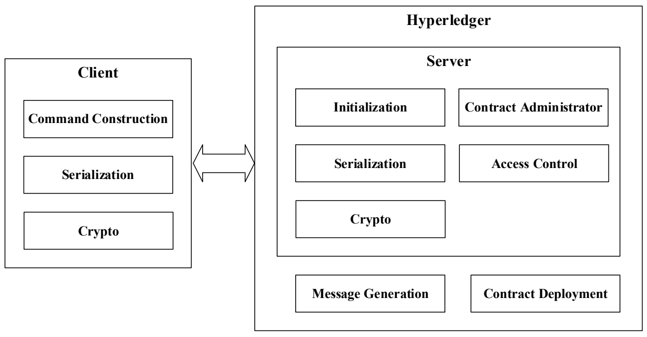

Fig. 1. Architecture of Inter-contract Access Control System

B. System Workflow

We introduced the administrators and owners for Hyperledger smart contracts, to strictly manage the rights to deploy contracts. When initializing the system, all related members jointly agree on an initial list of contract administrators, who can be added or deleted by voting. The administrator can manage the contract owners, who can manage all his smart contracts related states such as access control rules by client. Calling other contracts will trigger the access control module to check the access rights. This module will verify whether requesters are qualified based on callee's access rule and requester's attribute derived from Subject Management Module. Concretely, the expression in the rule is stuffed with the attribute value, then there will be an output revealing whether the requester has right.

C. Utility Module

All emitted commands need to be signed to determine their initiators' identities. The Crypto Module mainly handles the signature and verification for commands. By default, the client generates digests based on SHA256 for commands, then uses administrator's private key to sign them. The type of private key can be RSA or ECDSA produced by Openssl. Subsequently, the client will send the signed command to the server through HTTP. In this process, the command will firstly be serialized by the Serialization Module because the server can only deal with a string message. Then they are constructed a base on different parameters by the Command Construction Module and broadcast. Once received, the server performs deserialization, signature verification, and execution.

D. Contract Administrator Module

The Contract Administrator Module focuses on the management of the contract administrator. Once the blockchain platform is launched, the initialization module writes a list of administrators selected by the consortium into the system contract state. Then, they should control each member's join or withdraw by voting. For instance, as Fig. 2. shows, when adding a new contract administrator, an existing administrator

