Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Availability analysis of a permissioned blockchain with a lightweight consensus protocol

Amani Altarawneh[b,*], *Fei Sun*[a], *Richard R. Brooks*[a], *Owulakemi Hambolu*[a], *Lu Yu*[a], *Anthony Skjellum*[b]

[a] *Dept. of Electrical and Computer Engineering, Clemson University, Clemson, SC*
[b] *SimCenter & Dept. of Computer Science and Engineering, University of Tennessee at Chattanooga, Chattanooga, USA*

ARTICLE INFO

Article history:
Received 2 December 2019
Revised 21 September 2020
Accepted 23 October 2020
Available online 19 December 2020

Keywords:
Blockchain
Digital ledger technology
Non-cryptocurrency
Secure provenance
Scrybe
Lightweight mining
Consensus protocols

ABSTRACT

This paper offers a novel approach to the evaluation of provenance blockchain security and reliability using analytical methods for assessing system availability against malicious miner DoS attacks. In particular, we present the reliability and availability analysis of the **L**ight**W**eight **M**ining (LWM) protocol for securing data provenance. Our analysis shows the reliability of the protocol and its ability to protect against malicious miner DoS attacks. We use digital signatures to prove integrity and non-repudiation of messages passing the system. We describe system behaviors using communicating sequential processes (CSP) to check for synchronization within a number of concurrent processes. Queuing theory is used to determine the average waiting time for client blockchain transactions when malicious miners work to slow the system. CSP and queuing theory jointly test the blockchain's ability to make progress despite the presence of malicious miners. Further, the methodology described can be extended to other blockchain applications. Additional threats, beyond the malicious miner DoS attack, are reserved for future work.

1. Introduction

The first implementation and application of blockchain distributed ledger technology was Bitcoin (Nakamoto, 2008). Because of Bitcoin's ongoing success, there are now many industries seeking to utilize blockchain's technological potential in applications other than cryptocurrency. For instance, blockchain technology can be used to maintain provenance metadata for systems and ensure non-repudiation and integrity of data. Bitcoin's mining algorithm, Proof of Work (PoW) (Gervais et al., 2016), is resource-intensive in order to limit inflation of the underlying cryptocurrency. However, noneconomic applications of blockchain technology do not need to limit the number of blocks being mined, so using PoW would be both wasteful and exorbitant. Instead, our innovative blockchain technology, Scrybe (Brooks et al., 2018; Worley et al., 2018), incorporates a lightweight mining (LWM) protocol that poses minimal resource requirements and maintains the same integrity guarantees (Brooks et al. (2018)). To show this, we present a proof that Scrybe, using LWM, is both re-

* Corresponding author.
E-mail addresses: jwh247@mocs.utc.edu (A. Altarawneh), fsun@g.clemson.edu (F. Sun), rrb@g.clemson.edu (R.R. Brooks), ohambol@g.clemson.edu (O. Hambolu), lyu@g.clemson.edu (L. Yu), tony-skjellum@utc.edu (A. Skjellum).
https://doi.org/10.1016/j.cose.2020.102098

Table 1 – LWM vs. common consensus algorithms.

Consensus algorithms	Permission to join	Energy cost	Ability to fork	Blockchain technology
PoW	permissionless	High	Yes	Bitcoin
PoS	Both	Low	Yes	Ethereum soon
DPoS	permissionless	Low	Yes	Bitshares
Ripple Consensus	permissionless	Low	Yes	Ripple
PoET	Both	Low	Yes	Intel SawtoothLake
PBFT	permissioned	Low	No	HyperLedger Fabric
RAFT	permissioned	Low	No	Zookeeper (Zookeeper, 2017)
LWM	permissioned	Negligible	No	Scrybe

liable and available against malicious miner DoS attacks. A blockchain provides security by having miners with conflicting interests work together. By removing incentives for collusion, we can obtain guarantees of immutability for all participants, but we need to guarantee that miners cannot abuse their presence in the system to favor certain parties. In this context, we therefore can operate a feasible permissioned system that still includes conflicts. For example, with clinical trials (Brooks et al., 2018), we only allow medical institutions, pharmaceutical concerns, and regulators access to the chain. Our approach of issuing them X.509 certificates (Medury et al., 2018)does not compromise the blockchain's efficacy; it does, however, remove vulnerability to Sybil attacks (Douceur, 2002) and scaling problems arising from unrelated data being stored on chain.

