



AFRL-RI-RS-TR-2019-043

DATA PROVENANCE ASSURANCE IN CLOUD USING BLOCKCHAIN

OLD DOMINION UNIVERSITY

MARCH 2019

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2019-043 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

LAURENT NJILLA
Work Unit Manager

/ S /

JAMES S. PERRETTA
Deputy Chief, Information
Exploitation & Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) MARCH 2019		2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) AUG 2016 – SEP 2018	
4. TITLE AND SUBTITLE DATA PROVENANCE ASSURANCE IN CLOUD USING BLOCKCHAIN				5a. CONTRACT NUMBER N/A	
				5b. GRANT NUMBER FA8750-16-1-0301	
				5c. PROGRAM ELEMENT NUMBER 62788F	
6. AUTHOR(S) Sachin Shetty				5d. PROJECT NUMBER BLOC	
				5e. TASK NUMBER KC	
				5f. WORK UNIT NUMBER HN	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Old dominion University Research Foundation 4111 Monarch Way, Suite 204 Norfolk, VA 23508				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RIGA 525 Brooks Road Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI	
				11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2019-043	
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The project started on Aug 18, 2016. In the 24 months of executing the project, the team has conducted basic research on data provenance architecture for cloud using block chain, surveyed the vulnerabilities in block chain, in-depth analysis of the block discarding attack, developed a Proof-of-Stake consensus protocol for cloud based blockchain, architecture for secure BattlefieldIoT, cyber supply chain provenance, integration of software guard extensions on distributed ledgers for increased privacy.					
15. SUBJECT TERMS Blockchain, cloud, data provenance					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 142	19a. NAME OF RESPONSIBLE PERSON LAURENT NJILLA
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

Table of Contents

Section	Page
List of Figures.....	iii
List of Tables.....	iv
1.0 SUMMARY.....	1
2.0 INTRODUCTION.....	2
3.0 METHODS, ASSUMPTIONS, AND PROCEDURES.....	3
3.1 Blockchain Background.....	3
3.1.1 Blockchain Building Blocks.....	5
3.1.2 Blockchain Military Cyber Operations Use Cases.....	6
3.1.3 Blockchain Challenges.....	7
3.2 Provchain.....	10
3.2.1 Architecture Overview.....	11
3.2.2 Preliminaries and Concepts.....	12
3.2.3 Threat Model.....	13
3.2.4 Key Establishment.....	13
3.2.5 Provchain Implementation.....	15
3.2.5.1 Provchain Data Collection and Storage.....	15
3.2.5.2 Provenance Data Validation.....	18
3.3 BlockCloud Security Analysis.....	20
3.3.1 Exploring the Attack Surface of Blockchain.....	20
3.3.2 Countering Double Spending in Next-Generation Blockchains.....	21
3.3.3 BlockCloud Attack Surfaces.....	22
3.3.4 Block Withholding Attack.....	34
3.3.4.1 System Model.....	34
3.3.4.2 Augmenting with Extra Hashing Power.....	35
3.3.4.3 Disruptive Attack Strategy Analysis.....	36
3.3.5 DDOS Attack on Mempools.....	41
3.3.5.1 Data Collection for Evaluation.....	42
3.3.5.2 Threat Model.....	43
3.3.5.3 Attack Procedure.....	44
3.3.5.4 The Distribution Phase.....	45
3.3.5.5 The Attack Phase.....	45
3.3.5.6 Attack Cost.....	45
3.3.5.7 Countering the Mempool Attack.....	46
3.3.5.8 Fee-based Mempool Design.....	47
3.3.5.9 Age-based Countermeasures.....	51
3.4 Blockchain and SGX.....	55
3.4.1 Background.....	56
3.4.2 Blockchain Vulnerabilities and Attacks.....	56
3.4.3 Intel SGX Security Capabilities.....	57
3.4.4 Design of SGX-Enabled Distributed Ledger with Multi-Channel Architecture.....	59
3.4.4.1 SGX Enabled Connection Layer.....	60
3.4.4.2 Multi-Channel Transaction Layer.....	64

4.0	RESULTS AND DISCUSSIONS.....	66
4.1	Evaluation of Prochain.....	66
4.2	Evaluation of BWH Attack.....	70
4.3	DDOS Attack Results.....	73
4.4	Blockchain and SGX Evaluation.....	75
4.5	Blockchain Evaluation Platform Results.....	77
4.5.1	Architecture.....	77
4.5.2	Hyperledger Fabric.....	80
4.5.3	Measures of Performance.....	92
4.5.4	Simple Blockchain Simulation.....	102
4.5.5	Blockchain Simulation Introduction.....	105
4.6	Scalable Implementation in Hyperledger Fabric.....	111
4.6.1	Hyperledger Fabric Implementation.....	112
4.6.2	Performance Evaluation.....	114
5.0	CONCLUSIONS.....	118
6.0	RECOMMENDATIONS.....	121
7.0	REFERENCES.....	122
Appendix A-1 – List of Publications		130
Appendix A-2 – List of Presentations.....		132
Appendix A-3 – Blockchain For Distributed Systems Security Book Cover		133
Appendix A-4 – 2018 Top 50 Blockchain Papers and Research Team Award		134
LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS		135

List of Figures

Figure		Page
1	Block Structure.....	5
2	Blockchain Architecture.....	5
3	Provchain.....	12
4	ProvChain System Architecture.....	14
5	Provenance Data Collection and Storage.....	16
6	Blockchain Receipt.....	17
7	Merkle Tree.....	18
8	Transaction and Block Information.....	19
9	Hard Fork resulting from set of peers following conflicting rules due.....	28
10	Stale vs. orphan blocks.	28
11	Illustration of Selfish Mining.....	30
12	Temporal study of mempool size and mining fee paid by the users in a popular Blockchain system.....	42
13	Analysis of Fee-based Design.	50
14	Analysis of Age-based Design. design.....	54
15	SGX Enabled Blockchain Architecture in the Cloud.....	60
16	System Ledger and Subledger	65
17	Average Response Time with Different File Size	67
18	Throughput over time without Provchain.....	68
19	Throughput over time with Provchain.....	68
20	Number of Transactions per Second.....	69
21	Response Time Distribution without Provchain.....	69
22	Response Time Distribution with Provchain.....	69
23	Effect on Average Reward in Proportional Pool.....	71
24	Effect on Average # of Shares in Proportional Pool.....	72
25	Effect on Average Reward in PPLNS Pool.....	72
26	PPLNS Reward Variation Vs. Window Length (N).....	73
27	Performance of fee-based and age-based designs under mempool DDoS attack.....	74
28	Average Time Cost with Different Difficulty Bits.....	76
29	Average Hashing Power with Different Difficulty Bits.....	76
30	Screenshot of Artifact Generation Script Output.....	83
31	Screenshot of Console Outputs from Start Script for the Basic-Network Example.....	84
32	Example View of a Docker-Compose yml Configuration File.....	85
33	Example Terminal Results for the Docker ps Command of the Basic Network	86
34	Terminal View of the Output Logs Produced by the Orderer Node.....	87
35	Terminal Output for the Kafka Example.....	89
36	Terminal results for the docker ps command while running the kafka system..	89
37	Terminal Output from the Initialize all Script which Runs and Tests the Kafka Blockchain.....	90
38	Terminal Output from Docker Ps Command, Notice the Chaincode Containers That Have Been Started.....	91
39	Terminal Output After Stopping the Network.....	92
40	Illustration of the Process to Measure A Single Transaction Per Second.....	93
41	Illustration of Measuring Total System Transactions Per Second.....	94

42	Example Output Of Performance In Simulated Proof Of Stake Blockchain System.....	96
43	Example Output Of Performance In Simulated Proof Of Stake Blockchain System With Communication Delay Added.....	97
44	Example Output Reporting the Average Number of Times the Highest Staked Holder did not Get Elected as Validation Leader in the Proof of Stake Process.....	98
45	Illustration of a Basic Configuration of Hyperledger Fabric.....	99
46	Illustration of a Practical Byzantine Fault Tolerant Consensus Process.....	100
47	Performance in Transactions Per Second to Process a Single Transaction with PBFT Consensus (hyperledger fabric 0.6).....	101
48	Performance in Transactions Per Second to Process a Single Transaction with kafka Consensus (hyperledger fabric 1.1).....	101
49	Illustration of Linked List Similarities to Blockchain Data Structures.....	103
50	Illustration of a Simulated Blockchain System.....	104
51	Illustration of a Hyperledger Fabric System Connected and Interacting with a Simulated Environment.....	107
52	Image of the Type of Entities that are in the Simulated World.....	109
53	Top Down View of the Simulated Environment.....	110
54	Game View of the Simulated World With Entities Connected to the Real Blockchain System.....	111
55	Octopus Platform Architecture.....	112
56	Octopus Blockchain Architecture.....	113
57	Octopus Invoke Transaction Throughput.....	115
58	Octopus Invoke Transaction Scalability.....	116
59	Octopus Query Transaction Throughput.....	117
60	Octopus Query Transaction Scalability.....	118

List of Tables

Table		Page
1	Structure Definition of a File Provenance Entry.....	13
2	Symbol Table.....	35
3	Confusion Matrix.....	49
4	Design Rationale of Attack Oriented and SGX Enabled Blockchain	59
5	The Specifications of Software and the Version.....	66
6	Overhead for Provenance Data Retrieval.....	70
7	Sample Results of Performance Metrics of Single Transaction Per Second for the PBFT and Kafka Consensus.....	102

1. SUMMARY

The team has conducted basic research on data provenance architecture for cloud using block chain, surveyed the vulnerabilities in block chain, in-depth analysis of the block discarding attack, developed a Proof-of-Stake consensus protocol for cloud based blockchain, architecture for secure Battlefield IoT, cyber supply chain provenance, integration of software guard extensions on distributed ledgers for increased privacy. We have developed the data provenance framework based on block chain that can be deployed in a cloud computing environment. We studied the data provenance challenges in cloud computing environment and documented challenges in linking cloud logging and auditing data, collected from multiple entities, to provide the complete history of changes to data. We also investigated multiple vulnerabilities in block chain. We also focused on the block discarding attack and provided insights into the effectiveness of the attack in cloud environment. We propose techniques to develop access control and key management capabilities within the block chain based data provenance for cloud. We conducted an in-depth analysis of the potential vulnerabilities and attacks of the block-chain technology in the context of cloud for data provenance. We developed probabilistic reasoning-based models to assess the likelihood of the various vulnerabilities and associated risk, and the potential for their exploitation. We investigated concrete and practical mitigations to those attacks, investigate their practical relevance by developing the proper evaluation metrics, and demonstrate such evaluation based on the aforementioned likelihood of those attacks. We also identified vulnerabilities in proof of stake consensus protocol in permissioned block chain network. We have also conducted a systematic study of the attack surface of blockchain systems. We identify attacks attributed to the cryptographic constructs of the blockchain, attacks attributed to the distributed system of the blockchain, and attacks attributed to the application in which the blockchain is built. As a result, we propose various defenses to address each and every of those attacks. Through the rest of the project, we outline new approaches specific to the provenance-in-cloud application using blockchain to address various realities of those attacks. We addressed double spending (equivocation, in the case of provenance in clouds) through the design of a new primitive of one-time signatures.

The team has published 14 articles in conference proceedings, three articles in journals, one magazine article and upcoming book published by IEEE-Press entitled, "Blockchain for Distributed Systems Security". In less than a year, one research article, entitled "ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment has garnered over 100 citations as of to date. In addition to research articles, the team has developed three prototype tools. The Provchain tool provides Blockchain based data provenance capability in the cloud that allows cloud users to track the operations conducted on shared resources. A tool for Blockchain based information exchange provides an anonymous and privacy preserving mechanism to exchange cyber threat information. Finally, the tool for Blockchain based networked identity management provides the ability to detect rogue devices and unauthorized communication among authorized devices. We have disseminated the results of the project to Navy Research Lab, CERDEC, US Transcom, MITRE and Air Force Space Command. We also organized a symposium on Blockchain for Information sharing for NATO and shared the research results with the NATO community. In addition, we leveraged the results from the project to developing a networked identity management prototype for Air Force Space command.

2. INTRODUCTION

Cloud computing is widely adopted by commercial and military environments to support data storage, on-demand computing, and dynamic provisioning. Cloud computing environments are distributed and heterogeneous with a diversity of software and hardware components which are provided by different vendors, possibly introducing risks of vulnerabilities and incompatibility. The security assurance of intracloud and intercloud data management and transfer arises as a key issue. Cloud auditing can only be effective if all operations on the data can be tracked reliably. Provenance is a process that determines the history of a data product, starting from its original sources [1]. Assured provenance data can help detect access violations within the cloud computing infrastructure. However, developing assured data provenance remains a critical issue for cloud storage applications. Besides, provenance data may contain sensitive information about the original data and the data owners. Hence, there is a need to secure not only the cloud data but also ensure integrity and trustworthiness of provenance data. State-of-the-art, cloud-based provenance services are vulnerable to accidental corruption or malicious forgery of provenance data [2].

Blockchain technology has attracted interest due to a shared, distributed, and fault-tolerant database that every participant in the network can share the ability to nullify adversaries by harnessing the computational capabilities of the honest nodes, and information exchanged is resilient to manipulation. Blockchain network is a distributed public ledger where any single transaction is witnessed and verified by network nodes. Blockchain's decentralized architecture can be leveraged to develop an assured data provenance capability for a cloud computing environment. In decentralized architecture, every node participates in the network for providing services, thereby providing better efficiency. Availability is also ensured because of Blockchain's distributed characteristics. Since a centralized authority is frequently used in cloud services, there is a need to safeguard the personal data while maintaining privacy. With Blockchain-based cloud data provenance service, all data operations are transparently and permanently recorded. Thus, the trust between users and cloud service providers can easily be established. Furthermore, maintaining provenance can assist in improving the trust of cloud users toward cyber-threat information sharing [3] [4] to enable proactive cyber defense at a reduced security investment [5] [6].

In this report we present ProvChain, a Blockchain-based data provenance architecture that provides assurance of data operations in a cloud storage application while enhancing privacy and availability at the same time. ProvChain records the operation history as provenance data, which will be hashed into Merkle tree root nodes [7]. A list of hashes of provenance data will constitute a Merkle tree and the tree root node will be anchored to a Blockchain transaction. A list of Blockchain transactions will be used to form a block, and the block needs to be confirmed by a set of nodes in order to be included in the Blockchain. An attempt to modify a provenance data record will require an adversary to locate the transaction and the block. Blockchain's underlying cryptographic theory will allow one to modify a block record only if the adversary can present a longer chain of blocks than the rest of miners' Blockchain, which is quite difficult to achieve. By leveraging the global-scale computing power of Blockchain network, the Blockchain-based data provenance can provide integrity and trustworthiness. In our architecture, we keep the hashed identity of users in order to protect their privacy from rest of the nodes in Blockchain network.

We explored the attack surface of blockchains and possible ways in which this technology can

be compromised. Toward this goal, we attribute attack viability in the attack surface to: 1) the blockchain cryptographic constructs, 2) the distributed architecture of the systems using blockchain, and 3) the blockchain application context. To each of those contributing factors, we outline several attacks, including selfish mining and associated peer behaviors, 51% attack, Domain Name System (DNS) attacks, distributed denial-of-service attacks, equivocation, consensus delay (due to selfish behavior or distributed denial-of-service attacks), blockchain forking, orphaned and stale blocks, block ingestion, wallet thefts, and privacy attacks. We then explore the causal relationship between these attacks and show how one fraudulent activity can lead to the possibility of other attacks. A secondary contribution of this work is outlining effective defense measures taken by the blockchain technology or proposed by researchers to mitigate the effects of these attacks and patching the vulnerabilities in blockchains.

We will also provide analysis of vulnerabilities in Blockchain. We have studied various attack surfaces in Blockchain and have conducted rigorous analysis of the Block Withholding attack and the DDOS attack on mempools. In 2017, Blockchain-based systems have seen a rise in their value. Consequently, they have attracted several forms of denial-of-service attacks, and their attack surface is being widely explored in the fields of security and distributed systems. In this chapter, we present a new form of attack that can be carried out on the memory pools (mempools) of Blockchain systems in general. Towards that end, we study such an attack and explore its effects on transactions fee structures by legitimate users. We also propose countermeasures to contain such an attack. Our countermeasures include fee-based and age-based designs, which optimize the mempool size and help in countering the effects of this attack. We further evaluate our designs by simulations and analyze their usefulness in varying attack conditions. Our analyses can be extended to a wide variety of Blockchain systems using proof concepts, where fees are provided as an incentive of participation.

We focused on security analysis of blockchain-based data provenance in cloud. Blockchain's public and distributed peer-to-peer ledger capability benefits cloud computing services, which require functions such as assured data provenance, auditing, management of digital assets, and distributed consensus. Blockchain's underlying consensus mechanism allows building a tamper-proof environment where transactions on any digital assets are verified by a set of authentic participants or miners. However, achieving consensus demands computational power from the miners in exchange for a handsome reward. Therefore, greedy miners always try to exploit the system by augmenting their mining power. In this chapter, we first discuss blockchain's capability in providing assured data provenance in cloud and present vulnerabilities in blockchain cloud. We model the block withholding (BWH) attack in a blockchain cloud considering distinct pool reward mechanisms. BWH attack provides a rogue miner ample resources in the blockchain cloud for disrupting honest miners' mining efforts, which was verified through simulations.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

In this section, we provide the detailed design and implementation of Provchain. In addition, we also provide the survey of the attack surfaces in Blockchain and detailed analysis of Block Withholding Attack and DDOS attack on mempools.

3.1 Blockchain Background

Blockchain technology has attracted tremendous interest from a wide range of stakeholders, which include finance, healthcare, utilities, real estate and government agencies [8-12].

Examples of applications where technology has the potential to provide are, claims processing, transparency and auditing of operations, identity management, supply chain provenance to address the threat of counterfeit products, and integrity of information acquired from Internet of Things (IoT) devices. Blockchains are a shared, distributed and fault-tolerant database that every participant in the network can share, but no entity can control. The technology is designed to operate in a highly contested environment against adversaries who are determined to compromise. Blockchains assume the presence of adversaries in the network and nullify the adversarial strategies by harnessing the computational capabilities of the honest nodes and information exchanged is resilient to manipulation and destruction. Blockchains facilitate the development of trustworthy networks in a trustless environment.

The premise of blockchain is that applications do not need a trusted central authority to operate and can function in a decentralized fashion. Blockchain enables an exchange of information among distrusting entities. Blockchain enables trustless networks and allows entities to engage in transactions in absence of mutual trust. There is an assumption that a communication medium could be compromised by insiders or outsiders. The reconciliation process between entities is sped up due to absence of a trusted central authority or intermediary. Tampering of blockchains are extremely challenging due to use of a cryptographic data structure and no reliance of secrets. The blockchain networks are fault tolerant, which allows nodes to eliminate compromised nodes.

Blockchains have the following advantage over centralized databases: 1) Ability to directly share a database across diverse boundaries of trust in situations where it is difficult to identify a trusted, centralized arbitrator to enforce constraints of proof of authorization and validity. In a blockchain, transactions leverage their own proof of validity and authorization based on a verification process managed by multiple validating nodes and a consensus mechanism that ensures synchronization. 2) Ability to provide robustness in an economical fashion without the need for expensive infrastructure for replication and disaster recovery. Blockchain requires no configuration to connect and synchronize nodes in a peer-to-peer fashion, built-in redundancy, and no need for close monitoring. It can tolerate multiple communication link failures, allows external users to transmit transactions to any node, and ensures disconnected nodes will be caught up on missed transactions.

Blockchain's distributed database maintains a continuously growing list of records, called blocks, secured from tampering and revision by distributed storage and continuous verification. The blocks contain a temporal listing of transactions that are stored in a public ledger using a persistent, immutable and append-only data structure that is globally viewable by every participant in the underlying peer-to-peer network. When such elegant data structure is considered to track data transactions in a distributed environment, the block structure contains attributes such as the set of user transactions, a timestamp, a reference to a previous block in the blockchain, Merkle root of the transactions, and so on. In this manner, the blocks are linked together to form a chain, where the hash of previous blocks help to maintain the integrity of the whole blockchain

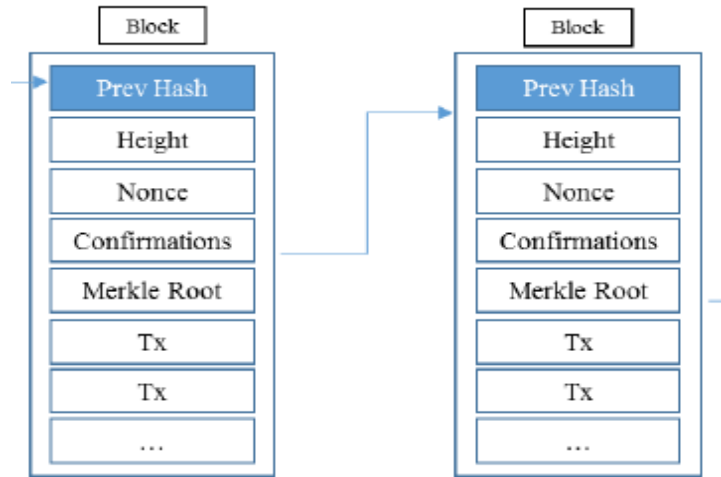


Figure 1. Block Structure

3.1.1 Blockchain Building Blocks

The blockchain technology's effectiveness hinges on the three main components: a decentralized network, distributed consensus, and cryptographically secure algorithms. Fig. 2 illustrates the basic blockchain architecture.

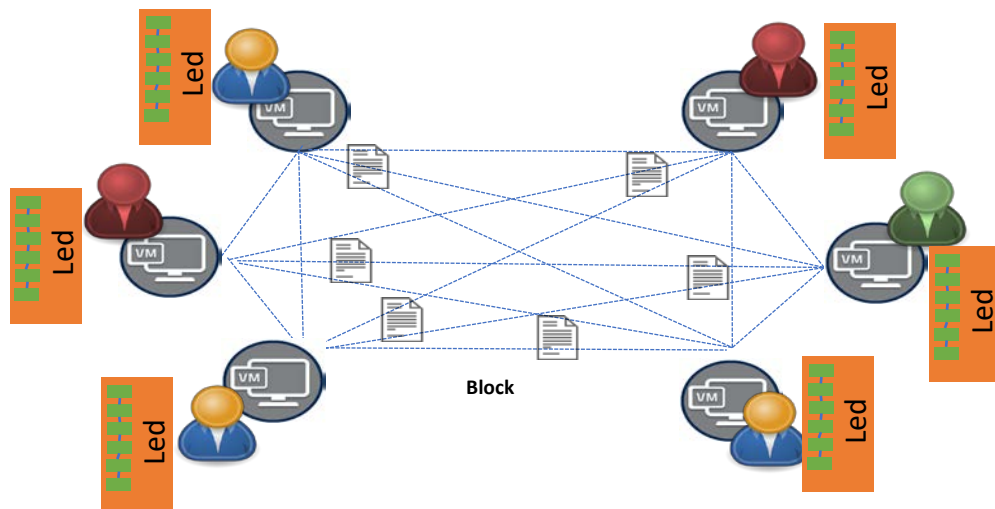


Figure 2. Blockchain Architecture

The key features of each of the components are as follows:

Decentralized Network – The function of the decentralized network is to ensure the propagation of transaction messages among the nodes responsible for maintaining the distributed ledger. The network protocol allows the transaction message to be broadcast from any node to all nodes in the decentralized network. However, the network is not a pure broadcast medium and allows nodes to propagate messages that represent valid transactions. The network can be part of a private or public blockchain that has ramifications on the network performance and security. Irrespective of public or private blockchain, the decentralized network is based on a peer-to-peer architecture. The nodes can join and leave freely. There is no centralized arbitrator.

The network has built-in redundancy and robustness to mitigate against node and link failures.

Distributed Consensus – Blockchain uses consensus protocols over a decentralized peer-to-peer network for verification of transactions prior to adding blocks to the public ledger. The consensus protocol receives messages from the peer-to-peer network and inserts transactions in the distributed ledger. The consensus protocol is responsible for mining blocks and reaching consensus on their integration in the blockchain. The consensus protocol chooses the set of transactions that are accepted after passing a verification process. The verification process is determined by users and does not require a centralized administrator. The consensus protocols ensure that the newly added transactions are not at odds with the confirmed transactions in the blockchain and maintain the correct chronological order. The newly added transactions that are waiting to be confirmed are packed in a block and submitted to the blockchain network for validation.

Cryptographically Secure Algorithm – The foundational component of the blockchain technology is the cryptosystem. State-of-the-art blockchains' cryptography systems use public key algorithms such as Elliptic Curve Cryptography, and message digests such as SHA3-256. In a typical blockchain application, an Elliptic Curve key pair that contains a public key and private key is generated based on Secp256K1 curves. The private key has the traditional usage of being kept secret and utilized to sign transactions. For instance, in the bitcoin use case when a user exchanges bitcoin with another user, the user will sign the transaction with their private key prior to announcing to the network. Once the transaction is signed, the miners in the network will use consensus algorithms to verify the validity of the transaction signature and validation is achieved.

3.1.2 Blockchain Military Cyber Operations Use Cases

Blockchain as a distributed ledger system provides many features and functionalities needed for cyber operations, such as audit of historical information, assured data provenance, guaranteed variability of integrity violations of historical data, and audit of contents from tampering. Besides, blockchain has both cost effectiveness merits, as well as transparency features, making it an appealing system for military cyber operations.

Generation of cyber assets – Blockchain can be utilized to generate cyber assets which will enable applications which rely on direct interaction between customers and assets. The blockchain system can aid in assuring the processes of issuance, transaction processing, housing of cyber assets and identities.

- **Transfer of ownership of cyber assets** – The blockchain system allows transfer of cyber assets between owners by leveraging the property of blockchain so that once a transaction is guaranteed, it cannot be reversed. Any changes will have to be appended and will not result in change of an already validated transaction, thereby ensuring nonreversibility of transfer of ownership.
- **Transparent and assured data provenance** – Every operation on the cyber asset is encoded in the blockchain transaction using a publicly available and immutable ledger. The blockchain system ensures that provenance of every operation on the cyber asset is recorded and traceable.
- **Verifiability and audit** – The distributed ledger keeps track of transactions pertaining to creation and transfer of cyber assets. The tamper-resistant property of the ledger facilitates variability and audit of operations.

Military Cyber Operations – Ensuring traceable and tamper-evident accountability and auditability of command and control, logistics, and other critical mission data among international partners is paramount as our future operations involve the convergence of multiple domains and a heavily contested cyberspace. Centralized or homogenous information systems and databases must evolve distributed, disintermediated, and secure capabilities. As such, trust with respect to operations involving international entities must not be rooted in one single entity. Trust must be decentralized and built around robust, innovative cryptographic paradigms transcending the traditional PKI typically utilized in most homogenous enterprises.

An innovative, distributed trust and identity management mechanism is a crucial enabling technology assuring identification, authentication, and authorization in a way that would further allow disintermediated accountability and auditability. Emerging blockchain and distributed ledger technologies as a whole demonstrate the potential of a truly distributed and disintermediated mechanism for accountability and auditability. The current production application of cryptocurrencies already created unprecedented accountability and auditability in a way that disrupts traditional fiat currencies and disintermediates the way people are able to transact internationally via pseudonymous identity management via wallets in permissionless and public blockchain implementations.

The nuances of disintermediated international partnerships and information exchange involve some mutually exclusive research and development challenges distinct from the permissionless and public implementations of blockchain. For the level of identification, authentication, authorization, accountability, and auditability challenges that encompass overall integrity concerns for international operations, even more robust distributed identity management mechanisms must be investigated. Furthermore, the underlying practice of consensus must vary from the latency-tolerant and performance-heavy implementations observed in cryptocurrencies. Most importantly, the distributed ledger technology mechanism may be private and permissioned; however, it also must not be vested in one single vendor's implementation in a way that inadvertently and ironically intermediates and centralizes information exchanges. We must thoroughly assess and demand interoperability and standardizations among blockchain implementations for mission data that further enables the disintermediated accountability and auditability required in our increasingly complex international operations in multiple domains.

3.1.3 Blockchain Challenges

Blockchain does have the potential to address cyber security issues in distributed systems. However, Blockchain cannot be considered a panacea to address all cyber security concerns. Blockchain does have inherent capabilities to address integrity violations. However, assurance of confidentiality, availability, and authenticity is not guaranteed by Blockchain and will require integration with several security solutions. Organizations who are considering Blockchain to address cyber security issues should carefully assess whether the technology is a good fit. Specifically the below outline is a good start to identify if the organization does need Blockchain to replace existing solution:

- Do you have concerns with the ability of centralized database to withstand failures?
- Are there multiple stakeholders responsible for modifying the contents of the database?
- Do the multiple parties operate under differing trust domains?

- Are there clear defined rules to control data input?
- Is there a value proposition in having validators in form of consensus?

Once the need for distributed Blockchain has been justified, the next steps are to identify the Blockchain solution that's a good fit for the organization. There are several aspects that need to be addressed, such as, Type of data encoded in the blockchain transactions, frequency of transactions, infrastructure used to store the blockchain (public or permissioned), key management system, number of validators, bootstrap time, ability of smart contracts to learn rules dynamically, attack surfaces in the blockchain solution, etc. For instance, for healthcare sector, the type of data stored in the blockchain needs to be carefully identified as any sensitive information can be subject to confidentiality attacks. Any organization that would like to ensure activities of participants in the blockchain are compliant would prefer permissioned blockchain [13].

Below is a summary of key research challenges that need to be addressed for realizing a practical Blockchain solution.

Scalability - Bitcoin's current implementations are not scalable due to the fact that it takes 10 min or longer to confirm transactions and 7 transactions/s is the maximum throughput that can be achieved. There is need for fundamental research to develop a scalable blockchain platform. Prior to developing a scalable blockchain platform, it is imperative to not only define scalability in the context of blockchain, but also identify metrics to quantify scalability. There have been efforts to improve the scalability in blockchain by modifying parameters such as block size and block intervals. However, these efforts to achieve scaling through reparameterization alone can only realize limited benefits and do not address the network performance issues. The network performance is exacerbated due to blockchain's peer-to-peer (p2p) overlay network protocol, degree of decentralization, and number of peers in the network. The throughput of blockchain will depend on the throughput of the overlay network that determines the rate at which blocks propagate and the percentage of nodes involved in the exchange of blocks in a given time interval. For example, if the transaction rate reaches 80% of the throughput, it is quite possible that 10%–20% of the p2p nodes will not be able to render the services and reduce effective network mining power.

There is a need to develop new architectures for blockchain to ensure sufficient scalability without sacrificing decentralization. The architecture should involve protocol design strategies across several layers, namely, Network, Consensus and Storage. There is also need to identify and measure scalability metrics such as throughput, latency, bootstrap time, storage, cost of confirmed transaction, fairness, and network utilization. The architecture will also be designed to address issues such as, "Does exploitation of system parameters to improve scalability sacrifice security properties?" and "What is the degree of resilience of the system during a cyberattack?".

Network Layer – The objective of the network layer in the blockchain architecture is to provide an effective mechanism to propagate transaction messages. The network layer ensures that messages from any participant can be transmitted to all the nodes in the blockchain network. However, the network layer does not operate in a full broadcast mode and nodes exchange messages that contain validated transactions. And, in most current implementations of the blockchain network, the network is heavily underutilized and limits throughput. Thus, the network layer in blockchain is a bottleneck in processing of transactions.

Consensus Layer - The consensus layer is responsible for validating transactions and uses the network layer to deliver messages and record the transactions in the distributed ledger. The consensus protocols include proof of work, proof of stake, and byzantine fault tolerance. The traditional blockchain technology relies heavily on the underlying proof of work (PoW) mechanism to achieve consensus in the decentralized system where the miner has to spend its computational power to solve the crypto-puzzle so as to successfully include its block in the blockchain. With such an approach, miners opt for various specialized hardware to achieve their computational ability. Knowing the eventual goal of the miners is to win the block-adding race so that they can be rewarded, a significant amount of energy is required to do so. For a simple example, if we consider the case of Bitcoin's blockchain, the miners compete to get the reward of 25 Bitcoin, which is worth approximately \$20,000 and is freshly minted for the winning miner every 10 min. Thus, the amount of reward per second is \$33.30 and if we assume the rate of industrial electricity is \$0.01/KW hr, then we can approximately state that the Bitcoin miners use energy of 1100 MW per second. This is substantial quantity that is spent to reach consensus using the PoW approach and most of it is used in computing the irreversible SHA256 hashing function. Since the value of direct incentives will diminish eventually, the critical question of "how will the PoW miners be motivated to mine?" has to be addressed so as to smoothly run the consensus process. Proof of stake (PoS) consensus protocol is an interestingly attractive one, which provides the block inclusion decision-making power to those entities that have stakes in the system irrespective of blockchain's length or history of the public ledger. The principal motivation behind this scheme is to place the power of leader-election in the blockchain update process into the hands of the stakeholders. This is done to ensure that the security of the system will be maintained while the members' stakes are at risk. Roughly speaking, this approach is similar to the PoW consensus except the computational part. Hence, a stakeholder's chances to extend the blockchain by including its own block depends proportionately on the amount of stake it has in the system.

There is a need for developing a customized consensus engine that will not require participants to make significant investments on computation and will balance the tradeoff between numbers of transactions processed, transaction validation time, incentives, and security rules set by participators. The customized consensus engine will choose the optimal combination of consensus protocols to achieve the aforementioned objectives.

Privacy - Permissioned blockchain platforms, such as Hyperledger Fabric and JP Morgan's Quorum, claim that privacy is a goal but the way they achieve it is actually quite limited. These systems consist of several validating nodes, "each" of which sees the entire transaction log "in plaintext". That is, while the systems are designed to provide availability/consistency even when some of the nodes fail, they cannot guarantee privacy if one of the nodes suffers a data breach. Some permissioned blockchain platforms offer a feature where you can create a "private channel" comprising just a subset of these nodes; however, among this subset it is still the case that any data breach would leak the transaction data and then the private channel cannot interact with the other channels. With the existing systems, there is an inherent tradeoff between resilience and expressiveness on the one hand, and privacy on the other.

There are a variety of technical approaches that can provide better operating points. These include threshold cryptography/multi-party computation, zero knowledge proofs, and homomorphic encryption. The theory for these approaches is well established in general, but in concrete terms it is a great open challenge to a) find efficient algorithms for applications of interest, and b) integrate with existing systems. So, focusing on building better privacy mechanisms could be a well-motivated and technically interesting challenge to add.

Security- Despite the advantages of using Blockchain for distributed systems security, there have been numerous instances of reported security risks associated with this technology [14-17]. In 2016, it was reported that an adversary was able to withdraw \$50 million from “The DAO”, a decentralized autonomous organization that operates on Blockchain-based smart contracts [18]. In June 2017, Bitfinex reported a distributed denial-of-service (DDoS) attack that led to a temporary suspension. Several exchanges of Bitcoin and Ethereum (a Blockchain-based distributed computing platform) have also suffered from DDoS attacks and DNS attacks frequently, hampering the service availability to the users

The attack surfaces in Blockchain can be broadly classified into three main categories: 1) Threats associated with the techniques employed for creating/maintaining the distributed ledger (e.g., Blockchain forks, stale blocks, orphaned blocks, etc.), 2) Threats to the Blockchain system’s underlying network infrastructure s, (e.g. attacks on consensus protocols which cause delays, decreased throughput, inconsistencies, DDoS attack, Domain Name System (DNS) attacks, Fork After Withholding (FAW) Attacks, etc.), and 3) Threats associated with frontend/backend applications integrated with Blockchain technology (Stealing of private keys, attacks on certificate authorities, attacks on membership services in permissioned blockchain, Blockchain ingestion, double-spending, wallet theft, etc.).

3.2 Prochain

ProvChain is a data provenance architecture built on a Blockchain that will provide the ability to audit data operations for cloud storage. ProvChain achieves the following four objectives.

- **Real-time Cloud Data Provenance:** User operations are monitored in real time to collect provenance data, which will further support access control policy enforcement [27] and intrusion detection.
- **Tamper-proof Environment:** Data provenance record is collected and then published to the Blockchain network that protects the provenance data. All data on the Blockchain is shared among the nodes. ProvChain builds a public time-stamped log of all user operations on cloud data without the presence of a trusted third party. Every provenance entry is assigned a Blockchain receipt for future validation.
- **Enhanced Privacy Preservation:** Data provenance record is associated with a hashed user ID to preserve privacy so that Blockchain network node cannot correlate data records associated with a specific user. Provenance auditor can access provenance data owned by the user but can never identify the true owner. Only the service provider can link each record with the owner of the record data.
- **Provenance Data Validation:** Data provenance record is published globally on Blockchain network, where a number of Blockchain nodes provide confirmation for every block. ProvChain uses Blockchain receipt to validate every provenance data entry.

To achieve the above objectives, we adopt the below methods to design ProvChain’s architecture.

- Monitor user activities in real time using hooks and listeners so that every user operation on file will be collected and recorded for generating provenance data.
- Store all hashed data operations in form of blocks in the Blockchain. Every node on the

Blockchain can verify the operations by mining the block so that data provenance is authentic and tamper-proof.

- Hash the user ID while publishing provenance data so that the Blockchain network and the provenance auditor cannot determine user identity and the data operations.
- Provenance auditor validates provenance data by retrieving transactions from the Blockchain network by using Blockchain receipt that contains block and transaction information.

3.2.1 Architecture Overview

An overview of ProvChain architecture is illustrated in Figure 3. The following are the critical components of ProvChain.

- **Cloud User.** A user, who owns its data and has sharing relationship on other users' data, may opt for the provenance service, where the provenance data are stored on the public Blockchain. Data changes made by the user can be monitored and validated by Blockchain nodes, but the nodes may not know about details of other users' activities. The provenance data will not expose real user identity.
- **Cloud Service Provider (CSP).** The cloud service provider offers a cloud storage service and is responsible for user registration. A CSP can benefit from our system in the following ways. They can audit the data changes all the time, and they can learn a lot about data operations performed by all the users to better improve their service. Through provenance data, they can also detect intrusion from anomalous behaviors. In addition, they can protect their own daily records, just like normal users. In the business realm, they can gain brand reputation from using Blockchain provenance services because they provide trustworthiness.
- **Provenance Database.** The provenance database records all provenance data on the Blockchain network, which is used for detecting malicious behaviors. All data records are anonymized.
- **Provenance Auditor (PA).** PA can retrieve all the provenance data from the Blockchain and move it into the provenance database as well as validate the Blockchain receipt. The PA maintains the provenance database but cannot correlate the provenance entry to the data owner.
- **Blockchain Network.** The Blockchain network consists of globally participating nodes. All of the provenance data will be recorded in form of blocks and verified by Blockchain nodes.

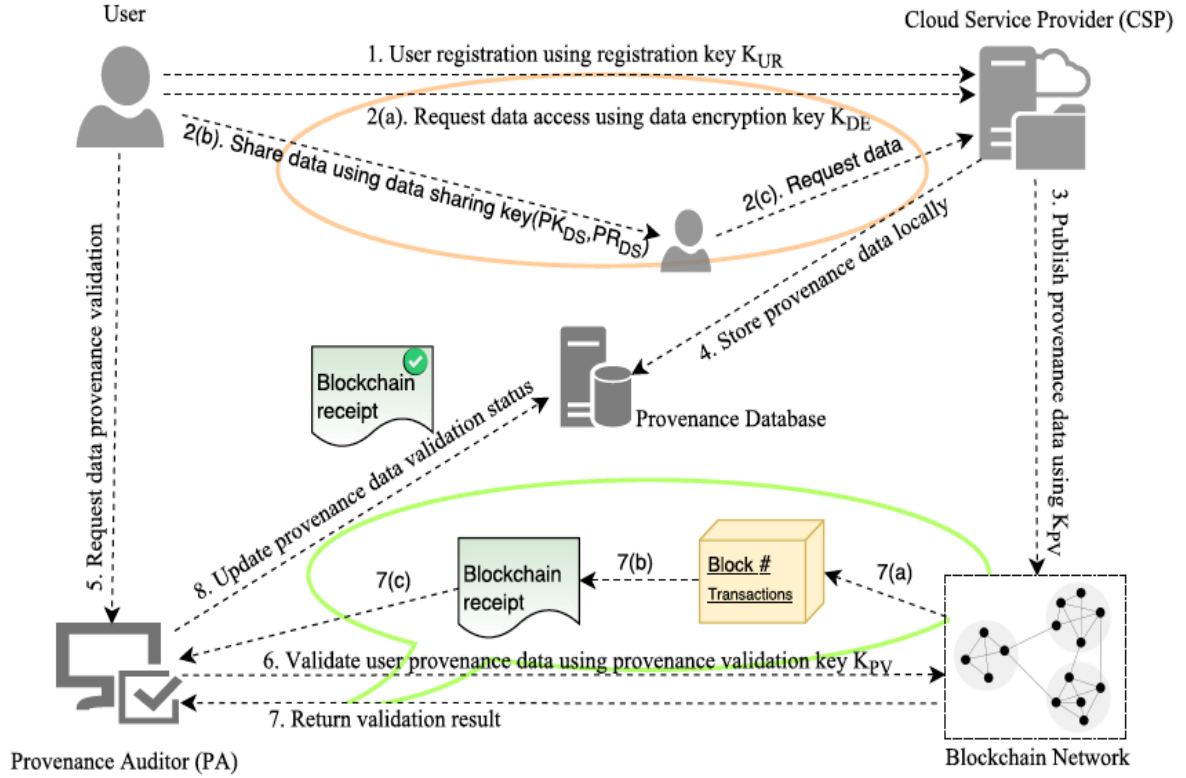


Figure 3. Provchain

3.2.2 Preliminaries and Concepts

ProvChain uses cloud file as data unit and monitors file operations to provide data provenance service. After each file operation is detected, a provenance entry will be generated. The cloud service provider will upload the provenance entry onto the Blockchain network. In this section, we describe the details on file provenance use case and block structure.

File Provenance Use Case. For each file provenance, we can record activities such as file creation, file modification, file copy, file share, and file delete. Examples are shown in Table I. A file can be created by user A, which refers to origin of file X. Then, user A copies file X to another location, probably for backup or other reasons. The read and write operation of user A on file X can also be recorded. If a user B asks for sharing file X from user A, there will also be a record both on user A and user B. User A shares the file X at pre-defined location and user B creates a new file Y from the shared file X. Then, user B can operate on file Y just the same as user A on file X, such as read and write operations. If user B deletes the file, there will be a record for deletion. At some point of time, user A decides to make the file X public so that the file access is changed. Anyone who accesses it will also create a new file at their own respective location. History of files (different versions of file) can be backed up for future use.

Table 1. Structure Definition of a File Provenance Entry

RecordID	DateTime	Username	Filename	AffectedUser	Action	TxHash	BlockHash	Validation
001	2016-9-7 10:00:00	A	X	None	Create at CloudA/A	m-bits	n-bits	True
002	2016-9-7 11:00:00	A	X	None	Copy to CloudB/A/backup	m-bits	n-bits	True
003	2016-9-7 12:00:00	A	X	None	Read at CloudA/A	m-bits	n-bits	True
004	2016-9-7 13:00:00	A	X	B	Share to User B	m-bits	n-bits	True
005	2016-9-7 14:00:00	A	Y	None	Write to CloudA/Y	m-bits	n-bits	True
006	2016-9-9 11:00:00	B	Z	A	Copy from CloudA/A to CloudB/B	m-bits	n-bits	True
007	2016-9-9 12:00:00	B	Z	A	Read at CloudB/B	m-bits	n-bits	True
008	2016-9-9 13:00:00	B	Z	A	Write at CloudB/B	m-bits	n-bits	True
009	2016-9-9 15:00:00	A	X	B	Share X to public	m-bits	n-bits	True
010	2016-9-9 14:00:00	B	Z	None	Delete from CloudB/B	m-bits	n-bits	True

Block Structure. ProvChain uses Blockchain network to provide data record verification and resist against tampering. The block structure is composed of two parts: block header and a list of transactions. The main attributes in the header are block hash, height, confirmations, nonce, and Merkle root. Block hash is computed using the previous block hash and a nonce. The height represents the block index in the global Blockchain network. The confirmation number of the block indicates the number of nodes that have mined this block, and the nonce is used by Blockchain nodes to check the integrity of the block. The Merkle root is the root of binary hash tree created out of all the transactions in a block. Transaction lists come after the block header. Each transaction has a hash, with inputs and outputs. In ProvChain, each data record is hashed into a Merkle tree node. The Merkle tree root node will be anchored to one transaction in a certain block.

3.2.3 Threat Model

Here, we analyze the potential vulnerabilities in ProvChain. The cloud service provider offers data provenance service as well as cloud storage service, which allow the user to store data on the cloud platform and provide the option to enable the data provenance service. The cloud service provider cannot guarantee that data records will remain unchanged due to known vulnerabilities in hypervisors and cloud operating systems. Once the data provenance service is enabled, the user will be able to trace the data, and the provenance auditor is allowed to access all the provenance data. However, the provenance auditor cannot be completely trusted. The adversary can potentially access or modify user data and/or user provenance data. Since ProvChain's main objective is to protect provenance data, we assume that user data stored on the cloud is encrypted and is not accessible to anyone without the decryption key.

3.2.4 Key Establishment

To use ProvChain, users are required to register the service and create their credentials. For cloud storage applications, users generate data encryption key pairs to encrypt their cloud data for confidentiality. If the user wants to share a file, a data sharing key will be provided. For provenance data, cloud service provider generates key pairs to encrypt provenance data for privacy considerations, because provenance data will be further uploaded and published to the Blockchain network. We describe each key below.

- **User Registration Key *KUR*.** In ProvChain, a user needs to register the cloud storage service to store data on the cloud. We denote the key as *KUR*. Every time a user wants to operate on cloud data, the registration key is needed.
- **Data Encryption Key *KDE*.** After registration, the user generates an encryption key

KDE, for encrypting all the data stored in the cloud. When a file is created, the user has the option to encrypt the file, which limits the file access only to key holders.

- **Data Sharing Public/Private Key Pair (*PKDS*, *PRDS*).** For data sharing, a public/private key pair will be generated, denoted as (*PKDS*, *PRDS*). For common cases, the private key is used to generate a signature from the owner, whereas the public key is used by others to verify the data ownership. When users share the data with others, they share the private key for data ownership changes.
- **Provenance Verification Key *KPV*.** Each block on the Blockchain holds several provenance data entries, and provenance data entry is produced upon detection of a file operation. Every data operation will trigger the cloud service provider to generate a key *KPV* to encrypt the provenance data. The key will be shared with PA if the user assigns a provenance auditor to audit the provenance data.

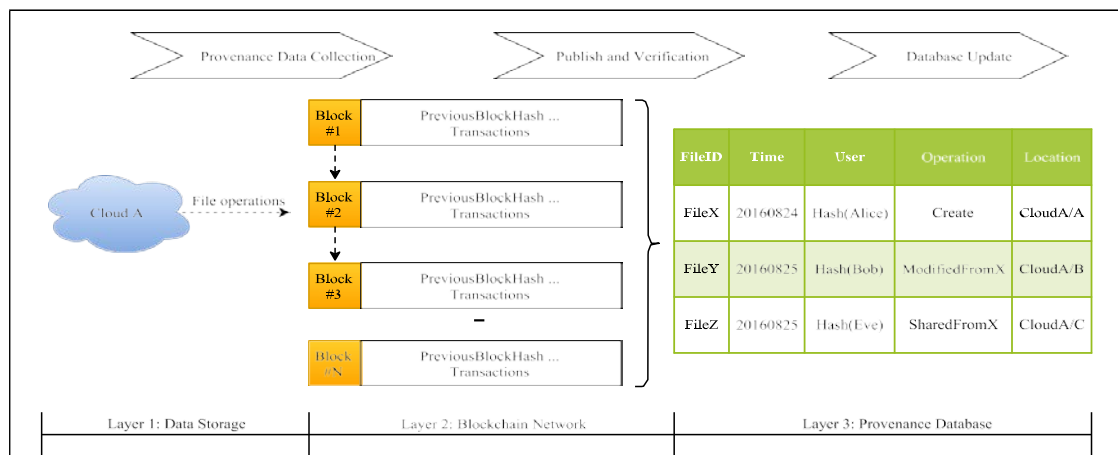


Figure 4. ProvChain System Architecture.

3.2.5 Provchain Implementation

The implementation of ProvChain is conducted using a three-layer architecture, comprising data storage layer, Blockchain layer, and provenance database layer, as shown in Figure 4. The functions for each layer are described as follows.

- **Data Storage Layer.** ProvChain is implemented to support cloud storage applications. Here, we use one cloud service provider, but our architecture can be scaled to multiple providers.
- **Blockchain Network Layer.** We use Blockchain network to record each provenance data entry. Each block can record multiple data operations. Here, we use file as a data unit, so we record each file operation with username and file name. File access operations include Create, Share, Change, and Delete.
- **Provenance Database Layer.** We build an extended database locally for recording the file operation as well as querying. In ProvChain, the service provider can assign a provenance auditor to verify the data from the Blockchain network. The response is a Blockchain receipt that gets validated and appended in the database.

There are three phases in the life cycle of data provenance for ProvChain, namely, provenance data collection, provenance data storage, and provenance data validation.

