MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Blockchain for Embedded System Accountability

Chiu, Michael; Goldsmith, Abraham; Kalabic, Uros

TR2021-041 May 04, 2021

Abstract

We present, in the form of a proof of concept, a permissioned blockchain framework that attains accountability within a system containing embedded devices. Accountability is a desirable property of distributed systems that enables the detection, identification, and removal of faulty or malicious behavior. It is a complementary approach to Byzantine fault tolerance, which is concerned with ensuring continued functioning of the system in the presence of Byzantine faults. Our proof of concept consists of a Raspberry Pi acting as a human interface; the blockchain is implemented in Hyperledger Fabric and the Raspberry Pi runs a lightweight blockchain client to minimize computational burden. The application shows that we are able to use smart contracts to detect and identify faulty or malicious hardware, and the permissioning framework to remove it.

IEEE International Conference on Blockchain and Cryptocurrency (ICBC)

© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Blockchain for Embedded System Accountability

Michael Chiu
Department of Computer Science
University of Toronto
Toronto, ON M5S 2E4
chiu@cs.toronto.edu

Abraham Goldsmith
Mitsubishi Electric Research Labs
Cambridge, MA 02139
goldsmith@merl.com

Uroš Kalabić
Mitsubishi Electric Research Labs
Cambridge, MA 02139
kalabic@merl.com

***Abstract*—We present, in the form of a proof of concept, a permissioned blockhain framework that attains accountability within a system containing embedded devices. Accountability is a desirable property of distributed systems that enables the detection, identification, and removal of faulty or malicious behavior. It is a complementary approach to Byzantine fault tolerance, which is concerned with ensuring continued functioning of the system in the presence of Byzantine faults. Our proof of concept consists of a Raspberry Pi acting as a human interface; the blockchain is implemented in Hyperledger Fabric and the Raspberry Pi runs a lightweight blockchain client to minimize computational burden. The application shows that we are able to use smart contracts to detect and identify faulty or malicious hardware, and the permissioning framework to remove it.**

***Index Terms*—Blockchain, accountability, distributed systems, fault detection**

I. Introduction

In distributed systems, apart from security, the primary problem is ensuring trust [1]. However, the distributed nature of networked systems makes trust difficult to guarantee because the system must not only demonstrate robustness to both malicious actors and faults, but also transparency. Broadly, there are two approaches to increasing the robustness of a system: implementing Byzantine fault tolerant (BFT) algorithms and introducing accountability. BFT algorithms robustify a distributed system by making it tolerant of up to roughly one third of participants exhibiting Byzantine-faulty behavior [2]. However, robustness does not mean that a system is able to identify and remove faulty participants, or that it increases transparency.

An improvement on robustness is *accountability*. Accountability is a property of networked systems enabling the detection, identification, and isolation of malicious participants. In contrast to a BFT-based approach, accountability also increases transparency within a system. This is because an accountable system is able to detect and identify malicious participants; accountability relies on the *auditability* of a system [3]. Auditability is a sub-property of accountable systems that is related to an increase in the transparency of a system, which increases its trust. The key to auditability is the enablement of secure, tamper resistant, tamper evident, and transparent audit logs.

This work was supported by Mitsubishi Electric Research Laboratories.

978-0-7381-1420-0/21/$31.00 ©2021 IEEE

Accountability has long been recognized as a desirable property of trustworthy systems [4] and is being increasingly recognized as a desirable property of distributed systems [5], [6], where it has been shown to increase the robustness of a system [7]. Indeed, it has been demonstrated that identifying and isolating faulty messages can improve throughput for state machine replication in faulty scenarios by up to 70% [8].

A number of accountable systems have been proposed in the literature but have proven difficult to put into practice. One of the earliest was PeerReview [9]. Another is FullReview [10], a protocol based on game-theoretic protocols that also aims to enable accountability in a system. However, significant limitations remain: both protocols require modification of the target software and must be included in each target application. Any scheme that attempts to implement accountability into a system must be able to secure audit logs and the importance of this has long been recognized [11], [12]. Early approaches to securing audit logs were mainly cryptographic, such as that of [13]. More recent approaches to secure logging are blockchain based [14], [15]. The blockchain data structure itself satisfies many of the properties, such as being append-only, required for securing an audit log.