The Scrybe blockchain provides reliability and availability by guaranteeing three characteristics; non-repudiation, data integrity, and resilience against denial of service (DoS) attacks[1], respectively. To ensure these characteristics, we study the performance of Scrybe as more malicious miners are added to the system. We say that a miner is malicious if it ignores one or more clients and excludes their transactions from the block that it is currently mining and adding to the blockchain. When malicious miners invade the mining process, the system will necessarily suffer from reduced efficiency and/or complete failure. We use the average client waiting time as an indicator of system performance. When the number of malicious miners increase, the client waiting time should concomitantly increase. We utilize queuing theory to calculate the client's expected waiting time until being served.

This paper presents the LWM consensus algorithm that achieves consensus on who is selected to be the next miner, as explained in Section 2.2. LWM achieves consensus with less overhead than current mining approaches; a comparison of relevant blockchain consensus properties is provided in Table 1. We also prove Scrybe's relevant security properties, which are integrity, non-repudiation, and availability. These are described in Section 4. Most importantly, we provide a theoretical proof, confirmed through simulation, of Scrybe's stability and resilience to insider threats from misbehaving miners. This is particularly significant because it is called out as an open problem in Narayanan et al.'s book (Narayanan et al., 2016).

The remainder of this paper addresses the above issues and is organized as follows: Background and related work are described in Section 2. The Scrybe blockchain provenance system is described in Section 3. Reliability and availability against malicious miner DoS attack verification for Scrybe is described in Section 4, which contains most of the key results of the paper. Section 5 describes LWM in the presence of malicious miner denial of service (DoS) attack. Finally, Section 6 offers conclusions and outlines some potential future work.

2. Background and related work

This section provides the definition of secure data provenance and implementations that are blockchain-based, and an overall comparison between LWM and other current consensus protocols.

2.1. *Blockchain-based data provenance implementations*

Provenance data consists of one or more artifacts of metadata that can be used to provide data provenance by tracking changes to data over time, which ensures the data's integrity (Technopedia, 2018). Secure provenance enforces accountability and non-repudiation, meaning a user cannot dispute responsibility for authoring a change (Benchoufi et al., 2017). Further, in case of an error (or tampering), secure provenance provides the capability to chronologically trace back changes in order to successfully identify the change responsible for the error and when that change occurred (Benchoufi et al., 2017). Because blockchains are designed to be a highly available distributed databases, they provide an ideal infrastructure for implementing secure provenance (Zinder, 2018). Additionally, the provenance fields inherent to the blockchain data structure—including timestamp, block number, and miner's signature—can be leveraged and incorporated into a secure provenance system like Scrybe. For example, the integrity of a cloud computing server needs to be ensured whether it is being used privately, commercially, and/or for defense-related purposes.

A full comparison of existing systems and technologies is beyond the scope of this paper[2]. However, we offer the follow-

[1] Consideration of client-based DoS/DDoS attacks is beyond the scope of this paper, which focuses on the system's availability of the blockchain itself while considering potentially malicious miners. Availability analysis in the presence of DDoS attacks is addressed in Bhat's thesis (Bhat, 2020).

[2] See Mukhopadhyay et al. (2016), in which some of us and others provided a further overview.

ing examples that are particularly relevant to our discussion of blockchain-based data provenance. ProvChain (Liang et al., 2017) is a blockchain-based provenance system that enables auditing data operations while preserving privacy and ensuring integrity. DApps (Wikipedia contributors, 2020a) are decentralized applications that run on the top of a blockchain system, like Ethereum blockchain technology, to enable traceability of certifications and other important information in supply chains. The Provenance Hyperledger (Benningfield, 2015) analyzes the possibility of using Hyperledger blockchain technology for tracing supply chains and avoiding counterfeit products by storing provenance data on a permissioned blockchain network.

2.2. *Lightweight mining vs. other consensuses protocols*

Blockchain technology is classified into two categories: permissioned and permissionless (Altarawneh et al., 2020; Baliga, 2017) based on how the miners participate. Permissionless blockchains are open for any miner to join, whereas permissioned blockchains require miners to be granted access (Baliga, 2017). Some examples of permissionless blockchains include Bitcoin, which uses the Proof-of-Work (PoW), and Ripple (Schwartz et al., 2014), which uses the Ripple Consensus Protocol. Miners in the Bitcoin PoW algorithm need to find a value that solves a puzzle quickly in order to add to the blockchain and earn cryptocurrency as a reward. Solving the puzzle involves computing many hashes, which consumes a lot of computational power. Additionally, the PoW algorithm is vulnerable to 51% attacks (Lin and Liao, 2017), which can occur if one entity owns more than half of the total processing power in the Bitcoin network (Xu, 2016). Similarly, the Ripple consensus protocol is designed so that each miner must create a list of other trusted miners that never collude against it. This list, called the Unique Node List (UNL) (Baliga, 2017), must have less than 40% overlap with any other UNL to prevent collusion. Each miner selects a set of transactions and broadcasts it to the miners in its UNL; these miners then validate the transactions and vote on them. Voting takes place in multiple rounds, and only transactions with a super-majority of 80% or more votes become a valid block.