3.2.5.1 Provenance Data Collection and Storage

Once the user performs actions on the data files stored in the cloud, the corresponding operations get recorded. The operation can be denoted in a metadata, including all the attributes mentioned in Table I. Note that for this phase, only RecordID, Date and Time, Username, Filename, AffectedUser, and Action attributes are recorded. The transaction hash, block hash, and validation field will be collected after provenance auditor queries the Blockchain network. The AffectedUser attribute is considered in two cases. One case is data modification, in which the same user is operating on the data, using the data encryption key, where there are no affected users other than the user itself. The other case is data sharing, in which the user shares a file with someone else. In the second case, the attribute, AffectedUser, in the file operation metadata, will include all the users in the sharing group.

We build ProvChain on top of an open source application called ownCloud [19] to collect the provenance data. OwnCloud is a self-hosted file synchronization and sharing server. ownCloud provides both web-based cloud storage services and desktop client, similar to Dropbox and Google Drive, which provides user control of personal data and universal file access to all of the data seamlessly. In addition, ownCloud is flexible, and developers can utilize their functions to develop various applications on top of it, allowing authorized users to enable and disable features, set policies, create backups, and manage users. The server also manages and secures application programming interface (API) access to ownCloud client and developers, while providing the internal processing engine needed to deliver high-performance, file-sharing services.

In order to collect provenance data, we use hooks to listen to file operations in ownCloud web interface. After an operation is monitored, the record will be generated, which is then uploaded to the Blockchain network and stored in the provenance database. Figure 5 shows the architecture of our provenance data collection and storage.

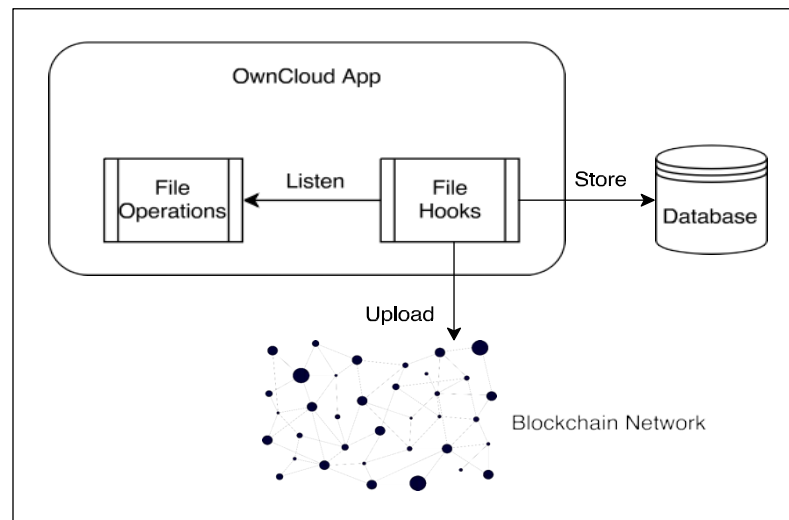


Figure 5. Provenance Data Collection and Storage

We take file change operation as an example to demonstrate the original provenance data in JSON format as follows.

```

{
  "app": "files",
  "type": "file_changed",
  "affecteduser": "test",
  "user": "test",
  "timestamp": "1475679929",
  "subject": "changed_self", "message": "",
  "messageparams": "[]",
  "priority": "30",
  "object_type": "files",
  "object_id": "142",
  "object_name": "66.txt",
  "link": "/apps/files/"
}

```

Blockchain receipt is shown in Figure 6, we use Data API to request the record along with the Blockchain receipt, using the following URL and GET method.

```

{
  "@context": "https://w3id.org/chainpoint/v2",
  "type": "ChainpointSHA256v2",
  "targetHash": "82e46ffd212d680b3e1a169e6a8b59472985ac55398b8740832fe94fd5e5fd63",
  "merkleRoot": "9f0100055a430539796817ce626d84ccb5485453e4d558cf3353e4d4a7e59031",
  "proof": [
    {
      "right": "0f6117e8bddd7fdc713aa5365e74aaf34f5cc31fd654ed84ea37976d873c087"
    },
    {
      "left": "f860e7697ba57d944d925f311cce786e6d20833071d1c16e6e5fef3fc4749c96"
    },
    {
      "right": "de4b5b29183d193b95905ae9741a928ab056cbbefb9a537ac9282fe180c78bd"
    },
    {
      "right": "e75da94bc44a3a9778b2ec7a5ffd58e4a622d4ce4c20676215eb88a4764bb335"
    }
  ],
  "anchors": [
    {
      "type": "BTCOpReturn",
      "sourceId": "0b956b057330591cd63c90e5572ba364c6f9f08299c3e8ee0c893411db1c30a6"
    }
  ]
}

```

Figure 6. Blockchain Receipt

For provenance data storage after data collection, we use Tierion API [20] to publish data records to Blockchain network. Tierion provides Data API primarily for collecting data and managing Datastores and Records in a personal account. Accessing Tierion's Data API requires an API Key, which is required in every request to the Data API. The submission of the credentials should contain the headers X-Username and X-API-Key for each data store owned by the account. In addition to using the Data API to create a Record, we can choose the option to submit an HTML form directly to Tierion, because our ownCloud application is web based and the provenance data is coming from a web site, which is easier to implement. The following URL is used to submit a data record to Blockchain network, by POST method- <https://tierion.com/form/submit>

For privacy consideration, ProvChain hashes user name. In that case, the provenance auditor cannot know to which user each provenance data belongs. Only the service provider can relate each user with the hashed user name because the provider keeps a list of user names. ProvChain also keeps the provenance data in a local provenance database for further update and validation. For publishing data records to Blockchain network, the Chainpoint standard [21] is adopted. Chainpoint is an open standard for creating a timestamp proof of any data, file, or series of events, which proposes a scalable protocol for publishing data records on the Blockchain and generating Blockchain receipts. By anchoring an unlimited amount of data to multiple Blockchains and verifying the integrity and existence of data without relying on a trusted third-party, Chainpoint standard is widely used in Blockchain applications. According to Chainpoint 2.0, data records are hashed so that each Merkle tree can host a large number of records, as is shown in Figure 7. The target hash of the specific record and the path to the Merkle root constitute the Merkle proof of the provenance data, which is a JSON-LD document that contains the information to cryptographically verify that a piece of data is anchored to a Blockchain. It proves the data existed at the time it was anchored. The Merkle root for each Merkle tree is related to one transaction in the Blockchain network.

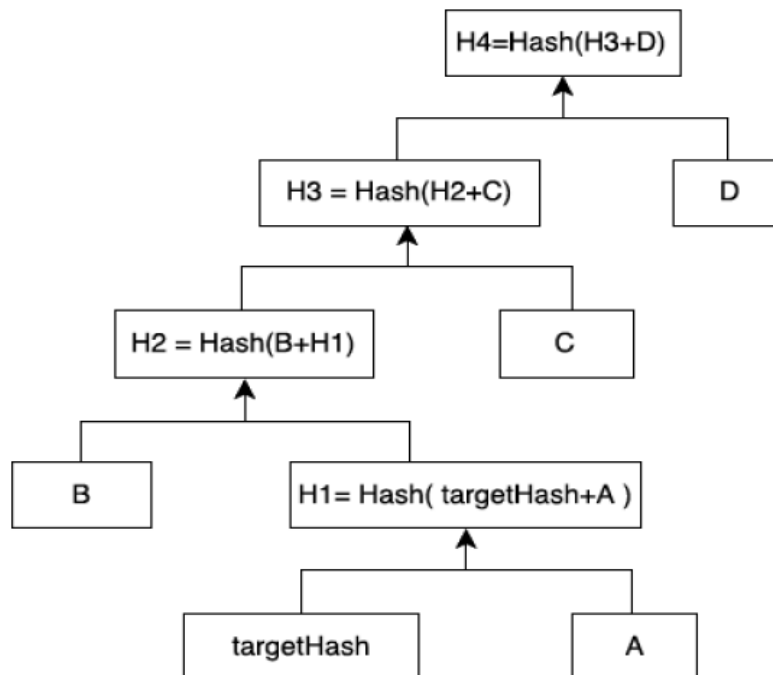


Figure 7. Merkle Tree

3.2.5.2 Provenance Data Validation

To validate the data records that are published in the Blockchain network, the provenance auditor requests the Blockchain receipt via Tierion API. The Data API offers a way to validate Blockchain Receipts. Before validating the <https://api.tierion.com/v1/records/<id>>

The request header should include *Content-Type: application/x-www-form-urlencoded* or *Content-Type: application/json* to set the data format to be received. The requests to the Data API is made over HTTPS. The Blockchain receipt contains information of the Blockchain transaction, and the Merkle proof is used to validate the transaction. Figure 8 is a sample Blockchain and transaction information.

Block	
Block Hash : 00000000000000002cd3c7cbe44e4d95dc1db4511632b79e35889a8fada1f1	
Prev Block : 000000000000000014c5703fc65a02a7be13c3b0af041456c480bf552a6d712	
Next Block : 0000000000000000169ca10626cff308e8bf5d56989bd73d51b870025d9c47	
Merkle Root : b1b8647c3f122cf3505b60dd3e4a4bbc442838bc69d4df1bc0710b990ec33ce6	
Height : 432992	
Confirmations : 1755	
Size : 998,130 Bytes	
Tx Count : 2,634	
Time : 2016-10-05 15:23:38	
Version : 0x20000000	
Difficulty : 392.44 G / 241.23 G	
Bits : 0x18048ed4	
Nonce : 0xf9526b4d	
Relayed : BW.COM	
Transaction	
Transaction	
Txid : 0b956b057330591cd63c90e5572ba364c6f9f08299c3e8ee0c893411db1c30a6	
Size : 234 Bytes	
Height : 432992	
Confirmations : 1754	
Timestamp : 2016-10-05 15:23:38	
Output Scripts:	
- OP_RETURN 9f0100055a430539796817ce626d84ccb5485453e4d558cf3353e4d4a7e59031 (decoded) - OP_DUP OP_HASH160 7003cc5915f6c23d512b38daeeecfdde7a587e9 OP_EQUALVERIFY OP_CHECKSIG	
Transaction	

Figure 8. Transaction and Block Information.

We can reconstruct the Merkle tree from the Blockchain receipt. Each provenance record is stored along with other records in the Blockchain network as a transaction, which is accessible in Blockchain Block Explorer [22]. Because the transaction attribute height represents the block index, we can find the exact block information as well. These data are illustrated in Figure 6. To validate the format and contents of a Blockchain receipt, and to confirm that the Merkle root of one record is stored in the Blockchain, we use the following URL and POST method, provided by Tierion API <https://api.tierion.com/v1/validatereceipt>

Algorithm 1 is used to validate the Blockchain receipt by the provenance auditor. In the algorithm, the proof, merkleRoot, and targetHash in the Blockchain receipt are inputs, and the output is a validation result. If true is returned, then the data record is validated based on the fact that the transaction and block is authentic. If false is returned, it means the block has tampered with and the data record is forged. Note that all of the hashes used in building Merkle trees and proofs are handled in binary format, according to the Chainpoint requirement. The anchors in the receipt indicates how the data record is anchored. Blockchain receipt validation confirms that a receipt's content is valid and true. Specifically, the validation process will confirm the following four elements: (1) the receipt is a well formatted JSON document; (2) all required fields are present; (3) the targetHash, merkleRoot, and proof values are valid; and (4) the merkleRoot value is anchored at the specified location(s).

After the validation of the Blockchain receipt, the provenance auditor can update the data record in the provenance database by filling in the remaining attributes including transaction hash, block hash, and validation result. If the validation result is true, then the provenance auditor can make sure that the provenance data is authentic. If the result is false, then the provenance auditor will report to the service provider that a tamper has happened.

Algorithm 1 Blockchain Receipt Validation Algorithm

```
1: procedure VALIDATE(proof, merkleRoot, targetHash)
2:   nodeNum  $\leftarrow$  number of Merkle tree nodes in proof
3:   h  $\leftarrow$  targetHash
4:   i  $\leftarrow$  0
5:   while i < nodeNum do
6:     if proof(i).key = right then
7:       h  $\leftarrow$  hash(h + proof(i).value).
8:     else
9:       h  $\leftarrow$  hash(proof(i).value + h).
10:    end if
11:    i  $\leftarrow$  i + 1
12:  end while
13:  if h = merkleRoot then return true
14:    return false
15:  end if
16: end procedure
```

3.3 BlockCloud Security Analysis

3.3.1 Exploring the Attack Surface of Blockchain

Blockchain's decentralized and append-only model provides transparency and verifiability without single point of failure. Despite these features, Blockchain is found to be vulnerable to various attacks that are linked to its cryptographic constructs, distributed architecture and the application context. In this paper, we explore the attack surface of the Blockchain along with the possible countermeasures.

The attacks linked to the Blockchain structure include, Blockchain Fork and stale and orphaned blocks. A fork represents a condition in which nodes in the network have diverging views about that state of the Blockchain. These forks can occur unintentionally due to protocol malfunction or unintentionally by implanting Sybil nodes that follow conflicting rules. Blockchain forks can be soft forks which can easily be recovered, or hard forks which can lead to splitting of Blockchain application. Bitcoin cash and Bitcoin gold were created due to hard forks in Bitcoin. While resolving soft forks is easy, hard forks are difficult to resolve. Stale blocks are valid blocks computed in the system which are not accepted in the main chain. Orphaned blocks are valid blocks whose parent hash points to an invalid block. Hence they are also not accepted in the main chain. The issue of stale blocks can be resolved by adding some incentive to the miner who found it. Ethereum applies this policy for stale blocks.

Attacks on Blockchain's peer-to-peer system involve selfish mining, the 51% attack, DNS attack, consensus delay and distributed denial-of-service (DDoS) attack. In selfish mining, the miner computes a block but does not publish it. The miner keeps on computing more blocks in the pursuit of gaining more rewards. When the miner eventually publishes his chain, it invalidates the smaller chain constructed by competing miners, thereby invalidating the chain and the transactions in it.

Selfish mining can be prevented by incorporating block life-time once a block is mined.

The 51% attack or the majority attack can be carried out if a mining pool acquires more than 51% of the hashing power of the network. With 51% hashing power, the pool can invalidate other blocks, prevent other miners from computing a block, and allow double-spending. A possible approach to counter the 51% attack is the use of two-phased proof of work (2P-PoW) that breaks a mining pool's dominance in the network.

DNS attacks can be carried out by isolating the mining pools or users towards a fake network. DNS attacks exploit the peer-to-peer discovery scheme in the network and hijack the nodes by implanting spoofed DNS. DDoS attacks, on the other hand, exploit the block size limit of the Blockchain to exchange low-valued transactions. This attack is also called "penny flooding." In penny flooding, attackers generate dust transactions and occupy most of the space in block. This prevents legitimate users from getting their transactions verified. Penny flooding attack is countered by sorting incoming transactions by mining fee and selecting only the high-valued transactions.

Consensus delay in Blockchain is achieved by injecting fake or false blocks in the network. The intent of consensus delay is to create a possibility of diverging opinion in the network and add delays. Delay can further lead to attacks like double-spending. Application oriented attacks in Blockchain include Blockchain ingestion, double-spending, and wallet theft. Blockchain ingestion exploits the public state of Blockchain to scrape useful information. Double-spending attack involves creating a duplicate of one transaction and sending it different users. Double-spending attacks exploit the delay factor in transaction verification time. It can be countered by using one-time signature scheme for transaction signing.

In wallet theft, the attacker steals the victim's keys. In Bitcoin, wallets are unencrypted by default which makes them vulnerable to wallet thefts. The attack can be countered by using sophisticated encryption for the wallets or getting them insured.

We have explored the attack surface of Blockchain and their possible countermeasures. We classify the attacks based on cryptographic constructs of the Blockchain, its peer-to-peer architecture, and the nature of the applications which use it. In the future, we aim to extend our study by analyzing DDoS attacks on Blockchains and finding better solutions for them.

3.3.2 Countering Double Spending in Next-Generation Blockchains

In Blockchain based applications, transaction verification is a function of block computation time and block size limit. In Bitcoin, the block size limit is 1 MB and average block computation time is 10 minutes. Therefore, a receiver has to wait a minimum of 10 minutes to verify the authenticity of a transaction generated by the sender. For fast transactions, a receiver may deliver the product before the transaction is verified. This creates an opportunity for the sender to reuse the previous transaction and double-spend it. Double-spending involves creating two transactions from same input balance and sending it to different receivers.

To counter the double-spending, we propose a change in the signature scheme of transaction.

Currently, most Blockchain based applications are using Elliptic Curve Digital Signature Algorithm (ECDSA) as signature scheme for transactions. We propose the replacement of ECDSA with one-time, hash-based signatures (OTS). One-time signatures reveal information about the signer's private key if the signer uses it over the same message. Therefore, if the sender generates two transactions from the same key, he risks losing information regarding his private key. The information can be used to forge the key and penalize the sender. One-time signature schemes suffer from two major challenges namely the amount of effort required to forge a key and the key length. Towards that end, we analyze the popular one-time signature schemes including "The Lamppost scheme", "Generalized Bos-Chaum" (GBC) scheme, "Lamport- Diffie-Winternitz"(LDW) scheme and "Generalized Bos-Chaum-Winternitz" (GBCW) scheme."

We introduce a notion of double-spending deterrence (DSD) and model a feasibility game between the double-spender and the forger who tries to replicate his key. The forger wins the feasibility game if he is able to sign a message given an amount of effort with high probability; otherwise, the signer wins. From that, we evaluate the performance of one-time signature schemes in the feasibility game. We use different key sizes for the aforementioned signature schemes to empirically and analytically evaluate the probability of success for the forger.

From our results, we conclude that LDW and GBC schemes are more suited for double-spending deterrence. Under these schemes, if the signer's tries are limited, the forger can forge the key with minimum effort and high probability. We also observe a trade-off between the size of the key and the probability of success. Within GBC and LDW schemes, if the key size is increased, the effort of forgery is reduced and the probability of success increases. We see this as an application specific problem. Blockchain applications that are more sensitive to double-spending attacks can select the appropriate signature scheme with key size which best suit their needs.

With future Blockchain applications such as Bitcoin vending machine, Bitcoin automated teller machine and fast smart contracts, the need for fast transactions will rise. This will increase the risk of double-spending attacks. In our work, we analyze how this threat can effectively be reduced. We analyze the popular signature schemes and show their effectiveness in countering double-spending attacks.

3.3.3 BlockCloud Attack Surfaces

In order to realize a Blockchain cloud, it is important to understand the vulnerabilities of the technology. In this paper, we focus on investigating the need of Blockchain in developing secure cloud system, and we perform in-depth analysis on various possible vulnerabilities in the Blockchain cloud. The vulnerabilities in a Blockchain cloud will primarily arise due to the requirement of computational power to achieve PoW-based consensus. There exists other consensus mechanisms, such as Proof-of-Stake (PoS), Perfect Byzantine Fault Tolerance (PBFT), Proof-of-Activity (PoA), and Proof-of-Elapsed-Time (PoET) etc., which do not necessarily require high computational abilities to mine blocks. However, current public Blockchain implementations rely on PoW to achieve consensus. So, the miners need to be computationally efficient to produce valid blocks as quickly as possible to get rewarded. At the same time, rogue miners may stand against this to disrupt the block mining process by empowering their hash powers.

The Proof of Work mechanism leverages the computational power of the miners to solve an extremely difficult crypto-puzzle. The one who solves the puzzle first becomes the leader of the consensus process, and its proposed block gets included in the Blockchain. Because the participating miners' computational ability are heterogeneous in general, the block generation happens to be asynchronous in most of the slots. However, the probability of finding more than one block is not completely nullified. In this situation, forking of the Blockchain occurs and miners start working on the longest chain they have seen so far. Before the consensus, each miner creates its block by including all transactions that satisfy the following: (1) the transaction originator must have assets in its account higher than that are transferred; (2) the originators' acquired assets are recognized as valid; (3) the recipient of the transaction will be the new owner of the asset; (4) the sender is not reclaiming the already transferred assets.

After verifying the transactions, only the valid ones are included in a block and their Merkle root is included in the block's header, as shown in Figure 2. The header also contains several other attributes such as hash of previous block, timestamp information, difficulty of the puzzle in bits, and the solution of the puzzle or nonce, which are required to compose the crypto-puzzle of the PoW consensus. The puzzle is to find a value (nonce) such that the hash of the block header, calculated by SHA-256 function, is less than a 256-bit target value that has "difficulty" number of zeros in the beginning. Mathematically, it tries to find the nonce (x) by trying difference x such that:

$$\text{Hash}(\text{block_header}(x)) \leq \frac{M}{D} \quad (1)$$

where M is the maximum possible value of the difficulty ($2^{256} - 1$), and D is the current difficulty. It can be observed that as the value of D increases, more trials are needed to find a valid solution. The expected value of required trials needed to find a solution is exactly D .

Although this mechanism provides a robust way to achieve consensus by utilizing the computational power of every participant, it has the following downsides too.

- 1) Reward out of this procedure clearly depends on the cost of electricity for solving such a hard puzzle. Therefore, miner's location also comes into play because this cost varies geographically.
- 2) The puzzle's difficulty is not feasible for a sole user with limited computational resources to mine because it will take years to find a solution, whereas the pool mining allows one to tract the problem in a limited time. Thus, it leads to some sort of centralization in the system, which is far away from the design goal.
- 3) As the mining rewards fade away in the future, it will be less motivating for the miners to devote their computational power for such consensus. The incentive model of this mechanism also needs to be revisited to keep motivating the miners.

Blockchain cloud is realizable, provided the majority of nodes in the network are honest and authentic. However, in the cloud environment, it is possible for rogue nodes to negatively impact the mining and/or consensus processes. In this section, we present the vulnerabilities in

Blockchain cloud.

- 1) *Double-Spending Attack*: A double-spending attack is particularly applicable in Bitcoin technology, where adversaries look forward to using the same digital cryptocurrency for more than one transaction. Because reproducing digital information is easy to achieve, the occurrence of double spending is possible in real-world settings. To conduct such an event, the attackers must have a strong hash power and must be able to generate a longer private chain than public Blockchain. The attack is carried out in the following steps:
 1. Starting from block N , privately mine to extend the Blockchain as much as possible, but do not publicize.
 2. Broadcast the transaction to the organization of interest.
 3. Wait patiently until enough confirmations are received and the transaction is successfully recorded in the Blockchain, so that the merchant dispatches the product.
 4. Secretly mine for extending the private branch until it is longer than the public branch. If it succeeds in doing so, publicize the secret branch that will be eventually accepted to be valid, and the block containing payment to merchant will be discarded.

From the hash rate-based analysis in [23], note that the probability of succeeding in a double-spending attack (a_z) is governed by the following expression

$$a_z = \min\left(\frac{q}{p}, 1\right)^{\max(z+1, 0)} = \begin{cases} 1 & \text{if } z < 0, q > p \\ (q/p)^{z+1} & \text{if } z \geq 0, q \leq p \end{cases} \quad (2)$$

where p, q are hash rate proportion of honest nodes pool and attacker pool, respectively, and z is the number of blocks by which the honest network has an advantage over the attacker. If z becomes negative (<0), then the attacker pool can publicize their blocks and overtake the honest users mined chain. Hence, the following remarks can be made on the success of such attack:

- The success of double spending depends on the hash power of the attacker and the number of blocks (leading or lagging).
- If the attacker's hash power (q) is more than 50%, then it always succeeds.
- If $q < p$, then success probability decreases exponentially.

The above analysis was conducted when the number of confirmations on a transaction did not affect the merchant's decision. However, when the requirement of n confirmations for validating a transaction is enforced, the expression of success probability is the given in Eq. 3.

$$a_z = \begin{cases} 1 & \text{if } q \geq p \\ 1 - \sum_{m=0}^n \binom{m+n-1}{m} p^n q^m - p^m q^n & \text{if } q < p \end{cases} \quad (3)$$

Thus, it is observed that double spending attack is successful for an attacker with any hash rate and the probability decreases exponentially as the number of confirmations increase. Hence, there is no relevancy of "6 confirmations requirement" (often cited) because this value was chosen based on the assumption that an attacker may not have more than 10% hash rate compared with

the rest of the network

2) Selfish Mining Attack [24-25]: Because mining cryptocurrencies such as Bitcoin are difficult for a single miner due to the requirement of high-computing power for solving the crypto-puzzle, a set of miners generally collude to form a pool among each other and share the received reward among themselves after successfully solution [24],[25]. This also helps the individual miners to generate a constant income instead of infrequent (random interval) payment, which occurs when they mine alone. It is argued that incentives to the pool of honest miners can be dominated if there is a pool of selfish miners who intentionally aim to invalidate the work of honest miners by following a selfish mining strategy [24] and generate better revenue for themselves. Similar to the double-spending attacks, in a selfish mining attack, the pool mines on their private chain and publishes it strategically depending on the state of the pool. The states are defined based on the parameter *lead* (difference in lengths of private chain and public chain) and branching (honest and selfish pools are working on different parent blocks). In brief, the selfish mining strategy can be stated as follows, assuming the honest pool always accepts the longest chain:

- If $lead = 2$ and honest pool mines the next block, then publishes the entire private chain.
- If $lead = 0$, part of honest pool works on selfish pool's mined block, and selfish pool mines the next block, then publishes the entire private chain.
- If $lead \geq 0$ and selfish pool mines the next block, then keep the mined block secret.

With the possibility of different states, Eyal and Sirer [24] calculated the following expected revenue (R_{pool}) of the selfish pool:

$$R_{pool} = \frac{\alpha(1 - \alpha)^2(4\alpha + \gamma(1 - 2\alpha)) - \alpha^3}{1 - \alpha(1 + (2 - \alpha)\alpha)} \quad (4)$$

where, α is the hash power of the selfish pool and γ is the proportion of honest miners those choose to mine on the pool's block. Thus the selfish pool's revenue is governed by their hashing power and propagation factor (γ). In general, $0 \leq \alpha \leq 0.5$ must be satisfied in order to avoid 51% attack, however [24] proposed that pool of selfish miners can gain higher revenue if the following constraint on their hashing power (α) is satisfied:

$$\frac{1 - \gamma}{3 - 2\gamma} < \alpha < 0.5 \quad (5)$$

It can be also observed that revenue of each selfish miner will increase if the pool size increases beyond the threshold. As a consequence, most of the honest miners would prefer to join the pool for generating higher incentives and eventually the pool becomes the major player that controls the Blockchain. Thus, the decentralization would not hold any more.

To resolve this issue, the authors suggest that it is necessary to raise the threshold so that no pool can benefit by executing selfish mining. Rather, miners must propagate all the blocks when they learn about competing branches of same length in the blockchain and randomly choose one to mine. In that case, $\gamma = 0.5$, thus threshold gets raised to 0.25. However, raising the threshold to 0.25 still keeps the range open to selfish mining attacks, which may be successful if a pool can be formed with hash power of at least 25% of the total network. An extension to the selfish

mining strategy has been proposed in [26], which states that revenue of a selfish miner can be even higher if it adopts their proposed stubborn strategies.

3) Eclipse Attack [27]: The Eclipse Attack [27] is performed to take the advantage of a P2P network that is used to broadcast information among Bitcoin nodes. In the Bitcoin network, the nodes randomly select 8 other peers to maintain a long-lived, outgoing connectivity for propagating and storing information about other peers. In addition, nodes with public IP can accept up to 117 unsolicited incoming connections from any other IPs at maximum. Thus, the openness and decentralization of the P2P network attracts adversaries to join and perform an eclipse attack, where “the attacker strategically takes the control of all of the incoming and outgoing communications of a victim node”, thereby stopping all connections from other legitimate nodes. The attack is performed by rapidly and repeatedly making unsolicited connection requests to the victim node from the attacker-controlled nodes by sending irrelevant information until the victim restarts. With such effort, there is a high chance that the victim will have the 8 outgoing connections to the attacker-controlled nodes.

In the core of Bitcoin’s P2P network, the network information is propagated through DNS seeders (servers that resolve DNS queries with respective IP addresses) and ADDR messages (that are used to obtain network information from the peers and contain up to 1000 IP addresses). Each node also locally maintains two tables (tried and new) to keep the public IPs. tried table contains the addresses of the peers with whom the node has successfully established connection along with the timestamp information. Whereas, new table contains addresses of peers with whom connection is not yet initiated. When a node restarts or a connections is dropped, the next peer selection follows a probabilistic approach, where an address for $(\omega+1)^{\text{th}}$ connection is chosen from the tried table with following probability:

$$P[\text{Select from tried}] = \frac{\sqrt{\rho}(9 - \omega)}{(\omega + 1) + \sqrt{\rho}(9 - \omega)} \quad (6)$$

where, ρ = ratio of # of addresses in tried and new table.

The eclipse attack takes the advantage of above selection process for monopolizing all connections of a victim node.

- i. Populates the tried table with attacker-controlled nodes’ IP address by sending unsolicited messages
- ii. Overwrite addresses of the new table with garbage addresses (not related to peers’ IPs)
- iii. Once the node restarts, with high probability all the connections are monopolized

4) Block Discarding Attack and Difficulty Raising Attack [28]: The Block Discarding Attack [28] is carried out by an attacker that has a good hold of network connections compared with a normal node. Since propagation of mined blocks is an important characteristic to add it into the mainstream Blockchain, the attacker would preferably place multiple slave nodes to improve its network superiority. With this placement, the attacker could easily get informed of freshly mined blocks and instantly propagate the attacker’s block faster than rest of the network. Thus, when any node publicizes a block, the attacker can immediately dispatch its own mined blocks, so as to discard others’ blocks. However, the difficulty raising attack takes the advantage of attacker’s hashing power to manipulate the difficulty level of the crypto-puzzle. In this attack,

Bahack [28] claims that the probability of discarding a block at depth n [i.e., $(n-1)$ blocks have been mined after this] is p^n , where p is ratio of hash power of attacker and rest of the network. To succeed in doing so, the attacker must wait for enough time.

5) Block Withholding Attack [29]: In the Block Withholding Attack [29], some pool members, who have joined to help mining blocks, would actually never publish any block, thus decreasing the expected revenue of the pool. These attacks are also known as “Sabotage” attacks, where the rogue miner never gains anything but makes everyone loose. However, an analysis with practical instantiation in [30] claims that a rogue miner could also gain profit from such attack.

6) Anonymity Issues in Blockchain Cloud: It has been acknowledged that the underlying blockchain technology of Bitcoin ecosystem is not completely anonymous in nature. The transactions are permanently recorded in the public ledger, hence everybody can see the balance, and transactions related to any Bitcoin address. The real identity and privacy of a user will not get exposed until the user reveals any information during purchase or any special circumstances. Therefore, Bitcoin is pseudo-anonymous, i.e. bitcoin addresses can be created by anyone but tracing back to the real person is not possible unless any

information is found from another source. To maintain a higher privacy and better anonymize the transactions in Bitcoin environment, the users are encouraged to have multiple Bitcoin addresses. Since, the convenience of e-cash system and pseudo-anonymity attract darknet markets to make illegal transactions anonymously, it has been a topic of interest for government and security industries to track down such illicit activities from the publicly available blockchain.

The work presented in [31] focuses to deanonymize the owner of Bitcoin transactions through mapping the Bitcoin address with the IP address of the actual owner. By collecting all of the network level traffic data, including IP information and performing offline processing, they found evidence of tracing back the IP address from the Bitcoin address. In addition to pruning irrelevant transaction data, there are five crucial steps they follow to achieve the mapping: (1) hypothesize an owner IP for each TX, (2) create granular pairings (of Bitcoin address, IP), (3) define statistical metrics for the pairings, (4) identify potential pairings that signify actual ownership, and (5) remove unwanted pairing based on a threshold. Targeting the Bitcoin peers behind NAT or firewall, [32] proposed a generic method to build a correlation between pseudonym of Bitcoin users and their public IPs. The method utilizes the connected user set or entry nodes for identifying the origin of transactions.

The outcome is a list, $\mathbf{I} = \{(IP, Id, PK)\}$, where Id is used to distinguish the clients using same IP and PK is the pseudonyms used in a transaction. Finding the entry nodes (at least 3) and mapping the transactions to these nodes are two important steps, as mentioned in the paper, to effectively deanonymize the clients.

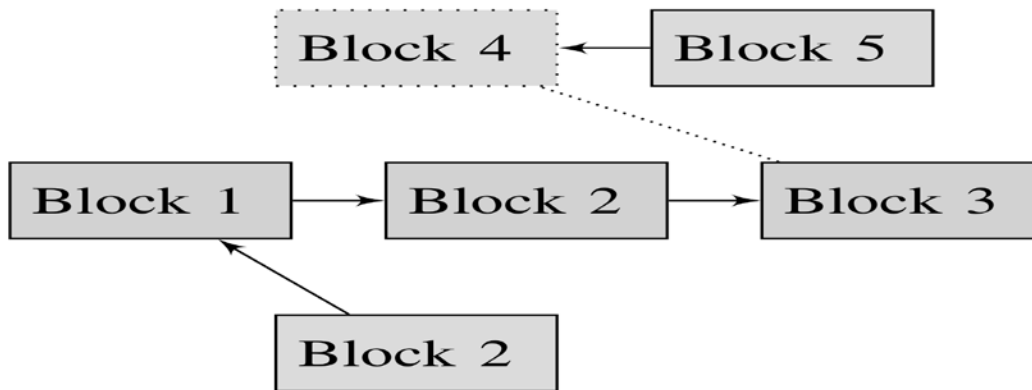


Figure 9. Hard Fork resulting from set of peers following conflicting rules due

7) Blockchain Fork A fork represents a condition in which nodes in the network have diverging views about that state of the Blockchain persisting over long periods of time or even indefinitely. These forks can be created unintentionally through protocol malfunctions or incompatibilities in client software upgrades. Forks can also be caused by malicious intents such as implanting “Sybil nodes” that follow conflicting validation rules or by carrying out “selfish mining” as discussed further in (section 3.4.1). Furthermore, intentional forks can either be soft or hard forks, the latter of which occurs when new blocks that the network accepts appear invalid to pre-fork nodes. On the other hand, soft forks occur when some blocks appear invalid to post-fork nodes. In either case, a Blockchain fork represents an inconsistent state that can be exploited by adversaries to cause confusion, fraudulent transactions, and distrust within network [33]. Figure 9 demonstrates a hard fork example that results from peers following conflicting rules about the state of Blockchain.

When “The DAO” was drained by more than one third of its digital cash by attackers [34], Ethereum used a hard fork to roll back transactions and retrieve millions of dollars’ worth of ether (the “fuel” for the Ethereum network). However, this required consensus by the majority of nodes in the network. In such a scenario, if a consensus delay happens due to a majority attack or DDoS event, fraudulent activities become somewhat difficult to deal with and prolonged delays can ultimately cause devaluation of cryptocurrency.

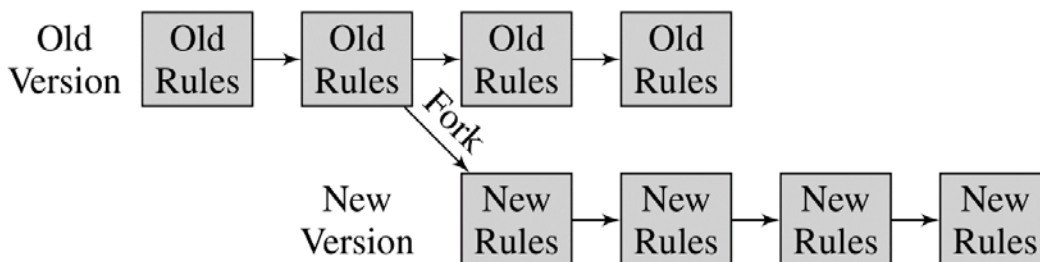


Figure 10. Stale vs. orphan blocks. Note that the stale block (block 2, bottom, and block 4) are valid but they are not part of the Blockchain. Orphan block (block 5) does not have a parent belonging to the Blockchain.

8) Stale Blocks and Orphaned Blocks: Two forms of inconsistencies can occur with the consensus process that can leave valid blocks out of the Blockchain. The first form is a “stale block”, which is a block that was successfully mined but is not accepted in the current best Blockchain (*i.e.*, the most-difficult-to-recreate chain). We will see in (section 3.4.1) that a form of Blockchain attack known as “selfish mining” can lead to the creation of stale blocks in the network, which deprives an honest miner of its reward.

The other form of inconsistency is an “orphan block”: a block whose previous (parent) hash field points to an unauthentic block that is detached from the Blockchain and thus cannot be validated. These inconsistencies can be introduced by an attacker or caused by race conditions in the work of the miners. Stale blocks may be initially accepted by the majority of the network, but they can be rejected later when proof of a longer Blockchain (*i.e.*, the current best) is received that does not include that particular block. Figure 10 demonstrates a chain where stale and orphan blocks can be found. The first orphaned block in Bitcoin was found on March 18, 2015, and that was the beginning of a period in which the most orphaned blocks were created. The trend was reduced in 2016, and from June 14, 2017 to the date of this chapter, no orphaned block has been added to the list [35].

9) Countering Blockchain Structure Attacks: Resolving soft forks in a Blockchain network is a relatively easy process. All peers in the network can come to a consensus about the true state of the Blockchain and resume activities from there. Resolving hard forks can be challenging because conflicting chains can be lengthy with transaction activities dating back to the time of the conflict. Although the stakes of rolling back from a hard fork are high, they can be resolved by the same principle of consensus as discussed earlier. As was the case with Ethereum, a hard fork was used to retrieve money for the investors after The DAO was attacked. Ultimately, the process of solving a fork depends upon the agreement of peers in the network. In Ethereum, stale blocks are actually rewarded and become part of Blockchain (as so-called uncle blocks) [36]. Recently, the number of orphan blocks in Bitcoin has decreased due to the shift towards highly centralized mining networks and thus reducing the high probability of orphan blocks prevalent in decentralized mining networks.

10) Blockchain’s Peer-to-Peer System: The underlying peer-to-peer architecture is the primary reason why certain guarantees are provided by Blockchain, including security. Counterintuitively, this peer-to-peer architecture that the Blockchain resides on actually contributes to several attacks, which we explore in this section.

a) Selfish Mining

As introduced by [37], the selfish mining attack is a strategy opted by certain miners who attempt to increase their rewards by deliberately keeping their blocks private. Rather than releasing them to the public upon discovery, these selfish miners continue to mine their own private blocks to obtain a longer chain than the public Blockchain.

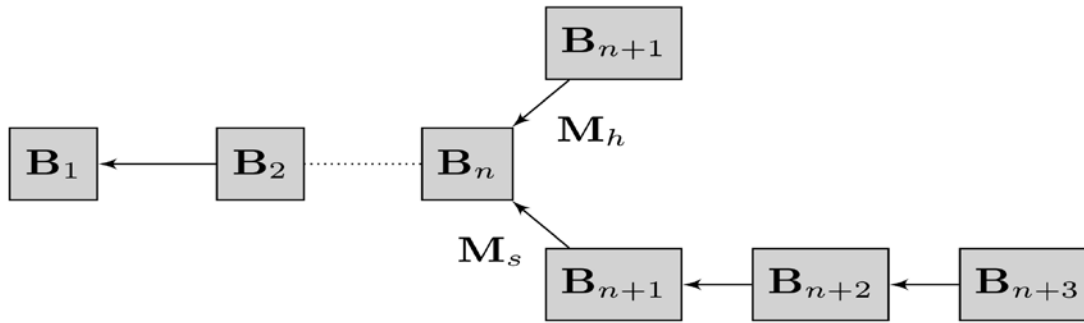


Figure 11. Illustration of Selfish Mining

Once the public Blockchain starts approaching the length of their private chain, the selfish miners finally release their blocks to the public to ultimately claim the reward. Figure 11 demonstrates how a selfish mining attack is carried out.

Consider a Blockchain with blocks ($B_1, B_2, \dots, B_n$). Suppose an honest miner M_h has successfully mined the next block B_{n+1} and in the same network, a selfish miner M_s has also solved $B_n + 1$. Instead of releasing her block, M_s chooses not to publish it and successfully mines two more blocks $B_n + 2$ and $B_n + 3$. Assuming favorable situations for M_h , we show that M_h can still be cheated despite having a majority of network's confidence in their work. Let the hash value of M_h 's block $B_n + 1$ be lower than both the target threshold and M_s 's block $B_n + 1$. If only these two blocks were presented to the network,

M_h 's block would be chosen (due to its greater computational complexity) over M_s 's block and appended to the public Blockchain. After some time, M_s releases all of her blocks $B_n + 1$, $B_n + 2$, and $B_n + 3$. Due to the design protocols of Blockchain, the network will invariably shift to M_s 's blocks and discard the block $B_n + 1$ of M_h . The effort put forth by M_h in computing its block will be entirely wasted due to selfish behavior of M_s . The incentive in adopting this selfish mining strategy is maximizing block rewards by publishing a longer chain. It should be noted that excluding the M_h 's block $B_n + 1$ from the Blockchain does not destroy the block, rather it leads to another significant problem in the network known as "stale blocks" (section 3.3.2).

Selfish mining attacks can produce undesirable results for the rest of the network by invalidating the blocks of honest miners who contribute to the Blockchain. In a case where two selfish miners compete to add their chains to the network, the chances of a "Blockchain fork" arise (section 3.3.1). These forks can cause a delay of consensus in the network, which can further lead to other potential attacks such as "double-spending", as discussed in (section 3.5.2). One selfish activity in the network has the potential to disrupt the overall network, and therefore it is imperative to study their relationship with one another.

b) The 51% Attack

The 51% attack happens when a single attacker, a group of sybil nodes, or a mining pool in the network attains the majority of the network's mining hash rate to manipulate the Blockchain. As such, the attackers would be able to 1) prevent transactions or blocks from being verified (thus making them invalid), 2) reverse transactions during the time they are in control to allow double-

spending, and 3) prevent other miners (verifiers) from finding any blocks for a short period of time. Having a majority of the hashing power at their disposal, the attacker's blocks will be appended to the Blockchain with higher probability compared to others [38]. Also, these blocks can possibly have fraudulent or double spent transactions. For example, if an attacker performs a transaction in exchange for any product with Alice, it can replicate the same transaction with Bob and put it on the block. Transactions on Blockchains are not reversible, and only one transaction can be considered valid among the two.

A 51% attack is not beyond the realm of possibilities. In July 2014, the mining pool "GHash.IO" held over 51% of the hash rate for a day in the Bitcoin [39], which raised many concerns in the press and media about Bitcoin and its vulnerabilities. "GHash.IO" later shrunk in size when miners left its pool and eventually closed in October 2016. It should be noted that mining pools do not always need 51% of the network's hashing power to carry out the fraudulent activities mentioned previously. Even with less hashing power, similar targets can be achieved with a significant probability of success. As shown in Bahack *et al.*[40], majority attacks are possible with one quarter of the network's hashing power.

c) DNS Attacks

When a node joins the Bitcoin network for the first time, it will be unaware of the active peers in the network. To discover the active peers (identified by their IP addresses) in the network, a bootstrapping step is required. The DNS can be used as a bootstrapping mechanism, and DNS seeds are queried by nodes upon joining the network to obtain further information about other active peers. As pointed out in the developer's guide of Bitcoin [41], DNS opens a wide attack surface to the Bitcoin network, namely, the DNS resolution is vulnerable to man-in-the-middle attacks (resolver), cache poisoning, etc. As a result, by using the attack surface of the DNS, an adversary can potentially isolate Blockchain peers (by feeding them with an invalid list of peers), feed the peers with fake blocks, invalidate transactions, etc. For more on DNS security in general, the reader may refer to the work in [42].

d) Distributed Denial of Service Attacks

One of the most common attacks on online services is the DDoS attack [43],[44]. Blockchain technology, and despite being a peer-to-peer system, is still prone to DDoS attacks. Blockchain-based applications, such as Bitcoin and Ethereum, have repeatedly suffered from these attacks [36], [45-48]. DDoS attacks manifest themselves in a number of ways, depending on the application nature, network architecture, and peers behavior. For example, in the Bitcoin network, the 51% attack can lead to denial of service. Specifically, if a group of miners acquire a significant hashing power, they can prevent other miners from adding their mined blocks to the Blockchain, invalidate ongoing transactions, and cause service failure in the network. Intentional forks; forks that are the result of malicious behavior; can turn into hard forks, resulting in similar outcomes of denial of service.

Another possibility for the attack is due to the limited number of transactions per block the Bitcoin network can process in a given time. For example, on average, it takes the Bitcoin network 10 minutes to mine a block, which has a maximum size of 1MB. Although the size of transactions in Bitcoin varies, the average size of a transaction in Bitcoin is approximately 500 bytes, allowing approximately 2000 transactions per block on average—the maximum number of transactions

added to a block in Bitcoin is reported to be 2210 [35]. Furthermore, the average time needed to mine one block, based on the predefined difficulty, is approximately 10 minutes. As such, for all current transactions in the network to be successfully included in the Blockchain, their number may not exceed 200 transactions per minute. Taking that into account, and the fact that each transaction requires a minimum of two peers (identified by two different public identifiers) to be involved in a transaction, the total active peers served by the network per minute (*i.e.* where a block containing their transaction will be mined) will not exceed 200 peers.

An adversary may exploit the aforementioned operational reality of the Bitcoin system by introducing Sybil identities; the same adversary also may control multiple wallets. Furthermore, using those identities, the adversary may issue several dust transactions (*e.g.*, 0.001 BTC per transaction) between the various Sybil identities under his control. By introducing a large number of transactions of small value over a short period of time, the network will be congested by creating blocks containing those transactions, and service to legitimate users in the network will be denied. As a result of this congestion the adversary may as well launch other attacks; *e.g.*, double-spending of tokens not mined due to the congestion.

One may argue that miners may choose which transactions to include in a block. However, this is discouraged by design in Bitcoin, as outlined by Satoshi. Blocks today even include transactions of value as low as 0.0001 BTC, which makes flooding the network with low-value transactions possible.

e) Consensus Delay

Another attack associated with the peer-to-peer nature of the network is the consensus delay. In this attack, an attacker may inject false blocks to delay or prevent peers from reaching consensus in the Blockchain. Such delays can be introduced by either forcing the network to mine small blocks with a small coin-base, or by forcing it to spend time to reach consensus on corrupted blocks. In particular, because accepting or rejecting false blocks can be time consuming, this process allows misbehaving nodes to game the system. The problem is further exacerbated for time-critical applications, where resolution needs to be achieved within a short period of time.

f) Countering Peer-to-Peer Attacks

Prior research has been conducted to address the problem of selfish mining, and researchers have suggested several possible solutions [50],[51],[52],[53],[54]. Solat and Potop-Butucaru [55], proposed a “lifetime” for blocks that prevents *block withholding* by selfish miners. If the expected lifetime of a block expires (calculated by the honest miners), it is rejected by the network. Heilman [52], impedes the profitability of selfish miners by introducing a defense scheme called “Freshness Preferred.” Heilman [52], builds on top of the previous work by Eyal and Sirer [51], by adding unforgeable timestamps to blocks and prefers blocks with more recent timestamps compared to older ones. Their work reduces the incentive for selfish miners to withhold their blocks for long periods of time.

Majority attacks have also been widely discussed with countermeasures proposed to overcome a monopoly in Blockchain networks. Bastiaan [39], introduced the concept of “two phase proof-of-work” (2P-PoW) to counter 51% attacks. 2P-PoW is a continuous-time Markov chain (CTMC)

model that incorporates two challenges for miners to solve instead of one. The states of the CTMCs prevent the pool from increasing beyond an alarming size by shrinking incentive for miners in the pool. 2P-PoW prevents large pools from creating a hegemony.

Johnson *et al.* [56] proposed a game theory model to address DDoS attacks against mining pools. Other countermeasures include putting a cap on the minimum amount in the transaction that a sender can have or increasing the block size to accommodate more transactions. Yet another approach is to reduce the target difficulty in mining blocks so that more blocks can be mined with no transactions going to waste. Each of these propositions have their own caveats. To prevent DNS-based attacks, there is an enormous amount of literature that can be used to equip the Blockchain system with DNS attack defenses [57],[58],[59]. The dimensions we explored in this chapter encourage additional research in Blockchain technology in the areas regarding DNS and DDoS attacks.

11) Application Oriented Attacks: The Blockchain and associated peer-to-peer system are separate from the application services using them. Based on the nature of the applications, they will have their own shortcomings. Thus, we expect a significant number of attacks based on the application, which we address in this section.

a) Blockchain Ingestion

Blockchain is a public ledger, making it accessible to anyone who wants to mine it. As such, the analysis of the public Blockchain can reveal useful information to an adversary. This process is known as Blockchain ingestion and it might not be desirable to the Blockchain application or its users. For example, a credit card company in the open market can use data analytics to delve into public information on the Blockchain and optimize its own schemes to compete with the digital currency. To demonstrate the potential exploitation of the public data, Fleder *et al.*[60] used graph analysis to create links between Blockchain data of Bitcoin and associated identities of the users.

b) Wallet Theft

Where credentials, such as keys, associated with peers in the system are stored in a digital wallet, the “wallet theft” attack arises with certain implications on the application. For example, in Bitcoin, the wallet is stored unencrypted by default, allowing an adversary to learn the credentials associated with it and the nature of transactions issued by it. Even when a wallet is safely guarded on a host, launching a malware attack on the host will allow the adversary to steal the wallet. Finally, with many third-party services enabling storage of wallets, those services can also be compromised and the wallets can be leaked to an adversary [34].

c) Countering Application Oriented Attacks

Application-oriented attacks have various possible countermeasures. For example, to secure blocks, it is advised to keep backups of the wallet and secure the keys used for signing transactions. Passwords are easy to compromise, and using a strong password is required. However, changing passwords does not change the keys secured by them, making those keys vulnerable due to a previous compromise of the password. Wallet encryption, a standard practice in the original Bitcoin design, is highly recommended to cope with vulnerable keys. Other mechanisms to cope

with wallet security include insurance, which technically does not address the problem by remedying its consequences.