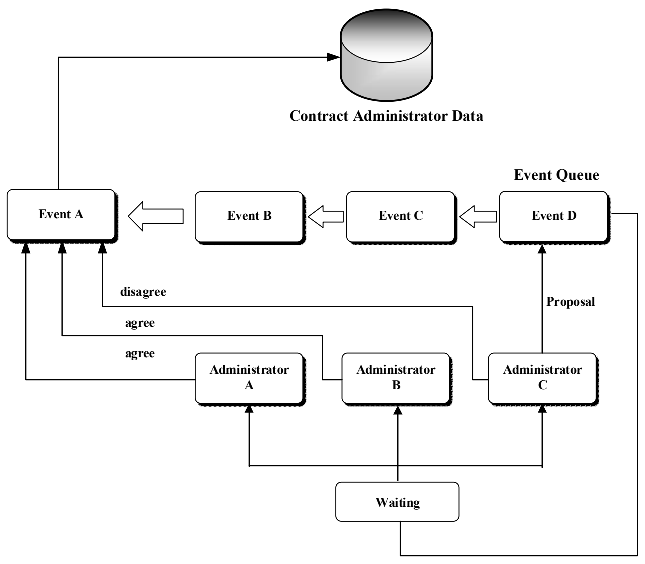

Fig. 2. Workflow of Contract Administrator Voting

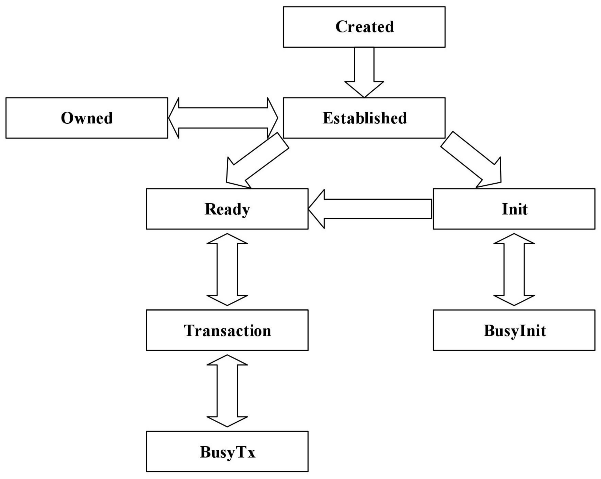

Fig. 3. Contract Handler's Finite State Machine

such as 'Administrator C' will initiate a proposal by sending a message to the pending event queue ('Event A/B/C/D'). Then other administrators will identify and vote for each pending event by polling this queue. Finally, the Administrator Module will collect all votes and judge whether they have reached a threshold, the new administrator will be added into 'Contract Adminstrator Data' if satisfying rules.

E. Access Control Module

The Access Control Module is used to store contracts access control management policies of their owner. Since there is no need to transfer access rights between contracts but ask for permission recovery, so we adopt access control list structure. When a user deploys a contract, becoming its owner, a relationship mapping between them will be established and stored in the state. The owner can distribute a corresponding function to manage access rules towards it. Latter, whenever this contract is called, the access control function will be triggered automatically to judge whether the caller has related rights based on access policies.

III. Implementation

In this section, we will introduce key technologies used in our system, including contract deployment process modification, inter-contract invocation improvement, replay attack defense, state tamper-proofing mechanism, and access rights history maintainance.

A. Contract Deployment Process Modification

Our system aims at limiting that only owners can deploy contracts to secure system. To reach the goal, we need to change original contract deployment command in Hyperledger Fabric. We first attach two additional parameters (-i, -s) to the command, corresponding to owner's ID and the signature respectively. These two parameters are enforced to specify when sending a deployment command, otherwise, an exception will throw. So the system can lookup this ID in an owner list and get corresponding public key, to check identity by verifying his signature. Retrospecting this process, we can understand the relationship mappings established by system smart contract after deploying contracts successfully, so other users can not call contracts arbitrarily unless being granted rights in access policies. To sum up, the Hyperledger smart contract consists of a finite state machine and a message mechanism. Therefore, a new state will be added to the state machine of the chaincode's handler and the chaincode's shim layer, that is, the state of adding the owner relationship is entered after the chaincode enters the registration completion state, shown in Fig. 3.

After starting to deploy, the ordinary contract communicates with the node to enter the created state, and sends a registration message to the node. When processing the message, the node adds the handler of the chaincode, then the chaincode enters the created state. At this point, if the contract is a normal contract, its ownership will be added by system, entering the owned state. Its current handler will add this relationship in system contract through system-generated messages. If added successfully, the contract rollbacks to created state, otherwise the contract will enter termination state. Then the contract can be normally initialized, invoked and so on. Therefore, the process of contract access control can be guaranteed to execute correctly by our system.

B. Inter-contract Invocation Improvement

Given a common contract interaction example that chaincode A attempts to call chaincode B. Firstly, A will send a InvokeChaincode message generated in the shim layer to its handler. Then the handler enters the BusyState state, constructing a call message and sending to B's handler, which will load corresponding chaincode and execute received a message. The result will return to A after completing execution. A will execute the subsequent code logic continuously. However, to realize access control checks each time when a call is triggered between contracts, that is, before A enters the BusyState state (send execution message to B) in this example, we improve original invocation process via the InvokeChaincode interface of stub code. To be concrete, A's handler will first construct a message leveraging the same transaction, and send to the system smart contract, which will check their access right based on access rules. If getting failure response, the invoking request aborts directly, otherwise, A obtains the result.