In this work, we present an application of permissioned blockchains in conjunction with kernel-level instrumentation technologies to the problem of enabling accountability within a distributed system containing embedded devices. We present an application of this idea in the form of a proof of concept (PoC), implemented using the permissioned blockchain framework of Hyperledger Fabric [16] and a Raspberry Pi. The Raspberry Pi serves as a human-computer interface via a button press; Fabric is implemented due to its modular architecture, which enables proper separation of concerns. We also note that Fabric has been put into use by many enterprises, and industrial and government projects, *e.g.*, [17].

The rest of the paper is structured as follows. Section II presents the case for permissioned blockchains as simple and robust solutions to enabling accountability within a distributed system. Section III presents the PoC implementing an accountable blockchain-based system in Fabric. Section IV is the conclusion.

II. Blockchain for Accountable Distributed Systems

As a primary principle behind the design of dependable network systems [18], accountability consists of the following three properties [19]:

1) Fault prevention: discouragement of malicious behavior by increasing its cost;
2) Fault tolerance: detection and isolation of faults and prevention of their spread to other parts of the system;
3) Fault removal: removal of misbehaving actors from the system to limit their impact on the rest of the system.

It is clear from the definitions that system accountability relies greatly on system auditability. Auditability, in turn, relies on the security of audit logs which must maintain the following to be suitable for use in audits [20]:

- Tamper resistance: Guarantee that only the creator can add valid entries and that entries cannot be altered;
- Verifiability: Ability to check that log entries have not been deleted or altered;
- Data access control and searchability: Search accessibility to only a subset of audit logs.

Blockchain technology satisfies all three and, indeed, the applicability of blockchain to securing audit logs has gained widespread recognition [14] and led to a large number of applications of blockchain to various domain-specific secure logging problems. In the following, we show that, in addition to auditability, a permissioned blockchain network of nodes satisfies the three properties of an accountable distributed system, thereby increasing the robustness and trustworthiness of the system.

A. Fault Prevention

Malicious actors within a distributed system can hamper the system by causing nodes to behave in unexpected ways. For example, a malicious actor can cause *equivocation*, which is when the actor takes control of a node and reports different truths to different participants within the system. The general problem of designing reliable distributed systems in the presence of unexpected behavior from the participants (whether due to failure or maliciousness) is known as the Byzantine generals problem [21] and has led to a class of algorithms known as practical Byzantine fault tolerant (PBFT) algorithms that provide tolerance of malfunctioning participants [2], [22].

An auditable system that is able to provide a trusted single source of truth of the operations of participating nodes greatly discourages and can, depending on the implementation, prevent equivocation entirely. For example, in [23], the proposed append-only log abstraction prevents equivocation and improves on PBFT consensus algorithms.

Blockchain technology, when permissioned, is able to realize a system capable of fault prevention. Smart contracts in permissioned blockchains are trusted and, in conjunction with kernel instrumentation,[1] can ensure that activity associated with a node can be trusted; the blockchain data structure can act as a shared data layer amongst participating nodes that decentralizes and removes the need for trusted log storage and ensures the immutability of the data. Thus, a permissioned blockchain can make transparent the actions of all participants, which would greatly increase the cost of malicious behavior.

[1]A family of technologies used by modern kernels to record and profile kernel events, *e.g.*, Linux Audit system, Probes, Tracepoints, eBPF

B. Fault Tolerance

The blockchain data structure, as an irrefutable source of truth within a network, enables the detection of faults and malicious behavior. Faults can arise due to malfunctioning code, perhaps due to faulty data or an internal node error, and may cause a node to fail; they may also be caused by maliciousness. Either way, faults can be isolated at the network level using smart contracts. In permissioned blockchain networks, such as Fabric, smart contracts are the interface between a node and the blockchain network. Inputs to the network from a node can be checked by a smart contract for validity. A smart contract, upon detecting data outside a valid set of parameters, is able to log the attempted call with the parameters for audit purposes and not execute the faulty program.

Fault tolerance also requires two properties: determinism and strong identity [9], both of which are satisfied by any permissioned blockchain system by default. Determinism stipulates that any node given the same input should return the same output. Strong identity stipulates that participants must be uniquely identifiable.

C. Fault Removal

Blockchain-based auditability provides a single source of truth for the actions taken by the participants and the activity within a node, which can be used to identify faulty behavior. Once identified, removal of participants can be accomplished simply by revoking the membership of malfunctioning nodes in the permissioned network.