New models of cryptocurrency, such as “Zcash”, hide the transactions and maintain its anonymity on the Blockchain, thus preventing the block ingestion attack. The double-spending attack is easily addressed in fast networks, but not when the network is characterized by high latency. One possible approach to deal with the problem is utilizing one-time (or a few time) signatures, such as XMSS [49], [61]. However, this requires the change in the current signature algorithms that Blockchain applications have used. Other proposals include reducing the difficulty parameter of Blockchain to enable swift block mining, which is a reasonable approach, except that it would further facilitate selfish mining.

3.3.4 Block Withholding Attack

We model a critical issue named block withholding attack, which may occur in Blockchain cloud during pool mining, to identify the constraints on attacker’s hashing power in order to defeat the purpose of pool mining. The scenario is analyzed individually when a pool operator uses different rewarding schemes, such as proportional reward and Pay-per-Last-N-Shares (PPLNS) reward. After understanding the important security vulnerabilities related to the Blockchain implementation, we undertake one important case called block-withholding attack and analyze the situation rigorously to understand the strategies of a powerful attacker that led to the introduction of bringing anomaly in the mining pool. The attacker joins in the pool with intentions to withhold the successful blocks and look for opportunities to specifically demotivate the honest members to mine in the pool. We theoretically analyze the attacker’s strategy by considering different types of pools whose rewarding schemes are different.

3.3.4.1 System Model

In this work, we consider a pool P , where n miners work continuously trying to solve the cryptographic puzzle using their hashing power. The pool is assigned with a pool operator, who is responsible for collecting transactions from the network, creating a block, keeping track of puzzle difficulty, recording the number of shares submitted by pool members, dispatching the successfully mined block to the P2P network, collecting and redistributing the reward among miners. The pool members are considered to be honest in nature, which means, they report their solutions or shares to the pool operator immediately as they find. Assuming the miner i has hashing power α_i of total mining power (M), where $0 \leq \alpha_i \leq 1$, the overall mining power of the pool is $B = \sum_{i=1}^n \alpha_i$. For simplicity, we assume that $\beta < 1$, thus the pool is not the only computing entity but there exists other solo miners or pools whose computational power is fixed with respect to the pool P . Additionally, the miners of pool P receive reward based on their contributions in a round, where a round is defined as the interval between two valid blocks found. The contribution of the miners is calculated based on the number of shares reported, and each share could be a full solution with probability $1/D$, where D is the overall difficulty of the crypto-puzzle which is assumed to be fixed. Finding a share is inevitably easier than finding the valid solution because the share is only meant to prove a miner has worked enough to find it. Thus, the difficulty for finding a share is determined by the pool operator, which is less than D . As a standard, a particular hash is a valid share with

probability $\frac{1}{2^k}$, where k is specified by the private blockchain creator so that $0 \ll 2^k \ll D$. The description of all the symbols used in the rest of the paper are given in Table II.

Table 2. Symbol Table

Symbol	Meaning
α_i	Hash power of miner i
α_A	Hash power of attacker
D	Difficulty of PoW puzzle
R	Block Reward
$\mathcal{M}$	Total mining power of the pool
δ	Percentage of reward for pool operator
κ	Difficulty required for a block to be a share
$h(.)$	Function determining number of shares produced by a miner
B	Average number of blocks found in last N shares
T	Mean amount of time required to find a block

On the other hand, the attacker is considered to be a powerful miner who willingly participates in the pool P and aims to maximally damage the mining activity of the pool by withholding successfully mined blocks. The attacker joins the pool P with initial hashing rate of α_A but we assume that the attacker has ability to augment its mining power by incorporating additional physical ASIC resources or leasing computing power from cloud vendors. Although, such efforts for increasing hashing power may be costly by itself, the attacker's irrationality makes it feasible by our assumption. In this chapter, we aim to understand how much of extra mining power a block withholding attacker may need to completely sabotage a pool, thus leading to an situation in which no single miner would prefer to honestly mine in the pool. This analysis would then help to perform a cost-benefit analysis of leasing cloud/non-cloud computational resources for successful attack. By saying sabotaging a pool, we mean that the attacker comes up with such strategies that dominate the reward distribution in the pool, therefore leading to a situation in which other members' reward variance out of pool mining is more than the variance out of solo mining.

3.3.4.2 Augmenting with Extra Hashing Power

For increasing the hashing ability of a miner, there are different possible options available to undertake. The first option is mining hardware. Mining has progressed from the era of CPU to GPU and finally to the Application Specific Integrated Circuit (ASIC) era. Currently, miners choose ASIC chips to mine blocks because these give a great amount of hashing power at a minimum cost. ASIC chips are designed purposely for block mining; hence, they cannot be used for any other task. The ASIC mining hardware offers $50\times$ to $100\times$ increase in hashing power while reducing the power usage by $7\times$ compared with previous technologies. Avalon [62] manufactures ASIC mining chips for the Bitcoin miners' market where each server can process 3.65 TH/s at a power efficiency of 0.29 W/GH.

The second option is renting mining services from cloud providers, which is called cloud mining or cloud hashing. There are three ways to perform remote mining: (1) Hosted mining, (2) Virtual hosted mining, and (3) Leased hashing power. In case of hosted mining, the user leases a machine that is capable of mining and hosted by the provider on cloud. In virtual hosted mining, the user creates a mining environment from scratch on the virtual private server. In the leased hashing power technique, which is mostly used in current scenarios, the user leases the required amount of hashing power from the provider without any hassle of managing the infrastructure. This allows users to purchase hashing capacity from the cloud providers' mining hardware that are already installed in their data centers. This process enables another viable way to get rid of the issues related to installing mining hardware, managing electricity consumption, or network connectivity and bandwidth requirements. Such services exist in practice, and providers, such as Hashflare [63], Genesis Mining [64], Hashnest [65], and Eobot [66], offer competitive prices in exchange for hashing capabilities.

3.3.4.3 Disruptive Attack Strategy Analysis

Considering the attacker is going to withhold the valid blocks only, it may publish the shares that are not exactly solutions of the crypt-puzzle. Thus, the only way an attacker can do damage to the pool is to take away as much reward as it can by submitting sufficient number of shares. Since mining pools are characterized by the schemes they adopt to reward the participating miners, the attacker's disruption strategy may vary accordingly, which we discuss in the following section.

1) Proportional Reward – Proportional Rewards is a very naive scheme [67], in which the total reward is divided proportionately according to the number of shares each miner contributed in that round of competition. The round in this context means the interval between finding two successfully mined valid blocks. As a pool operator receives a challenge from the Blockchain network, the competition round starts, where it assigns the task to participating miners in the pool. The members utilize their hashing ability to solve the crypto-puzzle. Upon finding a valid block, the honest members usually forward it to the pool operator, who then broadcasts in the Blockchain network. The pool operator receives a fixed reward R if the network of miners reach a consensus on its mined block, and from this point the next round starts. The pool operator may keep a fixed percentage of the reward, and the remaining is then distributed among the members in proportion to the number of *shares* each miner has contributed, with respect to total number of *shares* received in that round. Assuming that δ portion of reward is reserved for the pool operator, the pool members share the total reward of $(1 - \delta)R$. Now, to estimate the expected number of *shares* in a particular round of competition, we assume that the time taken to find a *share* by miner i is exponentially distributed with parameter α_i . Thus, the expected time of finding a *share* is $\frac{1}{\alpha_i}$, which can be a full solution with probability $\frac{1}{D}$. Considering the round lasts for T units of time, the miner i could produce $\alpha_i MT$ number of hashes on average in that round. However, the total number of *shares* the miner produces is modeled as the function $h(\alpha_i MT)$, where $h(.)$ is monotonically increasing function with respect to i 's mining power, i.e. $\frac{\delta h}{\delta \alpha_i} > 0$. Thus, the total number of *shares* submitted to the pool operator can be represented as-

$$H = \sum_{i \in \mathcal{P}} h(\alpha_i \mathcal{M}T) \quad (7)$$

So, the expected reward received by miner i out of the pool that adopts the proportional reward scheme is:

$$U_i = \frac{(1 - \delta)Rh(\alpha_i \mathcal{M}T)}{H} \quad (8)$$

Block withholding attack in the proportional reward pool: When a malicious miner with hashing power α_A joins the pool, then the inherent power of the pool increases to $\sum_{i=1}^n \alpha_i + \alpha_A$. Thus, the number of shares submitted in a round increases from H to H' , where $\sum_{i \in \mathcal{P}} h(\alpha_i \mathcal{M}T) + h(\alpha_A \mathcal{M}T)$. As the hashing power of attacker increases, its share contribution in the pool also increases proportionately. Since the attacker never submits valid blocks (solutions) rather withholds them, the length of the round does not depend on its mining power. The goal of the attacker is to impose a maximum disruption to the pool members so that pool mining becomes no longer profitable for them and forces them to leave the pool eventually. There are two different ways an attacker can affect the block mining of pool members: (1) eclipsing the blockchain network, where attacker could control the network connections and hence manipulate the victims mining activity directly, (2) increasing hashing power, which could produce more number of shares and hence overall reward received may dominate over rewards to all other members. We have chosen to analyze the second scenario, where the attacker could get extra mining power to dominate in the pool and increase the reward variance of pool members.

To start absorbing rewards, the attacker needs to know the amount of extra hashing power on top of $\alpha_A \mathcal{M}$ it requires to generate x more shares because the only way to decrease the payout of pool members is to submit more shares to the pool operator. Now, if the attacker generates extra x shares, the rewards to the pool member i can be represented as:

$$U_i = \frac{(1 - \delta)Rh(\alpha_i \mathcal{M}T)}{H' + h(x)} \quad (9)$$

This means, payoff of miner i is inversely proportional to the mining power of the attacker and therefore its net reward goes down if the attacker generates more shares. However, it is not possible to make the utility of honest miners very close to zero because it will need an enormous amount of shares to generate. Pursuing that would equivalently require large computing ability which may not be possible for an attacker. Instead, we plan to find the lower bound on x that will still paralyze the miners to be better off with solo mining. Thus, the attacker's optimization function can be characterized as:

$$x^* = \underset{x}{\text{minimize}} \left[\sum_{i=1}^n \frac{(1 - \delta)Rh(\alpha_i \mathcal{M}T)}{H' + h(x)} \right] \quad (10)$$

Subject to,

$$\text{prop_var}(U_i) \geq \text{var}(U_{solo}^i) + \epsilon, \forall i \in [1, \dots, n]$$

Where, $0 < \epsilon \ll 1$ is a very small amount that signifies the point at which the payout variance out of pool mining crosses the variance of solo mining. $\text{prop_var}(U_i)$ is the payout variance of miner i per each share while mining in the pool and $\text{var}(U_{solo}^i)$ is the payout variance per share while mining solo. The constraint signifies that if $\text{prop_var}(U_i)$ is higher than the that of solo mining, then the attacker successfully demotivates them to not participate in the pool. If the variance is high, then the reward will not be consistent, which is what the attacker wants to achieve.

Now, we find the variance of payout for miner i when mining solo with constant hash rate of α_i . In case of solo mining, the reward is given if it finds an actual block. Thus, the expected reward out of solo mining is $\text{Pr}(\text{finding a block})R = \frac{\alpha_i R}{2^{\kappa D}}$. Block finding as a solo miner can be characterized as a Poisson process with rate parameter $\lambda = \frac{\alpha_i}{2^{\kappa D}}$. Hence total expected reward that solo miner i can receive by mining for T amount of time is the following.

$$U_{solo}^i = \frac{\alpha_i T R}{2^{\kappa D}} \quad (11)$$

From the property of Poisson distribution, we know that variance (σ^2) is same as the rate parameter (λ). Hence, the variance of payout from solo mining can be

$$\text{var}(U_{solo}^i) = \frac{\alpha_i T R^2}{2^{\kappa D}} \quad (12)$$

To model the payout variance per share in pool mining, we first need to consider an appropriate distribution for total number of shares produced in the pool. Now, considering the $N = H' + h(x)$ as the random variable representing the total number of shares reported in the pool during a round, $P_r(N)$ represents the corresponding probability distribution function (PDF). The average value of reward per share to miner i in the round can be

$$\mathbb{E}[U_i] = \sum_{N=1}^{\infty} \frac{(1 - \delta)R}{N} P_r(H' + h(x) = N) \quad (13)$$

where, $\frac{(1-\delta)R}{N}$ is the per share reward as we found earlier. Now, the expected squared payout can be defined as:

$$\mathbb{E}[U_i^2] = \sum_{N=1}^{\infty} \frac{(1-\delta)^2 R^2}{N^2} Pr(H' + h(x) = N) \quad (14)$$

Hence, the payout variance can be expressed as:

$$prop_var(U_i) = (1-\delta)^2 R^2 \sum_{N=1}^{\infty} \frac{[Pr(N) - (Pr(N))^2]}{N^2}$$

Now, to find the concrete value of the payout variance, we can model the total number of shares using different standard probability distribution functions such as geometric and negative binomial. These PDFs provide near natural model to estimate the total number of shares in a round [67].

2) Pay-Per-Last N-Shares (PPLNS) Reward - Since proportional reward scheme is very naive in nature, it suffers from pool-hopping attack, where malicious miners could strategically choose different pools to mine in a round to obtain maximum reward. Therefore, several advanced schemes are proposed to avoid such scenario and PPLNS reward scheme one of them that supposedly resists the pool-hopping nature of greedy miners. Unlike proportional reward scheme, which is a round based, PPLNS considers temporal share submission activities to reward the pool members irrespective of rounds in the mining process. In this scheme, the reward is distributed among the miners who recently submitted their shares no matter how many actual blocks are found in the considered interval.

The simple variant of PPLNS is to set a threshold for total recent shares to N and the total reward is distributed among based on the last N shares submitted. Now taking the same notations into account and considering total B blocks are found during the last N shares, the payout per share (U_{pplns}) can be represented as:

$$U_{pplns} = \frac{(1-\delta)RB}{N} \quad (15)$$

where, B follows the Poisson distribution with mean $\frac{N}{2^{kD}}$. In this variant, it is assumed that the difficulty D and reward R are kept constant. Altogether, RB defines the total expected reward received by the pool operator during the last N shares.

The pool employing PPLNS reward function typically works in the following manner. The operator keeps track of the history of at least last N submitted shares irrespective of the round they

have been reported. Therefore, the rounds in this case are interdependent on each other. The ordering of shares is maintained using a sliding-window of length N , which later is used to divide the total reward proportionately among the contributors. Considering the sliding window at k^{th} share as $s_k = \{s_{k-N}, s_{k-N+1}, \dots, s_k\}$ the reward for miner i can be expressed as:

$$U_{pplns}^i = (1 - \delta) \times \frac{\#\{s_j : s_j \in s \text{ and } s_j = i\}}{N} \times RB \quad (16)$$

In the above expression, the total reward received in the window of last N shares is divided proportionately among each pool member $i \in P$, who submitted their shares in the considered window.

Block withholding attack on PPLNS reward pool:

Similar to the previous reward scheme, one malicious miner is introduced in the PPLNS pool who aims to disrupt the normal operation of pool mining via accumulating as much reward to itself. Unlike the strategy adopted in proportional reward pool, the attacker would have to opt for a different strategy in the PPLNS pool because submitting as many shares may not be the optimal case anymore due to the fact that only last N shares will be considered for the reward distribution irrespective of the rounds in which they are submitted. If we consider that there are N slots to place N shares, the attacker can opt for the following strategies to disturb the fairness of pool reward system. First the attacker with mining power α_A must mine at a faster rate compared to other pool members, where successfully mined blocks are withheld. Next, the shares are published in such a way that they reside in the window of last N shares, so that the majority of reward is returned to the attacker instead of the pool members.

The attacker's reward can equivalently defined as per Eqn.(16), where number of shares it contributes in the last N slots is dependent on its hashing power.

$$U_{pplns}^A = (1 - \delta)RB \left[\frac{\Phi(N, \alpha_A)}{N} \right] \quad (17)$$

where, $\Phi(.)$ is a function that models the average number of shares the attacker can contribute in the window of N slots. The function Φ depends on the hashing power of attacker and follows a Poisson distribution with mean parameter of $\lambda = \frac{\alpha_A MN}{2^k}$. At the same time, the random variable B also follows a Poisson distribution with mean $\frac{MN}{2^k D}$. However, due to the fact that the attacker withholds the successfully mined blocks, the mean value of B will exclude the hashing capability of the attacker. So, the mean of Poisson distributed random variable B will be $\frac{(1-\alpha_A)MN}{2^k D}$. Hence, the expected reward for the

$$\mathbb{E}[U_{pplns}^A] = (1 - \delta)R \frac{\alpha_A (1 - \alpha_A) \mathcal{M}^2 N}{2^{2k} D} \quad (18)$$

Understanding the overall utility of the attacker, its goal would be to maximize this quantity with

the help of additional hashing power from external sources. Compared to the optimization goal defined in the proportional reward pool, where the goal was to minimize the other pool members' reward by finding an optimal amount of computational power, here the objective is to maximize its own reward in the window of N -slots by submitting as many shares so that other members' net reward will automatically decline. Now, considering the attacker comes up with additional yM amount of computational power into the pool, then the net objective function of the attacker can be defined as follows:

$$y^* = \underset{y}{\text{maximize}} \left[(1 - \delta) R \frac{(\alpha_A + y)(1 - \alpha_A) \mathcal{M}^2 N}{2^{2\kappa} D} \right] \quad (19)$$

Subject to,

$$pplns_var(U_i) \geq var(U_{solo}^i) + \epsilon, \text{ for an } i \in \{1, \dots, n\}$$

The constraint posed in the above optimization problem is similar to the proportional pool but here we are looking for one such miner whose reward variance from pool mining is more than solo mining. If that case happens, then miner i would prefer to leave the pool and mine alone. Therefore, this process can be iterated to eliminate miners one by one and thereby harming the pool mining process. Now, to define the variance of a miner i from PPLNS pool when the attacker has incorporated extra yM amount of hashing power, the expected reward for miner i needs to be formulated, which is $\frac{(1-\delta)R\alpha_i M' N}{2^{\kappa} D N} = \frac{(1-\delta)R\alpha_i M'}{2^{\kappa} D}$. Due to the presence of block withholding attacker, the total mining power of the pool is modified to M' , which is defined as $(1 - \alpha_A - y)M$. Hence the corresponding reward variance can be presented as:

$$pplns_var(U_i) = \frac{(1 - \delta)^2 R^2 \alpha_i (1 - \alpha_A - y) \mathcal{M}}{2^{\kappa} D N} \quad (20)$$

The reward variance out of solo mining will be same as defined in the previous subsection, where the period T will be replaced with N

3.3.5 DDOS Attack on Mempools

There are two types of DDoS attacks on Blockchain-based systems. In the classical attack, the attacker exploits the size limitation of Block (e.g., 1MB) and the throughput of the network (3–7 transactions per second) and generates dust transactions to occupy the space in the block and prevent other transactions from mining. This type of DDoS attack has been addressed by the research community [68] and there are countermeasures adopted by the miners to prevent it. The miners prioritize the transactions based on fee and select the ones that pay higher mining fee as an incentive. As such, this prevents the dust transactions from occupying space in the block and preventing legitimate transaction from mining. The other form of DDoS attack is the attack on the

mempools of the Blockchain system. In this attack, the attacker floods the mempool by generating a series of unconfirmed dust transactions. Although these transactions may be rejected by the miners eventually, their presence in mempool creates another major problem. In Blockchain systems in general, the size of mining pool determines the fee paid to the miners. If the mempool size is big, miners have a limited choice of mining the transactions and as a result the users try to prioritize their transactions by paying higher mining fee. One such attack on mempool was carried out recently on a popular public Blockchain system, when the mempool was flooded with over 115k unconfirmed transactions resulting in the rise in the mining fee and transaction confirmation time substantially [69].

3.3.5.1 Data Collection for Evaluation

To observe the relationship between the size of the mempool and the fee paid by the users, we used a public dataset provided by the company called “Blockchain” [70, 71] which is a leading software platform that keeps record of digital assets and Blockchain systems measurements. For our study, we gathered the dataset of mempool size and fee over approximately a year of one such popular and widely deployed Blockchain systems. In Figure 12, we plot the results obtained from the dataset and we use the min-max normalization to scale our dataset in the range (0, 1). The min-max scaling is conducted as:

$$z = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad (21)$$

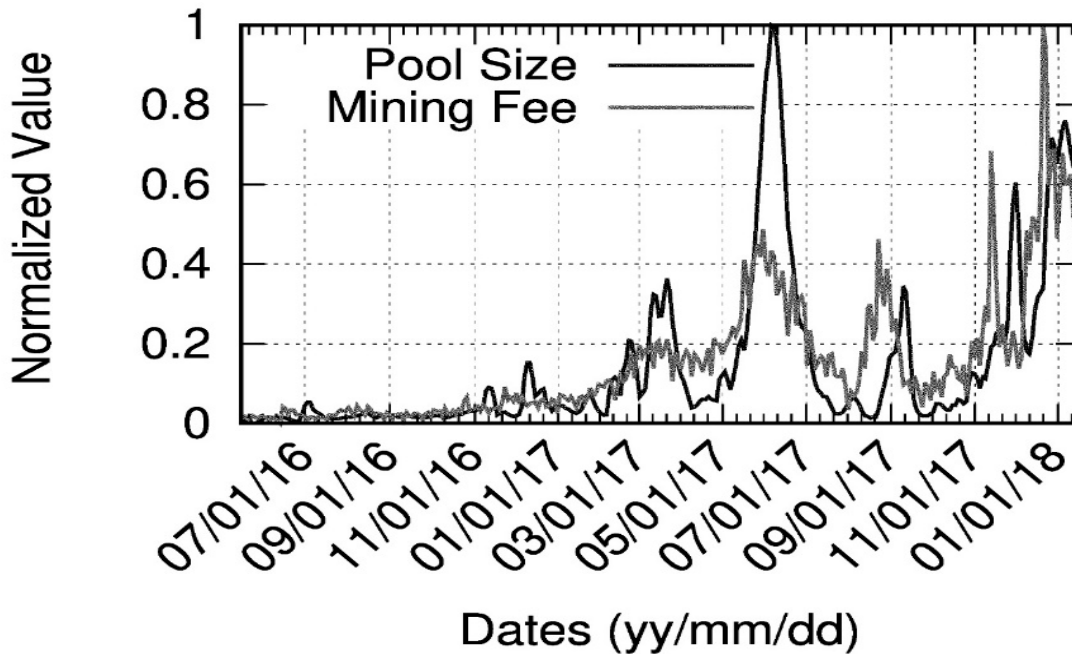


Figure 12. Temporal study of mempool size and mining fee paid by the users in a popular Blockchain system.

3.3.5.2 Threat Model

In this work, we assume that the attacker is a client with a complete Blockchain and a memory pool at his machine. The attacker has spendable transactions denoted by “UTXO’s”; those transactions have been previously mined into the Blockchain. In many Blockchain systems, transactions can be split into various small transactions [72]. We assume that a balance in the attacker’s possession is large enough that it can be split into fractions of dust transactions; each of those transactions is at least capable of providing the mining fee as an incentive to miners. We assume also that the attacker controls a group of Sybil accounts, each with multiple public addresses. These public addresses could be used to exchange transactions during an attack. The attacker and the Sybil accounts have an apriori knowledge of each other’s public addresses. Furthermore, the attacker and Sybil have client-side software and scripts [73, 74], which enables them to initiate a flood of “raw transactions” [75] in a short time span. We assume that Sybils (collectively) have a capacity of exchanging transactions at a much higher rate than the network’s throughput [76]. Although, being a full client in the network, we also assume that the attacker does not have the capability of mining transactions. This means that the attacker does not possess enough computational power to mine a block in PoW-based setting, discard a transaction, reverse a transaction or delay other transactions from being mined. Moreover, the attacker does not have control over other legitimate users in the network and, as such, cannot prevent them from broadcasting their transactions and accessing mempool nor other resources in the network. The attacker is also constrained by a “budget”: since every transaction requires a minimum fee to be relayed to the network, the fee limits the number of transactions that an attacker can generate.

Attacker’s Goal. The end goal of the attacker is to flood mempools in the network with *unconfirmed transactions*. The attacker will broadcast dust transactions at a higher rate than the throughput of the network. At mempools, the arrival rate corresponds to the flow of incoming transactions and the departure rate corresponds to the rate of transaction mining (posting on the Blockchain system). The departure rate is fixed, because the average block computation time and the size of the block are currently fixed. When the arrival rate increases due to a flood of dust transactions, it results in transactions backlog. As the queue size grows, the mempool size increases accordingly. Overwhelming the mempool size alarms the legitimate users, who naturally start paying higher mining fee to prioritize their transactions.

After flooding the mempools, the secondary objective of the attacker will be to reduce the cost of attack by getting his transactions rejected. For the attacker, mining transactions will result into losing the mining fee to the miners. However, if the transactions get rejected, the attacker will have yet another chance to carry out the same attack. In the prior related work, the target of attack is either a mining pool [77] or the Blockchain itself [78, 79]. In our threat model, the targets of attack are the mempools in the system as outlined in Figure 1. The victims in every attack are the benign users in the network who are denied of normal service. Another distinguishing feature of this threat model is that the attacker does not want his transactions to be mined. In the analysis performed by Baqer *et al.* [78], the intent of the spam attack was to flood blocks in the Blockchain by exploiting the block size limit; such an attack requires transaction mining in Blockchain. While there is a limit on the block size in Blockchain systems, there is no limit on the size of mempools and flooding them does not require transaction mining. Furthermore, the attack exploiting block size can be effectively countered by miners while the attack on mempools cannot be countered in the same way. As transaction mining involves mining fee while the attack objective is only mempool flooding, therefore for this attack, transaction acceptance in a block remains undesirable for the

attacker. According to the taxonomy of DDoS attack [80], mempool flooding attack can be characterized as a “semantic attack of variable rate.”

3.3.5.3 Attack Procedure

As mentioned earlier, when the rate of incoming transactions exceeds the network’s throughput, a backlog of unconfirmed transactions builds up at the mempool. As backlog grows, competition for transaction mining also increases. Users try to prioritize their transactions by offering more fee to the miners. As a result, the fee per transaction paid to the miners increases. To facilitate usage, public Blockchain systems are anticipated to provide online services to estimate mempool size and the average fee paid [81]. Based on that, a fee is recommended for users who want to get their transactions confirmed within a desired time.

Since the mempool size affects the way users pay the mining fee, this creates an attack possibility for an adversary to exploit the mempool size and create panic among legitimate users. When a legitimate user sees the mempool size growing, the user, as a rational agent, will try to prioritize his transactions by adding more mining fees to them. Dust transactions of an attacker will eventually be rejected by the miners, to protect Blockchain from spam. Although it protects the system from spam, that policy in itself also works in favor of the attacker, since the attacker loses no fees as a result of enforcing this policy. On the other hand, legitimate users end up paying more than the required fee to get their transactions confirmed. Upon rejection of transactions, the attacker can re-launch the attack multiple times to flood the mempools.

As shown in Figure 2, there is a high correlation between the mempool size and the transaction fee paid to the miners. During 2017, it was reported a spam attack of unconfirmed transactions which led to higher mining fee was launched on a popular Blockchain [29, 69]. From Figure 2, it can be observed that during this timeframe of the attack the mempool size was much larger than the average size. As a result, the mining fee pattern also followed similar growing trend as the mempool size. In December 2017, the problem of mempool flooding was highlighted by crypto analysts [84, 85] suggesting that it was an attempt to increase the mining fee and drive the users away from adoption and participating of the Blockchain system. To further establish a relationship between the mining fee and the mempool size, we computed Pearson correlation [86] on our dataset. The Pearson correlation coefficient is defined as:

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} \quad (22)$$

We observed a high correlation of 0.69 between the mempool size and the mining fee. As a result, we conclude that overwhelming the mempool size can also lead to other problems in the Blockchain. Delay in transaction verification can create multiple problems, including possibilities of *equivocation* and *double-spending* [87].

To sum up, and as described in the threat model, the objective of the attacker is to maximize the size of the mempool and minimize the cost of the attack. The cost of the attack is the fee paid to the miner if a transaction gets mined in a block. The fee consists of the relay fee and the mining

fee. Higher fee increases the priority of transaction and chances of a transaction mining [88]. To avoid that, the attacker will design his transactions in a way that they are less likely to be prioritized by miners. At the same time, the attacker wants his transactions to stay in the mempools for as long as possible. To this end, we envision that this attack can be carried out in two phases: the *distribution phase* and the *attack phase*.

3.3.5.4 The Distribution Phase

In the distribution phase, the attacker estimates the minimum relay fee of the network, divides his spendable transactions (“UTXO’s”) into various transactions and sends them to the Sybil accounts. This can be done in two ways. 1) The attacker may generate a dust transaction from previous UTXO, send it to a Sybil account and get in return a new transaction (unspent). The attacker uses the new transaction as new input and repeats the procedure multiple times for all Sybil addresses. 2) An alternative way is to use the spendable transactions and generate a series of outputs to all the addresses of Sybil nodes. Unlike the previous method, this will result in only one transaction to all the Sybil outputs. Transactions of this nature are known as “**send many**” transactions [89] because the user is sending transactions to various addresses within one transaction. Since the aim of the attacker is to generate as many transactions as possible, he will not opt for “send many” option. All transactions to the Sybil addresses will be generated *independently*. The transactions made in the distribution phase will have input “UTXO’s”, which will be previously mined in the Blockchain. Hence, these transactions will have greater-than-zero age and will be capable of paying the minimum mining fee.

3.3.5.5 The Attack Phase

Once the distribution phase is completed, all Sybil accounts will have sizable balance. In the attack phase, all Sybil will carry out “raw transactions” [75] from the balance received in the distribution phase. Sybil will generate dust transactions and exchange them with each other. To maximize the severity of attack, they will prefer to have one recipient per transaction. The rate of transactions will be much higher than the throughput of the network. As a result, the arrival rate of the transactions at the mempools will be higher than the departure rate of mined transactions. This will increase the transaction backlog and the size of the mempools over the duration of the attack. The attack will be carried out until all the spam transactions get into the mempools. The transactions made in the attack phase will have the transactions of distribution phase as input “UTXO’s”. These inputs will still be awaiting confirmation in the Blockchain. Due to that, their confirmation factor or age score will be zero.

3.3.5.6 Attack Cost

As mentioned earlier, one of the objectives of attacker is to minimize the *attack cost*. To be able to achieve that, the attacker requires transactions to be part of the mempool but not part of the Blockchain system. This can be achieved by adding the minimum relay fee R_f to each transaction but not the minimum mining fee. The relay fee is necessary for a transaction to be broadcast to all peers in the network and be accepted by the mempool. If the attacker adds the mining fee, his transactions will attain priority from a miner and might get mined. To avoid that, the Sybil only pay the relay fee. If a transaction has i inputs, where each input contributes a size of k Bytes, and

o outputs, where each output contributes a size of l Bytes, then the total size of the transaction and its associated cost are determined by (1) and (2), respectively:

$$\begin{aligned} S(\text{Bytes}) &= (i \times k) + (o \times l) + i \quad (1) \\ C(\text{cost unit}) &= R_f \times \frac{S}{1024} = R_f \times \frac{[(i \times k) + (o \times l) + i]}{1024} \quad (23) \end{aligned}$$

Assuming that the attacker is limited by a budget B and minimum value set by the network as T_{min} , then, using (2), the total number of transactions T_a that the attacker can generate can be computed in (3).

$$T_a = \frac{B \times 1024}{R_f \times T_{min} \times [(i \times k) + (o \times l) + i]} \quad (24)$$

Now we look at the system from standpoint of a legitimate user. A legitimate user who intends to get his transaction mined into the Blockchain pays relay fee for transaction broadcast and mining fee as an incentive to the miner [90]. For such a user, contributing T transactions, the cost incurred per transactions and the total cost of all transactions can be derived using (4) and (5).

$$C(\text{cost unit}) = [R_f + M_f] \times \frac{[(i \times k) + (o \times l) + i]}{1024} \quad (25)$$

$$T_l(\text{cost unit}) = T \times [R_f + M_f] \times \frac{[(i \times k) + (o \times l) + i]}{1024} \quad (26)$$

As mentioned in the threat model (section 10.4), the aim of the attacker is to increase the cost per transaction paid by the legitimate user (4). A legitimate user will aim to have his transactions mined, therefore he will pay relay fee and a high mining fee. The attacker will only aim to get his transactions into the mempool and eventually not get mined, so he will only pay the relay fee. In these settings, the maximum loss an attacker can incur would happen if all his transactions get mined. The cost in such a case will be equal to the product of the total number of transactions and the relay fee. The attacker can re-launch the same attack with a new balance of $B - (T_a \times R_f)$.

3.3.5.7 Countering the Mempool Attack

To counter DDoS on Blockchain's mempool, we propose two countermeasures that leverage the nature of incoming transactions and prevent spam on the system. One of the effective countermeasures against spam attack in Blockchain is to prevent the transmission of dust transactions in the network. We envision that if mempools can discard spam transactions and stop relaying them to other mempools (nodes in the Blockchain), the pool size can be effectively controlled, and spam can be countered. The existing countermeasures require increasing the block size or reducing the confirmation time to increase the throughput. We argue this is not a useful strategy: if the block size is increased to accommodate more transactions, attackers can increase the number of dust transactions to occupy even more space in the block. A better solution would be to filter spam transactions, which we explore in the rest of this chapter.

3.3.5.8 Fee-based Mempool Design

Any design that aims to optimize the size of mempool would require filtering of spam transactions upon arrival to nodes. As the threat model states, an attacker only intends to relay spam transactions between the mempools and does not want them to be mined. To achieve this goal, the attacker only pays minimum relay fee in transactions so that mempools accept and relay them. To prevent the transactions from mining, the attacker does not pay the mining fee. We use this insight to construct a “*Fee-based Mempool Design*”, as shown in Algorithm 1.

For this design, we assume that the mempool is initially empty when transactions begin to arrive at the node. We also assume that each incoming transaction has its associated relay fee and mining fee. We also fix a threshold beyond which the mempool starts spam filtering. Initially, when the transactions arrive in the pool, and for each transaction, the mempool checks if the transaction pays a minimum relay fee. If the transaction pays the minimum relay fee, it is accepted and the mempool size is updated. As the transactions get added into the mempool, the size of mempool grows. When the size reaches a threshold, the mempool starts applying the fee-based policy. Now, if the incoming transaction pays both the minimum relay fee and the minimum mining fee, only then it is accepted in the mempool. The key idea behind this scheme is that only those transactions should be accepted, which eventually get mined into the Blockchain. As a result, this technique puts a cap on the incoming transactions and filters spam transactions, thereby reducing effectively the mempool size. If the new size is less than the baseline size threshold then the mempool can proceed its operation of relay fee check. Otherwise, it will continue with the fee-based design.

Analysis of Fee-based Mempool Design. In the following, we will analyze the workings of fee-based design and its utility in light of our threat model. We will limit the number of transactions an attacker can generate within his budget by increasing the mining fee threshold. We also observe how this design affects other legitimate users within the same network.

Algorithm 1: Fee-based Mempool Design

Inputs : incoming transactions;
minimum relay fee;
minimum mining fee;
Threshold Size;
Output: Mempool Size
State: Mempool Empty

```
1 foreach transaction  $\in$  incoming transactions do
2   while (Mempool Size < Threshold Size) do
3     if (transaction relay fee > minimum relay fee) then
4       Mempool  $\leftarrow$  transaction
5       UPDATE(mempool); /* update mempool size
                           after accepting transaction */
6     else
7       (transaction relay fee < minimum relay fee)
8       transaction rejected;
9   State: Mempool Size Exceeds Threshold Size
10  while (Mempool Size > Threshold Size) do
11    while (transaction relay fee > minimum relay fee) do
12      if (transaction mining fee > minimum mining fee)
13        then
14          Mempool  $\leftarrow$  transaction;
15          UPDATE(mempool);
16        else
17          (transaction mining fee < minimum mining fee)
18          transaction rejected; /* transaction pays
                                   relay fee but does not pay mining fee */
19  return Mempool Size
Result: Spam Transactions Rejected
```

In the current settings of Blockchains, where an attacker only pays a relay fee to broadcast his transactions, if the mempools employ the fee-based design, all spam transactions will be rejected. As such, the mempool will only accept transactions which pay both the relay fee and the mining fee. Legitimate users, on the other hand, will benefit by this design, since they will always pay the relay and the mining fee, so their transactions will be accepted. Once the attacker becomes aware of the fee-based design, the only way it can carry out the attack is by adapting to the new settings and masquerading as a legitimate user. The attacker can do that by adding mining fee to each transaction. Given a budget B , adding mining fee to each transaction will reduce the total number of transactions T_a the attacker could generate in (3) will now become

$$T_a = \frac{1024 \times B}{[(i \times k) + (o \times l) + i] \times [R_f + M_f] \times T_{min}} \quad (27)$$

From (6), we now observe that the number of transactions the attacker can generate has an inverse relationship with the total fee paid per transactions, as naturally expected. Using that relationship, we can adjust the fee parameter and investigate how it limits the attacker's capabilities. To do that, we simulate the effect of increasing the mining fee on the volume of transactions that the mempool accepts. We allocate a fixed budget to the attacker and select thresholds of minimum mining fee

and maximum mining fee. Using (3), we select a suitable budget for attacker that results into 1000 transactions with a minimum mining fee. Then, we generate 1000 legitimate transactions, each with a mining fee normally distributed over the range of the minimum and maximum mining fee. Using a custom-built discrete-event time simulation, we increase the mining fee and monitor its effects on the attacker and the legitimate users.

Evaluation parameters: we use the confusion matrix as our evaluation metric, defined as follows (and highlighted in Table 3). We use the following parameters to test effectiveness of the results obtained from our simulations.

1. **Precision.** Precision is the measure of relevant information obtained from an experiment with respect to the total information. Mathematically it is defined as the ratio of true positive and the sum of true positive and false positive $\frac{TP}{TP+FP}$.
2. **Recall.** Recall is the measure of relevant information obtained from an experiment with respect to the total relevant information. Mathematically, it is defined as the ratio of true positive and the sum of true positive and false negative. $\frac{TP}{TP+FN}$.
3. **F1 Score.** F1 score uses both precision and recall and provides their harmonic average. F1 score can be computed as $\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$.
4. **Accuracy.** In machine learning, accuracy measures the strength of a classifier in determining the nature of experimental outcomes. Accuracy is can be computed as $\frac{TP+TN}{TP+TN+FP+FN}$.
5. **Negative Rate.** Negative rate or specificity is the measure of truly identified negatives in the complete set of negative values. Negative rate can be computed as $\frac{TN}{TN+FN}$.

Table 3. Confusion Matrix

		Actual Transaction	
		Legitimate	Malicious
Mempool Transaction	Legitimate	TP	FP
	Malicious	FN	TN

We will use this evaluation criteria for all the experiments described in the rest of the paper.

Evaluation Results. We plot the results in Figure 13 using the confusion matrix in Table 3 to evaluate the effect of the fee-based design on the mempool. We classify the true positives and the false positives as legitimate and malicious transactions accepted by the mempool respectively. We classify the false negatives and the true negatives as legitimate and malicious transactions rejected by the mempool. We plot the results of confusion matrix in Figure 13(a). The results show that with the increase in the mining fee threshold, the mempool size (TP+FP), malicious transactions (FP) and legitimate transactions (TP) decrease. With a fixed budget, increasing the mining fee decreases the total transactions. Accordingly, the size of the mempool also decreases due to fewer spam transactions (FP). However, increasing the mining fee also limits the fee-paying legitimate users. This, in turn, explains the trend of decreasing (TP).

Using the results from Figure 13(a) and the evaluation criteria defined above, we measure the precision and the accuracy of our design. From the plots in Figure 13(b), we observed that the accuracy increases with the mining fee to a maximum value and then decreased. Using that, we found a *minimum fee cutoff* corresponding to the *maximum accuracy*.

In Figure 13(c), we the plot accuracy and size ratio; the size ratio is the fraction of mempool transactions out of the total number of incoming transactions where a lower size ratio indicates higher size optimization. The results in Figure 13(c) show that at a fee threshold of 13 we achieve 60% accuracy, 70% size optimization, and 78% precision. Increasing the fee parameter further increases the size optimization but decreases the accuracy. Therefore, the fee-based design presents a trade-off between the size efficiency and the accurate detection of malicious transactions used for launching the DDoS attack on the mempool.

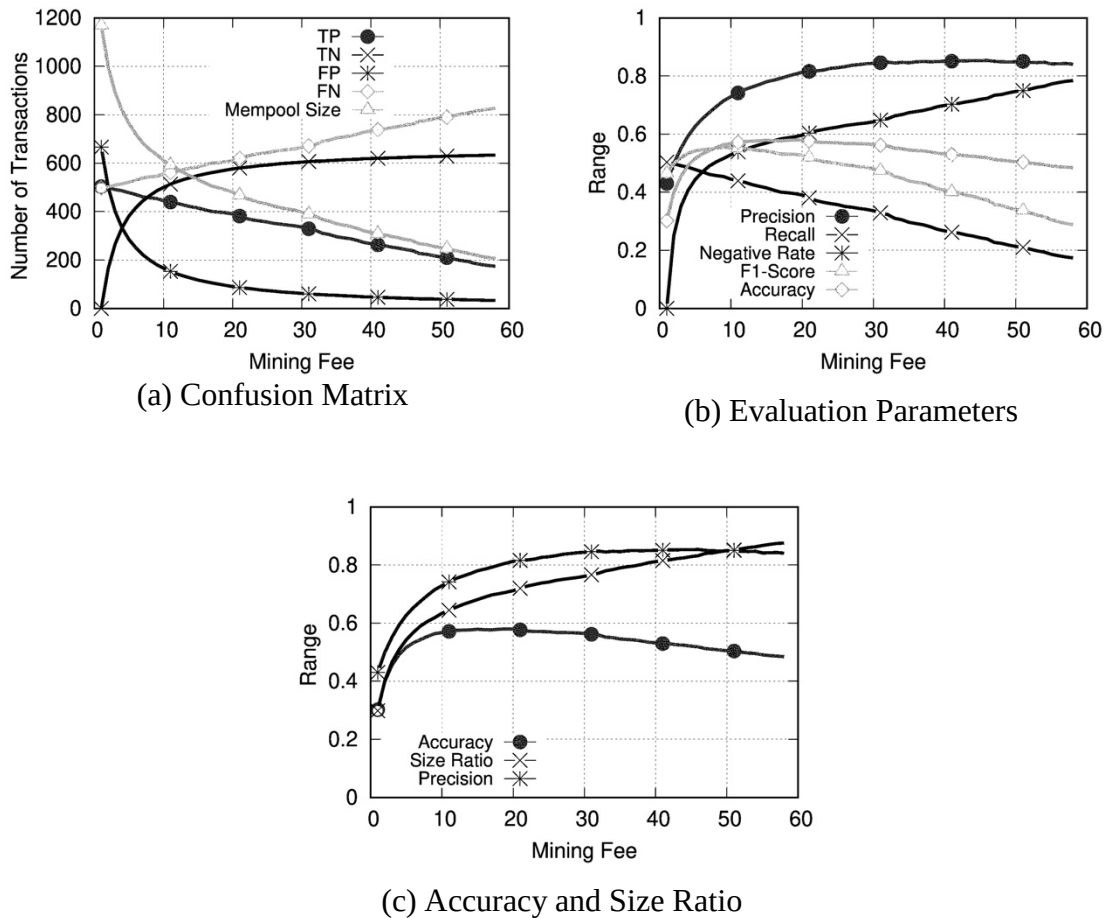


Figure 13 (a), (b), (c). Analysis of Fee-based Design. Notice that as the mining fee increases, the mempool size reduces. However, increasing mining fee also affects legitimate transactions which is why the accuracy of detection decreases with increasing mining fee.

Limitations of Fee-based Mempool Design. In the following we highlight the limitations of the fee-based design and motivate for our second design.

To understand the limitations of “Fee-based Mempool Design” we highlight the nature of some transactions in Blockchain. Suppose Alice sends a transaction to Bob (herein, such a transaction could be virtual, meaning that a digital resource, such as threat indicator in Blockchain-based information sharing is transferred between stakeholders). That transaction is yet to be verified and mined, but Bob spends them by sending 5 BTC to Charlie. For Bob’s transaction to be successfully mined, its parent transaction by Alice needs to be mined first. This sequence of transactions is known as parent-child transaction [8, 38]. For a child transaction to become legitimate, its parent transaction needs to be mined first. Oftentimes, however, when priority factor of a parent transaction is low, the child transaction increases the mining fee to increase the overall priority factor. This process is called “Child Pays for Parent” (CPFP) [91].

For legitimate users, this situation might be undesirable, since more child transactions can lead to transactions getting stuck in the mempool. However, the same situation can be viewed as an opportunity by the attacker to circumvent the fee-based design and carry out the same attack at even a lower cost. For transactions made in the attack phase, their parent transactions in the distribution phase need to be verified and mined. The attacker can minimize the probability of transaction acceptance in the first phase by reducing their priority factor; e.g., by paying a minimum relay fee and no mining fee. Once the parent transactions have lower probability of acceptance in the first phase, the child transactions can increase their priority factor by adding higher relay fee and mining fee. In such a situation, and when the mempools apply the fee-based countermeasures, spam transactions of the attack phase will get into the mempool. After the size of mempool reaches the size threshold, the mempool will check for incoming transactions that pay the minimum relay and mining fee. Since the transactions of all Sybil accounts will pay both the relay and the mining fee, they will be accepted by the mempool and the attacker will be successful, demonstrating a limitation of the fee-based design.

Countermeasure. One way to address this problem is to prioritize the incoming transactions on the basis of mining fee. Mempool can sort the incoming transactions for the fee value and accept the ones which have higher fee. As we increase the mining fee, the capability of attacker to produce transactions decreases (6). The attacker is constrained by the budget and increasing the mining fee reduces the number of transactions he can produce. We can observe this trend in Figure 3(a). Although this reduces the number of spam transactions in the mempool and optimizes its size, it also reduces the accuracy and the number of legitimate transactions that get accepted. As the fee parameter is increased, the capability of all of the legitimate users to pay higher fee also decreases. To this end, the fee-based countermeasures do limit the attacker from flooding the mempool, but they also limit the number of legitimate transactions that successfully pass the fee threshold. To address these limitations, we propose *age-based countermeasures*.

3.3.5.9 Age-based Countermeasures

Age-based Mempool Design. To limit the attacker’s chances of success, we propose the “Age-based Mempool Design” which addresses the limitations of our previous model. For this design, we leverage the confirmation factor or “age” of a transaction to distinguish between legitimate and malicious transactions. In Blockchain systems, the age of a transaction is generally used to determine how many block confirmations it has achieved over time.

For this design in algorithm 2, we assume that the baseline size threshold of the mempool has been reached, and the mempool is only accepting transactions which are paying the relay fee as well as the mining fee. Now, for each incoming transaction, as highlighted above, we count the number of inputs or parent transactions. We initialize a variable “average age” and set its value to 0. Next, we calculate the average age of the transaction by adding the age of each parent transaction and dividing by the total number of parent transactions. This gives an estimate of confirmation score of the incoming transaction. Then, we apply a “minimum age limit” filter on the mempool. The minimum age limit can take any arbitrary value greater than 0 [92]. If the transaction’s mean age value fulfills the criteria of age limit, then the mempool accepts the transaction. Otherwise, the mempool discards this transaction.

A transaction in Blockchain systems can have an input pointer pointing to the spendable transaction that it has previously been received. For spam transactions, these inputs are not spendable and therefore less likely to be mined. This serves the objectives of the attacker who intends to broadcast spam transactions which eventually get rejected. Although the age factor is taken into account for transactions, it is not considered while broadcasting those transactions. As such, attackers may exploit this feature of the system by broadcasting spam transactions and flooding the mempools without losing fees. In this design, we apply the check on the age of the incoming transactions. In the attack phase, the spam transactions will have input pointers of a parent transaction that will not be confirmed in any block. The age of all those parent transactions, made in the distribution phase, will be 0.

Using this knowledge about the nature of spam transactions, we compute the average age of all the input pointers (parent transactions); minimum age value of 1 means that all transactions coming into the pool are confirmed in at least the most recent block of the Blockchain. In particular, once the transaction is mined into a Blockchain, its parent transaction is removed from the “UTXO” set and cannot be spent again (double spending avoidance measure). The transaction itself becomes the new spendable “UTXO.” With these advantages, the age-based design can safeguard the Blockchain system from spam transactions and double-spending.

Once this design is implemented, if a user tries to spend his transactions, he needs to have at least one valid confirmation backing up his transaction. This gives an advantage to legitimate users who can make a normal transaction with confirmed parent transaction of significant age. On the other hand, most of the spam transactions of attacker will be rejected due to low confirmation factor despite paying high mining fee.