C. Replay Attack Defense

Since the system functions of the administrator and the contract owner are all implemented by the signature command, there is a threat of replay attacks. The replay attack means that attacks can intercept command messages sent by administrators and re-send them to the server, thus bypassing the authentication, which is one of the most common attacks in network systems. In this system, if an attacker eavesdrops on an administrator's command message for managing a contract's access control rules, then even if the administrator has other modifications to rules, the attacker can modify the access control rules by resending the message. Therefore, there must be a corresponding mechanism to prevent command replay attacks.

This system proposes an effective timestamp scheme by attacking messages with a current timestamp to avoid this problem. The client and server negotiate a valid time interval in advance, ie, a block generation period, in which messages are considered valid. The system sets a block generation time as the effective time window, which can effectively prevent the same command from being replayed. Meanwhile, two identical messages in the same block generation interval cannot be deemed as a malicious attack. Because only one of them will be recognized and executed , while left another invalid.

D. State Tamper-proofing Mechanism

Malicious users may commit or roll back the state in a single transaction, for which we propose two countermeasures. As to the first scheme, we can change the rollback flag when there is state transfer from the entire data state to the chaincode data state in the State Module. Moreover, once there is a state change in execution, the chaincode should return an execution result. If state changes caused by a single transaction are commited to a global state (Blockchain ledger), the system should deal with change result in each chaincode. The second solution logs all chaincode IDs that throw errors when called to change global state in a non-intrusive way, meanwhile, maintaining the data change of a single transaction. When these state changes going to be committed into persistent storage, the system will judge whether chaincode IDs related changes are in the above-mentioned set. If so, they will be discarded. It is noted that it is possible to predict whether the message will be thrown errors based on current state of the finite state machine and the received error message. Because chaincode execution will also return the same error message.

E. Access Rights History Maintainance

In order to trace history modification, the system will record changes towards permissions in the corresponding data interface added by the stub code of the chaincode, and connect these changes using the transaction ID. First, the system needs to pass the transaction ID to the stub code of the chaincode, and provide the corresponding interface for obtaining these IDs. Then the chaincode will record the current together with last transaction ID in each state change. Specifically, the system uses the status prefix plus the state content as a storage key for storing the current transaction ID. Every time the user executes a command that needs to record a history operation, the system will fetch the last transaction ID from the chaincode, together with current transaction ID committed to global state. Then, the current transaction ID will be updated as the last ID. Therefore the state modification history can be concatenated by these IDs.

IV. Experiment

In this section, we present an evaluation of the system, focusing on exploring how much overheads the system embedded with the proposed access control mechanism will introduce. All experiments were conducted on a server with 2.60GHz 64-bit Intel Xeon CPU E5-2670 v3 processor with 16-cores, 16 GB RAM, and the version of operating system is Ubuntu Server 16.04.4 LTS. All tests are performed in the context of Hyperledger fabric 0.6, a popular federated blockchain with PBFT consensus mechanism. Moreover, the detailed configurations of the server and the blockchain are shown in Table I.

TABLE I
Test Configurations

Parameter	Value
CPU	Intel Xeon E5-2670 @ 2.60GHz
Number of cores	16
RAM	16GB
Operating System	Ubuntu Server 16.04.4 LTS
Number of nodes	4
Blockchain	Fabric v0.6
Consensus	PBFT
Golang	1.8

In order to achieve the purpose of access control between smart contracts, the system checks the access control list before executing each call request, which may have a certain impact on the performance of the Hyperledger. The general performance metrics are throughput and latency. However, for blockchain, actual transaction delays include client-to-system delay, plus delay in system processing feedbacks. While the former is often related to the client network, which is immaterial to our system evaluation. Therefore, we do not consider it in experiments. Given that the feedback processing delay is closely related to the consensus algorithm and the topology relationship between peers, our evaluation will focus