III. Proof of Concept

We implement a PoC as an example of a blockchain-enabled accountable system. In the system, a human operator communicates with the blockchain through a client device implemented on a Raspberry Pi. The PoC shows how we can provide accountability by publishing interactions with the device to the blockchain using network-trusted smart contracts to record the data. The specific types of interactions we implement are login attempts and buttons presses on the Raspberry Pi, representing a common need in certain types of regulatory frameworks.

We are concerned with permissioned members of the network acting maliciously. We assume that the network, in part due to its permissionedness, is sufficiently protected from malicious actors outside of the network. The main concern is equivocation, which is only possible to achieve by editing the OS since smart contracts in permissionless blockchain networks are shared between all participants and therefore trusted by the network. Another relevant security threat is the compromise of a physical interface device, which in our case

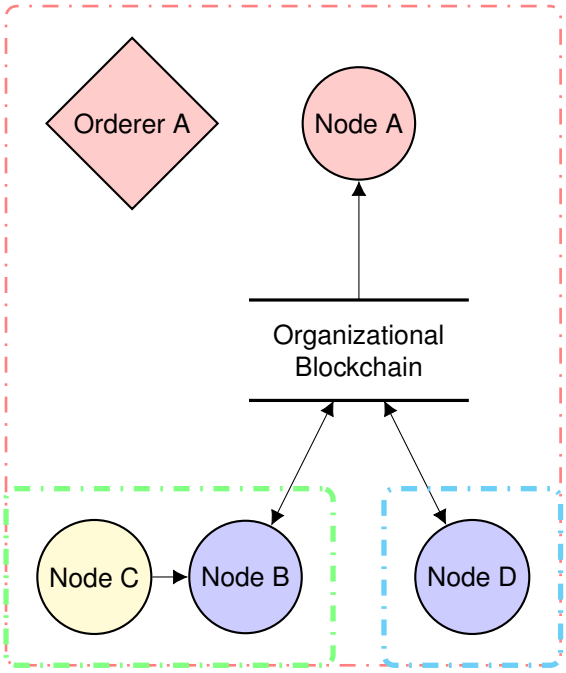

Fig. 1. PoC system level diagram; arrows represent flow of information

is the Raspberry Pi. The PoC is susceptible to this threat, but studying the problem is outside the scope of this short paper.

The implementation was done in Fabric, which is a Go-based open-source library for creating permissioned blockchain networks and was chosen because it is scalable and modular and because of the active open-source community of contributors that work on the Fabric project.

A. Blockchain Architecture

Fig. 1 shows a system-level diagram of the nodes implemented in the PoC. In the PoC, a parent organization (red node in the figure) observes the blockchain and its interaction with the blockchain is read-only. The parent organization also acts as a blockchain ordering service (red orderer); in the PoC, this is implemented through conventional application of Fabric libraries. The parent organization observes children organizations (boxes containing purple nodes). These can represent a company subject to regulations or local subsidiaries of a larger corporation subject to monitoring. Their interactions with the blockchain are capable of both reading and writing. In the figure, we show that the yellow Node C communicates with the blockchain through Node B. The communication between these two nodes is made possible with the implementation of a lightweight client that does not implement a full node. The reason for this architecture is to allow embedded devices, which are not typically capable of running a full node, to communicate with the blockchain. In the PoC, the embedded device is the Raspberry Pi on which we have implemented a lightweight client.

B. Implementation in Hyperledger Fabric

Fabric is a Docker-based framework where nodes in the network are represented using Docker containers and the Fabric library compiles into a number of binaries that contain the blockchain network functionality. The two most important binaries related to the execution of a Fabric blockchain network are the orderer and the peer binaries. The orderer binary is an executable containing the ordering service and is a Go-based daemon; the peer binary containing the functionality for running a blockchain node in Fabric is a command line application used to interact with the Fabric blockchain network through a node or, as referred to in Fabric documentation, peer.

Blockchain functionality is implemented by providing functions from the chaincode interface in Fabric. Chaincodes are compiled into Go binaries which are then run on peers. The separation of peer and chaincode executables has at least two advantages: a peer can run multiple chaincodes and chaincode failure should not crash the node.