Algorithm 2: Age-based Mempool Design

Inputs : incoming transactions;
minimum relay fee;
minimum mining fee;
minimum age limit;
age of each input of transaction;
Threshold Size;

Output: Mempool Size

State: Mempool Size Exceeds Threshold Size

```
1 foreach transaction  $\in$  incoming transactions do
2   initialize;
3   average age = 0;
4   N  $\leftarrow$  number of parent transactions of current transaction;
5   while (transaction relay fee > minimum relay fee) do
6     while (transaction mining fee > minimum mining fee)
7       do
8         average age =  $\frac{(\sum_{i=1}^N \text{parent}_i)}{N}$  ; /* apply age
9           filter */
10        if (average age > minimum age limit) then
11          Mempool  $\leftarrow$  transaction;
12          UPDATE(mempool); /* update mempool
13            size after accepting transaction */
14        else
15          (average age < minimum age limit)
16            transaction rejected; /* Reject
17              transaction if its age factor is
18              below the threshold */
19  return Mempool Size;
```

Result: Spam Transactions Rejected

Analysis of Age-based Mempool Design. We analyze the working of “Age-based Mempool Design” and how it counters DDoS attack. For this design, we have established that the attacker has the capability of circumventing the “fee-based design” and is willing to pay the relay and the mining fee in all transactions. Also, the attacker knows that his transactions will not be verified, so it pays comparatively higher relay and mining fee than the fee paid by the legitimate users.

We carried out our second experiment to analyze the working of the age-based countermeasures. We set a minimum age limit and a maximum age limit as thresholds for the incoming transactions. For the attacker, the only set of transactions with age value greater than 1 are generated in the distribution phase. Child transactions made in the attack phase were assigned 0 age value due to unconfirmed parent transactions. To capture that, we normally distribute the average age value of all malicious transactions from 0 to the minimum age limit. The average age value of all legitimate transactions was set from 0 to the maximum age limit. A total of 2000 transactions were generated with half of them being malicious and half being legitimate. Then, we applied the age-based design on all the incoming transactions at the mempool. We increased the age requirement for the

incoming transactions and evaluated the accuracy of detection and the state of mempool for each transaction.

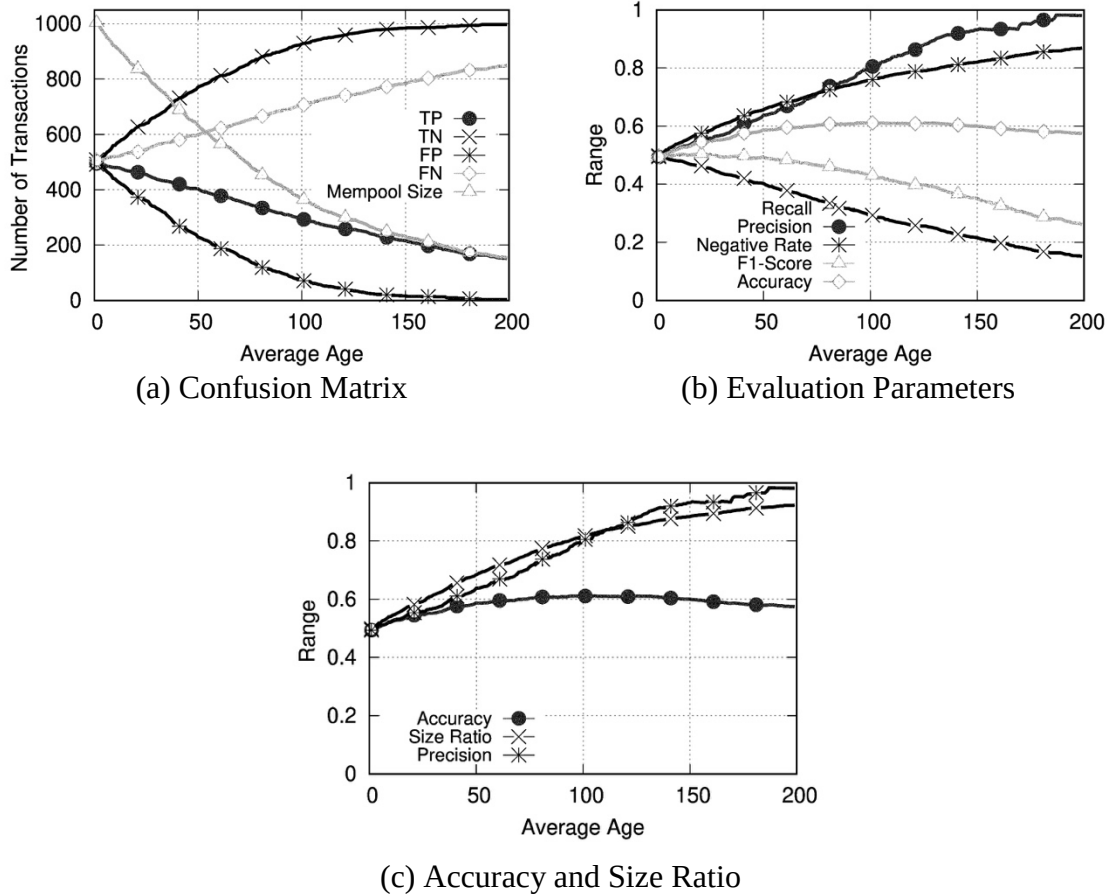


Figure 14. Analysis of Age-based Design. Notice that with age-based design, the accuracy, precision, and size ratio are comparatively higher than the fee-based design.

Evaluation Results. Using the same confusion matrix parameters in Table 3, the results in Figure 14 show that upon increasing the average age the malicious transactions (FP) decrease sharply. The mempool size decreases to a point where there are only legitimate transactions left in the mempool. Due to low (FP) and higher (TP), the precision reaches a close to 1 in Figure 14(b). In Figure 14(c), it can be observed that at an average age value of 100 we achieve accuracy, size optimization, and precision. As we increase the age parameter to 200, the accuracy does not decrease as in the fee-based design, while the size ratio increases up to 90% and the precision increases up to 98%. This shows that this policy prevents a majority of malicious transactions from entering the mempool and helps the legitimate users in getting their transactions accepted.

In these settings, if the attacker intends to spam the network, he needs to have majority of his transactions confirmed in the Blockchain. However, in our attack model, we have described that confirmation is undesirable for the attacker since it results in losing mining and relay fee, creating asymmetry in the attack cost. In PoW-based Blockchains, as highlighted earlier, we recall that the

average block mining time is 10 minutes. As a result, for a single confirmation of all of the transactions, the attacker has to wait on average for 10 minutes. Using the results from Figure 14(c), the attacker will have to wait a minimum of 100 blocks to relaunch the attack. With the average block computation time of 10 minutes, 100 blocks lead to 16 hours of delay, which can be further controlled by adjusting the Blockchain operation parameters.

Even if the attacker still plans to carry out the attack after waiting and paying all the fee, he will not be able to flood the mempool. The best the attacker might achieve will be occasional network stressing with series of transactions; higher attack cost and low incentive will discourage the attacker. Therefore, the age-based design offers more security against the DDoS attacks while ensuring regular service provision for the legitimate users.

Limitations of Age-based Countermeasures. Although the age-based countermeasures provide an effective defense to DDoS attacks on Blockchains, there are some limitations in this design. Primarily, it requires all the incoming transactions to have a confirmed parent transaction. Depending on the bottleneck and size of the mempool, transaction verification can take even longer time. In low-latency, high-throughput Blockchains, fast transactions are unavoidable, where users cannot wait for verification [93, 87], and their transactions will be rejected by the mempool [94]. However, we do not see Blockchains evolving into such applications any soon, so we do not consider it as a significant problem.

3.4 Blockchain and SGX

Distributed Ledger Technology hosts large numbers of nodes without pre-established trust to provide robust and decentralized services, of which blockchain is the most widely used where transactions across the whole network are visible to all nodes in the form of blocks. However, attacks do exist in such systems where participants are controlled by malicious users which are difficult to detect. To secure the system and protect user privacy, a multi-channel architecture that leverages one promising technology for trusted execution, Intel Software Guard Extensions, is proposed. Each distributed node is enrolled as a trusted entity while multiple channels help to separate different applications and provide better flexibility in different scenarios. We illustrate how to utilize SGX capabilities to defend against six existing attacks on distributed ledger, secure the blockchain mining process and enhance the transaction privacy. Evaluation shows the system is scalable and cost effective.

Blockchain applications in cloud computing environment inevitably suffer from these vulnerabilities. Besides, the distributed and decentralized nature brings a potential of privacy and confidentiality risks, which are the main concerns for the adoption of blockchain applications by industrial and military scenario. We presents multi-channel architecture based on Intel SGX to serve as a feasible solution and help defend effectively against adversarial attacks which are possible in distributed ledger system, as analyzed in this paper. The main contribution of a two-layer architecture is illustrated below.

1. **SGX Enabled Connection Layer.** By enabling Intel SGX features before blockchain peer connection, the trusted execution environment is provisioned per platform. Each execution

environment is isolated in an enclave which is a tamper-resistant zone in the memory, and even the administrator of the local machine cannot break into the enclave. Trusted communication is then provisioned by remote attestation between peers, ensuring the authorized participation and code execution. With the help of these two steps, the vulnerabilities can be effectively countered.

2. **Multi-channel Transaction Layer.** The multi-channel scheme provides a measure for isolating communication channels. With SGX enabled, the trusted participation of each peer in the communication channel is assured. Data transferred via the SGX enabled channel is kept confidential between channel members. Even if the channel member is compromised, data leakage from the trusted execution environment will be prevented.

3.4.1 Background

To better illustrate the proposed approach to secure blockchain architecture with the adoption of Intel SGX hardware, an overview of the existing attacks and vulnerabilities that remain as potential risks for the adoption of blockchain into the current systems is given below, followed by the SGX capabilities.

3.4.2 Blockchain Vulnerabilities and Attacks

Due to the P2P communication model and the trustless node involvement, most blockchain services and blockchain-based applications could be vulnerable to certain attacks that are difficult to detect or prevent. Six possible vulnerabilities existing in distributed ledgers are presented and analyzed below.

Block Withholding Attack (A1). Blockchain nodes usually join a pool to mine blocks with other nodes. Once a new block is mined, the rewards are shared among all the pool members. However, some malicious nodes intend to join the pool but never publish the block that has been mined, which decreases the overall rewards of the pool [95]. Such attack is hard to detect because the mining process is controlled by the owner of the mining platform. Even when the platform finds a solution to a new block, other nodes are not aware of the fact. Analysis [96] shows that block withholding attack makes it possible for a rogue miner to gain profit without effort.

Block Discarding Attack (A2). Block discarding attack [97] happens when a node controls the majority of network connections with other nodes. A mined block needs to be confirmed by most of the nodes will be confirmed and added to the blockchain. If the network is controlled by a single party that intends to discard a certain block mined by a normal node, the attack will discourage the nodes in the controlled network from confirming this block. In this way, the newly mined block by the normal node will be set invalid if a block from other nodes is confirmed earlier than itself.

Replay Attack (A3). Replay attack often happens in P2P networks where there are frequent message exchange and propagation. Attackers may try to repeat or delay an intercepted data transmission, thus preventing the honest party from communicating normally with the party where the message comes from [98]. Replay attack in blockchain network is discussed in several scenarios, such as smart grid [99] and Internet of Things environment [100], as well as blockchain based big data authentication protocols [101]. Replay attack could also be leveraged to launch

block withholding attack when the malicious node eavesdrops the block confirmation message and delays the message from immediate propagation. For blockchain applications where there are quantities of distributed nodes, it is challenging to mitigate such attacks using the traditional timestamping services since most message exchange is transferred in a simultaneous session and a concurrent manner.

Man-in-the-Middle Attack (A4). In the Man-in-the-Middle Attack [102], attackers try to intercept the message between two honest parties, possibly alter the message and deliver the modified message to both sides so that he can impersonate either side. Both two honest parties could leak sensitive information since they believe they are interacting with the authenticated parties. It is possible that a malicious node joins the block mining process and steal the mining result, that is, the solution to a puzzle in a proof of work blockchain, from another node and then pass it to other participants for confirmation. Even though there are methods to detect such attacks, but the timing is critical. When the node realizes that the solution is stolen, it is too late since the rewards are already distributed.

Potential Privacy Risks (A5). One major concern of blockchain is the privacy issue due to the distributed nature of node connection. Each node participates in the network actively to broadcast messages and receive rewards. Transactions made by each node is traceable by other nodes. This is also concerned with the anonymity issue which is acknowledged in Bitcoin transactions since the transactions are permanently recorded in the public ledger while everyone can see the detailed transaction balance and public addresses. Some scenarios such as financial institutes and military communications require isolated transactions built on top of tamper-proof blockchains. If the privacy issue is solved, users can communicate in a secure way without exposure of sensitive information and still make use of blockchain benefits for integrity protection.

Majority Hash Rate Attack (A6). To gain a large quantity of profit, some miners use special mining devices to control a great portion of mining power. When an attacker controls more than 50% of the entire mining power, he can reverse transactions he sent and even prevent some transactions from being confirmed, with a high possibility. Therefore, majority hash rate attack is also called 51% attack. This attack could also break the decentralized nature of blockchain since the majority is controlled by a powerful but single entity. A two-phase proof of work [103] is proposed to prevent such attack but the hardware security is not seriously considered.

3.4.3 Intel SGX Security Capabilities

Intel SGX provides enclaves, which is an isolated zone for trusted program running, to reduce the attack surface and minimize the trusted computing base. We illustrate three key notions adopted by SGX, including Enclave, Attestation and Sealing, as follows.

Enclave. When an enclave is created, the sensitive code and data will be stored inside a protected memory region called Enclave Page Cache (EPC). The EPC region is encrypted which ensures strong confidentiality. The code inside the enclave is not authorized to access memory beyond the enclave boundaries. If code inside one enclave accesses content in another enclave, there would be an error of aborted page access. Enclave Page Cache Map (EPCM) is a hardware structure which stores security and access control information for every page in the EPC. After the enclave

is provisioned with appropriate memory content, there will be an enclave measurement stored in two registers, MRENCLAVE and MRSIGNER. MRENCLAVE provides an identity of enclave code called Enclave Identity, which is a SHA-256 digest of enclave log by ECREATE instruction. The enclave log contains the content and relative position of the enclave pages as well as some security flags. MRSIGNER identifies an enclave signer's identity called Sealing Identity, which signs an enclave when the enclave is created. Inside the MRSIGNER, the hash of the signer's public key is stored. After enclave is initialized, the Enclave Identity and Sealing Identity will be available by calling EGETKEY and EREPORT instructions. Each enclave has a certificate issued by the enclave author in the format of SIGSTRUCT. Three metadata fields in SIGSTRUCT is used to determine an enclave's identity including the modulus of the RSA key used to sign the certificate (MODULUS), the enclave's product ID (ISVPRODID) and the security version number (ISVSVN). An enclave author can issue several enclave certificates using the same RSA key to indicate different modules of the same software.

Attestation. Attestation is a process that one entity proves that it is running on top of a trusted platform to another. Intel SGX provides two types of attestation, including local attestation (two enclaves running on the same platform) and remote attestation (extending local attestation to outside of the platform). Local attestation can be performed between two enclaves: target enclave and challenger enclave. First, challenger enclave sends attestation request to target enclave for verification as well as challenger enclave's MRENCLAVE value. Then target enclave uses EREPORT instruction as well as the received MRENCLAVE value to create a signed REPORT for challenger enclave and sends it back. Challenger enclave receives the REPORT and extracts the REPORT key to compute the MAC. If the MAC matches the value on the target enclave's REPORT, then challenger enclave can confirm that target enclave is running on the same trusted platform. Then challenger enclave can create a REPORT for target enclave and sends it to target enclave so that target enclave can confirm that challenger enclave is on the same platform in a similar way. In some cases, two enclaves on different platforms need to verify each other. Intel SGX enables a third enclave called quoting enclave to help launch a remote attestation. Quoting enclave adopts Intel Enhanced Privacy ID (EPID) [104] as a key to sign the REPORT from the target enclave on the same platform and generate a QUOTE which will be sent to the remote enclave for verification. Same as local attestation, there is a challenger enclave requesting to verify the target enclave. Here, the target enclave exists on a different platform from the challenger enclave. The target enclave will create a REPORT and send it to the quoting enclave. The quoting enclave signs the REPORT using the EPID key to generate a QUOTE and sends the QUOTE back to the target enclave. The target enclave forwards the QUOTE to challenger enclave as well as some other user data for authentication. Challenger enclave uses an EPID public key to verify the QUOTE. In this way, the target platform is verified, and further communication can be established.

Sealing. When enclave program finishes running, the code and data inside the enclave will be gone. To store the data for future use, Intel SGX provides a sealing key to encrypt data and ensure data integrity. The sealed data can only be unsealed when the trusted environment is restored locally [105]. By calling EGETKEY instruction, current enclave can access the sealing key, seal the data and export the data to a memory region outside the enclave. There are two sealing policies designed by Intel, including sealing to the enclave identity and sealing to the sealing identity. Depending on the access control policies of the enclave applications, different sealing policies can be adopted. Sealing to the enclave identity can produce a key only available to the exact enclave

instance. If a key available to different enclave instances under the same sealing identity is needed, the policy of sealing to the sealing identity (Sealing Authority) can make it.

The security capabilities of SGX can be summarized as follows. **Enclave Execution (C1)** is tamper-resistant against software attacks outside the enclave [106]. **Hardware-rooted Randomness (C2)** provides a valuable source for cryptographic key generation and protection. A true random function is provided in the tRTS (Trusted Run-Time System) library available in Intel CPUs [107]. **Remote Attestation (C3)** allows a client platform to attest itself to a remote party proving that the client is running in a trusted environment. This ensures the integrity of code execution in both the client and server side. **Trusted Elapsed time (C4)** provides a hardware-assisted measure for trusted timestamping service, which is critical for scenarios where it is time sensitive. **Confidentiality Assurance (C5)** prevents sensitive transactions and business contracts from leakage. The enclave identity key and provisioning key can be involved for secret protection and attestation. **Sealing and Unsealing (C6)** helps to store confidential information outside the enclave for future access after system shutdown. **Monotonic Counter (C7)** serves as a measure to defend against the replay attack. Altogether, these Intel SGX capabilities will greatly reduce the attack surface and minimize the trusted computing base.

3.4.4 Design of SGX-Enabled Distributed Ledger with Multi-Channel Architecture

Considering SGX security capabilities and the promising usage of enclave isolation, we use the SGX capabilities (C1-C7) mentioned above to defend against the six attacks (A1-A6) which are possibly existing in blockchain applications, following the basic design rationale as illustrated in Table 1, to achieve a reliable distributed ledger. Design of SGX-Enabled Distributed Ledger with Multi-Channel Architecture

Table 4. Design Rationale of Attack Oriented and SGX Enabled Blockchain

Potential Attacks	Intel SGX Capabilities
Block Withholding Attack (A1)	Enclave Execution (C1)
Block Discarding Attack (A2)	Enclave Execution (C1), Remote Attestation(C3), Monotonic Counter(C7)
Replay Attack (A3)	Hardware-rooted Randomness (C2), Remote Attestation(C3), Trusted Elapsed time (C4)
MITM Attack (A4)	Hardware-rooted Randomness (C2), Remote Attestation(C3), Confidentiality Assurance(C5)
Potential Privacy Risks (A5)	Enclave Execution (C1), Confidentiality Assurance(C5), Sealing and Unsealing(C6)
Majority Hash Rate Attack (A6)	Enclave Execution (C1)

Trusted execution could be established in an enclave zone (C1) which provides an isolated environment and thus the reduced attack surface. The code and data inside the enclave are integrity-protected. Even when the platform and the Operating System is controlled by a malicious entity, the mining is still protected from compromise. This guarantees that once a block is mined, the block will be immediately submitted to the network, making block withholding attack difficult. Hardware-rooted Randomness (C2) helps to generate random numbers used in key management

protocols, benefiting the message exchange in transactions between blockchain nodes and the resistance of replay attack. Remote Attestation (C3) is utilized during key negotiation and secret exchanges. The shared secrets established is platform dependent, making it effective for resisting the man-in-the-middle attack. To provide a message with a trusted timestamping, the SGX capability of trusted elapsed time (C4) calls *sgx_get_trusted_time()*. This trusted time can effectively prevent replay attack where a given expire time is set. Confidentiality Assurance (C5) protects peer to peer communication in the transaction process and preserve the confidentiality of both identity and transaction. Enclaves can seal and unseal the secret (C6) shared by two parties for multiple usages. Monotonic Counter(C7) provides a trusted counter by calling *sgx_create_monotonic_counter()* function, which is utilized to preserve message authenticity during communications, further enhancing resistance against replay attack and man-in-the-middle attack.

Based on the above design rationale, a further contribution is made to protect privacy by adopting a two-layer architecture, comprising an upper transaction layer featured with multiple channels and a connection layer enabled with SGX hardware, as is shown in Figure 15. The transaction layer uses multiple channels to separate different group of nodes. Each channel node is responsible for maintaining states of the current channel. Access to each channel is predefined during node enrollment. The connection layer will be used for the communication between nodes. The enclave process hosts and isolates the block mining code while the non-enclave process is responsible for broadcasting information among blockchain nodes.

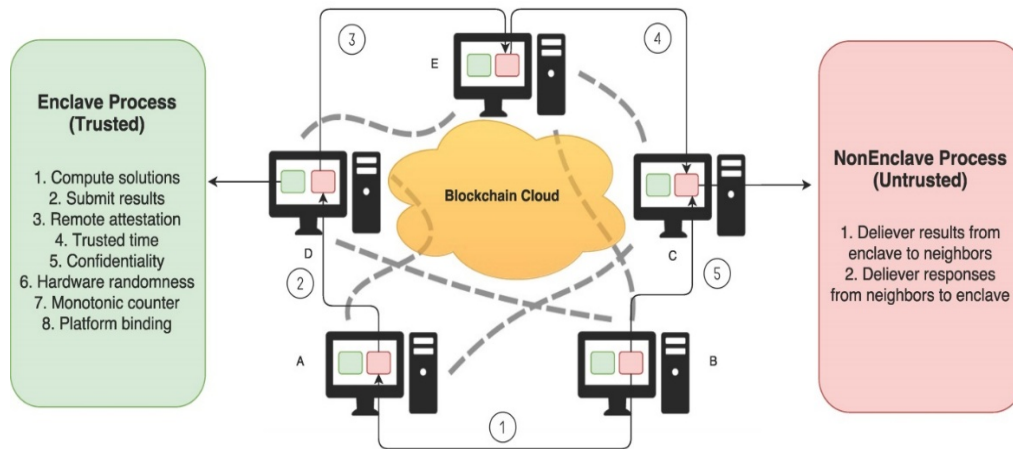


Figure 15. SGX Enabled Blockchain Architecture in the Cloud

3.4.4.1 SGX Enabled Connection Layer

In Figure 1, we design two processes for the program running of block mining and consensus in each node, enclave process and non-enclave process, to isolate critical code and data. We assume the enclave process is trusted and tamper-resistant against adversarial compromise. The communication between each node is established by the non-enclave process, which suffers from the above mentioned six attacks (A1-A6). There are four steps for the enclave provisioning and information exchange, as illustrated below while the authentication between the node and the network manager is out of the scope of this paper.

Node Involvement. To participate in the blockchain network, the node needs to connect to other neighboring nodes (or peers). During the first time when two nodes connect to each other, the remote attestation is launched with each node enabled by SGX. For example, in Figure 1, if peer A and B are trying to build a secure and trusted connection, they will request an attestation from each other. The attestation is launched in the following steps. First, peer A sends a request to B for attestation and the enclave running on peer B will generate a data structure called REPORT which is evidence for trusted enclave execution. The REPORT will be forwarded to a quoting enclave which is specific to the platform providing a signed QUOTE to be verified at the challenger platform (peer A). Peer A will request the Intel Attestation Service for QUOTE verification. If verification passed, the platform authenticity of peer B is confirmed. Likewise, peer A can be attested upon the request of peer B and other nodes perform similar actions as well for the first connection request. By remote attestation, the connection between two trusted platforms is established. Replay attack (A3) and Man-in-the-Middle attack (A4) can be mitigated since the message exchanged can be generated with a binding to the specific platform. If a middle man intercepts the message, the message origin and message sender will not be consistent. Any message replayed from another platform will be detected immediately. This phase is also responsible for key negotiation which will be used in the following message propagation including mined block broadcast from both neighbors and the peer itself. Due to performance considerations, the remote attestation is performed only when the peer receives the first connection request with the same peer. The following connection will only need to use the negotiated key for future communications since the key is originated from the authenticated party which is platform dependent, and is only known by the two attested peers. Only when the keys expire, then a new remote attestation process is needed.

Block Mining Isolation. To preserve the integrity of the mining code running and the mined solution, we keep the mining code inside the enclave (C1). This CPU-dependent mining also prevents the occupation of large numbers of resources which are the prerequisites of block withholding attack (A1) and majority hash rate attack (A6). By isolation, the enclave process will not be manipulated by the non-enclave process, which preserves the code integrity even when the non-enclave process is compromised. In this way, both block withholding attack and block discarding attack (A2) will be prevented since we can place the following two steps inside the enclave, including (1) mine the solution and submit the mining result regardless of whether there is a solution; (2) receive messages from neighbors regardless of whether it is a block mining solution or a transaction broadcast. All the process logic for the above two steps are handled inside the enclave, in case that the non-enclave process takes control and misbehaves. Note that it is important to include a nonce generated from the hardware (C2) in case of chosen plain-text attack, preventing the non-enclave process from guessing the message type.

A certain amount of blockchain transactions are grouped into one block. Each block contains the hash of the previous block and links to the previous block, making it a distributed and robust ledger in that multiple copies of the ledger is maintained among the participating miners. Integrity is preserved since it is technically hard to change a certain block with the necessity to change all the following blocks. Mining is the process to keep record of each block. To keep all blocks consistent throughout the blockchain network, miners repeatedly verify transactions, mine and confirm new blocks. Proof of work is the consensus scheme used in Bitcoin mining [108] and serves as an

example consensus algorithm here to illustrate how SGX enabled block mining works. To successfully add a new block to the blockchain network, the miners need to compute a difficult puzzle which requires considerable computing power. However, to make it easier for other miners to verify the block, the puzzle utilizes a one-way hash function. Therefore, the ability to solve the puzzle is dependent on the hashing power. Taking proof of work as an example consensus algorithm used in our design, the following algorithm (Pow) is used to isolate the mining process inside an enclave. Specifically, the enclave environment should be provisioned first by initialing the parameters required and then start to compute solutions for new blocks. If a valid block is mined successfully, the miner will notify the neighbors to broadcast the solution, which is encrypted. Once other peers receive a message from their neighbors, an enclave environment will be provisioned the same way to decrypt the message. If the block is valid, it will then notify its own neighbors for further propagation.

Algorithm Pow: SGX based Block Mining Using Proof of Work

```

Procedure Proof_of_Work()
{
    enclave_init();
    While (true)
    {
        newBlock = computeSolution();
        If (newBlock.isValid())
        {
            sendToNeighbor( encode( newBlock ) );
            neighbors: receive( encode( newBlock ) );
            break;
        }
    }
    enclave_exit( NULL );
    Return;
}
Procedure receive( message )
{
    enclave_init();
    Message = decode( message );
    If (message.type = NEWBLOCK)
    {
        confirm( message );
        If (message.isValid())
        sendToNeighbor( encode( newBlock ) );
        enclave\_exit( message );
    }
    Return;
}

```

The enclave initialization before the mining process generates a cryptographic hash of the code and data residing in the enclave, which is stored locally as an enclave measurement and used to

verify that the initialized enclave is what the enclave claims to be. During the creation of the enclave, a series of instructions including ECREATE, EADD, EEXTEND and EINIT are executed.

Mined Solution Submission. Once a peer finds the solution for a new block, it will submit to the pool manager for block confirmation and get rewarded. In public blockchains, proof of work is widely used as the consensus algorithm. Commonly, miners join a pool and mine continuously to gain competitive computing power. There is a pool manager to assign mining tasks to pool members. Once the pool manager receives a new block solution from honest miners, the manager will broadcast the block to the blockchain network. If the blockchain network miners reach a consensus on the newly mined block, a certain amount of reward will be received by the pool manager. Hence, it is critical that the new block is recorded and the miner who finds the block is recorded, both in a trusted and tamper resistant manner. With the adoption of Intel SGX features, two security guarantees can be achieved. For one thing, once a mined block is passed on to a malicious peer, the malicious peer cannot ignore the block since he has no control over the program running inside an enclave (C1). For another, the block confirmation message is encrypted by the key negotiated in node involvement phase, so it is impossible to distinguish a block confirmation message from other messages since only the enclave process can decrypt the message using the same key negotiated during the remote attestation process. The block solution is also sealed for secure local storage (C6). This is effective for defending against block withholding attack, in that the malicious peer is oblivious about whether the enclave process has mined a new block or not, thus it is difficult to decide whether to withhold the message or not. Besides, the attempt to modify the message is impossible since once it is modified, the integrity is disrupted, and any valid message cannot be recovered. To defend against block discarding attack, the data propagation process is designed to run inside an enclave to avoid malicious data discarding. The block mining process runs inside the enclave to provide integrity of the process while the block submission is bound to the platform to achieve tamper resistance.

Mining Rewards Distribution. When a new block is confirmed, the peer who finds the new block will get rewarded. There are several reward schemes existing for pool-based mining [95]. In proportional reward scheme, the reward is distributed proportionally according to the shares submitted by each miner. The share is the work load needed to find the final solution, that is the interval between two valid blocks. In this case, some malicious miners may frequently hop from different pools during a competition round for maximizing the rewards. Pay-Per-Last N Shares (PPLNS) reward scheme is proposed to defeat such greedy miners. PPLNS takes only the recent shares into considerations instead of the shares from the whole round of competition, so only recent active miners are eligible for sharing the reward. Both schemes require the combination of the miner identity and activity. It is critical that the mining rewards are distributed and that greedy miners are detectable. For peer platforms SGX enabled, the mining record can be tracked on a platform basis because both shares and solutions are generated from specific platforms. In this way, non-frameability is assured for the honest miners, and the greedy miners cannot be repudiated. Meanwhile, each share and solution submission are bound with a trusted elapsed time, which maximizes the trustworthiness of the node, providing a matrix for a fair reward distribution and an authenticated identity for confirming the deserved reward receiver. By using the platform dependent key for confirmation, all the neighbors of the peer that proposes the new block witnesses the reward distribution.

3.4.4.2 Multi-channel Transaction Layer

To preserve the privacy of participating nodes in the blockchain network, a multi-channel architecture is adopted to isolate transactions. Enclave isolation preserves the integrity of blockchain mining code and data, while channel isolation preserves the privacy of transactions by limiting the access to restricted participating nodes. Each channel maintains a ledger called subledger for efficient transaction signing and verification. Meanwhile, there is a system ledger maintained by all channels. The existence of a subledger for each channel helps advance the block indexing and confirmation operations, thus improving the system scalability and throughput. For some blockchain implementations, such as Hyperledger Fabric [109] and R3 Corda [110] which support multiple channels, a further step is taken to combine the multi-channel scheme with Intel SGX to provide a privacy enhanced architecture.

A specially assigned peer is responsible for generating the genesis block (the first block for the ledger) for each channel during channel establishment. Access control policy is defined in the genesis block configuration. To enhance user privacy, each channel hosts a certain number of peers while each peer joins the channel by binding the platform key with the channel ID which is publicly known to all nodes in the blockchain. SGX enabled channel can help defend attackers from hijacking a channel member by isolating channel transactions inside enclaves. The channel metadata and the platform key are used to bind a node to a channel so even an attacker that has access to the channel member's certificate cannot participate in the channel since there is no binding between the channel and the attacker platform. Meanwhile, Man in the Cloud (MITC) attack [111] is prevented by sealing confidential information locally.

In Figure 16, there is a system ledger and several subledgers, as well as three nodes. Each subledger represents one channel, such as Subledger 1 for Channel 1. Node A belongs to both Channel 1 and Channel 4. Both Node A and Node C participate in Channel 4. Therefore, blocks in Channel 4 are signed by both nodes. However, due to channel isolation, Node A is oblivious to transactions in Channel 2. The binding between the node and the channel is also included in a transaction which is public to all channel nodes, and stored in the node platform using SGX sealing feature (C6). The attacker who even holds the certificate maliciously cannot be involved in attempted channel mining because it will be easily detected that the certificate is originally from another platform instead of the attacker. This preserves both privacy of channel members and the integrity of the permission to participate in the channel.

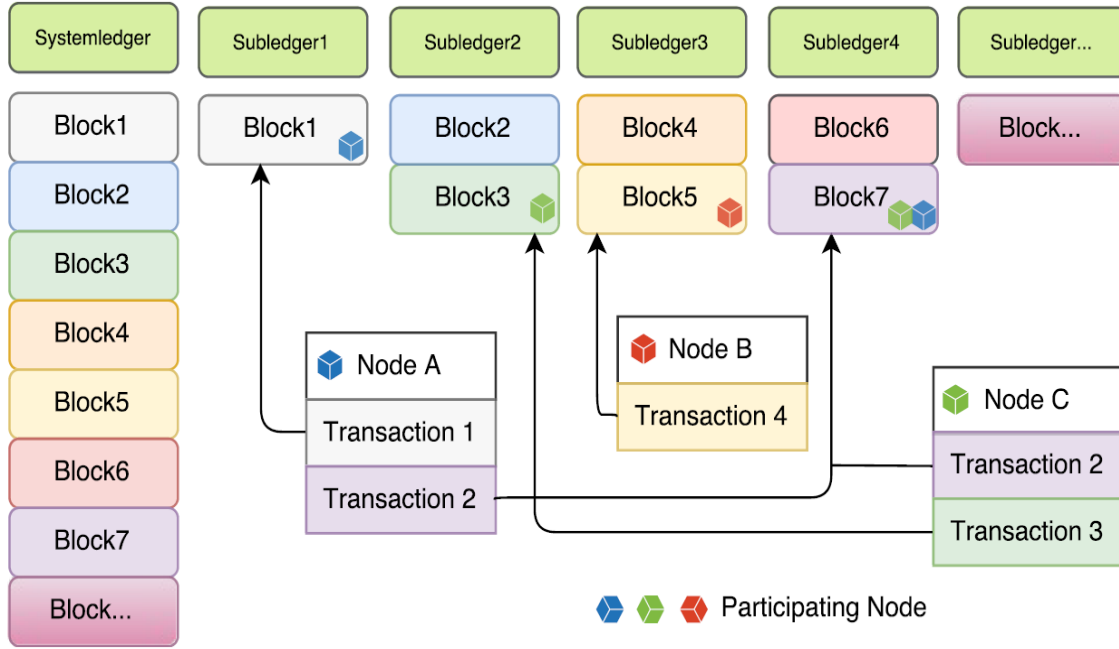


Figure 16. System Ledger and Subledger

In Hyperledger Fabric [109], there are enrollment certificate (ECert) and transaction certificate (TCert). Both certificates are signed by the Certificate Authority (CA) which is part of the membership service provider. On receiving certificate issuance request from potential channel members, the CA will build a trusted connection with the node using SGX remote attestation and obtain the platform-dependent key, noted as K_{platID} . To simplify the binding of a specific node to the channel and make certificates platform dependent, $ECert_{SGX}$ is used to represent the platform self-signed certificate (or token) for new member enrollment.

$$ECert_{SGX} = (ECert, platformID)_{sig} \quad (28)$$

The $TCert_{SGX}$ is used for transaction participation in the channel-isolated communication, which contains the identifier of the channel that the member has access to.

$$TCert_{SGX} = (TCert, platformID, channelIDs)_{sig} \quad (29)$$

The above certificate ensures that each participant is assigned an original certificate and generate a self-signed certificate accordingly. By doing so, the participating nodes can maintain the authorization from the CAs and providing additional security guarantee of the trusted platform. Most importantly, trusted members constitute a trusted channel with assured privacy preservation.

4 RESULTS AND DISCUSSION

4.1 Evaluation of Provchain

The evaluation environment setup includes three categories: the server side software, the ownCloud application configuration, and a benchmarking tool. For provenance collection, we use Apache JMeter [83] as a benchmarking tool to assess the performance of the provenance-enabled ownCloud application. The specifications of software and the version used are listed in Table V.

Apache JMeter is an open source and Java-based software designed to test functions and behaviors on a large scale, with various performance measures such as transaction time, response time, and throughput. Web applications are the main cases in which JMeter plays an important role. JMeter uses test plans to describe a series of steps that will execute when the test plan runs. A complete test plan usually consists of one or more Thread Groups, logic controllers, sample generating controllers, listeners, timers, scripts, and configuration elements. JMeter provides data analysis and visualization plugins, allowing great extensibility as well as personalization. User-defined functions can be used to provide dynamic input to a test or handle data manipulation. We build a test plan to measure the performance of a file created using both provenance-enabled ownCloud and non-provenance ownCloud. Our test plan aims to simulate the action of a user who logs into ownCloud and then creates a file using a filename. The simulation also uses random strings to represent file content to be stored when the user closes the file.

The test plan contains two controllers. One is for user login and the other is for file create. When simulating user login, we use HTTP Get to request access to the ownCloud server at our experimental server path `/owncloud/index.php/login` for a request token. Using the request token, we use HTTP POST to send the user credentials as well as the request token to the same path and exchange for the access token. Here, we use a regular expression to extract the access token from the returned HTML files. After acquiring the access token, we are able to create random files. Using HTTP POST, we can send the filename and file content at the same time to the path `/owncloud/index.php/apps/files/ajax/upload.php`.

Table 5. The specifications of software and the version

Software	Name	Version
Server Operating System	Ubuntu	14.04
Web server	Apache server	2.4.6
Database	MariaDB	5.5.44
Cloud storage	OwnCloud	9.0
Performance benchmarking	Apache JMeter	3.2

The following BeanShell Preprocessor script is used to simulate the file content.

```

import org.apache.commons.io.FileUtils; import
org.apache.commons.lang3.Random StringUtils;
import org.apache.commons.lang3.Random Utils;
// create a temp file
File f = File.createTempFile("f-", "");
// generate random string
// and write to file FileUtils.writeStringToFile(f,
RandomStringUtils.random(RandomUtils.nextInt(1000, 10000)),
"UTF-8");
// store file name
vars.put("fname", f.getCanonicalPath());

```

We perform file create with random file names and file contents for 500 repetitions in JMeter [83]. The file size ranges from 1KB to 2MB. Figure 15 shows the average response time of both provenance-enabled ownCloud and non-provenance ownCloud.

From Figure 17, we can conclude that the provenance service brings an average of 6.49% of total overhead against original ownCloud application in terms of the response time, which is acceptable considering the security features it provides. In addition, with the file size increases, the overhead is generally not as much as it is when the file size is smaller, because the larger the file size is, the more time will be spent on transmitting the file itself, and less time will be used for provenance service.

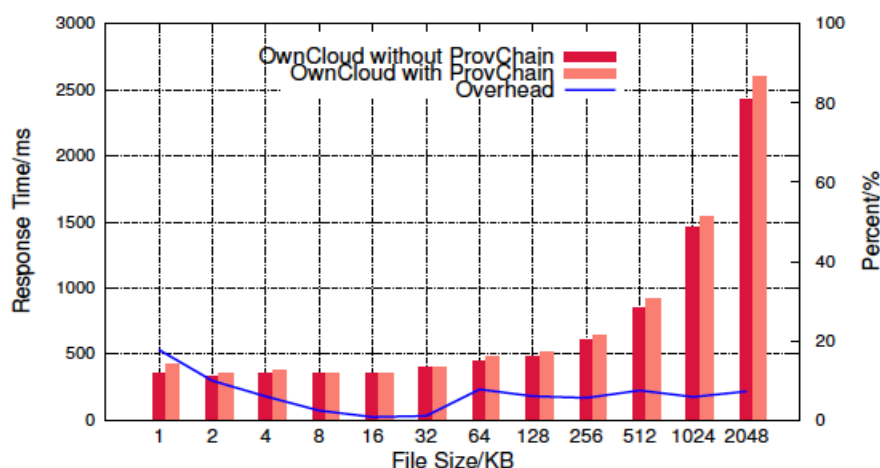


Figure 17. Average Response Time with Different File Size.

Figure 18 and 19 shows the throughput for both original ownCloud (Figure 18) and provenance-enabled ownCloud (Figure 19). We chose 64KB as the file size to assess the performance where only one server is responsible for the provenance service regardless of the production environment, which comprises a web server and services for load balancing and network flow optimization. The results show that both systems have the same amount of traffic received;

however, there is a difference in the amount of traffic sent. Overall, the transaction time distribution is considered to be acceptable, as shown in Figure 20. The provenance-enabled ownCloud has a comparable transaction rate, as depicted in Figure 21 and 22. More evaluations can be conducted with varying file types, operations, and file-sharing status. We use file create operation as a use case for our performance evaluation of provenance data collection. The evaluation for other file operations follow the same procedures.

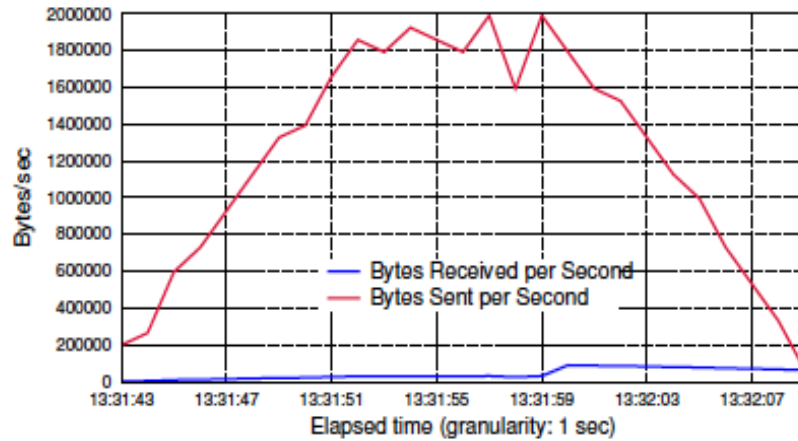


Figure 18. Throughput over Time without Provchain

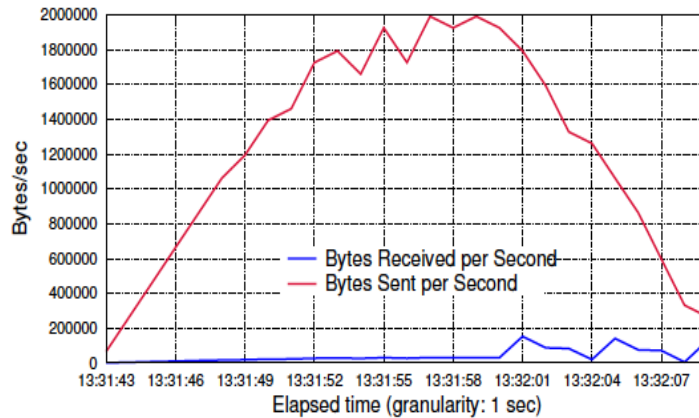


Figure 19. Throughput over time with Provchain

For provenance retrieval, we focus on the efficiency of requesting Blockchain receipt for each of the provenance data entries. In our experiment, we query 10 records each time with a total size of 1.004KB, which uses an average time of 221ms. For each retrieval of Blockchain receipt, we recorded the retrieval time for different file operations. The performance test for provenance data storage follows this in similar way. Table 5 is the provenance retrieval overhead, from which we can conclude that our retrieval methods have a low overhead for the cloud storage system.

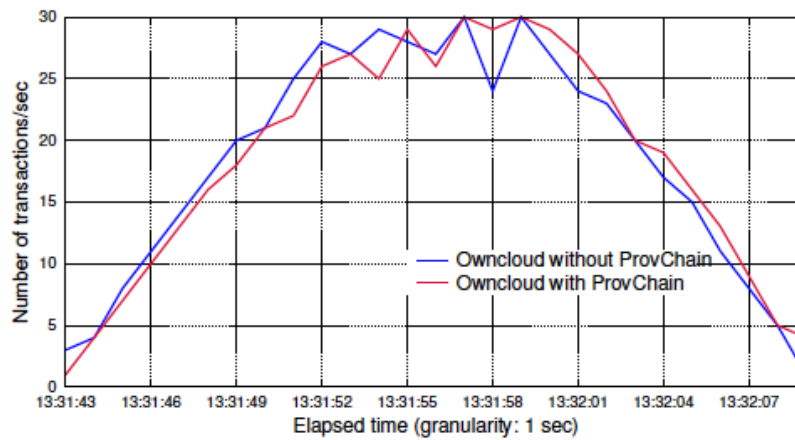


Figure 20. Number of Transactions per Second

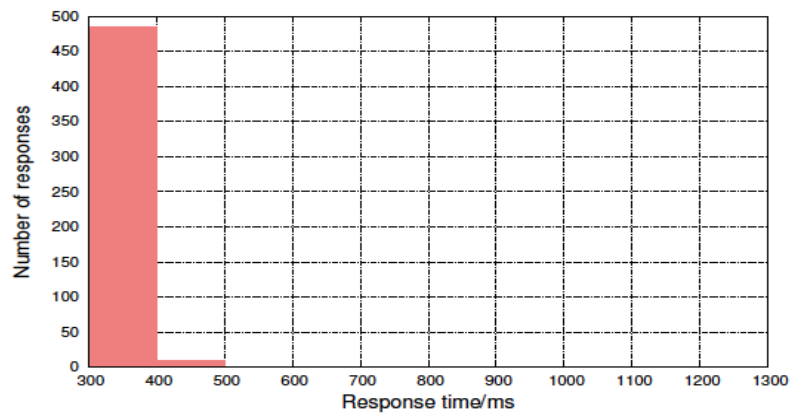


Figure 21. Response Time Distribution without Provchain

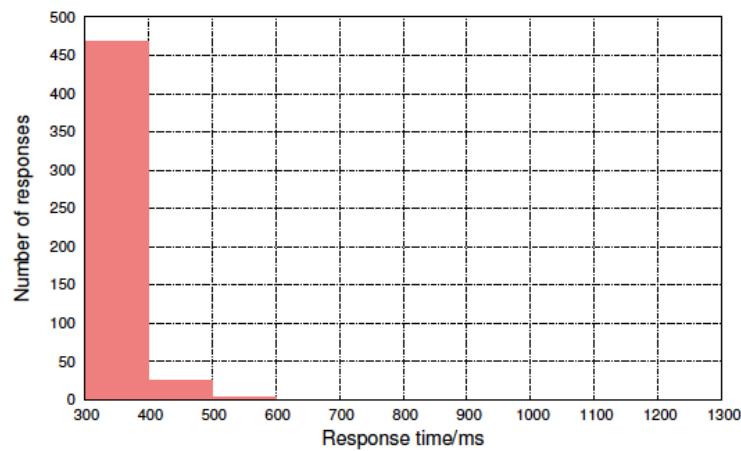


Figure 22. Response Time Distribution with Provchain

Table 6. Overhead for provenance data retrieval

RecordType	Size of Data Transferred	Average Time Cost
File Create	1.07KB	0.838s
File Change	1.06KB	0.676s
File Delete	1.07KB	0.675s
File Share	1.07KB	0.790s

4.2 Evaluation of BWH Attack

In this section, we evaluate the block withholding attack instance by considering a private PoW-based Blockchain that may be operable in small-scale enterprises. It is assumed that all nodes mine on a single pool, and one of them is malicious in nature and looks forward to disrupt the honest mining performed by rest of the nodes. We consider the initial total mining power of the pool (M) is 100 GH/s and the difficulty of crypto-puzzle (D) to be 2096, which is kept fixed throughout. Considering a homogeneous environment, where all the miners have equal computational power, we have experimented by provisioning extra hashing to the attacker and observing the average reward variation. All of the miners including the attacker are assumed to have 20% of the hashing power in the beginning. For computing number of shares in a round, we assume the value of κ as 10, which is typically half number of bits than the actual difficulty (D). The total shares submitted by a miner with power α_i , denoted by $h(\alpha_i)$, is sampled from a Poisson distribution of mean $\alpha_i \kappa$.

Since the attacker has possible means of increasing its hashing capability, we first analyze the impact of such action on the overall reward of the attacker as well as the honest miners. Considering a maximum reward (R) of 10 for successfully mining a block, we can observe from Figure 21 that increasing hash power of an attacker in the pool mining has an adverse effect on the honest miner group. Therefore, the honest miners' reward decreases gradually as depicted in the red dotted line. Although the attacker could potentially increase its hashing ability to a higher amount, such a costly attempt is not necessary for demotivating honest miners to leave the pool. Instead, the attacker would prefer to have a threshold amount of computational power so that the total reward of honest miners just goes below the attacker's reward, which happens to occur at $x = 0.6$ in our considered case. When the submitted shares are considered with respect to increasing hashing power of attacker, we could observe from Figure 22 that the total number of share submissions from attacker increases exponentially, whereas the honest pool's submissions are fixed on an overage. This situation is created because the attacker has growing computational power, whereas the honest miners' hashing limits are fixed.

In Figure 23, the reward variation with respect to attacker's increasing hashing ability is depicted, when the pool uses the PPLNS reward mechanism. Here, we observe that the overall reward a pool could obtain can be higher than the proportional pool. This is because more than one successful block could be found during the submission of last N shares. However, when the attacker gathers extra computational power, it can take away the honest miners' incentives of mining in a pool. We

can observe that with additional power of $y = 0.6 + E$, $E > 0$, the attacker dominates over the honest pool members. Another interesting observation can be made from Figure 24, which shows that with increasing the window size in multiples of difficulty (D), the total reward to the attacker does not increase at growing rate, whereas the honest miners are rewarded higher than attacker. This happens because an increased number of shares come from the honest pool, whereas the attacker with a constant hash rate could not make profit unless extra computational power is added. This gives us an idea of how the window size (N) can be chosen to build a barrier for a block withholding attacker that aims to take away the honest miners' rewards.