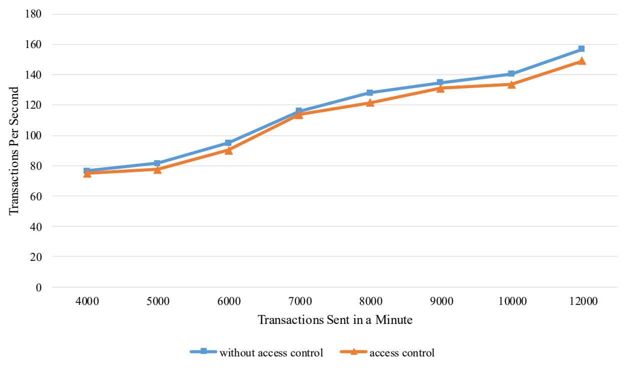

Fig. 4. Transactions Per Second With and Without Access Control

on transaction throughput rate based on the same default consensus configuration, reducing the interference of uncorrelated factors. Usually, throughput is computed by measuring amount of processed requests per unit of time. In this experiment, it is regarded as the number of processed contract execution requests per second. The consequences are shown in Fig. 4.

Because all transactions in the x axis are used to call smart contract, so the x axis shows the amount of smart contract call requests in a minute and the y axis represents the amount of successful execution in Fig.4. When the amount of requests is less than 9000, all transactions were successfully executed because its upper limit of throughput had not been reached. As the figure shows, when the number of transaction requests reaches the upper throughput limit of the Hyperledger, the proposed system has little effects on the throughput in fact. It can be figured out that the performance loss of the Hyperledger caused by introducing access control mechanism is about 2%-5%.

V. Conclusion and Future Work

This paper analyzes the implementation of Hyperledger smart contract, and eventually finds the state modification issue during invocations between smart contracts. Moreover, this paper exposes the lack of access control-related contract development interface in the Hyperledger. In view of the existing problems, this paper puts forward an access control mechanism between smart contracts and provides developers with access control interfaces. The solution not only reduces the instability caused by access control, but also increases the efficiency of smart contracts development for smart contract developers.

However, the study is not quite thorough within limited time. There are still some issues that remains to be discussed. For instance, with the development of blockchain contracts, they will be more complicated and functional. To provide sufficient security, we should consider implementing the relevant access control through its whole life such as deployment, invoking, upgrade, storage, and termination. What's more, fed access control lists or polices comes from external world, which may be tampered or faked. Therefore, it is necessary to design a secure oracle to transmit trusted data to blockchain.

References

[1] Ethereum. [Online]. Available: http://gavwood.com/Paper.pdf
[2] J. Liang, W. Han, Z. Guo, Y. Chen, C. Cao, X. S. Wang, and F. Li, "Desc: Enabling secure data exchange based on smart contracts," *Science China Information Sciences*, vol. 61, no. 4, pp. 262–264, 2018.
[3] S. Singh and N. Singh, "Blockchain: Future of financial and cyber security," in *Proceedings of the 2nd International Conference on Contemporary Computing and Informatics*, 2017, pp. 463–467.
[4] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, 2015, pp. 104–121.
[5] C. Lin and C. Liao, "A survey of blockchain security issues and challenges." *International Journal of Network Security*, vol. 19, no. 5, pp. 653–659, 2017.
[6] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Proceedings of the 6th International Conference on Principles of Security and Trust*, 2017, pp. 164–186.
[7] B. Marino and A. Juels, "Setting standards for altering and undoing smart contracts," in *Proceedings of the 10th International Symposium on Rules and Rule Markup Languages for the Semantic Web*, 2016, pp. 151–166.
[8] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: Platforms, applications, and design patterns," in *Proceedings of the 2017 International Conference on Financial Cryptography and Data Security*, 2017, pp. 494–509.
[9] L. Luu, H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 254–269.
[10] E. Yuan and J. Tong, "Attributed based access control (abac) for web services," in *Proceedings of the 2005 IEEE International Conference on Web Services*, 2005, pp. 561–569.
[11] 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, pp. 5943–5964, 2016.
[12] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Proceedings of the 2nd International Conference on Open and Big Data*, 2016, pp. 25–30.
[13] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control," in *Proceedings of the 17th IFIP International Conference on Distributed Applications and Interoperable Systems*, 2017, pp. 206–220.