As shown in Fig. 1, the PoC consists of four types of nodes: ordering node (red diamond), observing node (red circle), operating node (purple circles), and lightweight client (yellow circle). We describe each in the following.

a) Orderer nodes: These nodes order transactions in the network. In the PoC, we implement the orderer through the conventional application of Fabric libraries.

b) Observer nodes: These are run by observers in order to carry out real-time auditing. They are full nodes, meaning that they synchronize a complete copy of the blockchain and are able to participate in the consensus process by proposing transactions.

c) Operator nodes: These participate actively in the consensus process by constructing transactions containing data from OS auditing mechanisms. Operator nodes can be servers running a dedicated node aggregating OS auditing data from multiple OSes, or they can run on top of an individual OS.

d) Lightweight clients: Most modern computers are embedded devices with have limited computational resources that make them unable to run a full node and therefore store a full copy of the blockchain. Lightweight blockchain clients enable embedded devices to be administered by the blockchain network protocol itself and, in the PoC, we implement a lightweight client on the Raspberry Pi. We do this by running a peer binary on the embedded device; due to the architecture of Fabric, the peer binary itself acts as a lightweight client since it is a trusted binary used to interact with the permissioned blockchain network.

C. Raspberry Pi and Permissioning

The Raspberry Pi acts as a permissioned, embedded device providing a human control interface device to the network. We log the action of a human operator, who may or may not be authorized, and who interacts with an embedded terminal that is trusted by the network. We note that, since the Raspberry Pi runs on the ARM instruction set, the available Fabric binaries and Docker images do not work. To enable compatibility, we built from source with ARM64 as the target backend.

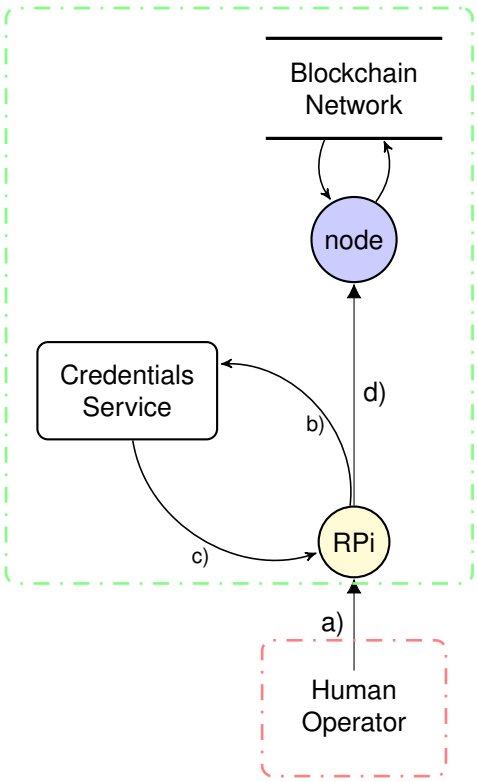

Fig. 2. Schematic of permissioning involving human operators in the PoC where a) the human operator inputs some commands into the Raspberry Pi along with some credentials; b) the credentials are sent to an LDAP credentialing service; c) the results are returned to the lightweight client; if the credentials are valid, the lightweight client proceeds with packaging a transaction containing the operator's command and some audit data; if invalid, the lightweight client constructs a transaction indicating a failed credential attempt and the requested command for audit purposes; finally d) the transaction is sent to a full node for inclusion into the blockchain

Fig. 2 illustrates this functionality as implemented in the PoC. In the figure, nodes that are permissioned by the network during setup of the network are contained in the green box. The human operator is unpermissioned and not part of the network. In the PoC, the human operator provides authentications through a terminal, after which an LDAP credentialing service determines whether he is permitted to log a button press onto the blockchain. The exercise of pressing a button on a Raspberry Pi shows how physical processes can be audited with blockchain, satisfying fault tolerance. The permissioning of the Raspberry Pi demonstrates fault removal.

D. Discussion

The PoC shows that a permissioned blockchain is able to provide a simple and robust solution to enabling accountability within a permissioned blockchain system. It illustrates that physical processes can be made accountable, digitally, through blockchain enabled auditability and smart contracts as trusted network services. Fig. 3 provides the result of the invocation of a chaincode checking the history, *i.e.*, audit trail, of the button presses on the Raspberry Pi recorded on the blockchain. The blue and green boxes show successful attempts of the button

Fig. 3. Audit trail of button presses on the Raspberry Pi recorded on the blockchain

presses. A recorded failed attempt at altering the state, due to incorrect credentials, is shown by the red box. Our prototype demonstrates that a permissioned blockchain network prevents equivocating the occurrence of a physical button press.

IV. Conclusion

In this paper, we considered the application of blockchain to ensuring accountability in distributed systems. We discussed how permissioned blockchains are well suited to this problem because the blockchain, as an immutable, append-only data structure that is replicated among all participants, provides a single source of truth that makes the detection and identification of malicious behavior straightforward, and because permissionedness makes removal of participants straightforward.