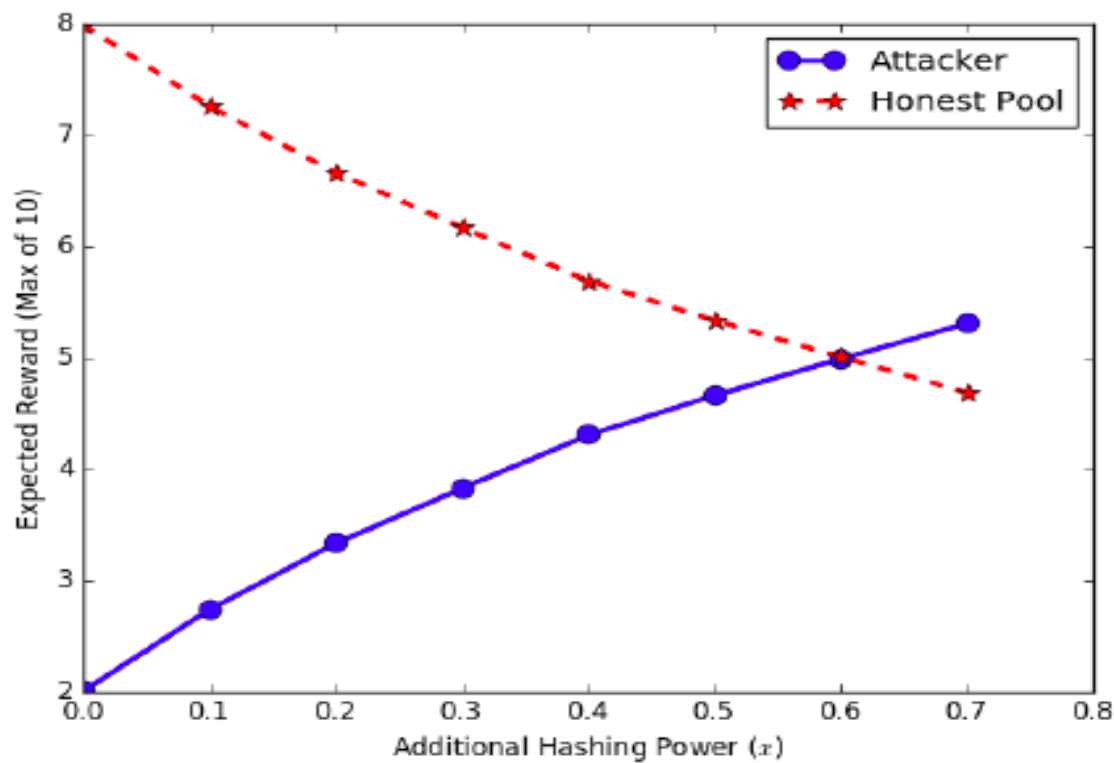


Figure 23. Effect on Average Reward in Proportional Pool

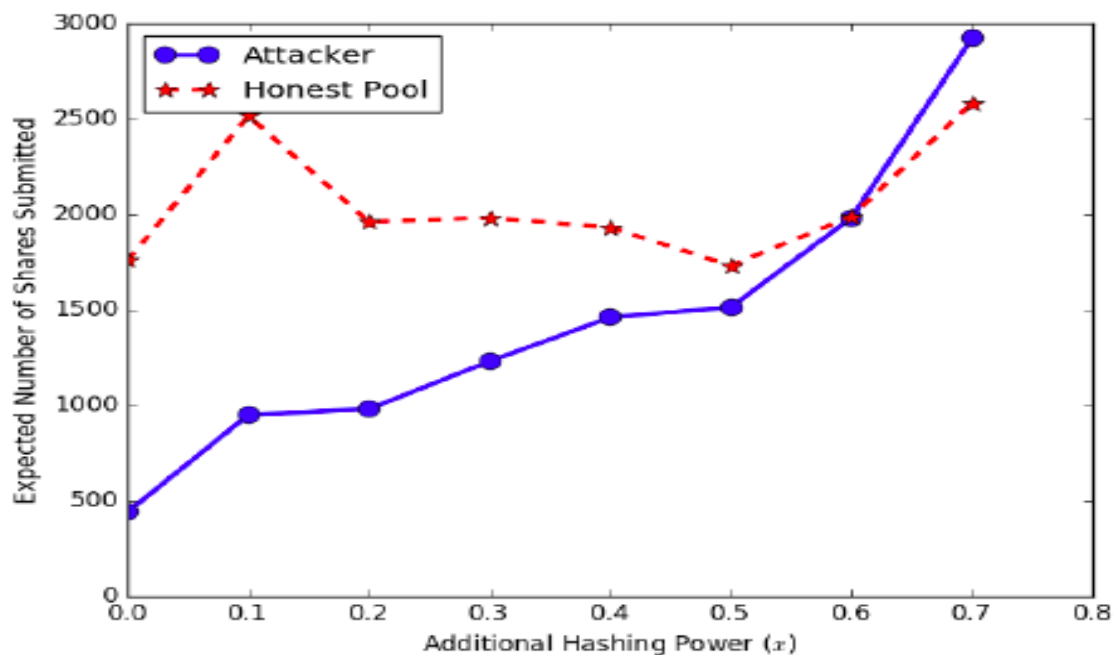


Figure 24. Effect on Average # of Shares in Proportional Pool

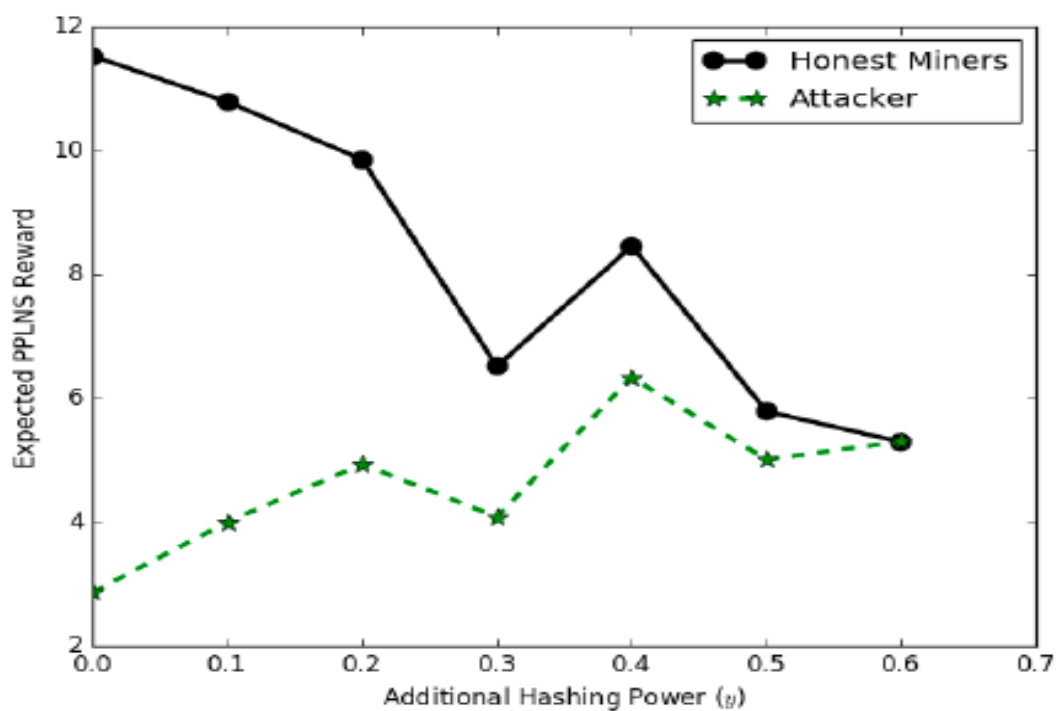


Figure 25. Effect on Average Reward in PPLNS Pool

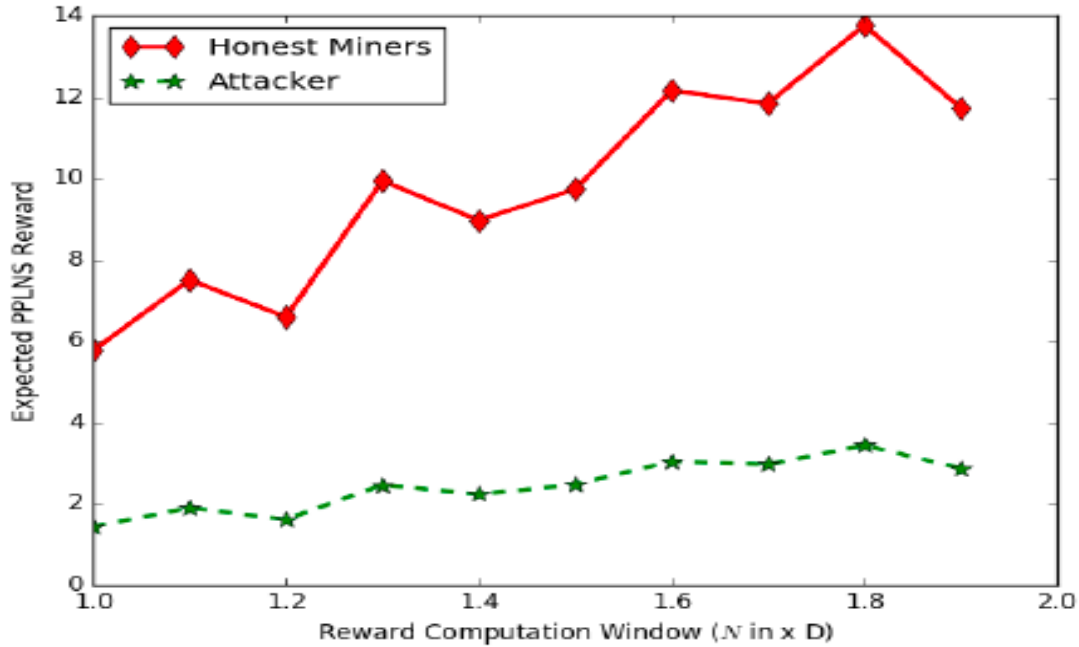


Figure 26. PPLNS Reward Variation Vs. Window Length (N)

4.3 DDOS Attack Results

In this section, we describe the experiments we performed to compare our designs when the mempool is under attack. We present a scenario similar to [69], where a group of attackers flooded Blockchain mempools with dust transactions.

For our simulations, we generate a series of legitimate and spam transactions. Each transaction has age, relay fee and mining fee features. For both types of transactions, we normally distribute the relay fee and mining fee over a selected range of fee values. As per our attack procedure, we generate the spam transactions with low age factor. Although, we observed in the distribution phase that an attacker may possess some transactions which can have some age value. To capture that, we normally distribute the age of spam transactions from 0 to a lower age limit threshold. For legitimate transactions, we normally distribute the age factor from 0 to an upper age limit.

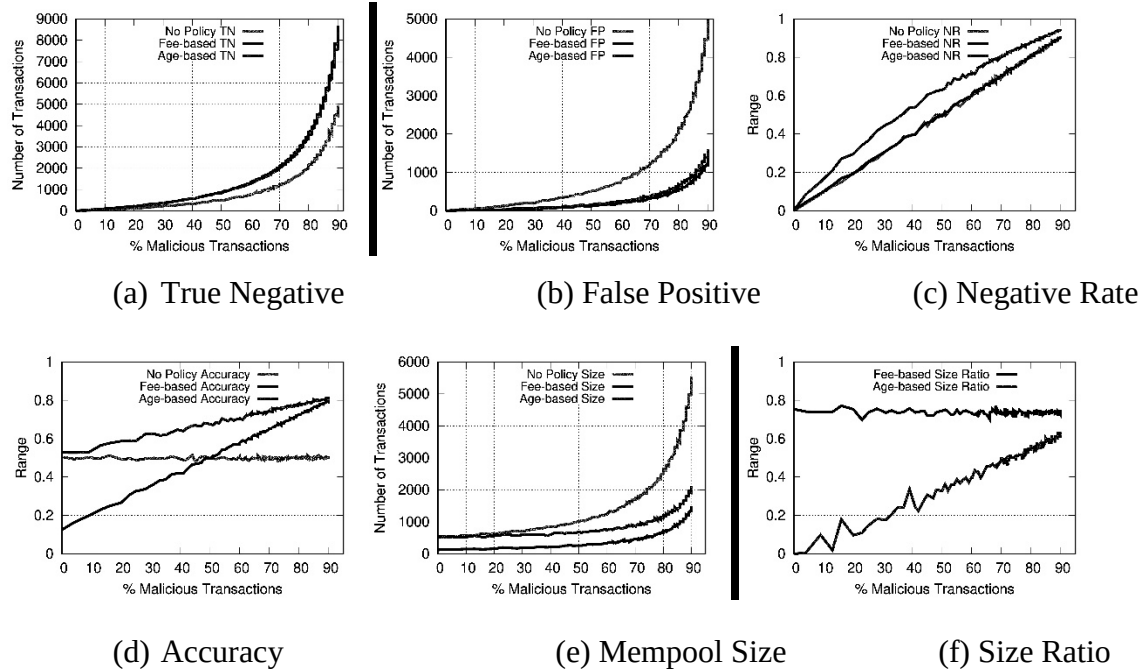


Figure 27. Performance of fee-based and age-based designs under mempool DDoS attack. Notice that when the percentage of malicious transactions is low, indicating less severe attack, the fee-based policy is effective in terms of accuracy and size optimization. However, as the attack rate increases, the age-based policy also becomes effective with better negative rate.

For our experiment, we fixed the size of legitimate transactions and sequentially increased the number of malicious transactions from 0 to 90% of the total transactions. We found (TN) and (FP) from the confusion matrix as suitable measures to determine efficiency of fee- and age-based models in detecting spam. In Figure 25(a) and Figure 25(b), we plot (TN) and (FP) for mempools under no policy, fee-based design and age-based design. It can be observed that both designs are efficient in detecting and discarding malicious transactions when the attack grows. Although both designs effectively reduce (FP), the age-based design achieves better efficiency. As malicious transactions increase, the age-based design discards all the unconfirmed transactions and the fee-based design caps the fee-paying capacity of the attacker.

In Figure 25(c) and Figure 25(d), we plot the negative rate and accuracy. The figure shows that the accuracy of the age-based design increases as malicious transactions increase. The accuracy of fee-based design is low, at the beginning, but it grows as percentage of malicious transactions grows. The reason for the low accuracy at the start is due to the low detection rate (TP). Although, the age-based design appears to be the best choice for detecting malicious transactions, we see in Figure 25(e) that the fee-based model achieves better size efficiency.

The mempool size is determined by the true positive and false positive values (TP+FP). For both designs, we plot the mempool size in Figure 25(e). The figures show that just below the number

of malicious transactions, fee-based design has lower mempool size. After that, the age-based design becomes more size-efficient due to the low (FP). To understand the mempool size optimization ratio, we use the model in (7).

$$sizeratio = 1 - \frac{mempool_size_under_design}{mempool_size_no_design} \quad (7)$$

This equation gives a ratio of size difference of mempool when countermeasures are applied. The results in Figure 25(f) show that the fee-based design achieves a consistent size optimization irrespective of malicious transactions percentage. The size ratio for age-based design increases with the percentage increase in malicious transactions. At 88% of transactions being malicious, the age-based design achieves a size ratio of 60%. From Figure 25, it can be inferred that there is a trade-off between the detection accuracy and size optimization.

We can use the size and accuracy trade-off to select appropriate countermeasures during DDoS attack. If the attack is less severe but the pending transaction backlog is high, the fee-based design will limit the incoming malicious transactions and optimize the mempool size until the backlog is cleared. If the attack is severe [39] and majority of incoming transactions are unconfirmed then age-based design will be more useful in detecting malicious transactions, reducing the FP and optimizing mempool size.

4.4 Blockchain and SGX Evaluation

Intel SGX provides an isolated environment, protecting critical and vulnerable programs from compromise and malicious control. By enclave isolation, we preserve the integrity of code and data, as well as node privacy. By channel isolation, we preserve the privacy of authorized transactors. By the way, the original transaction certificate is used for limited transactions to provide unlinkability. Therefore, the SGX secured certificates also provides untraceability for different transaction parties. Based on the effectiveness of the proposed architecture to defend against all six attacks and vulnerabilities, we evaluate the efficiency of the SGX enabled blockchain service performance. The channel scalability is important in cloud applications where hardware requirement is concerned as a key factor. With the growth of node number, the hardware cost grows linearly while the number of channels scales exponentially. This shows the effectiveness for applications with a larger quantity of nodes, especially cloud applications built on top of distributed ledgers.

To evaluate the time cost with enclave execution, we utilize Intel SDK to test on real SGX machines and compute the time cost for the execution of proof of work algorithms inside an enclave. The test platform has an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz (Skylake with 8MB cache) with 32GB RAM. The mining process uses different value of difficulty defined in the proof of work consensus algorithm. Considering the limited computing capability of a desktop machine, we suppose the value of difficulty ranges between 0 and 13, and observe the performance to gain a basic view of the performance. The time cost for SGX enclave execution is shown in Figure 3.

The concept of difficulty in proof of work represents the degree of the difficulty to which the mining task is set. The higher the difficulty is, the more time is needed to compute and find a solution. From Figure 3 we can conclude that SGX enabled mining algorithm brings an acceptable overhead to find a solution under a certain difficulty, with an average overhead of 21%. The

solution divided by the total time used to find the solution represents the hashing power. We also evaluate the average hashing power of SGX enabled machine against a non SGX enabled machine. Figure 4 shows that SGX enabled machine brings trivial effect on the hashing power.

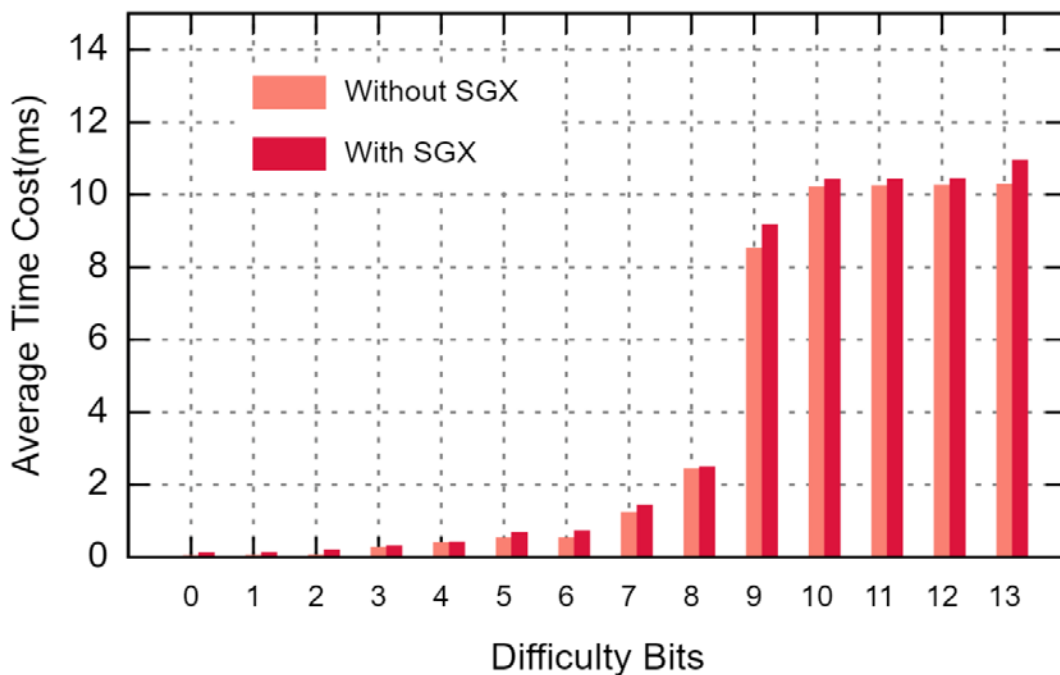


Figure 28. Average Time Cost with Different Difficulty Bits

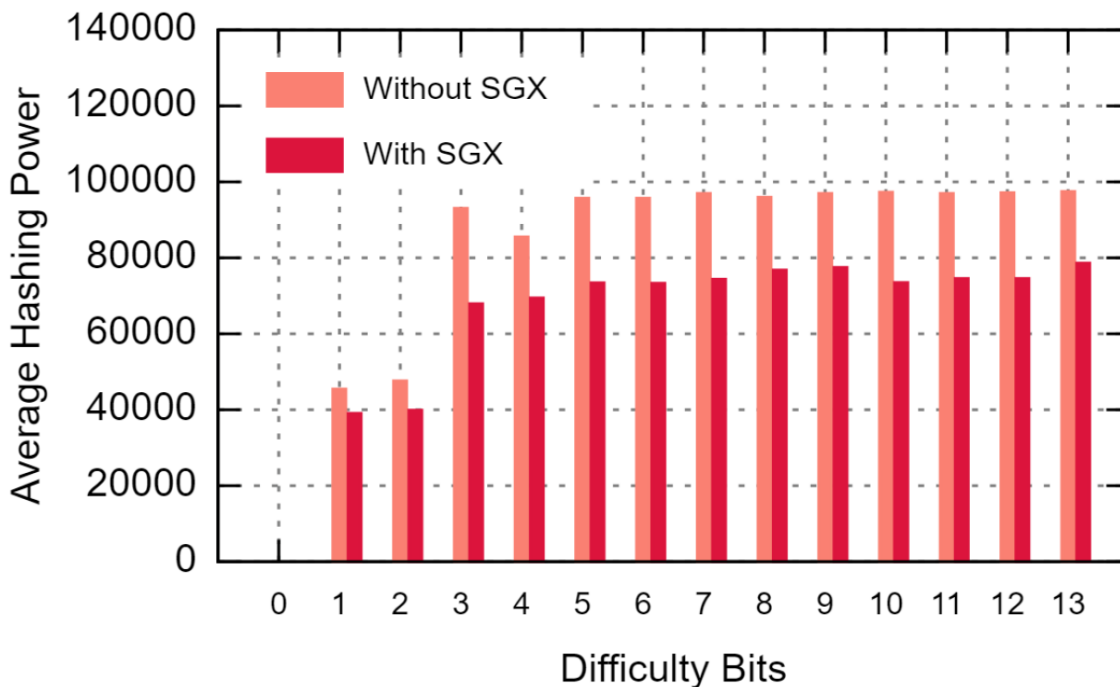


Figure 29. Average Hashing Power with Different Difficulty Bits

The hashing power of SGX enabled machine decreases approximately 21.3% from non SGX enabled machine. This adds to the difficulty of the succeeding of a malicious node to launch various attacks and compromise the blockchain service. Therefore, if 51% of nodes are to be controlled by malicious parties, the hardware cost is considerably high and the time to modify a transaction will increase dramatically. This ensures that the security of SGX blockchain mining is enhanced with acceptable overhead, besides the preserved privacy and trust guarantee.

4.5 Blockchain Evaluation Platform Results

In this section we will focus on the development of the testing platform to evaluate the approaches presented in prior chapters. This chapter will assume the reader has some knowledge of programming or scripting, object-oriented design, and will be most beneficial to readers who are familiar with open source software. Software code will be available to the reader via an online repository and can be downloaded and run on any system that is capable to meet the open sourced software requirements. One example available to the reader is a custom built, simplistic, blockchain simulation application built in C#. This application will require a freely available Microsoft .NET framework. Another example that will be used is a Hyperledger Fabric example. Hyperledger Fabric is an open sourced blockchain application and toolset managed by the Linux Foundation. It will require the user to run the open sourced framework docker, which will allow prebuilt applications containers to be hosted, and a few open sourced software/scripting languages such as GoLang and JavaScript.

The two examples offer opportunity to evaluate algorithms and protocols proposed in previous chapters in a couple of different environments. The simple implementation in C# is a way to simulate some of the theories in an iterative way. Simulating some of the protocols with blockchain is very beneficial in that it allows researchers to evaluate algorithms or protocols without the need to know the operational aspects of the entire system. The Hyperledger Fabric example is a fully capable blockchain platform that can be modified to work in a practical environment. Both examples can be scaled to evaluate performance of algorithms and protocols.

The source code, materials, and additional documentation needed to run the examples can be found on Github at this location :

https://github.com/odu-vmasc/Blockchain_BookChapter

4.5.1 Architecture

Ledgers have been used to record state changes in software and physical systems throughout the existence of humans and documentation. Simply put the ledger is a key value pair where the key and the value can be any number of representations. Every change in state of this key value pair is associated or represented with a hash value that is computed with a combination of the prior hash value and the data representing the key value pair including the change in state. Key pieces of implementation that need to be planned and discussed is how the representation of key aspects will be implemented. These key aspects are;

1. Distributed Ledger
2. Participating nodes
3. Consensus nodes or integration to the peer nodes
4. Messaging methods between participating nodes

Distributed Ledger

From the perspective of software implementation which this chapter focuses on, the ledgers exist natively in many programming languages but referred to more common forms such as; linked list, array, dictionary, or map. Any of these forms of data structures can be used as a ledger though limited to the functionality of the specific nature of those data types.

The most commonly used data structure to represent the blockchain ledger is the linked list. The linked list provides a dynamic structure that can grow without having to specify an initial size. The linked list can be described as a chain of nodes where each node can be a data structure. The structure of the node (block) can be customized based on the type of data that needs to be stored in the list. At a minimum for it to function as a blockchain it will need to contain a link or a pointer that links the node to the prior node. The current nodes hash value needs to be based on the prior nodes state and hash value. Each node needs to be able to link to the next node in the same manner that it linked to the prior node.

In addition to the aforementioned aspects, a developer has great deal of leeway in determining information represented in the node. This should mostly focus on representation of states for the system the blockchain is supporting. A great deal of research is being done to determine limits on the amount of data that is beneficial to store in blockchains. The consensus is that for large data systems the blockchain will only manage the states of the system and the data be stored in more traditional database systems. The traditional data and state for blockchains is typically a key and value pair. The key would be an id that represent the object in which a state is being observed for, and the value being a representation of the state for that object. For more flexible implementations the key and the value would be represented using a string which would allow maximum flexibility. However, there is no reason that other data types could be used for the key and the value such as integers or Boolean values. Strategically choosing the data type could potentially reduce the size of the data allowing for better performing blockchains.

The hash function that is used should be determined based on the latest standards as well as the size of the data that is to be included in the hash. The developer should be mindful of the vulnerabilities of the hash function chosen as well as the performance of the hash function regarding the size of the data that will be hashed. The data that should be hashed is the value of the hash from the prior block and some aspect of the key value pair data associated with the current block. Producing a value from the hash function will then represent a uniquely identified value for the current state block within the blockchain. This hash value then will not only represent a clear value that provides integrity of the data, but also a clear value that provides integrity of the location in the blockchain of that data. Many of the hash functions that exist openly (SHA-2, SHA-3, BLAKE2) are very good and have been tested by the community and the performance of these functions is greatly documented. Implementations of these hash functions exist as libraries in C# and within the Hyperledger Fabric tool kit that can be utilized by the developer for ease of use.

Participating Nodes

Since the blockchain is decentralized in nature, the components that are to be decentralized need to be handled as participating nodes within the system. These nodes ideally would be decentralized on many machines. Depending on the implementation of the blockchain a number of nodes can be defined to do the decentralized tasks. The types of nodes required for the decentralized blockchain system will vary based on the implementation strategies used. At a minimum, peer nodes need to be defined that can communicate with each other in a peer to peer network. These peer nodes would need to be able to read and write to a local ledger to maintain and keep the state of the blockchain. In most cases, the peers need to be able to communicate with each other, but in some instances a manager node can be used. Clearly adding manager nodes makes the system more centralized and strays away from the decentralized purpose and goal of blockchain systems.

Decentralized in nature, the nodes need to be modular and developed as objects that can be scaled in numbers. This can be done using independent computers that run the node application, from virtual machines running the node application, containerized instances of the node application, or using programmed objects within the software code of a simulation. The last example is a way to test blockchain protocols from a centralized environment.

Communication

Protocols of communication from the participating nodes can vary based on the nature and strategy of the implementation. At a minimum the nodes need to be able to communicate with each other in a one to one manner. Each participating node might have the ability to communicate with each other in a peer to peer manner or at least be able to communicate with a managing node that can relay or communicate with all of the other nodes. In many cases a method of broadcast is desired to ensure the most efficient message transfer to all participating nodes. The broadcast method sends a message to all nodes or a portion of nodes on the participating network simultaneously.

Another strategy is gossip network protocol where information propagates throughout the entire network through neighboring peers. This strategy utilizes properties of the behavior of disease spread and epidemics to propagate information through a network of connected nodes. Knowledge of neighboring nodes is needed for this method to work as well as a highly connected network. If there are large portions of the network that are only connected through one link this can greatly degrade the performance of the information spread.

Implementing the communication can be done using standard protocols based on design of the blockchain system. Individual machines can be the participating nodes connected to the internet using HTTP to communicate, in a private setting through a local area network, or even from a virtual network on a single machine. Each one of these setups can be designed with the same communication protocols to allow for future scaling and duplication of work from various settings.

In a simulated environment the communication method becomes very simple as a message can be as simple as a function call to the peer object. The challenge in this case becomes the accurate representation of the asynchronous nature of the real system in the simulated environment. This can be achieved in a couple of ways. By using threading and parallel processing of the individual participating nodes at times when there are simultaneous processes, the slight delay that can occur in the process for each thread or forked process will offer variation in the system more. Additionally, delays can be applied to those message function calls that are simulating the communication of peer objects. The delay value can be a static pause, or a looped process based

on a static iterator value. To make things even more asynchronous it could be more beneficial to utilize a randomly selected value for the pause or looped iterator.

Consensus

A core issue in any asynchronous decentralized systems is consensus of the order of messages to the system. Good consensus methods on the decentralized system ensures that there is trust in the order of messages that happen on the systems. If two messages enter the system at two different ends of the network at nearly the same time, how does the decentralized system determine which message occurred first? Naturally the local participating node to each message will perceive the nearest message as entering before the further message. Ensuring that message that reached the network first is processed in that order is the main challenge in blockchain systems consensus algorithms.

This chapter will briefly describe three consensus algorithms that can be implemented by the reader. The focus on this chapter regarding the consensus algorithms is how these algorithms affect the blockchain systems ability to perform. In many cases the biggest bottleneck on blockchain systems is the consensus algorithms. This is because finding consensus among a large network of nodes is very difficult and time consuming. Consider the speed of a network hop, then consider it having to go through nodes to get to others until all of the nodes or at least a majority of the nodes have received the message. Very quickly one can start to see that depending on the method used to find consensus, the more decentralized the system becomes worse the performance might get. When it comes to performance there tends to be a trade off in blockchain systems, where more performance comes with more centralization. The challenge is to strive for good performance and good decentralization that is appropriate for the use case the blockchain is being designed or implemented for.

4.5.2 Hyperledger Fabric

Hyperledger Fabric is an open sourced toolkit that is managed and maintained by the Hyperledger product of the Linux foundation with much of the information provided here: <https://www.hyperledger.org/>, it allows developers to build a permissioned blockchain system. Permissioned blockchain as it is with Hyperledger Fabric, distributes a ledger amongst participating nodes that are granted permission using cryptographic certifications that are assigned either beforehand or through a certificate authority. By doing this the ledger is not publicly distributed in large numbers but privately distributed to a much smaller number of nodes but guarantee that only those nodes have access to the information in and of the ledger.

Since nodes require access, Hyperledger Fabric is considered a permissioned blockchain. This type of system has pro's and con's. The benefits of permissioned systems are that the content and system remains inaccessible to the public. Permission and access control require a protocol for distributing and enforcing access such as certificate authority nodes. The cost of having the data private is that the immutable properties are limited to a sub-network of limited machines. The public networks though open to all connected devices offer a bigger opportunity to distribute the blockchain ensuring a larger network and a much smaller probability that large portion of the network would be controlled in order gain majority.

Node Types

- o Client - A node that is given permission or allowed to communicate with chaincode installed on peer nodes.
- o Peer - A node that participates on the network as validating node, host to a local copy of the ledger, and host of chaincode.
- o Orderer - A node that provides organization of the participating peer nodes. Since Hyperledger Fabric v1 the orderer node determines the consensus method used and can handle much of the message broadcasts to all participating nodes.
- o Certificate Authority - Node that provides cryptographic files with permissions to communicate and operate with other Hyperledger Fabric nodes.
- o Kafka and Zookeeper - Special nodes that allow for the operation of Kafka consensus with Hyperledger Fabric.

Docker

Docker is a framework to build containerized images of aspects of a computer system to provide a means to rapidly start, scale, and run varying types of systems and applications on any environment. The idea is built on the premise that no matter what your system setup is (os, parameters, software), one could host the docker platform which allows the user to launch docker containers. Once the docker container is launched the container can interact with other containers as well as the physical machine the developer is on through the network interfaces that the docker platform sets up. The developer can also port forward internet traffic to that machine to specific docker containers to interface a container with the network.

A docker container can run on any machine that is hosting the docker platform regardless of the operating system. The docker containers represent and function as a complete system that is running a application or service. The container is not a virtualized machine and only utilizes enough resources to perform the tasks required for the application or service. By doing this a more efficient deployment of the applications and services are provided. This type of system also allows for an easy way to scale up systems where instead of having to create new machines and adding them to the network, one could just start up more docker containers of a type.

Starting docker containers firstly requires the development machine to have the docker platform installed. The developer then can install toolsets that help manage the docker containers such as docker-compose. The docker platform and the docker-compose toolset can both be installed on windows, Linux, and apple environments. For the most part, starting, stopping, and general management of the docker containers is done from the command line interface or terminal of the developer machine. Containers are represented as image files that are stored locally on the machine that is hosting the docker platform. These image files can be generated by the developers by building or modifying existing base images, or by creating completely new images from their own development environment. Docker provides a hub service where they host publicly many types of docker containers that developers can use or build from.

Hyperledger Fabric Example Exercise

In this section, we will demonstrate the operations required to conduct development on hyperledger fabric. For more details on the operations, the reader can refer to the Hyperledger Fabric documentation site (<https://hyperledger-fabric.readthedocs.io/>). Using docker-compose the

hyperledger fabric network can be started from a single command using a docker-compose *.yaml file. The yaml file contains the information needed to define which images to start, what information to copy to the images on startup, what ports to communicate with the images on, what scripts or applications to start up on image boot, and what environment variables should be established and set. Assuming the docker and docker-compose environment has been set up and tested on the development machine the user should be able to follow the instructions in this section. The graphics and examples will be shown using an Ubuntu linux terminal for reference.

Using a terminal/command line interface, navigate to the directory with the scenario example files. This will be referred to as the directory *bookchapter*. Inside *bookchapter* you will notice a few other directories and several files. All filetypes of *.yaml are configure files that are used by docker-compose to either start up or assist in starting up several docker containers. The directory contains yaml files for various hyperledger fabric setups. The directory *e2e_cli* contains files and directories needed for the fabric client, the participating node that will allow us to interact with the hyperledger fabric system. The directory *kafka* contains the files and directories needed to start up a hyperledger fabric system that uses the kafka consensus algorithm. The directory titled *fabric-samples* that contains several samples provided by the hyperledger fabric community. One of the directories inside the directory *fabric-samples* is the directory titled *basic-network* that contains the necessary files to start up hyperledger fabrics basic test network. The directory titled *scripts* contains bash script files that will be loaded into the fabric client docker container. The developer can modify these script files before starting up the fabric client using the text editor of their choice to provide an easier development platform.

Running the First-Network

The Basic-Network is built to run on a computer that is capable of executing bash script files. This chapter describes the execution of the first-network in a linux environment. Once the developer has installed the docker environment, the docker compose tool, and run a test (example the docker whale hello world test), the developer will be ready to build a basic hyperledger fabric system. The most basic example is the hyperledger fabrics first-network. Located in the directory *basic-network* there are a few script files (file type *.sh), a few config files (file types *.yaml or *.yml), and two directories; *config* and *crypto-config* containing the needed cryptographic security files and initial states for the blockchain setup.

To start up the first network the developer needs to navigate to the directory *basic-network*. Once inside the directory the developer will need to run the script file *generate.sh*. This file will generate the blockchain initial state files such as the genesis block. After the script has run successfully, the user will see an output similar to Figure 30.

```

pfoytik@vmasc-001780: /usr/local/go/src/github.com/hyperledger/fabric-samples/basic-network$ ./generate.sh
org1.example.com
2018-03-23 15:42:27.833 EDT [common/configtx/tool] main -> INFO 001 Loading configuration
2018-03-23 15:42:27.849 EDT [common/configtx/tool] doOutputBlock -> INFO 002 Generating genesis block
2018-03-23 15:42:27.850 EDT [common/configtx/tool] doOutputBlock -> INFO 003 Writing genesis block
2018-03-23 15:42:27.857 EDT [common/configtx/tool] main -> INFO 001 Loading configuration
2018-03-23 15:42:27.859 EDT [common/configtx/tool] doOutputChannelCreateTx -> INFO 002 Generating new channel configtx
2018-03-23 15:42:27.859 EDT [common/configtx/tool] doOutputChannelCreateTx -> INFO 003 Writing new channel tx
2018-03-23 15:42:27.867 EDT [common/configtx/tool] main -> INFO 001 Loading configuration
2018-03-23 15:42:27.868 EDT [common/configtx/tool] doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2018-03-23 15:42:27.868 EDT [common/configtx/tool] doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update

```

Figure 30. Screenshot of artifact generation script output

The generate.sh script uses precompiled applications such as configtxgen and cryptogen. The application configtxgen generates initial blocks for the hyperledger fabric and cryptogen generates the cryptographic cert files for the different peers and participating nodes of the hyperledger fabric system.

Once these files have been generated, the developer can now start the basic-network by running the script file start.sh. Using a linux environment from the terminal simply run:

./start.sh

This script file starts up the hyperledger fabric basic network as defined in the config file docker-compose.yml. It then establishes a blockchain channel on a single peer. If the script file has run correctly the developer should see results like Figure 31.

```

Creating network "net_basic" with the default driver
Creating orderer.example.com
Creating couchdb
Creating ca.example.com
Creating peer0.org1.example.com

# wait for Hyperledger Fabric to start
# In case of errors when running later commands, issue export FABRIC_START_TIMEOUT=<larger number>
export FABRIC_START_TIMEOUT=10
echo $(FABRIC_START_TIMEOUT)
sleep $(FABRIC_START_TIMEOUT)

# Create the channel
docker exec -e "CORE_PEER_LOCALMSPID=org1nsp" -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/nsp/users/Admin@org1.example.com/nsp" peer0.org1.example.com peer channel create -o orderer.example.com:7050 -c mychannel -f /etc/hyperledger/configtx/channel.tx
2018-03-23 20:07:44.230 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-03-23 20:07:44.238 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-03-23 20:07:44.239 UTC [channelCmd] InitCndFactory -> INFO 003 Endorser and orderer connections initialized
2018-03-23 20:07:44.239 UTC [msp] GetLocalMSP -> DEBU 004 Returning existing local MSP
2018-03-23 20:07:44.239 UTC [msp] GetDefaultSigningIdentity -> DEBU 005 Obtaining default signing identity
2018-03-23 20:07:44.239 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing local MSP
2018-03-23 20:07:44.239 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining default signing identity
2018-03-23 20:07:44.239 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0A0C060A0747267314D53501280002D...536160706C65436F6E736F7274697560
2018-03-23 20:07:44.239 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: CA3485130E84BE4F1FAE2E6E344335AB2C40540F3316BDEEE9FD062D9D02D
2018-03-23 20:07:44.239 UTC [msp] GetLocalMSP -> DEBU 00a Returning existing local MSP
2018-03-23 20:07:44.239 UTC [msp] GetDefaultSigningIdentity -> DEBU 00b Obtaining default signing identity
2018-03-23 20:07:44.239 UTC [msp] GetLocalMSP -> DEBU 00c Returning existing local MSP
2018-03-23 20:07:44.239 UTC [msp] GetDefaultSigningIdentity -> DEBU 00d Obtaining default signing identity
2018-03-23 20:07:44.239 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0A0C060A1508021A060809080D050522...793B758F0E23C14B8272A79636E692F4
2018-03-23 20:07:44.239 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: EA55089C9508FF1B185DBA174B47A078B455975A2B286442FF12CA68C34383AF
2018-03-23 20:07:44.261 UTC [msp] GetLocalMSP -> DEBU 010 Returning existing local MSP
2018-03-23 20:07:44.261 UTC [msp] GetDefaultSigningIdentity -> DEBU 011 Obtaining default signing identity
2018-03-23 20:07:44.261 UTC [msp] GetLocalMSP -> DEBU 012 Returning existing local MSP
2018-03-23 20:07:44.261 UTC [msp] GetDefaultSigningIdentity -> DEBU 013 Obtaining default signing identity
2018-03-23 20:07:44.261 UTC [msp/identity] Sign -> DEBU 014 Sign: plaintext: 0A0C060A1508021A060809080D050522...74C2D1738C5412080A021A0612021A06
2018-03-23 20:07:44.261 UTC [msp/identity] Sign -> DEBU 015 Sign: digest: 696880852861EDCFD18501F05E5A037A8613A253454083FCCA6278E67A3321
2018-03-23 20:07:44.262 UTC [channelCmd] readBlock -> DEBU 016 Got status: "orderer.deliverResponse_Status
2018-03-23 20:07:44.262 UTC [msp] GetLocalMSP -> DEBU 017 Returning existing local MSP
2018-03-23 20:07:44.262 UTC [msp] GetDefaultSigningIdentity -> DEBU 018 Obtaining default signing identity
2018-03-23 20:07:44.262 UTC [channelCmd] InitCndFactory -> INFO 019 Endorser and orderer connections initialized
2018-03-23 20:07:44.402 UTC [msp] GetLocalMSP -> DEBU 01a Returning existing local MSP
2018-03-23 20:07:44.402 UTC [msp] GetDefaultSigningIdentity -> DEBU 01b Obtaining default signing identity
2018-03-23 20:07:44.403 UTC [msp] GetLocalMSP -> DEBU 01c Returning existing local MSP
2018-03-23 20:07:44.403 UTC [msp] GetDefaultSigningIdentity -> DEBU 01d Obtaining default signing identity
2018-03-23 20:07:44.403 UTC [msp/identity] Sign -> DEBU 01e Sign: plaintext: 0A0C060A1508021A060809080D050522...A6D04DC405712080A021A0612021A06
2018-03-23 20:07:44.403 UTC [msp/identity] Sign -> DEBU 01f Sign: digest: 5F460463F40ABC004719A0C5A866C98AE509CF32E052AA502ABE508101D97
2018-03-23 20:07:44.408 UTC [channelCmd] readBlock -> DEBU 020 Received block:0
2018-03-23 20:07:44.469 UTC [main] main -> INFO 021 Exiting....
# Join peer0.org1.example.com to the channel.
docker exec -e "CORE_PEER_LOCALMSPID=org1nsp" -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/nsp/users/Admin@org1.example.com/nsp" peer0.org1.example.com peer channel join -b mychannel.block
2018-03-23 20:07:44.568 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-03-23 20:07:44.568 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-03-23 20:07:44.569 UTC [channelCmd] InitCndFactory -> INFO 003 Endorser and orderer connections initialized
2018-03-23 20:07:44.569 UTC [msp/identity] Sign -> DEBU 004 Sign: plaintext: 0A0A070A5C08011A0C0809080D05050510...426120D088991A080A080A080A080A080A08
2018-03-23 20:07:44.569 UTC [msp/identity] Sign -> DEBU 005 Sign: digest: F93E3C360A4167630C204E0422B6121C5776E2347EA63571AC5507C9E008B0
2018-03-23 20:07:44.925 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!
2018-03-23 20:07:44.925 UTC [main] main -> INFO 007 Exiting....

```

Figure 31. Screenshot of console outputs from start script for the basic-network example

The start.sh script uses docker-compose and the config file docker-compose.yml to start up the basic-network. The docker-compose toolset reads the config file to determine what container images need to start, what data will be added to the image, how the image will communicate with the network, and what commands should the container run on startup. The docker-compose toolset when used will then check the local container repository stored on the machine and check to see if the container image exists locally, if not then it will download it from either a specified location or by default will reach out to the public docker hub. If necessary the toolset will download the container image, then proceed to start it up based on the information specified in the .yml config file. The main command used to start up the network that start.sh uses is:

docker-compose -f docker-compose.yml up

The docker-compose command uses the -f flag indicating that a file name will be specified for the config file. In this case the config file used is docker-compose.yml. Finally the command up is used indicating that the network should be started up. This command uses the docker compose tool to read in a config file and start up the services or containers that are described in the yml file using the settings and attributes defined in the config file.

Looking at the docker-compose.yml file in a text editor, the developer will see the details shown in Figure 32. The config yml and yaml files splits up sections based on the service or container. The snapshot of figure 3 shows the service of the hyperledger fabric certificate authority (ca). In this config file the label associated with this container is ca.example.com. After the label the image is defined for the container to use, in this file the image is hyperledger/fabric-ca:x86_64-1.0.0. The

characters after the colon indicate the tag or version to use. If there is not a colon and version the docker-compose uses the latest tag or version. This can be problematic as some containers do not have a latest version specified and this can result in an error. For simplistic purposes the config files included with this book all have the x86_64-1.0.0 tag or version applied. More information on the other sections of detail can be found from the online docker documentation.

```
version: '2'

networks:
  basic:

services:
  ca.example.com:
    image: hyperledger/fabric-ca:x86_64-1.0.0
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca.example.com
    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start --ca.certfile /etc/hyperledger/fabric-ca-server-config/org1.example.com-cert.pem
--ca.keyfile /etc/hyperledger/fabric-ca-server-config/a22daf356b2aab5792ea53e35f66fccefd7f1aa2b3a2b92dbfbf96a448ea26a_sk -bb
admin:adminpw -d'
    volumes:
      - ./crypto-config/peerOrganizations/org1.example.com/ca:/etc/hyperledger/fabric-ca-server-config
    container_name: ca.example.com
    networks:
      - basic

  orderer.example.com:
    container_name: orderer.example.com
    image: hyperledger/fabric-orderer:x86_64-1.0.0
    environment:
      - ORDERER_GENERAL_LOGLEVEL=debug
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/orderer
    command: orderer
    ports:
      - 7050:7050
    volumes:
      - ./config:/etc/hyperledger/configtx
      - ./crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com:/etc/hyperledger/msp/orderer
      - ./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com:/etc/hyperledger/msp/peerOrg1
    networks:
      - basic
```

Figure 32. Example view of a docker-compose yml configuration file

Looking at the docker-compose.yml file the developer will notice that there are several services. Each of these services should start up as its own docker container. To check your system to ensure that the containers started up and are running simply use the docker command to view the containers that are currently running on your machine. This can be done from a new terminal window if the prior terminal is displaying logs from the start script. To view the running containers, use the command:

docker ps

Running this command will produce a result like Figure 33

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
6ce400ce0abd	hyperledger/fabric-peer:x86_64-1.0.0	"peer node start"	25 minutes ago	Up 25 minutes
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp	peer0.org1.example.com			
bbe61328047f	hyperledger/fabric-couchdb:x86_64-1.0.6	"tini -- /docker-e..."	25 minutes ago	Up 25 minutes
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp	couchdb			
7d362b698b48	hyperledger/fabric-orderer:x86_64-1.0.0	"orderer"	25 minutes ago	Up 25 minutes
0.0.0.0:7050->7050/tcp	orderer.example.com			
beffb4ce1f11	hyperledger/fabric-ca:x86_64-1.0.0	"sh -c 'fabric-ca-..."	25 minutes ago	Up 25 minutes
0.0.0.0:7054->7054/tcp	ca.example.com			

Figure 33. Example terminal results for the docker ps command of the basic network

This report provides useful information of the id of the containers, the image used for that container, the name, and the status to list a few. At this point the system is running and the developer can build applications that either interact with that system, or to execute commands on the containers that are currently running. At any point a developer can view any console output or logging done by the different containers by using the docker command:

docker logs <id or name>

For example:

docker logs orderer.example.com

The above example will provide an output of all logged outputs that the peer has produced from start to the current time and is shown in Figure 34

```

2018-03-26 13:37:04.171 UTC [orderer/main] func1 -> DEBU 6a3 Closing Deliver stream
2018-03-26 13:37:04.613 UTC [orderer/main] Deliver -> DEBU 6a4 Starting new Deliver handler
2018-03-26 13:37:04.613 UTC [orderer/common/deliver] Handle -> DEBU 6a5 Starting new deliver loop
2018-03-26 13:37:04.614 UTC [orderer/common/deliver] Handle -> DEBU 6a6 Attempting to read seek info message
2018-03-26 13:37:04.614 UTC [policies] GetPolicy -> DEBU 6a7 Returning policy Readers for evaluation
2018-03-26 13:37:04.614 UTC [cauthdsl] func1 -> DEBU 6a8 0xc420122010 gate 1522071424614100888 evaluation starts
2018-03-26 13:37:04.614 UTC [cauthdsl] func2 -> DEBU 6a9 0xc420122010 signed by 0 principal evaluation starts (used [false])
2018-03-26 13:37:04.614 UTC [cauthdsl] func2 -> DEBU 6aa 0xc420122010 processing identity 0 with bytes of 0a07477267314d553012
fc052d2d2d2d424547494e202d2d2d2d0a4d4940434744434341622b6741774942416749515672466b6a587531366f4d764d766f7a5475148456a414b
42676771686b6a4f50515144416a427a4d5173770a435159445651514745774a565574a5544d4245474131554543424d4b5132467361575a76636d35705954
45574d4251474131554542784d4e5532467549455a790a5957356a61584e6a627a45544d4263474131554543684d5162334a6e4d5356c654746746347786c
4c6d4e76625445634d426f474131554541784d54593245750a62334a6e4d5356c654746746347786c4c6d4e7662544165467730784f44417a4d6a4d794d44
41334d6a4661467730794f44417a4d6a41794d4441334d6a46610a4d467378437a414a42674e5642415954416c56544d524d77455159445651514945777044
595778705a6d3979626d6c684d5259774641594456515148457731540a59573467526e4a68626d4e7063324e764d523877485159445651514445785a775a57
56794d433576636d63784c6d56345957317762475575593239744d466b770a457759484b6f5a497a6a3043415159494b6f5a497a6a30444151634451674145
6d62467268626b326e45785748736d2b71346e2f395a594141302f44666461560a3970785639502b434e6344664d426b6d6369377176794970326274457253
3552685972356c57776173414a6f694d68774930495175364e4e4d45737744675940a565230504151482f42415144416765414d417747413155645774542
2f7751434d4141774b7759445652306a42435177496f4167393971474b37356b424770560a703651766f4c476d3242686430556836786e496d41655868494c
786337717777436759494b6f5a497a6a3045417749445277417752414967637a4e5a61422b5a0a4f59674359505a6b7045682b3547654c4a4579445a59450
36336e564e657333484645434948497048765a703748677a445679726e7972556f714534566d33340a7831326a2b4b3568734c796e5a785249a02d2d2d2d2d
454e44202d2d2d2d2d0a
2018-03-26 13:37:04.614 UTC [msp/identity] newIdentity -> DEBU 6ab Creating identity instance for ID &[OrgMSP 9fa8fde63594209
402fcd1a13866e5393cf43172784e1039ff793ffcf1e212]
2018-03-26 13:37:04.614 UTC [cauthdsl] func2 -> DEBU 6ac 0xc420122010 identity 0 does not satisfy principal: The identity is a
member of a different MSP (expected OrdererMSP, got OrgMSP)
2018-03-26 13:37:04.614 UTC [cauthdsl] func2 -> DEBU 6ad 0xc420122010 principal evaluation fails
2018-03-26 13:37:04.614 UTC [cauthdsl] func1 -> DEBU 6ae 0xc420122010 gate 1522071424614100888 evaluation fails
2018-03-26 13:37:04.614 UTC [cauthdsl] func1 -> DEBU 6af 0xc420122010 gate 1522071424614605886 evaluation starts
2018-03-26 13:37:04.614 UTC [cauthdsl] func2 -> DEBU 6b0 0xc420122010 signed by 0 principal evaluation starts (used [false])
2018-03-26 13:37:04.614 UTC [cauthdsl] func2 -> DEBU 6b1 0xc420122010 processing identity 0 with bytes of 0a07477267314d553012
fc052d2d2d2d424547494e202d2d2d2d0a4d4940434744434341622b6741774942416749515672466b6a587531366f4d764d766f7a5475148456a414b
42676771686b6a4f50515144416a427a4d5173770a435159445651514745774a565574a5544d4245474131554543424d4b5132467361575a76636d35705954
45574d4251474131554542784d4e5532467549455a790a5957356a61584e6a627a45544d4263474131554543684d5162334a6e4d5356c654746746347786c
4c6d4e76625445634d426f474131554541784d54593245750a62334a6e4d5356c654746746347786c4c6d4e7662544165467730784f44417a4d6a4d794d44
41334d6a4661467730794f44417a4d6a41794d4441334d6a46610a4d467378437a414a42674e5642415954416c56544d524d77455159445651514945777044
595778705a6d3979626d6c684d5259774641594456515148457731540a59573467526e4a68626d4e7063324e764d523877485159445651514445785a775a57
56794d433576636d63784c6d56345957317762475575593239744d466b770a457759484b6f5a497a6a3043415159494b6f5a497a6a30444151634451674145
6d62467268626b326e45785748736d2b71346e2f395a594141302f44666461560a3970785639502b434e6344664d426b6d6369377176794970326274457253
3552685972356c57776173414a6f694d68774930495175364e4e4d45737744675940a565230504151482f42415144416765414d417747413155645774542
2f7751434d4141774b7759445652306a42435177496f4167393971474b37356b424770560a703651766f4c476d3242686430556836786e496d41655868494c
786337717777436759494b6f5a497a6a3045417749445277417752414967637a4e5a61422b5a0a4f59674359505a6b7045682b3547654c4a4579445a59450
36336e564e657333484645434948497048765a703748677a445679726e7972556f714534566d33340a7831326a2b4b3568734c796e5a785249a02d2d2d2d2d
454e44202d2d2d2d2d0a
2018-03-26 13:37:04.614 UTC [msp/identity] newIdentity -> DEBU 6b2 Creating identity instance for ID &[OrgMSP 9fa8fde63594209
402fcd1a13866e5393cf43172784e1039ff793ffcf1e212]
2018-03-26 13:37:04.614 UTC [msp] SatisfiesPrincipal -> DEBU 6b3 Checking if identity satisfies MEMBER role for OrgMSP
2018-03-26 13:37:04.614 UTC [msp] Validate -> DEBU 6b4 MSP OrgMSP validating identity
2018-03-26 13:37:04.615 UTC [cauthdsl] func2 -> DEBU 6b5 0xc420122010 principal matched by identity 0
2018-03-26 13:37:04.615 UTC [msp/identity] Verify -> DEBU 6b6 Verify: digest = 00000000 3d 16 07 d3 a4 cd a4 e6 9f 77 0f 44
cd 3c 54 8f |a.....w.D.<T.|
00000010 25 a4 06 3b 47 4a fe 16 6e a1 00 0d b3 85 05 4c |%...;GJ..n.....L|
2018-03-26 13:37:04.615 UTC [msp/identity] Verify -> DEBU 6b7 Verify: sig = 00000000 30 45 02 21 00 90 36 b8 1e 0b 44 fc 1f
50 45 f2 |0E...1..6...D..PE.|
00000010 99 ee af 4c ed a1 3b a1 2a 76 c4 5b 09 d2 dc b0 |...L...;.*v[....|
00000020 02 53 8f 6b 72 02 29 1c 7f f5 dd e3 d4 84 f3 15 |.S.kr.....|
00000030 a8 ae ab b5 12 70 26 10 8b 34 17 ad 40 e8 e5 c8 |.....p8..4..0...|
00000040 27 cb cb 4b 32 29 28 |'..K2)(|
2018-03-26 13:37:04.615 UTC [cauthdsl] func2 -> DEBU 6b8 0xc420122010 principal evaluation succeeds for identity 0
2018-03-26 13:37:04.615 UTC [cauthdsl] func1 -> DEBU 6b9 0xc420122010 gate 1522071424614605886 evaluation succeeds
2018-03-26 13:37:04.616 UTC [orderer/common/signfilter] Apply -> DEBU 6ba Forwarding validly signed message for policy &[%s(*c
common.ImplicitMetaPolicy=&(Readers 0)] [%s(int=1)] [%s(*policies.ImplicitMetaPolicy=&{0xc4202cfe40 1 [0xc420195008]}] [%s(*p
olicies.ImplicitMetaPolicy=&{0xc4202cfe40 1 [0xc420194f68]}]]
2018-03-26 13:37:04.616 UTC [orderer/common/deliver] Handle -> DEBU 6bb [channel: mychannel] Received seekInfo (0xc420bfe3e0)
start:cspecified:cnnumber:1 -> stop:cspecified:cnnumber:18446744073709551615 ->

```

Figure 34. Terminal view of the output logs produced by the orderer node

In this example, the orderer logs are displayed where timestamps are provided with debug outputs associated with the actions of the orderer at that time. For situations where the developer wants to keep display a real-time feed of the logs, the developer can include the `--follow` command for example:

docker logs --follow orderer.example.com

This command will occupy your terminal indefinitely until the docker container is stopped. If the developer intends to do this make sure that it is done in a spare opened terminal dedicated to the real-time log feed of that docker container.

Congratulations, at this point a basic Hyperledger Fabric network was started up and observed. To take the network down, the developer can use the stop script file. The stop file can be executed in a linux terminal using the following command:

./stop.sh

The main command that is used is the following command:

docker-compose -f docker-compose.yml stop

This command will stop the containers but save their current settings in the docker environment for later use. To stop and kill the containers in order to use the name or other unique settings with other containers the developer could use the teardown.sh by executing the following command in a linux terminal:

./teardown.sh

The main command that is used is the following command:

docker-compose -f docker-compose.yml down

This command essentially does the opposite of the up command, the up command initializes and starts up the containers, the down stops the containers and then removes their content from the docker environment. In general, the docker environment will keep a cache of previously used container images in its local library to speed up the startup process in case the developer decides to use those images later.

Running the Kafka Network

In the prior section the basic network was started. The basic network used a simple solo consensus mechanism where all transactions in the system are verified with one of the peers only. This type of system is great for starting up very simplistic prototype networks. It is beneficial because the basic communication structure between a single peer, orderer, and client can be tested. Once this is confirmed the user can run a more complex system where a more distributed decentralized consensus is required. In these types of systems more peer nodes are required to run certain consensus algorithms. In the case of the next example the Kafka consensus will require a minimum of one orderer nodes, a peer node per channel or organization, 4 kafka nodes, and 3 zookeeper nodes. In addition, a client node will be started to interact with the kafka network.

To start this network, navigate to the directory titled kafka. In this directory there are a few config yaml files, a directory with generated system files such as the genesis block and channel information and a directory with the cryptographic credentials for each node to use. In the basic-network example a script was used to start up the network, in this example we will use the docker-compose command to start up the orderer-kafka.yaml network. The command to do this is:

docker-compose -f orderer-kafka.yaml up

After running this the developer should see outputs from each of the peer nodes, the kafka nodes, the zookeeper nodes, and the orderer. These log outputs let you the developer know each container is up and running. If all of the containers start up but an error is presented this is not catastrophic and the system can still perform as expected. Figure 35 shows an image of the orderer-kafka start up and a minor http error:

```

peer1.org1.example.com | 2018-03-26 20:08:36.596 UTC | [chaincode] processStream -> DEBU 181 [74485c27]Received message COMPLETED from shim
peer0.org1.example.com | 2018-03-26 20:08:37.965 UTC | [logging] setModuleLevel -> DEBU 194 Module 'grpc' logger enabled for log level 'ERROR'
peer0.org2.example.com | 2018-03-26 20:08:38.220 UTC | [logging] setModuleLevel -> DEBU 184 Module 'configvalues/msp' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.598 UTC | [chaincode] HandleMessage -> DEBU 182 [74485c27]Fabric side Handling ChaincodeMessage of type: COMPLETED in state ready
peer1.org1.example.com | 2018-03-26 20:08:38.598 UTC | [chaincode] HandleMessage -> DEBU 183 [74485c27-8668-42fc-93d6-ce7f8d17ede3]HandleMessage- COMPLETED. Notify
peer0.org2.example.com | 2018-03-26 20:08:38.598 UTC | [chaincode] notify -> DEBU 184 notifying TxId:74485c27-8668-42fc-93d6-ce7f8d17ede3
peer1.org1.example.com | 2018-03-26 20:08:38.220 UTC | [logging] setModuleLevel -> DEBU 180 Module 'msp' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.598 UTC | [chaincode] Execute -> DEBU 185 Exit
peer1.org1.example.com | 2018-03-26 20:08:38.598 UTC | [sccapi] deploySyscc -> INFO 186 system chaincode qscs/[github.com/hyperledger/fabric/core/chaincode/qscs] deployed
peer1.org1.example.com | 2018-03-26 20:08:38.598 UTC | [nodeCmd] initSysccs -> INFO 187 Deployed system chaincode
peer1.org1.example.com | 2018-03-26 20:08:38.598 UTC | [nodeCmd] serve -> INFO 188 Starting peer with ID=[name:'peer1.org1.example.com' ], network ID=[dev], address=[172.22.0.11:7051]
peer1.org1.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 180 Module 'peer/gossip/sa' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.598 UTC | [nodeCmd] serve -> INFO 189 Started peer with ID=[name:'peer1.org1.example.com' ], network ID=[dev], address=[172.22.0.11:7051]
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 180 Module 'peer/gossip/mcs' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 186 Module 'gossip/gossip' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 187 Module 'gossip/comm' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 188 Module 'configvalues/msp' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 190 Module 'gossip/service' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 180 Module 'msp/identity' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 191 Module 'gossip/discovery' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 180 Module 'msp' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 192 Module 'gossip/state' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 193 Module 'gossip/pull' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 184 Module 'peer/gossip/sa' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 194 Module 'gossip/election' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 195 Module 'kvledger.utli' logger enabled for log level 'INFO'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 196 Module 'gossip/pull' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 196 Module 'kvledger' logger enabled for log level 'INFO'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 187 Module 'peer/gossip/mcs' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.221 UTC | [logging] setModuleLevel -> DEBU 197 Module 'ledgermgmt' logger enabled for log level 'INFO'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 198 Module 'gossip/comm' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.222 UTC | [logging] setModuleLevel -> DEBU 198 Module 'cauthdsl' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 191 Module 'gossip/gossip' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.222 UTC | [logging] setModuleLevel -> DEBU 199 Module 'policies' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 192 Module 'gossip/discovery' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 193 Module 'gossip/state' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.224 UTC | [logging] setModuleLevel -> DEBU 194 Module 'gossip/service' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 194 Module 'grpc' logger enabled for log level 'ERROR'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 195 Module 'gossip/election' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 196 Module 'ledgermgmt' logger enabled for log level 'INFO'
peer1.org1.example.com | 2018-03-26 20:08:38.599 UTC | [logging] setModuleLevel -> DEBU 197 Module 'kvledger' logger enabled for log level 'INFO'
peer0.org2.example.com | 2018-03-26 20:08:38.631 UTC | [logging] setModuleLevel -> DEBU 198 Module 'kvledger.utli' logger enabled for log level 'INFO'
peer1.org1.example.com | 2018-03-26 20:08:38.631 UTC | [logging] setModuleLevel -> DEBU 199 Module 'cauthdsl' logger enabled for log level 'WARNING'
peer0.org2.example.com | 2018-03-26 20:08:38.632 UTC | [logging] setModuleLevel -> DEBU 199 Module 'policies' logger enabled for log level 'WARNING'
peer1.org1.example.com | 2018-03-26 20:08:38.632 UTC | [logging] setModuleLevel -> DEBU 196 Module 'grpc' logger enabled for log level 'ERROR'
WARN: An HTTP request took too long to complete. Retry with --verbose to obtain debug information.
If you encounter this issue regularly because of slow network conditions, consider setting COMPOSE_HTTP_TIMEOUT to a higher value (current value: 60).

```

Figure 35. Terminal output for the kafka example

To ensure that everything is running correctly a docker ps command is done, the developer should see something like Figure 36, where there is a client node, 4 peer nodes (2 for each organization), an orderer, 4 kafka nodes, and 3 zookeeper nodes.

docker ps

```

o->3888|rcb | zookeeper
272c02309904 | mibeijsqdel\lprlrc-zookeeper:x86_64-T*0*0 | \qoqkel-euclibozu*** | 11 mjunf62 980 | nb 11 mjunf62 | 0*0*0*0:35855->5787|rcb| 0*0*0*0:35857->5888|rcb| 0*0*0*0:3585
o->3888|rcb | zookeeper
1524203023 | mibeijsqdel\lprlrc-zookeeper:x86_64-T*0*0 | \qoqkel-euclibozu*** | 11 mjunf62 980 | nb 11 mjunf62 | 0*0*0*0:35852->5787|rcb| 0*0*0*0:35854->5888|rcb| 0*0*0*0:3585
o->3888|rcb | zookeeper
871444519180 | mibeijsqdel\lprlrc-zookeeper:x86_64-T*0*0 | \qoqkel-euclibozu*** | 11 mjunf62 980 | nb 11 mjunf62 | 0*0*0*0:35858->5787|rcb| 0*0*0*0:35860->5888|rcb| 0*0*0*0:3585
230504510009 | mibeijsqdel\lprlrc-k9lk9:x86_64-T*0*0 | \qoqkel-euclibozu*** | 11 mjunf62 980 | nb 11 mjunf62 | 8083|rcb| 0*0*0*0:35858->5885|rcb
5108963155P | mibeijsqdel\lprlrc-k9lk9:x86_64-T*0*0 | \qoqkel-euclibozu*** | 11 mjunf62 980 | nb 11 mjunf62 | 8083|rcb| 0*0*0*0:35880->8085|rcb
86081569006 | mibeijsqdel\lprlrc-k9lk9:x86_64-T*0*0 | \qoqkel-euclibozu*** | 11 mjunf62 980 | nb 11 mjunf62 | 8083|rcb| 0*0*0*0:35881->8085|rcb
40040812819C | mibeijsqdel\lprlrc-k9lk9:x86_64-T*0*0 | \qoqkel-euclibozu*** | 11 mjunf62 980 | nb 11 mjunf62 | 8083|rcb| 0*0*0*0:35885->8085|rcb
030301502729 | mibeijsqdel\lprlrc-olqel6l:x86_64-T*0*0 | \olqel6l 2f94f. | 11 mjunf62 980 | nb 11 mjunf62 | 0*0*0*0:1020->1020|rcb
5->1025|rcb| 0*0*0*0:10023->1023|rcb | bee17*0105*6x9ubf6*com | 11 mjunf62 980 | nb 11 mjunf62 | 1020|rcb| 1024-1023|rcb| 0*0*0*0:10027->1027|rcb| 0*0*0*0:1002
C90812030030 | mibeijsqdel\lprlrc-beel:x86_64-T*0*0 | \beel uoq6 2f94f. | 11 mjunf62 980 | nb 11 mjunf62 | 1020|rcb| 1024-1023|rcb| 0*0*0*0:8027->1027|rcb| 0*0*0*0:8025-
->1025|rcb| 0*0*0*0:8023->1023|rcb | bee17*0107*6x9ubf6*com | 11 mjunf62 980 | nb 11 mjunf62 | 1020|rcb| 1024-1023|rcb| 0*0*0*0:8027->1027|rcb| 0*0*0*0:8025-
59030804099 | mibeijsqdel\lprlrc-beel:x86_64-T*0*0 | \beel uoq6 2f94f. | 11 mjunf62 980 | nb 11 mjunf62 | 1020|rcb| 1024-1023|rcb| 0*0*0*0:1027-1023->1027-1023|rcb
3550104030710 | mibeijsqdel\lprlrc-beel:x86_64-T*0*0 | \beel uoq6 2f94f. | 11 mjunf62 980 | nb 11 mjunf62 | 1020|rcb| 1024-1023|rcb| 0*0*0*0:1027-1023->1027-1023|rcb
->1025|rcb| 0*0*0*0:8023->1023|rcb | bee17*0105*6x9ubf6*com | 11 mjunf62 980 | nb 11 mjunf62 | 1020|rcb| 1024-1023|rcb| 0*0*0*0:8027->1027|rcb| 0*0*0*0:8025-
00030802012 | mibeijsqdel\lprlrc-beel:x86_64-T*0*0 | \beel uoq6 2f94f. | 11 mjunf62 980 | nb 11 mjunf62 | 1020|rcb| 1024-1023|rcb| 0*0*0*0:8027->1027|rcb| 0*0*0*0:8025-
lprlrc-cff | 11 mjunf62 980 | nb 11 mjunf62
000800000008 | mibeijsqdel\lprlrc-zooj2:x86_64-T*0*0 | \p92u -C, mufje 61*** | 11 mjunf62 980 | nb 11 mjunf62
COMMITTEE ID | IHWCE | IHWCE2 | COMMITTEE | CBEVLED | 21W102 | 60812

```

Figure 36. Terminal results for the docker ps command while running the kafka system

In this example the client node will communicate with a peer node which will communicate with

the orderer node, and the orderer node will trust the kafka nodes to verify the order of the message based on a consensus amongst the nodes utilizing the zookeepers. To execute commands with the client node the developer could ssh into that container using the exec command with the -it flag and the containers name. For example to ssh into the client we would use the command:

docker exec -it fabric-cli bash

This will bring up a terminal of the fabric-cli container with the user as root. The developer can now execute commands on the client node. Displaying the contents of the default directory of the container (opt/gopath/src/github.com/hyperledger/fabric/peer) the developer will see three directories, with the one of most importance to the developer being the scripts directory. The scripts directory is a copy from the development machine. Located in the bookchapter directory as described at the beginning of this section under project files, the directory scripts is the folder that is copied to the client node. If the developer wishes to make changes to the script files in this directory it is best to do so before starting up the docker network as the files are copied at startup of the containers. The scripts folder contains script files (.sh) that can be used to handle the large and tedious commands to interact with the blockchain network. The script to run is the initialize_all.sh file by running the command from the default folder:

bash ./scripts/initialize_all.sh

Executing this command will do several things that will take a minute or so, it is important to wait until the script prints all good and shows a large END sign like Figure 37

```
2018-03-26 20:42:37.924 UTC [chaincodeCmd] Install -> DEBU 00f Installed remotely response:<status:200 payload:"OK" >
2018-03-26 20:42:37.924 UTC [main] main -> INFO 010 Exiting.....
===== Chaincode is installed on remote peer PEER3 =====

Instantiating chaincode on all 2 channels (once for each channel)...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=fabric-cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
cat: 'log.txt[A]': No such file or directory
===== Chaincode Instantiation on PEER0 on channel 'businesschannel' is successful =====

CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org2MSP
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=fabric-cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
cat: 'log.txt[A]': No such file or directory
===== Chaincode Instantiation on PEER2 on channel 'businesschannel' is successful =====

===== All GOOD, initialization completed =====

  END

root@fabric-cli:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

Figure 37. Terminal output from the initialize_all script which runs and tests the kafka blockchain

Once the script has finished running, the developer can scroll up to see everything that has run. The script has initialized a channel to operate in called businesschannel, established the peer roles for the multiple organizations, then installed the example chaincode on 2 of the peers one for each

organization. Once the chaincode is started, a new docker container is started up on the machine that is hosting the docker containers. A chaincode container will start for each peer node to use. After the chaincode is installed it is instantiated with an initial transaction. To ensure that the chaincode was installed for each peer, check the docker ps command one more time. The developer should see something similar to Figure 38.

CONTAINER ID	PORTS	IMAGE	COMMAND	CREATED	STATUS
f2e4d99a9978		dev-peer0.org2.example.com-myc9-1.0	"chaincode -peer.a..."	2 hours ago	Up 2 hours
org2.example.com-myc9-1.0		dev-peer0.org1.example.com-myc9-1.0	"chaincode -peer.a..."	2 hours ago	Up 2 hours
16fb35bd76d8		hyperledger/fabric-tools:x86_64-1.0.0	"bash -c 'while tr..."	2 hours ago	Up 2 hours
org1.example.com-myc9-1.0		hyperledger/fabric-peer:x86_64-1.0.0	"peer node start"	2 hours ago	Up 2 hours
bdb8c0be89c8		hyperledger/fabric-peer:x86_64-1.0.0	"peer node start"	2 hours ago	Up 2 hours
b6939cb5b3f5	7050/tcp, 7054-7059/tcp, 0.0.0.0:9051->7051/tcp, 0.0.0.0:9052->7052/tcp, 0.0.0.0:9053->7053/tcp	hyperledger/fabric-peer:x86_64-1.0.0	"peer node start"	2 hours ago	Up 2 hours
322d7dc9b170	7050/tcp, 7054-7059/tcp, 0.0.0.0:7051->7051/tcp, 0.0.0.0:7052->7052/tcp, 0.0.0.0:7053->7053/tcp	hyperledger/fabric-peer:x86_64-1.0.0	"peer node start"	2 hours ago	Up 2 hours
2a9038dc46ad	7050/tcp, 7054-7059/tcp, 0.0.0.0:8051->7051/tcp, 0.0.0.0:8052->7052/tcp, 0.0.0.0:8053->7053/tcp	hyperledger/fabric-peer:x86_64-1.0.0	"peer node start"	2 hours ago	Up 2 hours
ca96f5828039	7050/tcp, 7054-7059/tcp, 0.0.0.0:10051->7051/tcp, 0.0.0.0:10052->7052/tcp, 0.0.0.0:10053->7053/tcp	hyperledger/fabric-peer:x86_64-1.0.0	"peer node start"	2 hours ago	Up 2 hours
03d38f26915a	0.0.0.0:7050->7050/tcp	hyperledger/fabric-orderer:x86_64-1.0.0	"orderer start"	2 hours ago	Up 2 hours
446648f597dc	9093/tcp, 0.0.0.0:32832->9092/tcp	hyperledger/fabric-kafka:x86_64-1.0.0	"/docker-entrypoint..."	2 hours ago	Up 2 hours
4ed972ea06ef	9093/tcp, 0.0.0.0:32831->9092/tcp	hyperledger/fabric-kafka:x86_64-1.0.0	"/docker-entrypoint..."	2 hours ago	Up 2 hours
27b9ae3bf22b	9093/tcp, 0.0.0.0:32830->9092/tcp	hyperledger/fabric-kafka:x86_64-1.0.0	"/docker-entrypoint..."	2 hours ago	Up 2 hours
453024270d6d	9093/tcp, 0.0.0.0:32829->9092/tcp	hyperledger/fabric-kafka:x86_64-1.0.0	"/docker-entrypoint..."	2 hours ago	Up 2 hours
ab14b427a780	0.0.0.0:32828->2181/tcp, 0.0.0.0:32827->2888/tcp, 0.0.0.0:32826->3888/tcp	hyperledger/fabric-zookeeper:x86_64-1.0.0	"/docker-entrypoint..."	2 hours ago	Up 2 hours
e1254563d563	0.0.0.0:32825->2181/tcp, 0.0.0.0:32824->2888/tcp, 0.0.0.0:32823->3888/tcp	hyperledger/fabric-zookeeper:x86_64-1.0.0	"/docker-entrypoint..."	2 hours ago	Up 2 hours
515c053baad4	0.0.0.0:32822->2181/tcp, 0.0.0.0:32821->2888/tcp, 0.0.0.0:32820->3888/tcp	hyperledger/fabric-zookeeper:x86_64-1.0.0	"/docker-entrypoint..."	2 hours ago	Up 2 hours

Figure 38. Terminal output from docker ps command, notice the chaincode containers that have been started

The images with the name that starts with dev- are the chaincode. Just as was done with other containers, the developer can always check the logs of the chaincode containers to receive any logs established in the chaincode source code.

To take down the hyperledger fabric kafka network, first exit the fabric client by simply using the exit command:

exit

Once back in the development machine, use the docker-compose tool to bring down the network with the command:

docker-compose -f orderer-kafka.yaml down

This will stop all the containers and clear out the necessary content to allow the system to be restarted with a different network setup. This process will take a few seconds to a minute and will produce an output like Figure 39 when it is finished.