We presented an application in the form of a proof of concept implemented in Hyperledger Fabric and showed some of the additional benefits of using a permissioned blockchain framework. Our PoC included an embedded hardware component demonstrating an example of how a physical processes may be made digitally accountable.

Acknowledgments

The authors acknowledge Masafumi Yamada, Daiki Nakashima, and Emi Sugiyama of Mitsubishi Electric Corporation for technical discussions.

REFERENCES

[1] S. Pearson and A. Benameur, "Privacy, security and trust issues arising from cloud computing," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. and Sci.*, Indianapolis, IN, 2010, pp. 693–702.

[2] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. Symp. Operating Syst. Des. and Implementation*, New Orleans, 1999, pp. 173–186.

[3] R. K. L. Ko, B. S. Lee, and S. Pearson, "Towards achieving accountability, auditability and trust in cloud computing," in *Proc. Int. Conf. Advances Comput. Commun.*, Kochi, India, 2011, pp. 432–444.

[4] Department of Defense, "Trusted Computer System Evaluation Criteria," Standard DoD 5200.28-STD, 1985.

[5] A. Haeberlen, "A case for the accountable cloud," *ACM SIGOPS OSR*, vol. 44, no. 2, pp. 52–57, 2010.

[6] B. E. Ujcich, A. Miller, A. Bates, and W. H. Sanders, "Towards an accountable software-defined networking architecture," in *IEEE Conf. Netw Softwarization*, Bologna, Italy, 2017.

[7] A. Haeberlen, P. Kouznetsov, and P. Druschel, "The case for Byzantine fault detection." in *Proc. Workshop Hot Topics Syst. Dependability*, Seattle, WA, 2006.

[8] M. Correia, D. G. Ferro, F. P. Junqueira, and M. Serafini, "Practical hardening of crash-tolerant systems," in *Proc. USENIX Annu. Tech. Conf.*, Boston, 2012, pp. 453–466.

[9] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical Accountability for Distributed Systems," *ACM SIGOPS OSR*, vol. 41, no. 6, pp. 175–188, 2007.

[10] A. Diarra, S. B. Mokhtar, P.-L. Aublin, and V. Quéma, "Fullreview: Practical accountability in presence of selfish nodes," in *Proc. IEEE Int. Symp. Rel. Distrib. Syst.*, Nara, Japan, 2014, pp. 271–280.

[11] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 2, p. 159–176, 1999.

[12] M. Swanson and B. Guttman, "Generally accepted principles and practices for securing information technology systems," NIST, Special Publication 800-14, 1996.

[13] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines," in *Proc. USENIX Secur. Symp.*, San Antonio, TX, 1998, pp. 53–62.

[14] L. Shekhtman and E. Waisbard, "Engravechain: Tamper-proof distributed log system," in *Proc. Workshop Blockchain-Enabled Netw. Sensor*, New York, 2019, pp. 8–14.

[15] B. Putz, F. Menges, and G. Pernul, "A secure and auditable logging infrastructure based on a permissioned blockchain," *Comput. Secur.*, vol. 87, no. 101602, 2019.

[16] E. Androulaki *et al.*, "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in *Proc. EuroSys Conf.*, no. 30, 2018.

[17] Federal Reserve Bank of Boston, "Beyond theory: Getting practical with blockchain; Boston Fed learns by doing with blockchain technology," White Paper, 2019.

[18] A. R. Yumerefendi and J. S. Chase, "The role of accountability in dependable distributed systems," in *Proc. Workshop Hot Topics Syst. Dependability*, Yokohama, Japan, 2005.

[19] J.-C. Laprie, *Dependability: Basic concepts and terminology*. Vienna: Springer, 1992.

[20] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an encrypted and searchable audit log," in *Proc. Netw. Distr. Syst. Secur. Symp.*, San Diego, CA, 2004.

[21] L. Lamport, "The weak Byzantine generals problem," *J. ACM*, vol. 30, no. 3, pp. 668–676, 1983.

[22] M. Barborak, A. Dahbura, and M. Malek, "The consensus problem in fault-tolerant computing," *ACM Comput. Surv.*, vol. 25, no. 2, pp. 171–220, 1993.

[23] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested append-only memory: Making adversaries stick to their word," *ACM SIGOPS OSR*, vol. 41, no. 6, pp. 189–204, 2007.