```
Stopping fabric-cli ... done
Stopping peer0.org2.example.com ... done
Stopping peer0.org1.example.com ... done
Stopping peer1.org1.example.com ... done
Stopping peer1.org2.example.com ... done
Stopping orderer.example.com ... done
Stopping kafka0 ... done
Stopping kafka1 ... done
Stopping kafka2 ... done
Stopping kafka3 ... done
Stopping zookeeper0 ... done
Stopping zookeeper1 ... done
Stopping zookeeper2 ... done
Removing fabric-cli ... done
Removing peer0.org2.example.com ... done
Removing peer0.org1.example.com ... done
Removing peer1.org1.example.com ... done
Removing peer1.org2.example.com ... done
Removing orderer.example.com ... done
Removing kafka0 ... done
Removing kafka1 ... done
Removing kafka2 ... done
Removing kafka3 ... done
Removing zookeeper0 ... done
Removing zookeeper1 ... done
Removing zookeeper2 ... done
Removing network kafka_default
```

Figure 39. Terminal output after stopping the network

The developer can confirm that the containers have been taking down by doing a `docker ps` command to ensure there aren't any containers running from the given examples.

4.5.3 Measures of Performance

When it comes to blockchain performance there are several quantitative measures that are important. The performance measures can be improved but often come at a cost to security from increasing the centralization of the blockchain system. It is recognized that centralized systems have a better ability to function at a higher performance. This is mostly because consensus is not needed, leaving the performance metrics to how quickly the single machine can receive the message and process the request. The decentralized blockchain systems require latency depended message and a process of consensus among the verifying nodes. When it comes to performance of blockchain systems it is very common to refer to the best performance metric as the measure of transactions per second (TPS).

Transactions per second can be defined in a couple of ways. As its basic measure, transactions per second is the rate at which the system can process a transaction. This can be measured in several ways, the time it takes one of the verifying nodes to completely write a transaction to the local ledger, the time it takes a majority of nodes to completely write the transaction to their individual ledgers, or the total number of transactions per time segment and often utilizes some method to batch or the time it takes to process multiple transactions at the same time. The first described transaction can be referred to as the single transaction per second measure, the second type of measure is the total system transactions. In this case the system might not have the fastest single

transaction per second measure but can process many transactions a little slower resulting in a larger number of transaction per time segment. In most real-world scenarios, the developers are mostly concerned with the maximum number of transactions the system can process within a second and tend to not be as concerned with the metric of a single transaction. However, the single transaction metric can be an important metric when comparing system performance from the developer's perspective, especially when there are opportunities to improve systems by adding methods of batching or parallel processing. The single transaction can offer a common ground measurement that allows for an apple to apples comparison.

Figure 40 shows the process of measuring the single TPS for a blockchain system. The simplest way to measure TPS is to record the start time a transaction was sent, then immediately start querying the blockchain system, when transaction that was sent finally shows up in the query record the finish time. Take the difference of the finish time and the start time. This is the time to process a single transaction. In the decentralized systems there is often noise that can cause delays in the transaction such as networking issues, or processing load on the verifying nodes, so it is better idea to repeat this for a number of transactions. The number of transactions should depend on the variance that is observed from initial tests, the more variance the developer sees in the measures of transactions per second, the more measures or samples that should be recorded in order to produce a good representation of the system. The transactions per second can then be described with simple descriptive statistics such as maximum, minimum, mean, mode, and standard deviation.

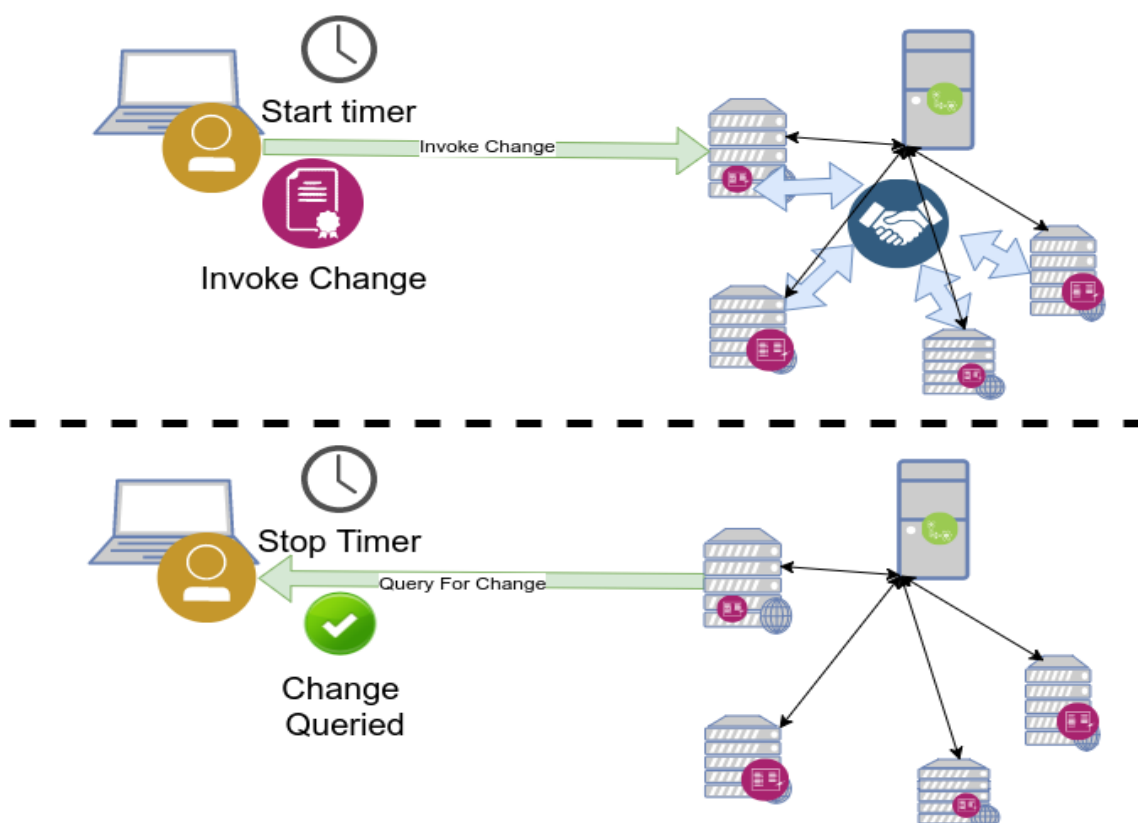


Figure 40. Illustration of the process to measure a single transaction per second

Figure 41 shows the process of measuring the total system TPS for a blockchain system. This

becomes a little more challenging because transactions likely would be coming from many sources all at the same time. To measure total system TPS, the developer will often need to create a recording mechanism on one or many of the validating nodes. The measuring tool can now internally measure the time at which a transaction is reported on the system, and the time at which the peer node is authorized or recognizes that consensus is found to write the transaction to the ledger. There is a lot of detail that is not recorded in this method as the network time of when the transaction is initiated is not recorded. This method also provides the total system time as perceived from one of the verifying nodes. If this method is used it is probably best to do the same method on multiple verifying nodes, then compare the results of many transactions. This will provide a sample that best represents the total system TPS and descriptive statistics can then be used to show the performance such as maximum, minimum, mean, median, mode, and standard deviation.

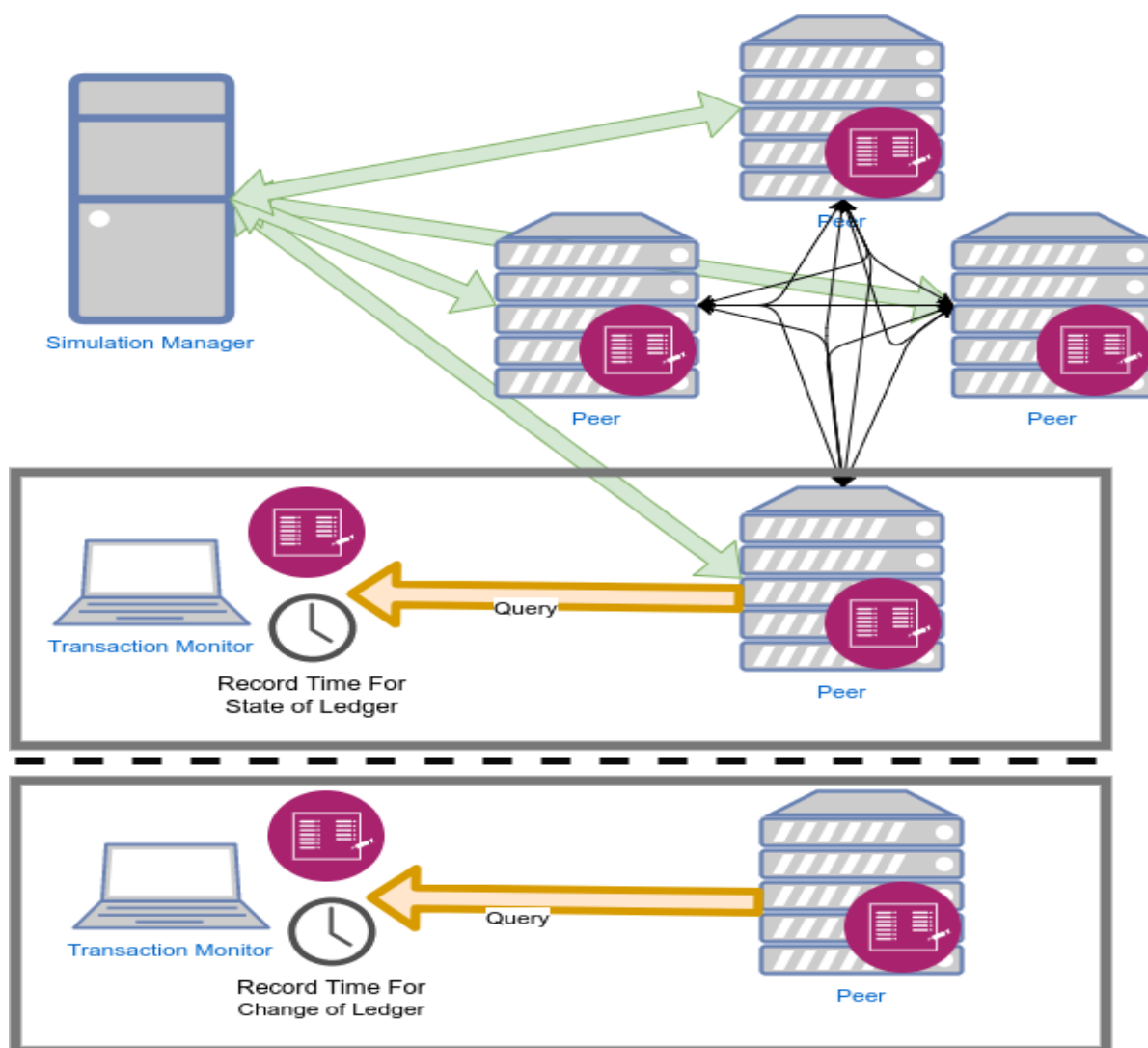


Figure 41. Illustration of measuring total system transactions per second

Performance metrics with the Proof of Stake Simulation

In this section, we provide the performance metrics for a Proof of Stake protocol. Performance metrics are set up in the described simulation and include:

- Average stake claimed per validator.
- Average and total times each validator was the leader.
- Total number of times a leader was selected as validator but did not have the highest stake amount.
- Average, max/min iterations to find a leader.
- Average, max/min time in milliseconds to make progress and extend the blockchain with a new block.

These performance metrics were recorded for various number of validator nodes and various number of transactions for 30 different tests to allow for a good representation of the results since the simulation is using stochastic variables for selection of stake and delay. Initially best-case scenarios are tested without delay to get a ground truth understanding of how the proof of stake application is working. The number of validators has a very large effect on performance of this algorithm mostly because of the round robin nature of all the validators when selecting a leader. The Figure 42 illustrates the performance of PoS enabled blockchain given an ideal scenario of negligible network delay and stake allocation delay. In this case, the latency of the consensus process is majorly dependent on the leader selection procedure. This latency variation over 30 experiments gives us an estimated ideal time of extending the blockchain with a new block. In other words, the duration of an ideal epoch is going to be in the range of 0.4 to 0.8 milliseconds when there are 10 validators and 0.1 milliseconds when 5 validators are present. As the number of validators increase, it is obvious that the leader election process takes more time to definitively find a leader, which is why average latency is increased.

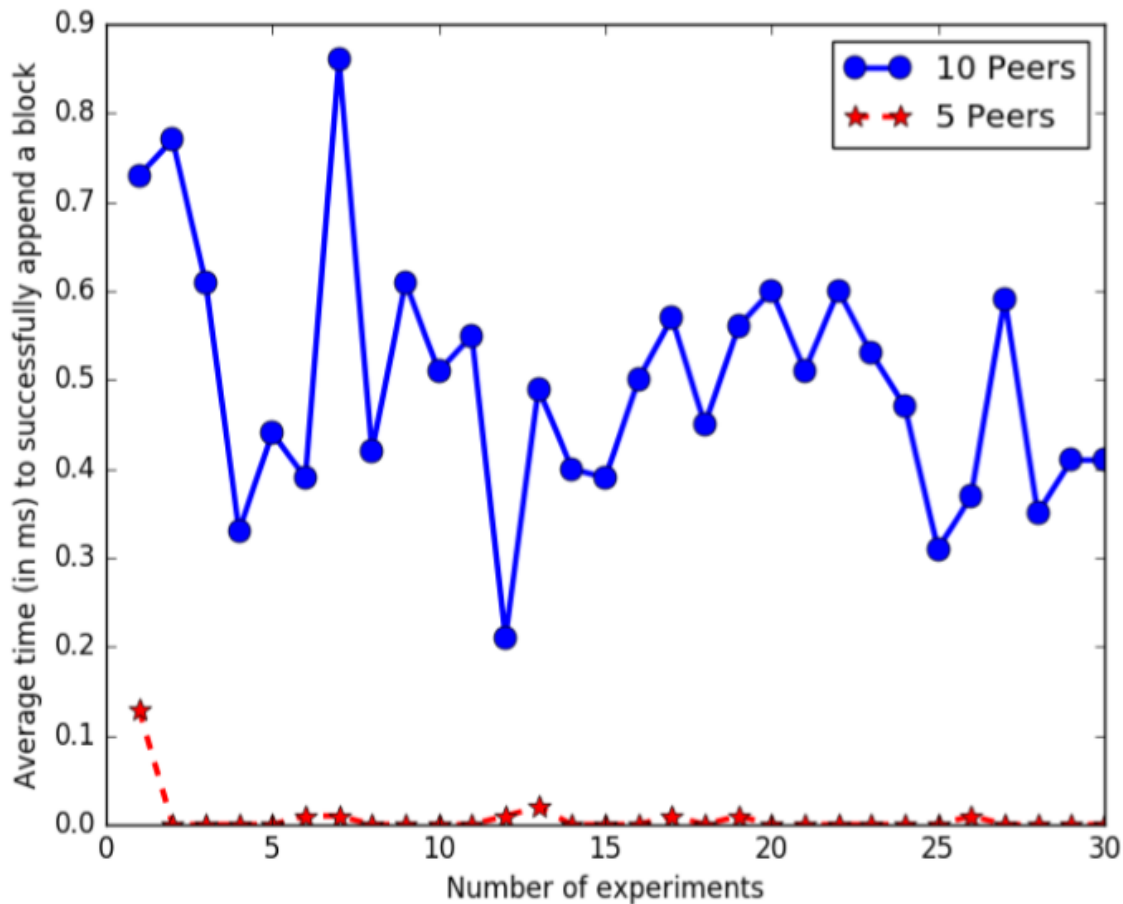


Figure 42. Example output of performance in simulated proof of stake blockchain system

In Figure 43 the similar experiment is conducted but with introduction of random communication delay. The delay component is sampled from a normal distribution in the range of 1 to 5 milliseconds. The 30 experimental rounds show that even with the presence of network delay, our PoS consensus eventually extends the blockchain and hence satisfies the safety and liveness properties.

Experiments were and can be run to see whether the majority stakeholders are always taking the opportunity to become leader, or the minority group of validators have a chance to include their block in the blockchain. For this, tests simulated with the prototype for 100 blocks and checked how many times a leader is selected in an epoch that does not have highest stake in the system. The variation depicted in Figure 44 shows that a validator from the minority stakeholders is selected more than 50% of the time on average, which means that

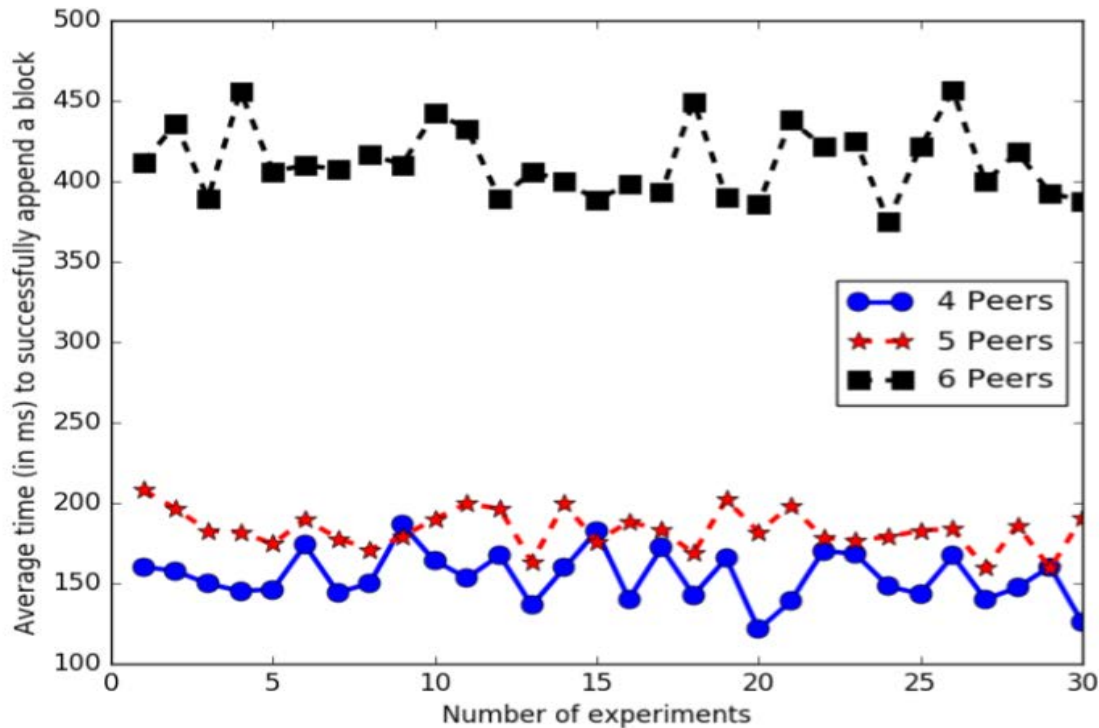


Figure 43. Example output of performance in simulated proof of stake blockchain system with communication delay added

the highest stake holding validator does not get elected as leader all the time. It is interesting to notice that as the number of validators increase, the selected leader is most likely to fall in the minority category because the stake values will be closely distributed and hence the election timeouts will also be similarly distributed. Thus, it is preferred more validators in the system to make it fairer in terms of leader selection process.

Optimization Goals: Current implementation considers that the stake allocation and verification occur at the beginning of every epoch, which can be a major bottleneck in the protocol. Since, stake allocation requires to adjust the actual resource utilization dynamically, it could result extra delay before any validator can verify the staked resource. In order to bypass this issue, we can enforce the validators to stake their resources for a fixed amount of time that is more than a typical epoch duration. A smart contract can be used for this purpose to prohibit release of stakes before the agreed staking time, which can be a regulatory condition for becoming a validator in the distributed computing system.

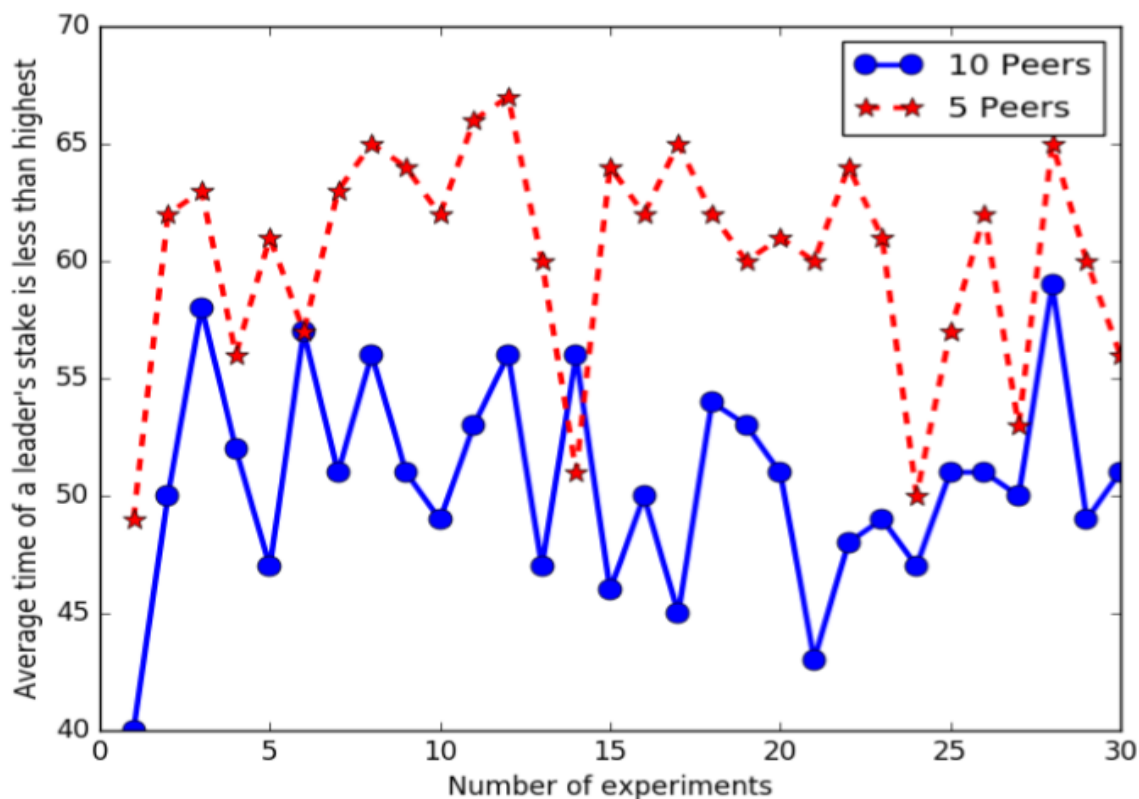


Figure 44. Example output reporting the average number of times the highest staked holder did not get elected as validation leader in the Proof of Stake process

Performance measures with the Hyperledger Fabric Example

The Hyperledger Fabric blockchain system has the ability to plug in various consensus methods. Throughout the development of hyperledger fabric there has been several versions and consensus methods that have been tried and provided. Each consensus algorithm can provide a means for the decentralized peer to peer system to agree on the order and validity of asynchronous transactions. Each algorithm does it with a different strategy. The provided consensus methods within hyperledger fabric that have been provided are, solo consensus, practical byzantine fault tolerant consensus, and kafka consensus.

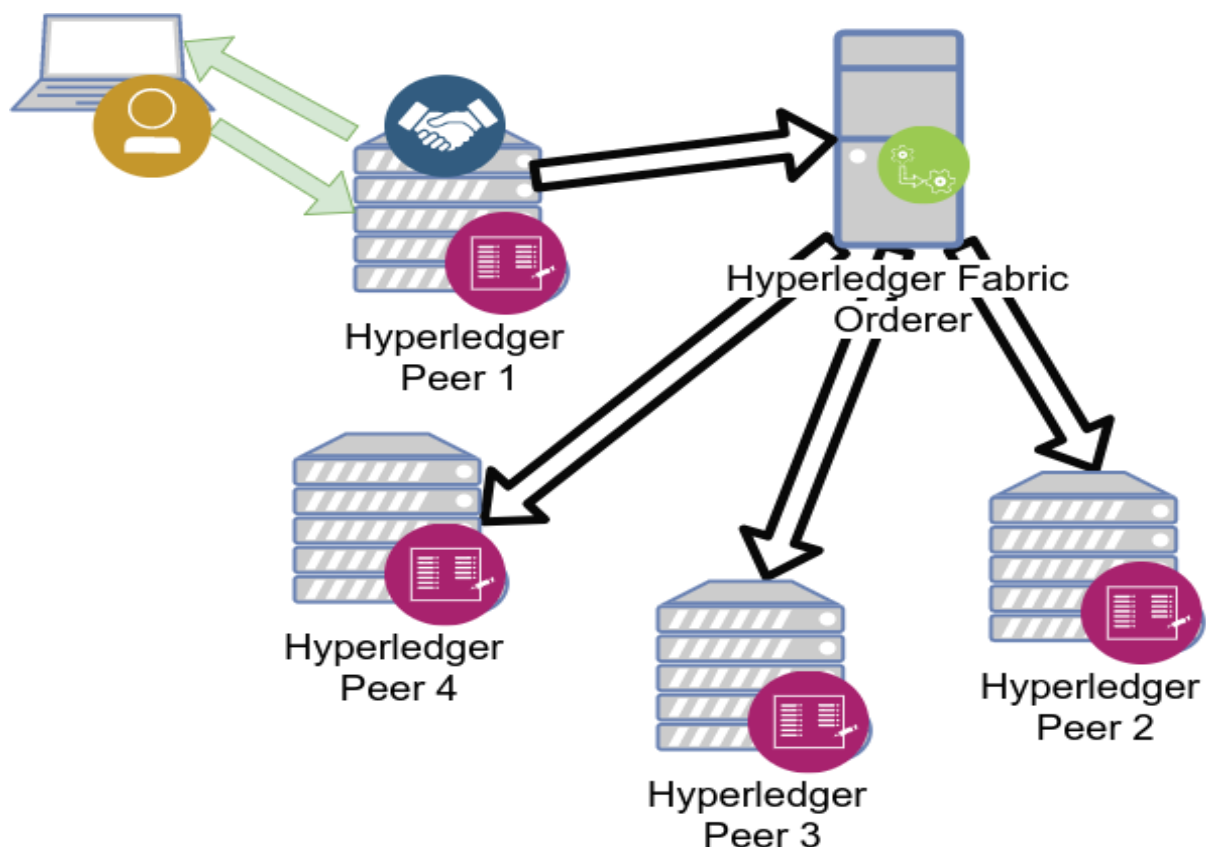


Figure 45. Illustration of a basic configuration of hyperledger fabric

The basic consensus method which is more centralized is referred to as solo consensus and is a good method to test systems. It is the method where a single node controls the consensus for the transactions in the system. This is not recommended for a final production system as it is essentially a centralized blockchain system. In this case a transaction is sent to an assigned peer, the peer verifies the order of the transaction and writes the transaction to its ledger, if there are other nodes on the network it broadcasts the transaction to the other nodes requesting that their ledgers also be updated. This system is not the best as there is a single point of failure to compromise the order of the transactions.

Practical Byzantine Fault Tolerant (PBFT) algorithm is based on the algorithm specified in the following publication. This algorithm focuses on a strategy of many broadcasted messages. This consensus relies heavily on networked communication between all participating peer nodes. Figure 46 shows process of the consensus, where a client starts with the message requesting an update to the blockchain. It is important to remember the scale of the situation, in an actual scenario, there is likely many clients sending requests to multiple peer nodes. The peer nodes need to utilize the PBFT algorithm to agree on the order of the messages received.

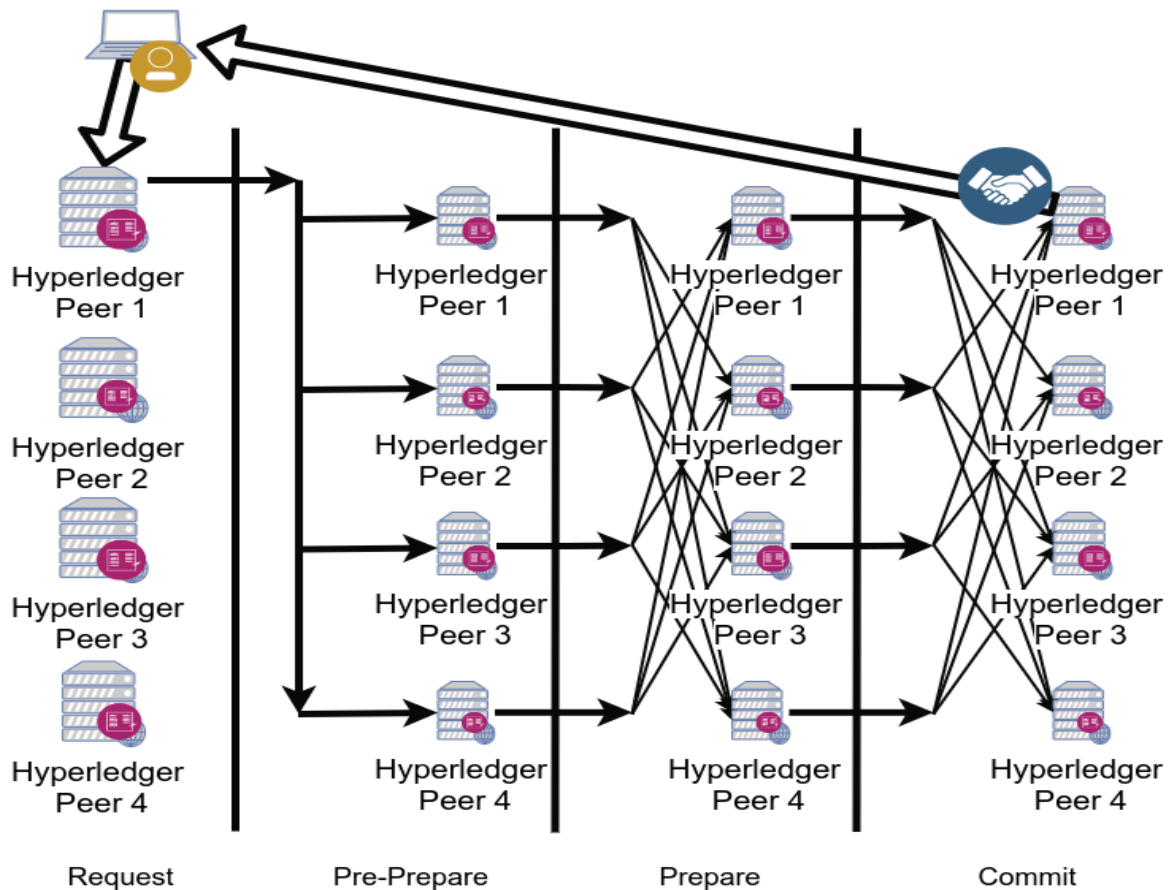


Figure 46. Illustration of a practical byzantine fault tolerant consensus process

As a peer receives the message, it promptly broadcasts the message to all participating nodes. Each node receives the messages for a short period of time. Then responds to all nodes with a broadcast declaring what it believes is the order of the messages received to all participating nodes. Each node then observes the perspective of every other nodes order of transactions. Each node then checks all the responses and if a consensus among all participating nodes is found, the peer nodes will write the request to the blockchain, and finally a response is sent to the client.

Comparisons for transactions per second (TPS) was done for the PBFT with Hyperledger Fabric 0.6, and Kafka with Hyperledger Fabric 1.0. The measure of TPS performed is the single transaction TPS. A client was run using a python script that sends a transaction to the peer network, starts a timer, then repeatedly queries the blockchain until the expected result is produced, finally the transaction timer is stopped and recorded. This occurs one thousand times to observe any fluctuations in the data. Example plots of the data produced by these tests are shown in Figure 47 for the PBFT results and Figure 48 for the Kafka results.

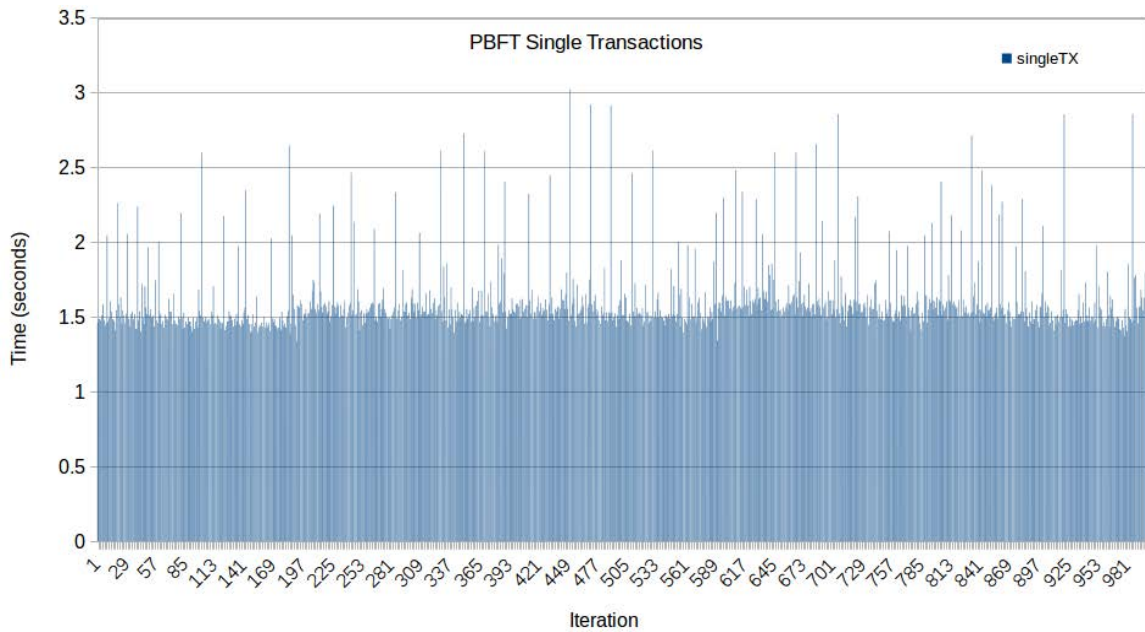


Figure 47. Performance in transactions per second to process a single transaction with PBFT consensus (hyperledger fabric 0.6)

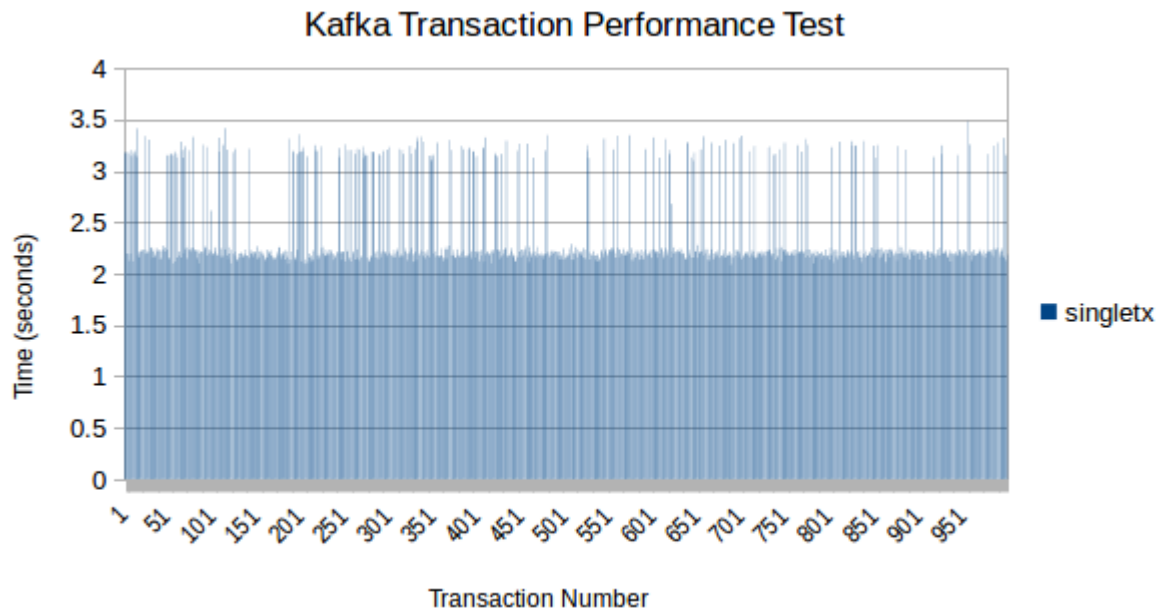


Figure 48. Performance in transactions per second to process a single transaction with kafka consensus (hyperledger fabric 1.1)

The figures show how the TPS results can fluctuate and offers at a quick glance what the amount of time it takes to process the single transaction. These results were generated with a simple network consisting of 4 peer nodes, one client, and all on local machine with virtual network using docker containers. The data can further be processed to populate a comparison table with descriptive statistics. Table 1 provides results of the descriptive statistics for both consensus algorithms.

Table 7. Sample results of performance metrics of single transaction per second for the PBFT and KAFKA consensus

Consensus	PBFT (Hyperledger Fabric 0.6)	Kafka (Hyperledger Fabric 1.0)
Average (TPS)	1.59	2.37
Max (TPS)	3.02	3.48
Min (TPS)	1.34	2.09
Mode (TPS)	1.52	2.2

The results can vary based on different scenarios, but in this simple test, the PBFT was able to perform slightly better than the Kafka results. On average the Kafka test showed to take almost 1 second more to do a single transaction. The single transaction is a good benchmark to understand but the systems should also be observed in the total system TPS. Similar tests should be done with system TSP to give a representation of the system rather than just a single.

4.5.4 Simple Blockchain Simulation

A way to test different blockchain protocols is within a modeling and simulation environment. The benefits to using modeling and simulation is that the entire environment can be controlled. This can be important when specific components need to be tested. In actual implementations within a physical system, a lot of unknown variance can occur in the results from the additional components involved in the decentralized system. These additional variances are important to note and be aware of but can distort the analysis of components such as the consensus method. A simulation environment was developed to be able to test consensus methods and is described in this chapter. Information on how the simulation was developed, how other simulations can be developed for blockchain systems and how to use the simulation described in this chapter are provided.

The implementation of the blockchain system is built within a custom C# application and simulation. By implementing the application within a simulated environment first, a full spectrum of tests can be performed on the various algorithms. This allows for testing extreme scenarios or provides a better control of the environment. With this capability a good understanding of the best case and worst case can be observed, and lessons can be learned from these exercises. This also offers a unique opportunity to verify that the algorithm is working as expected and provides a good way to document its effectiveness before a live or production version is implemented.

The simulation is developed as a test application where several validator nodes, a simulation controller, and associated blockchain infrastructure is spun up when started. The blockchain architecture is a simple data structure based on a linked list where each block on the list is an object consisting of a hash value, key value, and message. The hash and message are represented by a byte array and the key value is represented as an integer. Adding a new block utilizes the hash value of the prior block and the message of the new block to compute the new hash value. The hash algorithm used is SHA 256 from the C# system cryptography library. Different hash algorithms can be used instead if it is desired to test other hashing algorithms.

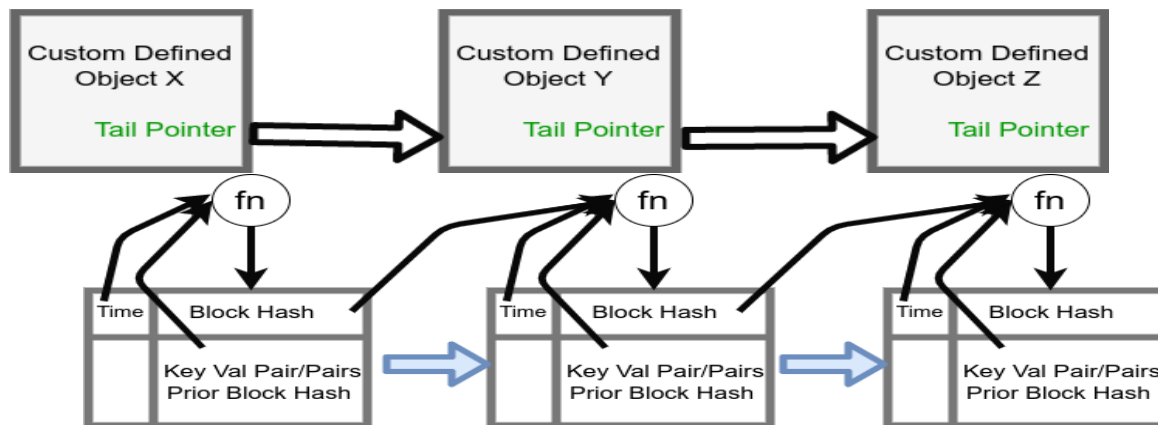


Figure 49. Illustration of linked list similarities to blockchain data structures

The original consensus algorithm that is tested with the simulated environment is a proof of stake algorithm. Each validator object in the simulation has a copy of the ledger that it hosts and uses to compare to the latest state of other validators and to append future data to when consensus has been derived. The validator object can choose a desired stake based on a random distribution or a set value for testing. The desired stake value is used as the mean value in a normal distribution that is used to select its current stake. Each validator's stake value is designed as a value between 0 and 100 which would be the amount of resources the validator is putting on hold for the proof of stake system. A delay is introduced in the simulation to the process where the validator selects its stake representing time it would take to occupy the resources. A delay can also be introduced in the process where validator nodes communicate with each other to represent some network delay.

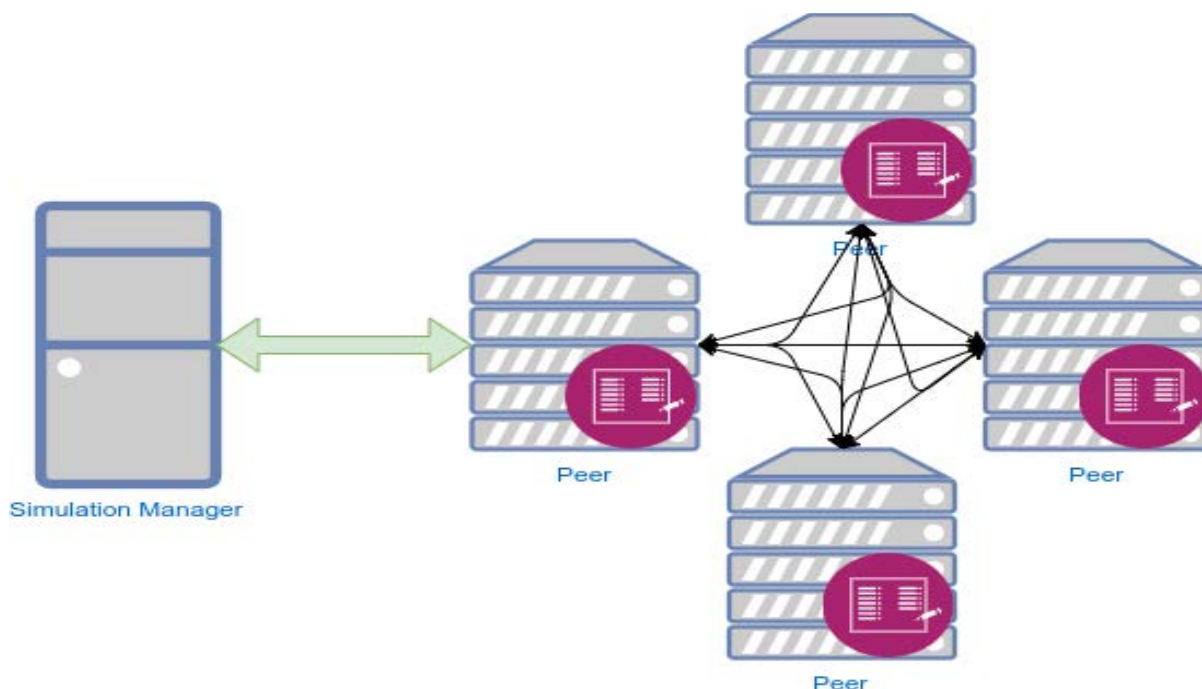


Figure 50. Illustration of a simulated blockchain system

The simulation controller runs the simulation directing the flow of communication and keeps track of statistical measures for analysis. The algorithm described above is implemented in the simulation controller and the validator object. The user tells the simulation controller how many validators to use and the number of transactions to process. With this information the controller initializes everything and creates the gen block on the ledger that all validators receive.

Once the validators have their ledger they are the sole controller of what can be written to their ledger. The controller then directs all validator objects to select their stake for the upcoming transaction. The controller compiles all validator nodes' claimed stake along with a weighted list based on each validators stake and shares that information with the all nodes on the network. Next the select leader process is started. At this point a timer is started based on a set loop iteration and each validator will have the ability to request a leader a set number of times before the leader select process is started over. That selected number of times is based on their chosen stake. Of course, if consensus is met in this process with all validator nodes to who the leader should be, the simulation controller exits the loop, proceeds allowing the chosen leader to validate, and broadcast to all validators to submit the transaction to their local ledgers. When the validators receive this broadcast message they will commence to writing the transaction to their blockchain.

The select leader process is initiated by each validator in the system in a round robin fashion where each validator broadcasts its recommended leader based on a random selection from the weighted list of validator ids by current stake allocation. Each validator receives the request then checks to ensure that the sender's ID exists in the system, the claimed leader stake is what the sender reported. If any of these are incorrect the validator will return false indicating it does not agree with the recommended leader. If the preliminary data is good, then the validator processing the recommendation will then select its recommended leader from the same weighted list of validator ids. If its selected leader id equals the recommended leader id from the broadcast message, then it will respond with a true value. If a validator can obtain a true value from more than half of the

network, it will be granted the leadership role. Once a validator is selected as the leader it is granted permission to broadcast the validated message to all validators indicating they can submit the current transaction block to their ledger

Performance metrics are set up in the described simulation and include but the quantification and measurement of performance is discussed in more detail in the blockchain performance section.

- o Average stake claimed per validator.
- o Average and total times each validator was the leader.
- o Total number of times a leader was selected as validator but did not have the highest stake amount.
- o Average, max/min iterations to find a leader.
- o Average, max/min time in milliseconds to make progress and extend the blockchain with a new block.

Simulation of a decentralized system has its challenges. The decentralized nature can be difficult because of the asynchronous properties, the potential resources or scalability, and the parallel processing of the distributed system. This can be accomplished in one of two ways. The simulation can be distributed in a parallel processing type of simulated environment. This is quite possible, where each object representing a peer runs as an individual thread or as a node on a multi-core and multi-processor hardware system. This creates a system closer to the real system where each peer can process simultaneously. If the resources are not available, then the modeler can always use round robin types of techniques. The round robin techniques

Optimization Goals: Current implementation considers that the stake allocation and verification occur at the beginning of every epoch, which can be a major bottleneck in the protocol. Since, stake allocation requires to adjust the actual resource utilization dynamically, it could result extra delay before any validator can verify the staked resource. In order to bypass this issue, we can enforce the validators to stake their resources for a fixed amount of time that is more than a typical epoch duration. A smart contract can be used for this purpose to prohibit release of stakes before the agreed staking time, which can be a regulatory condition for becoming a validator in the distributed computing system.

4.5.5 Blockchain Simulation Introduction

Described in the prior section was examples on how to implement a blockchain system using the open source toolset Hyperledger Fabric and through a simulated environment. The section focuses more on the idea of integrating a blockchain system that is real and not simulated (often this is referred to as live), with a simulation of an environment or system. The reasons for doing this would be to see the effects that a blockchain would have on an existing system. Tests like this can be built to show how the performance of the blockchain would make a difference to the simulation. It also provides a means to demonstrate the effectiveness of a blockchain system. Before a blockchain system is implemented it might be best to test the proposed system in a simulation of the environment or process that it will be working with. This is also a benefit to very large or costly systems as it allows exploration of the benefits before a costly implementation.

Methodology

Of the different paradigms of modeling and simulation, the paradigms that have entity representation tend to be better suited for use with blockchain technologies. In addition to entity-based models and simulation, the more dynamic types of simulation also make more sense, where there is a heavy influence of time to the system. These types of simulation systems tend to be agent-based models and simulations and discrete event models and simulation. Agent based models and simulations (ABMS) are representations or an abstraction of reality in which there are entities that make decisions for themselves based on environment variables around them. Discrete event simulation (DES) are representations or an abstraction of reality in which the actions or timing of the simulation is based on significant events. The difference between these two paradigms comes down to the representation or the advancement of time and the amount of intelligence required by an entity. It would be possible to apply the effects of a blockchain system as a measure of rate to other M&S paradigms such as system dynamics or other types of models without simulation. This could be achieved by producing quantitative values for performance and security metrics that can be used as parameters, weights, or rates in static models or flow based dynamic models.

Simulation Integration with Live Blockchain

The prior sections described blockchain examples that were simulated and real/functioning. The simulation integration can be set up to utilize either one. The major challenges or restrictions preventing the simulation and the blockchain to be integrated is the challenge of timing between the two and the ability to integrate a communication between the two. Many simulations are not built by default with communication protocols outside of the simulation but do offer advanced programming interfaces (API) and ways to program this capability.

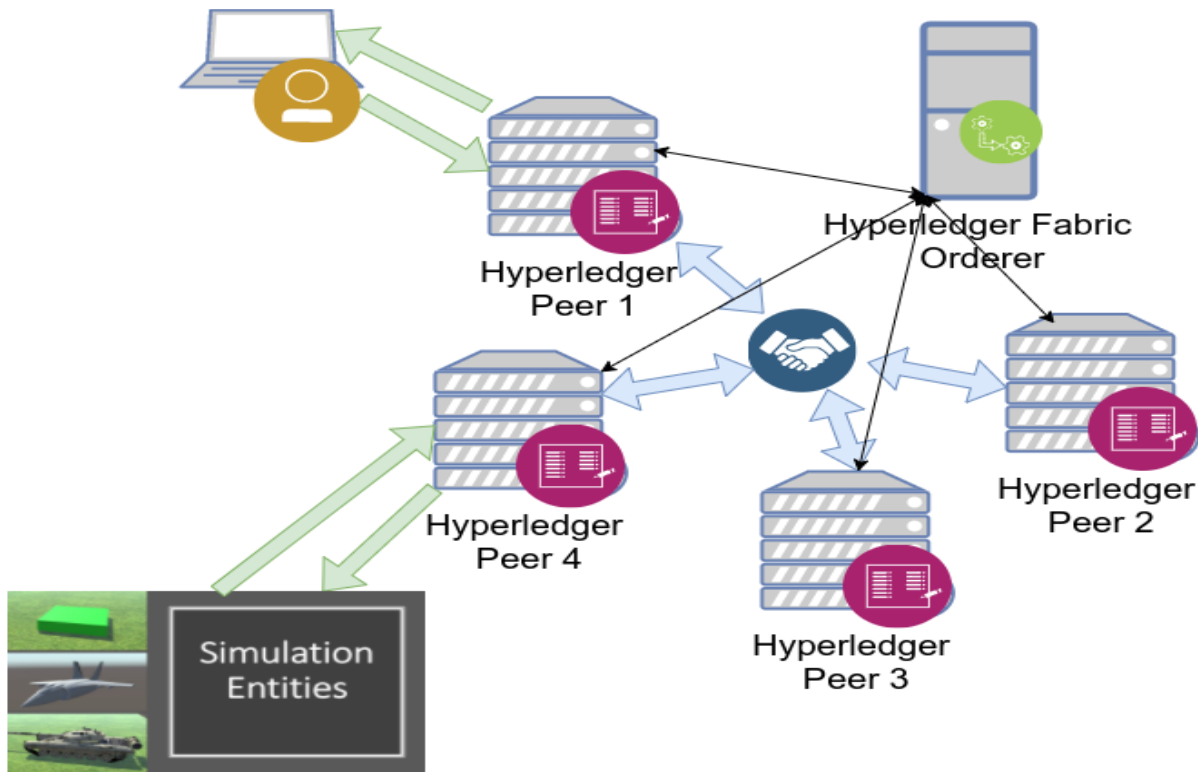


Figure 51. illustration of a hyperledger fabric system connected and interacting with a simulated environment

The timing issue might be the hardest challenge, especially if the simulation is to interact with a real time, actual blockchain system and not a simulated one. In this case the simulation might be performing faster than real time. In the DES paradigm this can be throttled to advance in time based on events from the blockchain system when interacting with the blockchain. In the case of a time stepped simulation that runs faster than real time, it can also be throttled back to only advance until after its interaction with the blockchain. The result will be an overall slower simulation but should provide a successful integration of the real blockchain system with the simulation.

The actual integration might be an easier problem to solve, but this heavily depends on the ability of the simulator to be able to communicate with other systems outside of the simulation. If the simulator provides the ability program in these points of communication, then the developers can simply program the messaging functions that are needed to interact with the blockchain using that API and any networking libraries that might be available to the language the API supports. The simulation can then be run, utilize inputs based on queries to the blockchain, and provide values and calls to the blockchain when state changes need to be recorded to the blockchain.

Simulation Integration with Simulated Blockchain

Simulation integration with simulated blockchain systems is a more natural combination as the timing and the communication can be tailored to be the same between both. If the modeler has access to the source code of the simulation, it is best to implement a blockchain representation within the simulation source code itself. Otherwise, the modeler will have to build a

communication link between the simulation and the blockchain simulation. If a communication link is needed, then the developer will need to make decisions on the implementation of the timing and communication in the same manner described earlier.

The biggest challenge again will come down to synchronization of the timing of the simulation and the blockchain simulation. For efficiency of the total system it will be best to attempt to match the blockchain simulation time step scope with the simulation time step. Modeling a blockchain system in real time clear as the representation of the communication flow and events are near real time. In the case of discrete event simulation, the blockchain does not necessarily need to model every aspect of the blockchain system. It should be modeled more as a queueing logistics problem. In order to achieve this, knowledge of transaction time metrics, bandwidth, and latency is needed. With this data the interactions happening within the blockchain simulation will become actions based on values from distributions formed from the observed data. The queueing of messages and delay caused by the blockchain can still be simulated, but the details will be abstracted out to values of distributions. To extend the blockchain simulation to an even higher scope for flow-based models (for example system dynamics), the level of abstraction from the blockchain system will need to be further observed in higher level measures. This could simply be utilizing measure of rates obtained from observations of the real blockchain system. The rates could reflect the average transaction time in various conditions, like bad network connectivity, bandwidth, and even structure of the blockchain like, number of peers, required computation of the smart contract, or consensus algorithm used.

Verification and Validation

It would be irresponsible to discuss the use of simulation involving blockchain systems without mentioning the necessity of verification and validation of the models and simulations. Modelers must be aware of the inherent assumptions that exist when building abstractions of the real system. These assumptions can exist in how the data was generated, in how the data was collected, and how the data was used in the actual modeling of the system. Once the blockchain simulation system is developed it is strongly recommended that some time is dedicated to verification of the system or observing the systems behavior to ensure that results to actions are behaving in the way that they were intended to.

It is also strongly recommended that the modeler dedicate data and time to validate the results of the blockchain simulation. An independent set of data should be recorded and stored for the intention of validation at the end of development. This independent set of data should not be used in the development of the model and simulation as this would be considered calibration, keep one set of data to be used to calibrate the model and separate set of data to be used as a single check at the end to give a validation score. After the calibration process score the model and simulation system with one of the many measures of error using the saved data set. This measure will let the modeler and any other party that uses the simulation to have an idea of how well the blockchain simulation represents the real system from the perspective of the validation data set.

If the model and simulation is exploratory and is simulating a fictitious system in which a real system does not exist yet. Then the modeler might not have a data set that can be generated to perform validation. In this all checks must be made to the best of modeler to verify the actions the simulation is what the developers, the users, and the modelers would intend. This can be done

using sensitivity analysis, where the modeler tests the extremes or ideal scenarios to see if the results are realistic to the intentions of the subject matter experts.

4.5.6 Example

As an example, a simulation was built in Unity 3d of a simple battlefield. A blockchain network using the Hyperledger Fabric platform is started up on a network server. The blockchain can utilize any supported consensus algorithm that is implemented with Hyperledger Fabric, the example that was tested uses Kafka consensus with 3 peer nodes, an orderer node, client node, 3 kafka nodes, and associated zookeeper nodes. This blockchain system can be simplified to use a solo consensus algorithm and fewer nodes. The simulation that was developed operates in near real time as a first person shoot style simulation, where the user can move around the world freely and look around. Unity 3d is a game engine that is free to use in an academic sense but requires a license if it is used for commercial purposes. Unity provides many free models and has a fully capable physics engine that allows developers to quickly prototype simulation or games.



Figure 52. image of the type of entities that are in the simulated world

The simulation is simple in nature, there are tank entities, plane entities, and cube entities. A top down view of the world is provided in Figure 53. The cube entities represent locations hosting parts of the blockchain network, validating nodes, in this case peer nodes. The tank and air entities

are associated with id's that are maintained on the live blockchain.

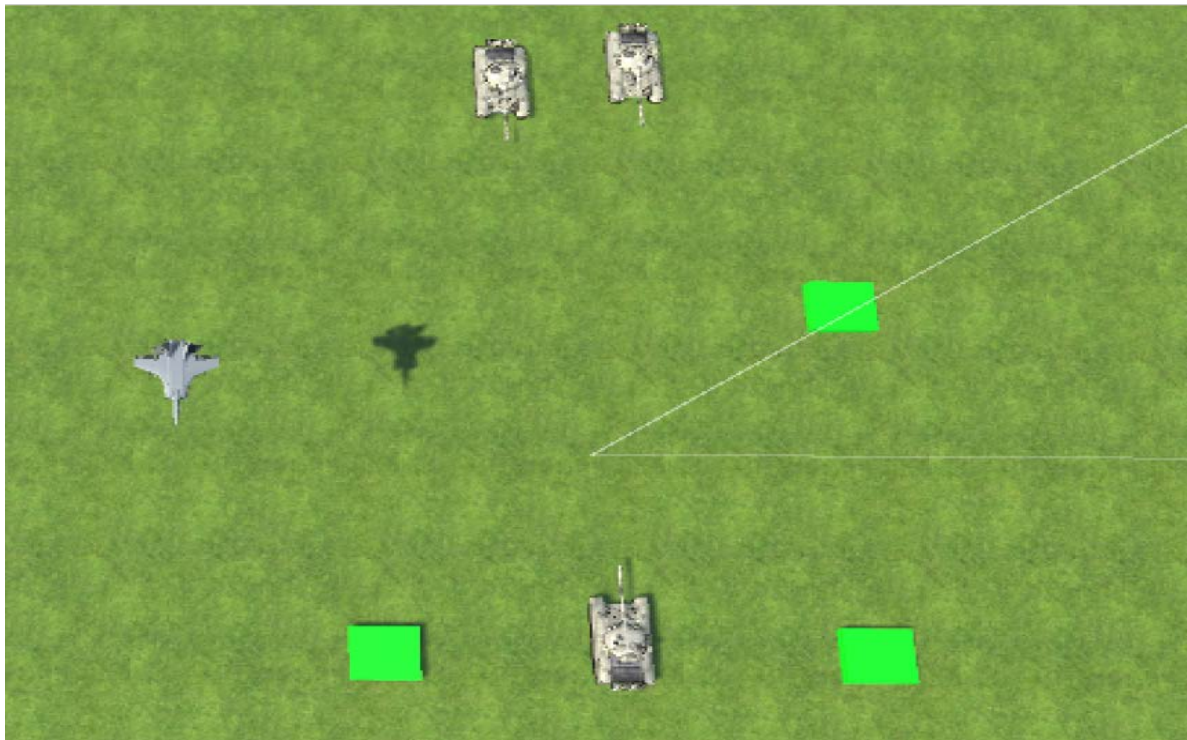


Figure 53. Top down view of the simulated environment

When the simulation starts the user is presented with a world that it can move around in. In the top right corner there is a minimap, the minimap is a static camera above the battlefield that includes vision on the three cubes representing peer nodes. The minimap acts as a sensor that utilizes the blockchain to determine if the entity that is in view is friendly or unknown. The user will also notice that on the minimap is an air entity that is circling above, if the camera is moved to look up the 3d jet will be in view and circling. Also, on the minimap is a tank nearby the users starting position, this tank acts as a static starting node for the user (think that the player stepped out of the tank and can move around). If the user maneuvers the character north, 2 other tanks will come into view, but are not within range of the minimap sensor. Pressing spacebar will move the tanks within range of the minimap as they advance south. The minimap reveals which entities are known friendly's that have been verified on the blockchain network by indicating a blue or green cube on the entities. If any entities are unknown and not verified on the blockchain, the minimap will show those entities with a red cube over them. Figure 54 shows the view from the user, including the minimap with the colored cubes based on the entity id's and the status of them on the blockchain.

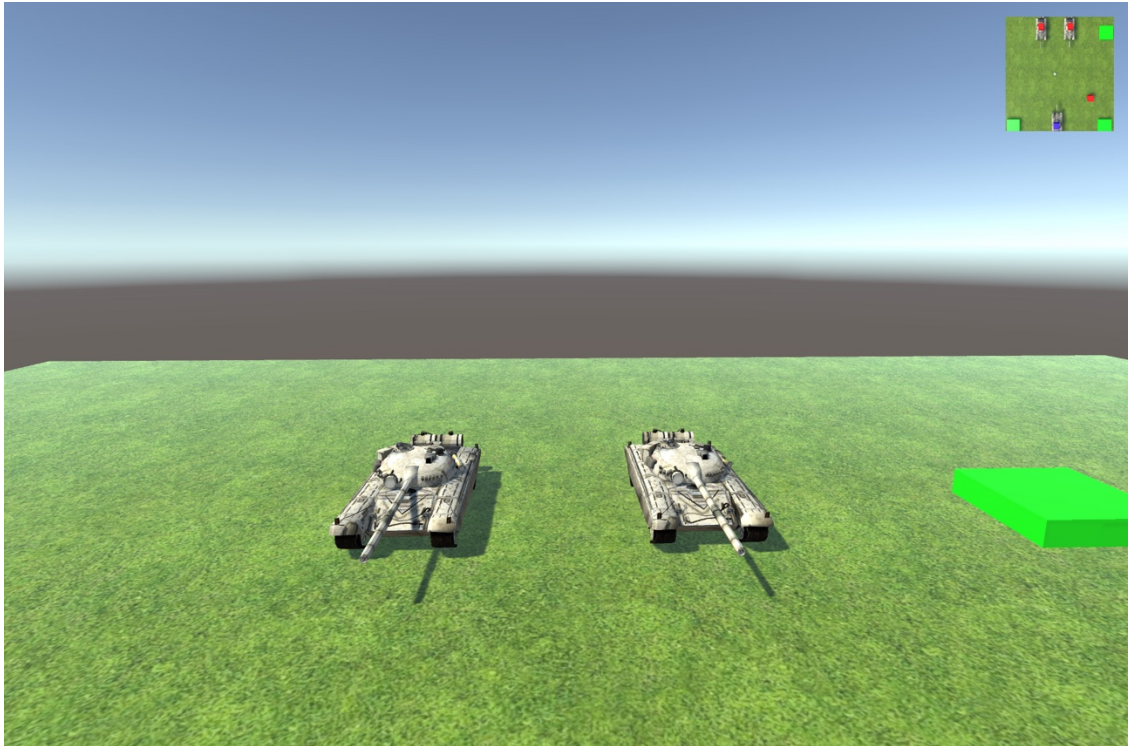


Figure 54. Game view of the simulated world with entities connected to the real blockchain system

In this example, the blockchain runs chaincode that writes simple datastructure to the blockchain. The datastructure is contains the following values:

1. EntityID: string
2. EntityVal: string
3. Message: string

The chaincode queries the blockchain for the EntityID. If the EntityID does not exist or if the Message value is empty or null, then the entity is considered unknown. This chaincode allows priveleged accounts to add entities to the blockchain or invoke new Messages and EntityVal strings. The blockchain is started and run using the JavaScript software development kit (SDK) for Hyperledger Fabric, to provide a cleaner interface between the clients and the peers hosting the chaincode. Once the blockchain network is started, the webservice is also started listening to requests on a specific port from the servers IP address.

4.6 Scalable Implementation in Hyperledger Fabric

We have followed microservices architecture to build Octopus, the scalable Hyperledger Fabric platform. We describe the scalable platform from a networked device identity management use case. The scalable platform contains following services. All these services are dockerized and deploying via kubernetes.

1. Hyperledger fabric blockchain
2. Admin service
3. Netflower service
4. Execer service
5. Storage service
6. Notification service
7. Kafka

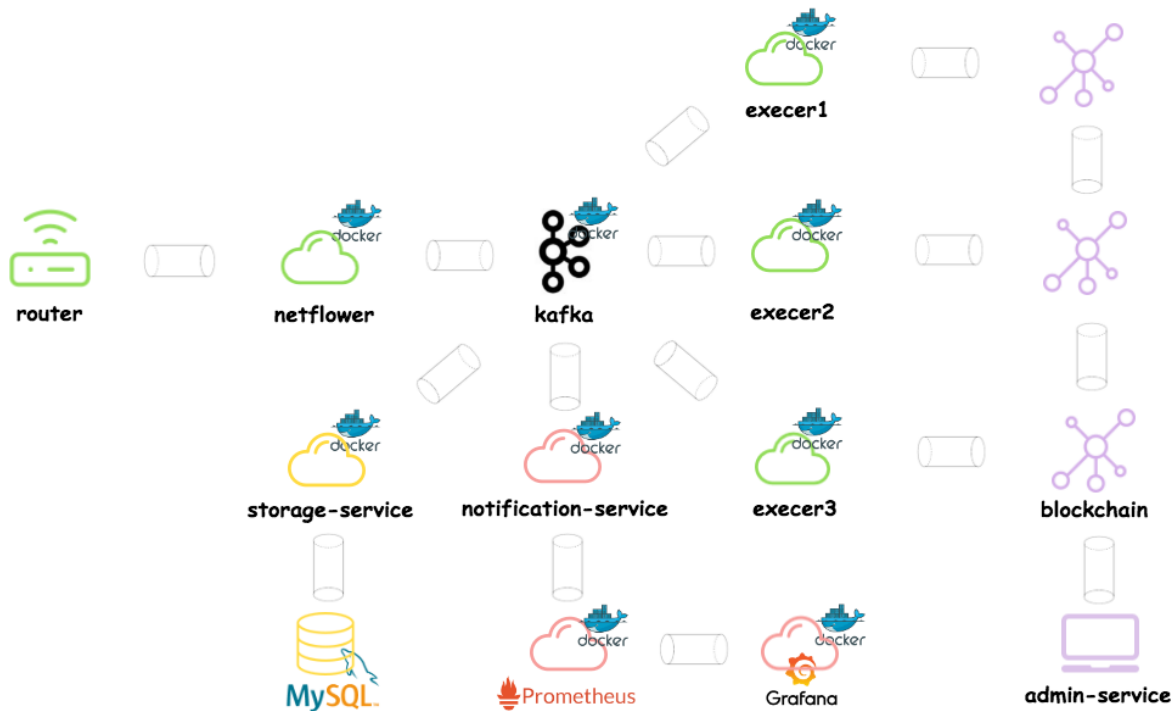


Figure 55 - Octopus platform architecture

4.6.1 Hyperledger fabric blockchain

Hyperledger fabric blockchain keeps all the medical device identity informations. We are recording the **IP address**, **MAC address** kind of informations from medical device. In octopus platform we have deployed blockchain network with below configurations.

1. One organization
2. Four peer nodes
3. One certificate authority node
4. Three orderer nodes
5. Three zookeeper nodes
6. Four kafka nodes

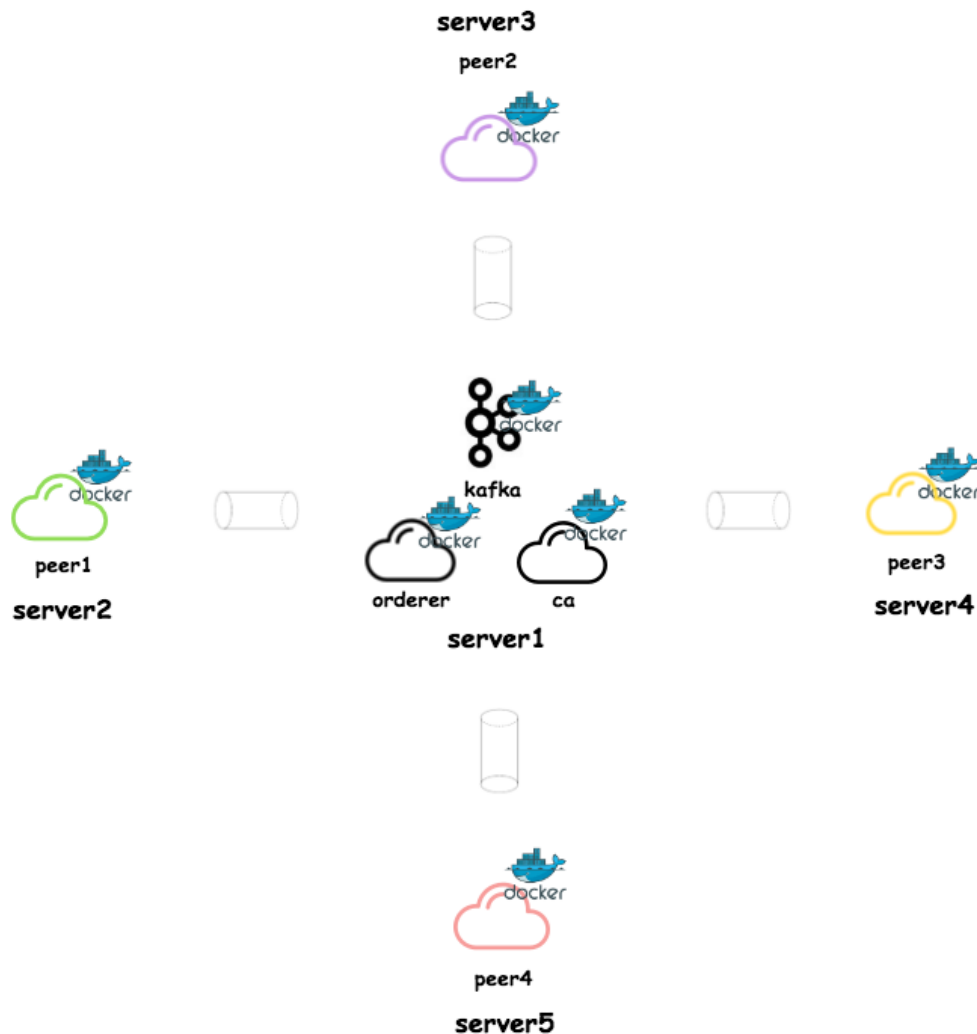


Figure 56 - Octopus blockchain architecture

Admin service- Admin service is responsible to register device identities in hyperledger fabric blockchain. When adding new device to the network system admin needs to register the device identity informations(IP address, MAC address etc) in the blockchain. We are providing web interface with admin service. When administrator registering a device, it connects to hyperledger blockchain peer via GRPC and make invoke transaction with the given device identity informations.

Netflower service - The network traffic which generates via devices will be goes through a netflow router. Then router redirect these traffic to octopus netflower service via UDP connection (netflower is a UDP server which implemented with Golang). When netflow packet received to

netflower it parse the packet and extract it's attributes(basically device informations, for an example **IP Addresses, Source Address, Source Port, Destination Address, Destination Port**). Finally netflower construct JSON message with these informations and forward it to execer service via kafka.

Execer service - Execer is the hyperledger fabric blockchain client service which implemented with **Scala** language. Messages which are coming from netflower service will be received to execer services via kafka. We are using kafka streams to handle the incoming messages. There are multiple execer services exists in the octopus network. Each execer service connects to separate hyperledger fabric blockchain peer. When message receives, it query the blockchain(with the device attributes) to check whether given device identity already registered in the blockchain. If no device identity found in the blockchain, its an unauthorized device. Execer sends this unauthorized device/event informations to storage service and notification service via kafka to take further actions.

Storage service - The unauthorized device informations(which sends from execer service) receives to storage service via kafka. These informations will be stored in mysql/mongodb data storage for further analysis. Administrative staff can view these informations for their admin dashboard.

Notification service - The unauthorized device informations(which sends from execer service) receives to notification service via kafka. Based on these information the security alerts will be generated. These alerts will be notified via prometheus(email alerts) and grafana dashboard.

Kafka - Kafka is the message broker of octopus platform. We have deployed three kafka broker nodes in octopus platform. Each microservice communicate with other services via kafka. We are using reactive programming approach in the octopus microservice platform. All the services published events/messages with kafka. There event will subscribed by relevant services and take corresponding actions.

4.6.2 Performance evaluation

We have conducted performance evaluation of octopus blockchain platform. The evaluation has been performed based on the hyperledger fabric transaction types. There are two types of transaction in hyperledger fabric, **invoke** and **query**. Invoke transactions will be recorded in the leader(it alter the ledger). Query transactions only read the current state from the blockchain (it not alter the ledger). Based on these two transaction types we have obtained the following statistics. To obtained the statistics we have deployed octopus network in five AWS 2xlarge instances(16 GB RAM and 8 CPUS).

1. Invoke transaction throughput
2. Invoke transaction scalability
3. Query transaction throughput
4. Query transaction scalability

Invoke transaction throughput- For this evaluation we have recorded no of invoke transactions(per second) that can be executed in each octopus blockchain peer. We have flooded transactions for each blockchain peer and recorded no of executed transactions. As shown in **Figure 57** each peer has consistent transaction throughput.

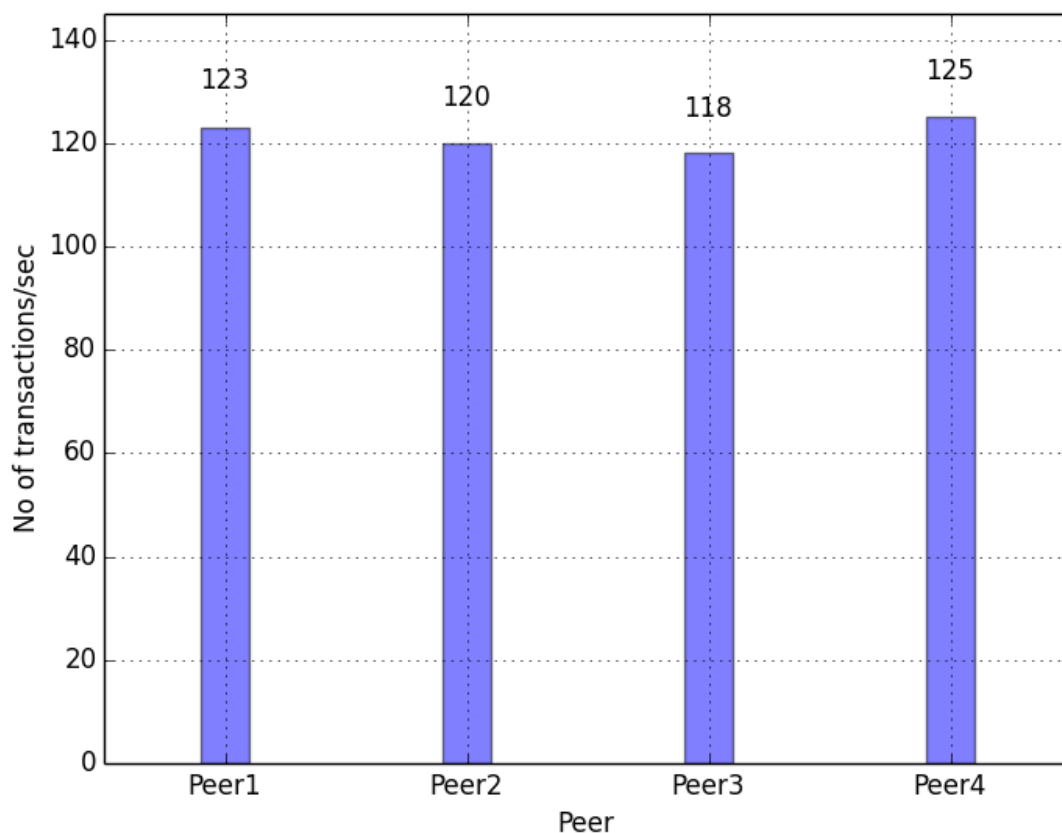


Figure 57 - Octopus invoke transaction throughput

Invoke transaction scalability - For this evaluation we have recorded no of invoke transactions(per second) over the no of blockchain peers in the network. We have flooded concurrent transaction in each blockchain peer and recorded the no of executed transactions. As shown in **Figure 58** when we increasing the no of blockchain peers the transaction throughput linearly increased.

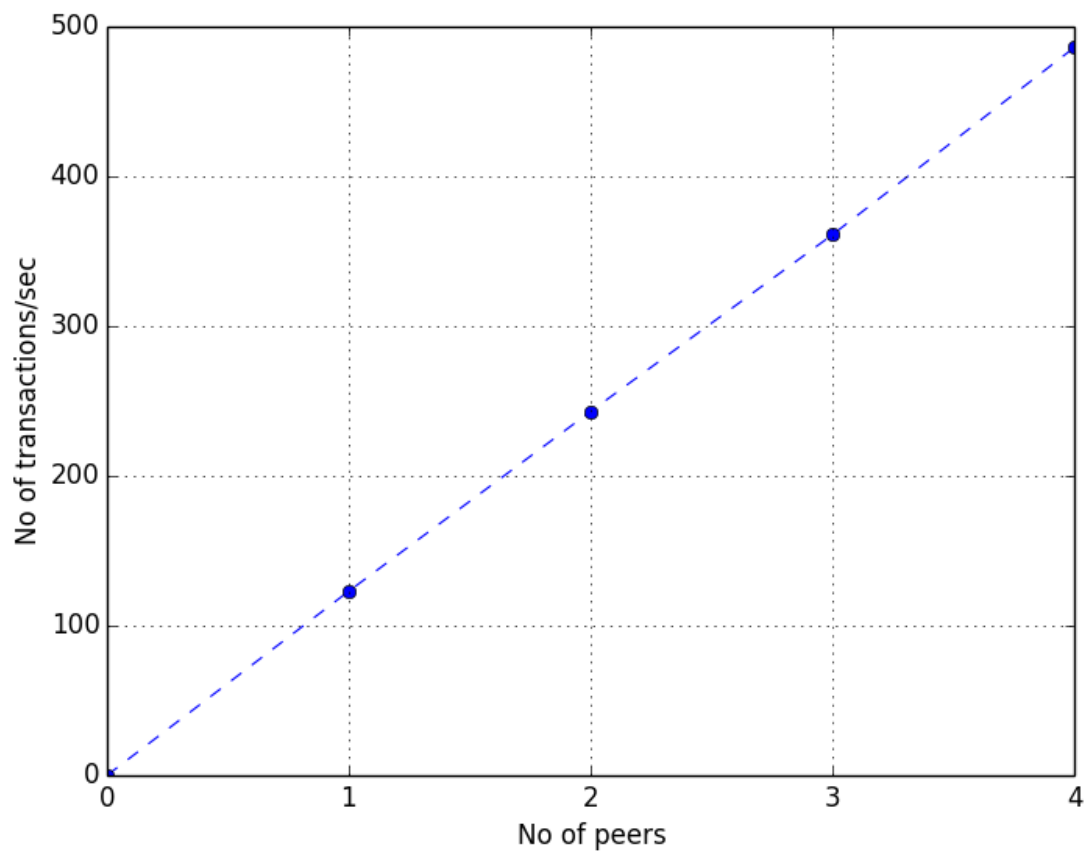


Figure 58 - Octopus invoke transaction scalability

Query transaction throughput - To measure the query transaction throughput we have recorded no of query transactions(per second) that can be executed in each octopus blockchain peer. We have flooded transactions for each blockchain peer and recorded no of executed transactions. As shown in **Figure 59** each peer has consistent transaction throughput. When comparing this result with the no of invoke transactions, we can observe that peers can execute high no of query transactions than invoke transaction. The main reason is, query transactions not altering the ledger, it just query the status form the ledger.

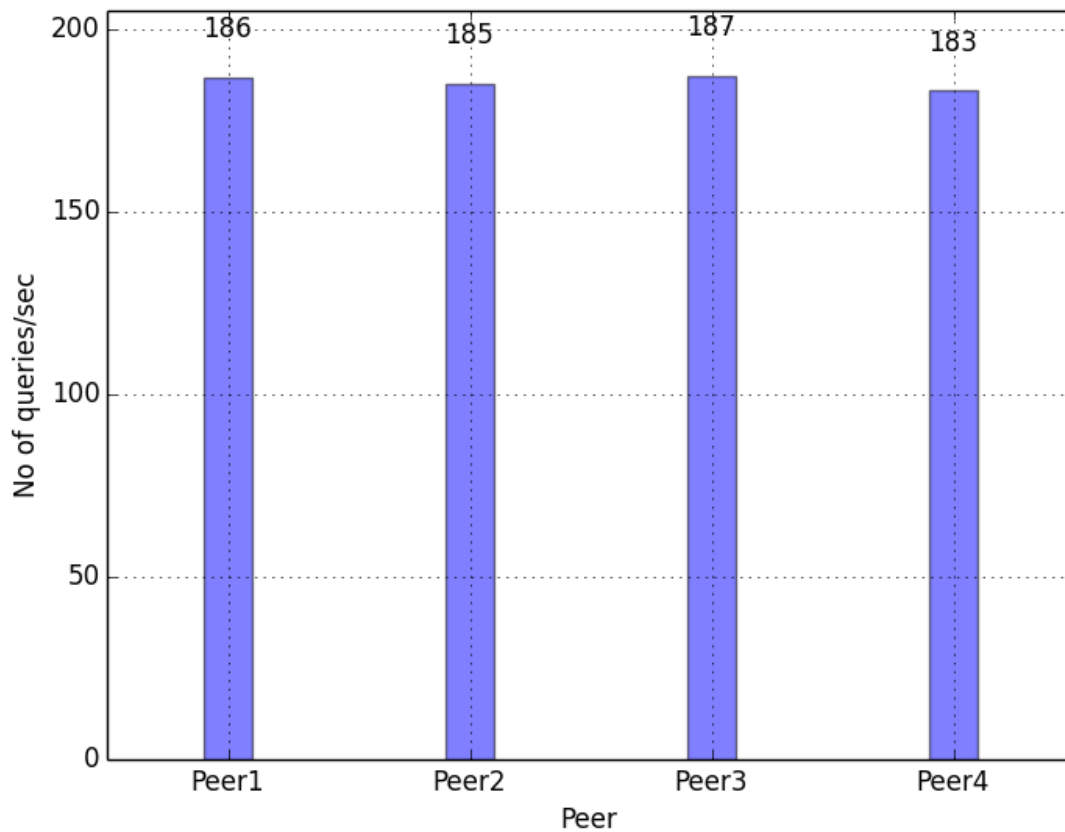


Figure 59 - Octopus query transaction throughput

Query transaction scalability - In this evaluation we have recorded no of executed query transactions(per second) against the no of peers in the blockchain network. We have flooded concurrent transaction in each blockchain peer and recorded the no of executed transactions. As shown in **Figure 60** when we increasing the no of blockchain peers the transaction throughput linearly increased.

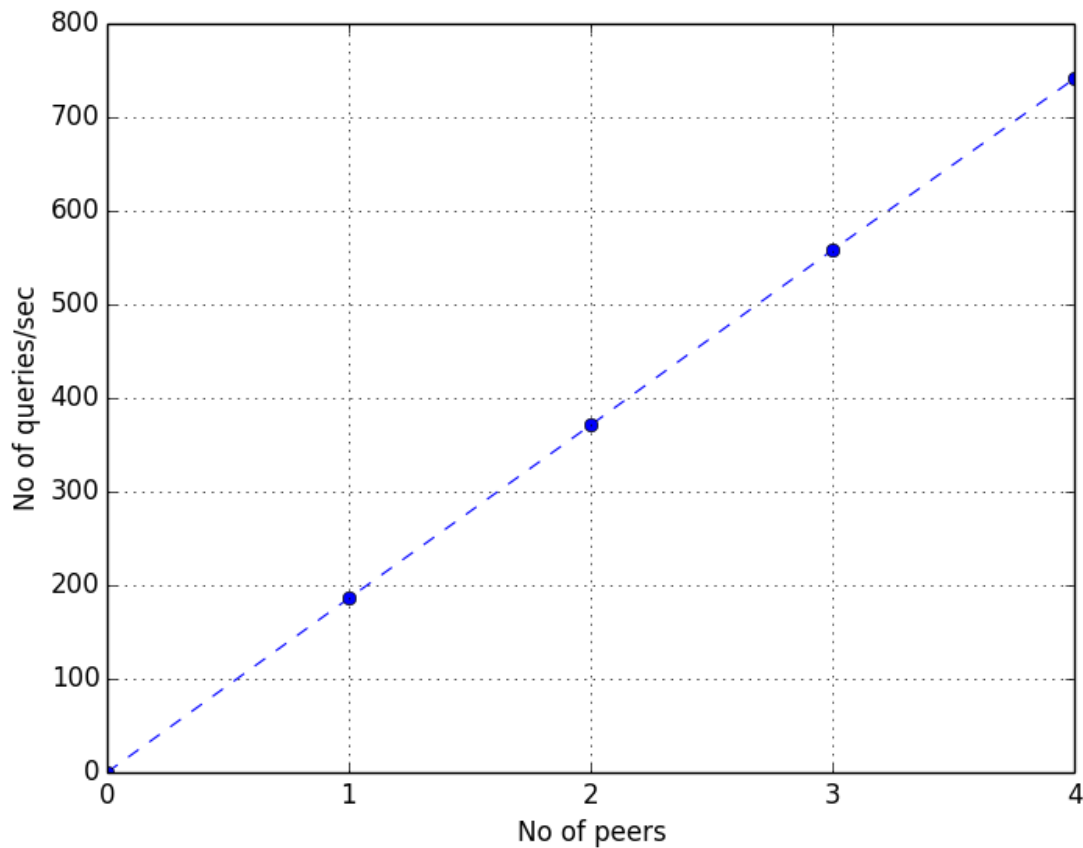


Figure 60 - Octopus query transaction scalability

5 CONCLUSIONS

In this report, we present the design and implementation of ProvChain, a Blockchain-based data provenance system for cloud auditing, with preserved user privacy and increased availability. Using Blockchain technology, we make the record with unalterable timestamping and generate Blockchain receipt for each of the data records for validation. With each data provenance record provided with a Blockchain receipt, our system performs functions with stability and scalability. Based on the current work, we can extend the system to various use cases where globally verified proof is needed. Our evaluation shows that provenance-enabled ownCloud brings a low overhead, according to our evaluation for the cloud storage performance. Instead of file as the data unit, in the future, we can also use other granularity such as data chunk in cloud storage, at a finer-grain level.

As for the rewards of Blockchain miners, cloud users may have to pay for a fee to enable data provenance services by the cloud service provider. The service provider can then pay for the Blockchain network. In this way, miners can be paid for continuous mining on blocks and validation of block authenticity. The fee can be determined depending on the different levels of data usage of each user.

Currently, we collect provenance data inside one cloud service provider and one cloud application. For future work, we plan to develop ProvChain for federated cloud service providers. Cloud storage applications on federated cloud providers will require the need to address interoperability, cross-provider data sharing, and management. We will collect data provenance across different cloud providers and different cloud storage applications to provide better provenance services and enhance data security. For provenance validation, we currently use the Tierion API to validate the Blockchain receipt, which is effective and flexible with regard to API implementation and usage.

For future work, we will implement the validation on top of an open source architecture that will not only improve overall performance but also security and reliability. The actual cost for each Blockchain receipt should be analyzed and calculated for business scenarios. We can also define the data attributes to be recorded according to client/user needs to prevent sensitive information leakage and thus preserving privacy. We will use the collected provenance data to check for access control violations [112] by way of machine learning strategies. The automatically generated access control rules will better serve the purpose of malicious behavior detection and intrusion prevention, which will in return provide better protection for the cloud storage application. Meanwhile, the same architecture can be applied to IoT scenarios where large numbers of mobile devices are responsible for data collection and processing.

Since Blockchain technology is one of the next-generation technologies that uses a cryptographically enforced distributed ledger system, its security evaluation is necessary to assure its usefulness in the cloud computing domain. Therefore, we investigate the applicability and security implications of Blockchain in realizing Blockchain cloud. A number of security vulnerabilities are discussed that may have harmful impact while integrating Blockchain on cloud system. We particularly model the issue of block withholding attack, which is prevalent in PoW-based mining pools, to understand the attacker's strategy toward taking over the pool members' rewards. Simulation results demonstrate that attacker's access to extra computational power could disrupt the honest mining operation in Blockchain cloud. The attacker's strategy is analyzed based on two different pools in which reward schemes are different. We found that pay per last N shares (PPLNS) scheme could be useful in keeping the attacker's impact to a lesser amount than the proportional reward scheme. In future studies, we aim to extend our analysis on PoS-based Blockchain cloud and test the instances on real-time private Blockchain platform.

We explore the attack surface of Blockchain technology. We attribute attacks to the cryptographic constructs of the block chain, the underlying communication architecture, and the context in which they are applied. In doing so, we highlight major threats and ongoing defense research activities. We believe that various attacks against Blockchain can be still launched, notwithstanding the current and existing defenses, and that some of those attacks can be used to facilitate several others. In the future, we will explore exact parameters and settings to empirically understand the cost of launching those attacks, under various defense capabilities.

We identify a DDoS attack on Blockchain mempools that trap users into paying higher mining fee. Attacks on Blockchain mempools have not been addressed previously, and we propose two countermeasures to the problem: fee-based and age-based designs. From our analysis and simulation results, we conclude that when the attack is not severe, the fee-based design is more

effective in mempool size optimization. However, it does so by affecting both the attacker and the legitimate users. In contrast, when the attack is severe, the age-based design is more useful in helping legitimate users while discarding maximum spam transactions. Although the size optimization achieved by the age-based design is less compared to the age-based design, its accuracy of spam detection is higher. Extending our analysis and design towards a hybrid model that leverages the benefits of both designs and achieves a maximum spam detection and size optimization is an open question.

We analyzed the potential vulnerabilities in blockchain architecture and utilize Intel SGX to help prevent these attacks. Meanwhile, a multi-channel scheme is adopted to effectively address the potential privacy issues, removing the need of trust between participating nodes during transactions. The evaluation shows that our architecture serves as a feasible solution with acceptable overhead. Future work will utilize the design, implement a privacy preserving blockchain service in a cloud data sharing application, and evaluate the overall application performance.

Blockchain systems are very early in development and work continues to prove the theories that exist. Testing the theories in simulation or an actual blockchain system is critical in understanding the true effects of the decisions made in the system. Discussing and thinking about how the decentralized blockchain systems will work best is important but until proper tests show that these ideas work and provide a means to compare the technology will not be able to advance from paper to production.

Simulation of blockchain systems can be a very valuable tool when used appropriately. If decisions are to be made with simulations for real systems, developers should ensure proper verification and validation of the system that is being modeled and simulated. Even with this point, simulation offers a unique ability to test extreme variations of the system that can be expensive or impossible to implement.

The future is bright for the open sourced blockchain systems as they will be better vetted by a large community and will offer opportunity for fast deployment. Additionally, they provide a means for developers to tailor the tools to work best for specific scenarios. Hyperledger fabric provides a platform that is heavily supported and will be a great way for new developers to implement systems to build upon. Years beyond this book, the success of the platform will greatly be based on the community and where the developers choose to support and build on. When choosing a platform look to see the quantity and the quality of developers that are supporting the tool to get an idea of the future of that platform.

Moving forward, the researchers will see a great deal of work in improving the scalability and performance of the decentralized blockchain systems. Further work will be done on better understanding the best use cases for these types of systems. Early developers and researchers might quickly jump to try blockchain systems to solve many problems, this does not mean that blockchain will be great at solving all of these problems, but over time the success stories will rise to the top. Hopefully stories of applications of blockchain systems to solve problems will be written for the good use cases as well as the bad use cases. There needs to be a better understanding of when blockchain works and when it does not work. Finally, there needs to be more documented

evidence of the performance metrics and capabilities of these systems that are peer reviewed and vetted by the academic community in addition to the open source community.

6 RECOMMENDATIONS

Although there are multiple Blockchain platforms on the market and considerable research and development on some Blockchain features is ongoing, further research is required in the following areas:

- Public, Private, or Hybrid Blockchain Architecture
- Incentives
- Anonymity and Data Privacy

Public, Private, or Hybrid Blockchain Architecture - Public Blockchain architecture provides a truly decentralized mechanism for performing transactions. However, commercial enterprises are skeptical of integrating public Blockchain into their enterprise solutions due to data privacy, performance, and response concerns. Further research and development within public Blockchain platform is necessary to address these concerns.

In the meantime, commercial stakeholders are increasingly gravitating towards a private/permissioned/ consortium Blockchain architecture. This architecture varies according to the governance style, which ranges from a single member to a consortium overseeing the Blockchain platform. These Blockchain platforms have centralized components in the architecture, protocol, or the process in which the transactions are validated.

Incentives - In Bitcoin, the incentive to participate in Blockchain platform is monetary; in use cases such as provenance and identity management, the incentives are non-monetary. The incentive structure needs to be incorporated within the protocol to ensure maximum participation. The incentive design mechanisms may be derived from the use cases to ensure that the Blockchain protocols are working toward maximizing the benefits for the use case. Though the Blockchain platform's trust property may eliminate tampering or cheating, it is important for the design to include incentive mechanisms with properties that discourage cheating or unfairness. There is a need for theoretic models to capture the dynamics associated with incentivizing participants.

Anonymity and Data Privacy - The public availability of Blockchain transactions makes it possible to use data analytics techniques to evaluate the vast quantities of data within them. This analysis could reveal valuable information, including the identity of participants and specific transactions they have performed. Several schemes address privacy issues in public Blockchain platforms, including stealth addresses, Homomorphic encryption, and zero knowledge proof. A combination of techniques is necessary to achieve the desired level of privacy.

7 REFERENCES

- [1] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM Sigmod Record*, vol. 34, no. 3, pp. 31–36, 2005.
- [2] B. Lee, A. Awad, and M. Awad, "Towards secure provenance in the cloud: A survey," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 577–582.
- [3] D. Tosh, S. Sengupta, C.A. Kamhoua, and K.A. Kwiat, "Establishing evolutionary game models for cyber security information exchange (cybex)," *Journal of Computer and System Sciences*, 2016.
- [4] C. Kamhoua, A. Martin, D.K. Tosh, K. Kwiat, C. Heitzenrater, and S. Sengupta, "Cyber-threats information sharing in cloud computing: A game theoretic approach," in *IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2015, pp. 382–389.
- [5] D.K. Tosh, S. Sengupta, S. Mukhopadhyay, C. Kamhoua, and K. Kwiat, "Game theoretic modeling to enforce security information sharing among firms," in *IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2015, pp. 7–12.
- [6] D.K. Tosh, M. Molloy, S. Sengupta, C.A. Kamhoua, and K.A. Kwiat, "Cyber-investment and cyber-information exchange decision modeling," in *IEEE 7th International Symposium on Cyberspace Safety and Security*, 2015, pp. 1219–1224.
- [7] R.C. Merkle, "Protocols for public key cryptosystems," in *Security and Privacy*, 1980 IEEE Symposium on, April 1980, pp. 122–122.
- [8] Kelly, J. and Williams, A., "Forty big banks test blockchain based bond trading system," [online] (2016).
- [9] Kar, I., "Estonian citizens will soon have the world's most hack-proof health-care records," [online] (2016).
- [10] Lacey, S., "The energy blockchain: How bitcoin could be a catalyst for the distributed grid," [online] (2016).
- [11] Oparah, D., "3 ways that the blockchain will change the real estate market," (2016).
- [12] Mizrahi, A., "A blockchain-based property ownership recording system," [online] (2015).
- [13] Before the Blockchain: 4 questions to answer, <https://gcn.com/articles/2018/06/25/blockchain-questions.aspx>
- [14] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *CoRR*, vol. abs/1802.06993, 2018. [Online]. Available: <http://arxiv.org/abs/1802.06993>
- [15] I.-C. Lin and T.-C. Liao, "A survey of blockchain security issues and challenges." *IJ Network Security*, vol. 19, no. 5, pp. 653–659, 2017.

- [16] 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 - Volume 10204, 2017, pp. 164–186. [Online]. Available: https://doi.org/10.1007/978-3-662-54455-6_8
- [17] D. Siegel, “Understanding The DAO Attack,” <https://www.coindesk.com/understanding-dao-hack-journalists/>
- [18] Do I need a Blockchain, : <http://www.multichain.com/blog/2015/11/avoiding-pointless-blockchain-project/>
- [19] “ownCloud,” Retrieved from <https://owncloud.org/>.
- [20] “Tierion api,” Retrieved from <https://tierion.com/app/api>.
- [21] “Chainpoint: A scalable protocol for anchoring data in the Blockchain and generating Blockchain receipts,” Retrieved from <http://www.chainpoint.org/>.
- [22] “Bitcoin block explorer,” Retrieved from <https://btc.com/>.
- [23] M. Rosenfeld, Analysis of hashrate-based double spending, arXiv preprint arXiv:1402.2009.
- [24] I. Eyal, E. G. Sirer, Majority is not enough: Bitcoin mining is vulnerable, In: International Conference on Financial Cryptography and Data Security, Springer (2014) pp. 436–454.
- [25] A. Sapirshtein, Y. Sompolinsky, A. Zohar, Optimal selfish mining strategies in bitcoin, arXiv preprint arXiv:1507.06183.
- [26] K. Nayak, S. Kumar, A. Miller, E. Shi, Stubborn mining: Generalizing selfish mining and combining with an eclipse attack, In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE (2016), pp. 305–320.
- [27] E. Heilman, A. Kendler, A. Zohar, S. Goldberg, Eclipse attacks on bitcoins peer-to-peer network, in: 24th USENIX Security Symposium (USENIX Security 15), (2015), pp. 129–144.
- [28] L. Bahack, Theoretical bitcoin attacks with less than half of the computational power (draft), arXiv preprint arXiv:1312.7013.
- [29] M. Rosenfeld, Analysis of bitcoin pooled mining reward systems, arXiv preprint. Available at <http://arxiv.org/abs/1112.4980>.
- [30] N. T. Courtois, L. Bahack, On subversive miner strategies and block withholding attack in bitcoin digital currency. arXiv preprint arXiv:1402.1718.
- [31] P. Koshy, D. Koshy, P. McDaniel, An analysis of anonymity in bitcoin using P2P network traffic, In: International Conference on Financial Cryptography and Data Security, Springer, (2014), pp. 469–485.
- [32] A. Biryukov, D. Khovratovich, I. Pustogarov, Deanonymisation of clients in bitcoin P2P network, In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, (2014), pp. 15–29.
- [33] Kwon, Yujin, Dohyun Kim, Yunmok Son, Eugene Vasserman, and Yongdae Kim.

2017. "Be Selfish and Avoid Dilemmas: Fork After Withholding (Faw) Attacks on Bitcoin." In Proceeding of Acm Ccs.
- [34] Siegel, David. 2016. "Understanding the DAO Attack." <https://www.coindesk.com/understanding-dao-hack-journalists/>.
 - [35] "Bitcoin Block Explorer - Blockchain." n.d. Bitcoin Block Explorer - Blockchain. <http://bit.ly/1srPhPs>.
 - [36] Vasek, Marie, Micah Thornton, and Tyler Moore. 2014. "Empirical Analysis of Denial-of-Service Attacks in the Bitcoin Ecosystem." In Financial Cryptography and Data Security, 57–71. Springer.
 - [37] Ittay Eyal, Emin Gun Sirer 2014b. "Majority Is Not Enough: Bitcoin Mining Is Vulnerable." In Financial Cryptography and Data Security, 436–54. Springer.
 - [38] Bitcoin Community. 2017. "51% Attack." <https://learncryptography.com/cryptocurrency/51-attack>.
 - [39] Bastiaan, Martijn. 2015. "Preventing the 51%-Attack: A Stochastic Analysis of Two Phase Proof of Work in Bitcoin." In. <https://goo.gl/nJsMzV>.
 - [40] Bahack, Lear. 2013. "Theoretical Bitcoin Attacks with Less Than Half of the Computational Power (Draft)." arXiv Preprint arXiv:1312.7013.
 - [41] "Bitcoin Developer Guide." 2017 Developer Guide - Bitcoin. <https://bitcoin.org/en/developer-guide>
 - [42] Kang, Ah Reum, Jeffrey Spaulding, and Aziz Mohaisen. 2016. "Domain Name System Security and Privacy: Old Problems and New Challenges." CoRR. <http://arxiv.org/abs/1606.07080>.
 - [43] An Wang, Aziz Mohaisen, and Songqing Chen. 2017. "An Adversary-Centric Behavior Modeling of DDoS Attacks." In 37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, Ga, USA, June 5-8, 2017, 1126–36.
 - [44] Muhammad Saad, My T. Thai, and Aziz Mohaisen. "POSTER: Deterring DDoS Attacks on Blockchain-based Cryptocurrencies through Mempool Optimization". Proceedings of 13th ACM ASIA Conference on Information, Computer and Communications Security, ASIACCS 2018.
 - [45] Muncaster, Phil. 2017. "World's Largest Bitcoin Exchange Bitfinex Crippled by Ddos." <http://bit.ly/2kqo6HU>.
 - [46] Cimpanu, Catalin. 2017. "Bitcoin Trader Hit by "Severe Ddos Attack" as Bitcoin Price Nears All-Time High." <http://bit.ly/2lA5iT6>.
 - [47] Vitalik Buterin 2016. "Ethereum Responds to Recent DDoS Attack." <http://bit.ly/2gcrn9d>.

- [48] Jeffrey Wilcke. 2016b. “The Ethereum Network Is Currently Undergoing a DoS Attack.” <http://bit.ly/2cwlB0D>.
- [49] Hülsing, Andreas, Denis Butin, Stefan Gazdag, and Aziz Mohaisen. 2015. “XMSS: Extended Hash-Based Signatures.” <https://www.ietf.org/id/draft-irtf-cfrg-xmss-hash-based-signatures-10.txt>.
- [50] Sapirshtein, Ayelet, Yonatan Sompolinsky, and Aviv Zohar. 2016. “Optimal Selfish Mining Strategies in Bitcoin.” In *Financial Cryptography and Data Security*, 515–32. Springer.
- [51] Ittay Eyal, Emin Gun Sirer 2014b. “Majority Is Not Enough: Bitcoin Mining Is Vulnerable.” In *Financial Cryptography and Data Security*, 436–54. Springer.
- [52] Heilman, Ethan. 2014. “One Weird Trick to Stop Selfish Miners: Fresh Bitcoins, a Solution for the Honest Miner.” In *Financial Cryptography and Data Security*, 161–62. Springer.
- [53] Courtois, Nicolas T, and Lear Bahack. 2014. “On Subversive Miner Strategies and Block Withholding Attack in Bitcoin Digital Currency.” *arXiv Preprint arXiv:1402.1718*.
- [54] Mehrdad Nojoumian, Arash Golchubian, Laurent Njilla, Kevin Kwiat, Charles A. Kamhoua, “Incentivizing Blockchain Miners to Avoid Dishonest Mining Strategies Using a Reputation-Based Paradigm” in the proceedings of the 2018 Computing Conference, London, United Kingdom, July 2018.
- [55] Solat, Siamak, and Maria Potop-Butucaru. 2016. “ZeroBlock: Preventing Selfish Mining in Bitcoin.” *arXiv Preprint arXiv:1605.02435*.
- [56] Johnson, Benjamin, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. 2014. “Game-Theoretic Analysis of Ddos Attacks Against Bitcoin Mining Pools.” In *Financial Cryptography and Data Security*, 72–86. Springer.
- [57] Silva, Peter. 2009. “DNSSEC: The Antidote to Dns Cache Poisoning and Other Dns Attacks.” *A F5 Networks, Inc. Technical Brief*.
- [58] Peng, Tao, Christopher Leckie, and Kotagiri Ramamohanarao. 2007. “Survey of Network-Based Defense Mechanisms Countering the Dos and Ddos Problems.” *ACM Computing Surveys (CSUR)* 39 (1). ACM:3.
- [59] Etheridge, James, and Richard Anton. 2002. “System and Method for Detecting and Countering a Network Attack.” Google Patents.
- [60] Fleder, Michael, Michael S Kester, and Sudeep Pillai. 2015. “Bitcoin Transaction Graph Analysis.” *arXiv Preprint arXiv:1502.01657*.

- [61] Muhammad Saad, Aziz Mohaisen, Charles Kamhoua, Kevin Kwait, and Laurent Njilla. 2018. "Countering Double Spending in Next-Generation Blockchains." In. IEEE International Conference on Communications.
- [62] Avalon Releases New ASIC Miner & Begins Shipping Worldwide through BlockC Partnership, <https://bitcoinmagazine.com/articles/avalon-releases-new-asic-miner-begins-shipping-worldwide-through-blockc-partnership-1447268188/>
- [63] Hashflare: Bitcoin cloud mining, Available at <https://hashflare.io>.
- [64] Genesis mining, <https://www.genesis-mining.com>.
- [65] Hashnest cloud mining, <https://www.hashnest.com>.
- [66] Eobot: Bitcoin cloud mining and the best way to mine cryptocurrency, Available at <https://www.eobot.com>.
- [67] M. Rosenfeld, Analysis of bitcoin pooled mining reward systems, arXiv preprint. Available at <http://arxiv.org/abs/1112.4980>.
- [68] Baqer, Khaled, Danny Yuxing Huang, Damon McCoy, and Nicholas Weaver. 2016. "Stressing Out: Bitcoin 'Stress Testing'." In International Conference on Financial Cryptography and Data Security, 3–18. Springer.
- [69] Memoria, Francisco. 2017a. "700 Million Stuck in 115,000 Unconfirmed Transactions." CryptoCoinsNews. <https://goo.gl/mYX14V>.
- [70] Community, Blockchain. 2018a. "Confirmation Score, Confirmed Transaction." Confirmation Score, Confirmed Transaction. <https://goo.gl/ZY6Prg>.
- [71] Community, Blockchain. 2018b. "Minimum Relay Fee." <https://goo.gl/rAvtMu>.
- [72] Nakamoto, Satoshi. 2008. "Bitcoin: A Peer-to-Peer Electronic Cash System."
- [73] Community, Blockchain. 2017a. "Developer Guide: Peer Discovery." Making Transactions. <https://goo.gl/famYFA>.
- [74] Dcousens. 2018. "BitcoinJS:"The Clean, Readable, Proven Library for Bitcoin JavaScript Development"." BitcoinJS. <https://goo.gl/2E7Yfi>.
- [75] Wiki, Bitcoin. 2018. "Raw Transactions." Raw Transactions - Bitcoin Wiki. <https://goo.gl/gP1Z4Y>.
- [76] Croman, Kyle, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, et al. 2016. "On Scaling Decentralized Blockchains." In International Conference on Financial Cryptography and Data Security, 106–25. Springer.
- [77] Johnson, Benjamin, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. 2014. "Game-Theoretic Analysis of Ddos Attacks Against Bitcoin Mining Pools." In Financial Cryptography and Data Security: FC 2014 Workshops, Bitcoin and Wahc

- 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers, edited by Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, 72–86. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-44774-1_6.
- [78] Baqer, Khaled, Danny Yuxing Huang, Damon McCoy, and Nicholas Weaver. 2016. “Stressing Out: Bitcoin ‘Stress Testing’.” In International Conference on Financial Cryptography and Data Security, 3–18. Springer.
 - [79] Wueest, Candid. 2014. “The Continued Rise of DDoS Attacks.” White Paper: Security Response, Symantec Corporation.
 - [80] Mirkovic, J., and P. Reiher. 2004. “A Taxonomy of Ddos Attack and Ddos Defense Mechanisms.” ACM SIGCOMM Computer Communication Review 34 (2). ACM.
 - [81] Community, Blockchain. 2017d. “Fee Estimation.” Estimatefee.com — Recommended Optimal Fees. <https://estimatefee.com/>.
 - [82] GuySir. 2017. “Someone Is Spamming the Mempool with Extremely Low-Fee Transactions.” Reddit. <https://goo.gl/AadTcu> /.
 - [83] “Apache JMeter,” Retrieved from jmeter.apache.org/.
 - [84] Butnix, JP. 2017. “Transaction Fees Continue to Cripple the Bitcoin Network.” Blockchain Network, News, Charts, Guides & Analysis. <https://goo.gl/4NKhb9>.
 - [85] Young, Joseph. 2018. “Analyst: Suspicious Bitcoin Mempool Activity, Transaction Fees Spike to \$16.” Cointelegraph. Cointelegraph. <https://goo.gl/hKU8X7>.
 - [86] Nagelkerke, Nico JD. 1991. “A Note on a General Definition of the Coefficient of Determination.” Biometrika 78 (3). Oxford University Press:691–92.
 - [87] Saad, Muhammad and Aziz Mohaisen, "Towards Characterizing Blockchain-based Cryptocurrencies for Highly-Accurate Predictions". International Workshop on Hot Topics in Pervasive Mobile and Online Social Networking, HotPOST 2018.
 - [88] Möser, Malte, and Rainer Böhme. 2015. “Trends, Tips, Tolls: A Longitudinal Study of Bitcoin Transaction Fees.” In International Conference on Financial Cryptography and Data Security, 19–33. Springer.
 - [89] Query, Chain. 2018. “The Sendmany Rpc Creates and Broadcasts a Transaction Which Sends Outputs to Multiple Addresses.” Chain Query: Bitcoin API: Sendmany. <https://goo.gl/Auj5ZC>.
 - [90] Murch. 2017. “Minimum Transaction” Stack Exchange. <https://goo.gl/wCqDWa>.
 - [91] Buntinx, JP. 2017. “What Is Child Pays for Parent?” The Merkle. <https://goo.gl/AhN8j2>.

- [92] Community, Blockchain. 2017a. "Developer Guide: Peer Discovery." Blockchain Transactions. <https://goo.gl/taHqbJ>.
- [93] Karame, Ghassan, Elli Androulaki, and Srdjan Capkun. 2012. "Double-Spending Fast Payments in Bitcoin." In The ACM Conference on Computer and Communications Security, Ccs'12, Raleigh, Nc, Usa, October 16-18, 2012, 906–17. <https://doi.org/10.1145/2382196.2382292>.
- [94] Bamert, Tobias, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. 2013. "Have a Snack, Pay with Bitcoins." In Peer-to-Peer Computing (P2p), 2013 IEEE Thirteenth International Conference on, 1–5. IEEE.
- [95] Rosenfeld, M. (2011). Analysis of bitcoin pooled mining reward systems. arXiv preprint arXiv:1112.4980.
- [96] Courtois, N. T., & Bahack, L. (2014). On subversive miner strategies and block withholding attack in bitcoin digital currency. arXiv preprint arXiv:1402.1718.
- [97] Bahack, L. (2013). Theoretical bitcoin attacks with less than half of the computational power (draft). arXiv preprint arXiv:1312.7013.
- [98] Dua, G., Gautam, N., Sharma, D., & Arora, A. (2013). Replay attack prevention in kerberos authentication protocol using triple password. CoRR, abs/1304.3550. Retrieved from <http://arxiv.org/abs/1304.3550>
- [99] Kim, M., Song, S., & Jun, M.-S. (2016). A study of block chain-based peer-to-peer energy loan service in smart grid environments. Advanced Science Letters, 22(9), 2543–2546.
- [100] Lee, B., & Lee, J.-H. (2017). Blockchain-based secure firmware update for embedded devices in an internet of things environment. The Journal of Supercomputing, 73(3), 1152–1167.
- [101] Abdullah, N., Hakansson, A., & Moradian, E. (2017). Blockchain based approach to enhance big data authentication in distributed environment. In Ubiquitous and future networks (icufn), 2017 ninth international conference on (pp. 887–892).
- [102] Callegati, F., Cerroni, W., & Ramilli, M. (2009, Jan). Man-in-the-middle attack to the https protocol. IEEE Security Privacy, 7(1), 78-81.
- [103] Bastiaan, M. (2015). Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin. In Available at <http://referaat.cs.utwente.nl/conference/22/paper/7473/preventingthe-51-attack-a-stochastic-analysis-of-two-phase-proof-of-work-in-bitcoin.pdf>.
- [104] Brickell, E., & Li, J. (2011). Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. International Journal of Information Privacy, Security and Integrity 2, 1(1), 3–33.

- [105] Anati, I., Gueron, S., Johnson, S., & Scarlata, V. (2013). Innovative technology for CPU based attestation and sealing. In Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy (Vol. 13).
- [106] McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., & Savagaonkar, U. R. (2013). Innovative instructions and software model for isolated execution. In Hasp@isca (p. 10).
- [107] Aumasson, L. (2016). Sgx secure enclaves in practice: Security and crypto review—kudelski security. Black Hat USA.
- [108] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.
- [109] Androulaki, E. (2017). Hyperledger. (<https://www.hyperledger.org/>)
- [110] Brown, R. G. (2017). Introducing r3 corda: A distributed ledger designed for financial services, 2016.
- [111] Liang, X., Shetty, S., Zhang, L., Kamhoua, C., & Kwiat, K. (2017). Man in the cloud (mitc) defender: Sgx-based user credential protection for synchronization applications in cloud computing platform. In The 10th ieee international conference on cloud computing (cloud 2017).
- [112] T. Ma, H. Wang, J. Cao, J. Yong, and Y. Zhao, “Access control management with provenance in healthcare environments,” in Computer Supported Cooperative Work in Design (CSCWD), 2016 IEEE 20th International Conference on. IEEE, 2016, pp. 545–550.

A-1 – List of Publications

1. Sachin Shetty, Charles Kamhoua, Laurent Njilla, "Blockchain for Distributed Systems Security", Wiley-IEEE Computer Society Press, 1 edition, March 19, 2019.
2. Deepak Tosh, Sachin Shetty, Xueping Liang, Charles Kamhoua, Laurent Njilla, "Towards Assured Data Provenance in Blockchain Integrated Cloud", IEEE Consumer Electronics Magazine, 2019
3. Deepak Tosh, Sachin Shetty, Peter Foytik, Laurent Njilla, Charles Kamhoua, "Blockchain Empowered Secure Internet-of-Battlefield Things (IoBT) Architecture", Milcom, Los Angeles, October 2018.
4. Deepak Tosh, Sachin Shetty, Peter Foytik, Charles Kamhoua, Laurent Njilla, "CloudPoS: A Proof-of-Stake Consensus Design for Blockchain Integrated Cloud", IEEE International Conference on Cloud Computing (Cloud 2018), July 2-7, 2018, San Francisco, CA, USA, August 2018 (Acceptance ratio: 15 %)
5. Xueping Liang, Sachin Shetty, Deepak Tosh, Peter Foytik, Lingchen Zhang, "A Reliable Data Provenance and Privacy Preservation Architecture for Business-Driven Cyber-Physical Systems Using Blockchain", International Journal of Information Security and Privacy, 2018 (Accepted)
6. Xueping Liang, Sachin Shetty, Deepak Tosh, Peter Foytik, Lingchen Zhang, "Leveraging Intel SGX to Enable Trusted and Privacy Preserving Membership Service in Distributed Ledgers", International Journal of Information and Computer Security, 2018 (Accepted)
7. Xueping Liang, Sachin Shetty, Deepak Tosh, Peter Foytik, Lingchen Zhang, "Leveraging Intel SGX to Enable Trusted and Privacy Preserving Membership Service in Distributed Ledgers", International Journal of Information and Computer Security (accepted), 2018
8. Deepak Tosh, Sachin Shetty, Xueping Liang, Charles A. Kamhoua, Laurent Njilla, "Consensus Protocols for Blockchain-based Data Provenance: Challenges and Opportunities," 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON) (Short Paper), NY, October 19-21, 2017
9. Xueping Liang, Juan Zhao, Sachin Shetty, Danyi Li, "Towards Data Assurance and Resilience in IoT Using Blockchain," IEEE Milcom, October 23-25, 2017, Baltimore, MD, USA
10. Sachin Shetty, Val Red, Charles Kamhoua, Kevin Kwiat, Laurent Njilla, "Data provenance assurance in the cloud using blockchain," Proc. SPIE 10206, Disruptive Technologies in Sensors and Sensor Systems, 102060I (2 May 2017);
11. Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, Laurent Njilla, "ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability", The 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), May 14-17 2017.
12. Deepak Tosh Sachin Shetty, Xueping Liang, Charles Kamhoua, Kevin Kwiat, Laurent Njilla, "Security Implications of Blockchain Cloud with Analysis of Block Withholding Attack", 17th IEEE/ACM International Symposium on Cluster, Cloud

- and Grid Computing (CCGRID), May 14-17 2017.
13. Muhammad Saad, Aziz Mohaisen, Charles Kamhoua, Kevin Kwait, & Laurent Njilla (2018). Countering Double Spending in Next-Generation Blockchains. 28th IEEE International Conference on Communications(ICC).
 14. Muhammad Saad, Jeffrey Spaulding, Aziz Mohaisen, Charles Kamhoua, Laurent Njilla ,& DaeHun Nyang (2018). Exploring the Attack Surface of Blockchain. 28th IEEE International Conference on Communications (ICC).

A-2 - List of Presentations

1. "Blockchain & AI Technology: New Tools for Cyber Defense", AAAE/Planet Defense Aviation Cybersecurity Workshop, Arlington, VA, December 23, 2018
2. "Blockchain and Process Control Systems Security", *Supply Chain Cybersecurity & Cloud-based Services Workshop*, October 23, 2018, FortWorth, Texas.
3. "Cyber Resilience Metrics", EPRI Webinar, October 18, 2018
4. "Blockchain Research Overview", ODU CyberOps, October 19, 2018
5. Panel Discussion, "Real-World Look at Blockchain- Its hear to stay", WEDI Blockchain Conference, Arlington, VA, October 15-17, 2018
6. "Blockchain for Cyber Supply Chain Provenance", AFPM/OPCAT Cybersecurity Conference, Atlanta, GA, October 1, 2018
7. "Blockchain for Energy Delivery Systems", Argonne National Laboratory (ANL) DOE CEDS Outreach Energy Sector Industry Advisory Team (IAT), Third Quarter Teleconference Call, September 18, 2018.
8. "Blockchain and IoT Security", VA-DC HFMA Fall Education Conference, September 19-21, 2018, Virginia Beach, VA
9. "Blockchain and Cyber Security", ODU NSA project on Blockchain and Cybersecurity, August 27, 2018
10. Panel Discussion, "Blockchain and Cyber Security, VA HIMMS, April 27, 2018
11. "Blockchain empowered cyber threat sharing platform", US Transacom, December 2017
12. "Provchain- A blockchain based data provenance for cloud computing", MITRE, November 2017
13. "Blockchain and Data Provenance", CERDEC, November 2017
14. "Blockchain based Trusted Information Sharing", NATO Workshop, October 2017
15. "Networked Identity Management of Internet of Things using Blockchain", AFPM Cyber security seminar, September 2017
16. "Data Provenance Assurance using Blockchain in Cloud Environment", UTRC, March 2017
17. "Blockchain for Cloud Security", Air Force Space Command, January 2017.

A-3 Blockchain For Distributed Systems Security Book Cover

- Sachin Shetty, Charles Kamhoua, Laurent Njilla, "Blockchain for Distributed Systems Security", Wiley-IEEE Computer Society Press, 1 edition, March 19, 2019.
- Contains a description of the properties that underpin the formal foundations of Blockchain technologies and explores the practical issues for deployment in cloud and IoT platforms.
- Topics including data provenance in cloud storage, secure IoT models, auditing architecture, and empirical validation of permissioned Blockchain platforms.



A-4 2018 Top 50 Blockchain Papers and Research Team Award



**BlockchainConnect
Conference**

Academic 2019

Hosted by *SV Insight*

2018 Top Blockchain Papers and Research Teams

SELECTED PAPER

ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability

Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, Laurent Njilla

A gold circular award ribbon is attached to the top right corner of the graphic, reading: BlockchainConnect Conference, 2018 Top Blockchain Papers and Research Teams.



**BlockchainConnect
Conference**

Academic 2019

Hosted by *SV Insight*

2018 Top Blockchain Papers and Research Teams

SELECTED PAPER

Security Implications of Blockchain Cloud with Analysis of Block Withholding Attack

Deepak K. Tosh, Sachin Shetty, Xueping Liang, Charles A. Kamhoua, Kevin A. Kwiat, Laurent Njilla

A gold circular award ribbon is attached to the top right corner of the graphic, reading: BlockchainConnect Conference, 2018 Top Blockchain Papers and Research Teams.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BTC	Bitcoin
BWH	Block Withholding
CA	Certificate Authority
CERDEC	Communications-Electronics Research, Development and Engineering Center
CPU	Central Processing Unit
CSP	Cloud Service Provider
CTMC	Continuous Time Markov Chain
DAO	Data Access Object
DDOS	Distributed Denial-of-Service
DNS	Domain Name System
ECDSA	Elliptic Curve Digital Signature Algorithm
EPC	Enclave Page Cache
EPCM	Enclave Page Cache Map
FAW	Fork After Withholding
GBC	Generalized Bso-Chaum
GBCW	Generalized Bso-Chaum-Winternite
GPU	Graphical Processing Unit
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LDW	Lamport-Diffie-Winternite
MAC	Media Access Control
MB	Megabyte
MITC	Man in the Cloud
NATO	North Atlantic Treaty Organization
P2P	Peer-to-Peer
PA	Provenance Auditor
PBFT	Practical Byzantine Fault Tolerance
PKI	Public Key Infrastructure
PoA	Proof of Activity
PoET	Proof of Elapsed Time
PoS	Proof of Stake
PoW	Proof of Work
PPLNS	Pay Per Last N Shares
SGX	Software Guard Extensions
TPS	Transactions Per Second
URL	Uniform Resource Locator
UTXO	Unspent transaction output