Ethereum and Intel SawtoothLake (Baliga, 2017) are blockchain platforms that are considered both permissioned and permissionless. In Proof-of-Stake (PoS), which Ethereum is said to be adopting soon (Wood et al., 2014), miners need to maintain a sufficiently large amount of cryptocurrency in order to have a higher chance to be selected to create the block and earn that new block's reward. Miners are selected randomly to prevent malicious behavior, but miners with more stake are more likely to be selected. Another variant of the Proof of Stake (PoS) consensus algorithm that is implemented in BitShares (bitshares.org, 0000) is Delegated Proof of Stake (DPoS) (Block.one, 2018); DPoS is used in permissionless blockchain where participants stake an amount of cryptocurrency in order to qualify as a candidate to mine the next block.

Intel SawtoothLake uses the Proof of Elapsed Time (PoET) consensus algorithm (Baliga, 2017), which relies on specialized hardware. All miners must run a Trusted Execution Environment (TEE), which randomly generates waiting times for each miner. The miner with the shortest waiting time generated by the EET will mine the next block. Practical Byzantine Fault Tolerance (PBFT) (Castro and Liskov, 1999) is a consensus algorithm that is used in the HyperLedger Fabric (Baliga, 2017), a permissioned blockchain. PBFT uses the state machine replication concept (Cachin, 2010) and requires $3n+1$ replicas to be able to tolerate n faulty nodes. Messages sent between nodes and replicas are signed and encrypted to decrease the number of messages needed to maintain trust. However, overhead in the network increases with the number of replicas. Another variant of the Byzantine fault tolerance algorithms is the Reliable, Replicated, Redundant, and Fault-Tolerant (RAFT) (Ongaro and Ousterhout, 2013) algorithm, which was developed as an alternative to Paxos (Lamport, 1998) and is both simple and practical (Ongaro and Ousterhout, 2013). RAFT is used in permissioned blockchain where the nodes collaborate with each other to elect a leader who produces the next block. Once a leader has been elected, another phase begins where the leader receives a transaction, adds it to its log, and broadcasts the log to all other nodes (Ongaro and Ousterhout, 2013). Although Scrybe is a permissioned blockchain like those described, its consensus algorithm is distinct. Table 1 shows a comparison between LWM and other common consensus algorithms based on the blockchain type for which the algorithm is used, the cost, the ability to fork into different chains, and its application(s). A deep comparison for LWM with other consensus algorithms based on other metrics (Vukolić, 2015) such as the ability of the algorithm to tolerate faults or crashes is required; however, LWM is under development and such comparisons will be offered in the future work. Scrybe, with the LWM algorithm, is a fast, scalable, reliable, and eco-friendly alternative for a permissioned network of miners.

3. Scrybe: A secure-provenance blockchain

This section provides an overview of Scrybe—a secure-provenance blockchain—by explaining the blocks and transactions and the algorithm itself.

3.1. *Blocks and transactions*

Transactions and blocks are the main components in the Scrybe blockchain. Computational actors called miners are responsible for mining the blocks and adding them to the blockchain. Mining is the process of including a transaction in a data block and adding that block to the blockchain. A transaction in Scrybe includes client details such as name, signature, and public key, as well as the submission timestamp, the hash of the source data using the SHA-3 algorithm, and the persistent URLs (PURLs) that point to the source data (extraneous to the blockchain). Miners aggregate a group of transactions and add them as one block to the blockchain upon a successful mining step. Miners use the Merkle root (Becker, 2008) to quickly verify if any of the transactions under consideration are already included in another block. The LWM algorithm selects a miner that will be responsible for adding the block to the blockchain and broadcasting the block to other miners to prevent duplicate blocks. In addition to the Merkle root, miners use the hash of the previous block and the miner's signature to check for potential malicious behavior

such as a miner purposely omitting transactions or unauthorized miner(s) broadcasting invalid blocks (Guin et al., 2018).

3.2. *Lightweight mining algorithm*

Hash numbers broadcast by miners can be based on a miner-generated random number. Collusion can occur when several miners remain to manipulate the process. However, LWM has the ability to function as long as good miners protect against collusion (Pirretti et al., 2006). The LWM algorithm's strength comes from requiring all miners to share their hash first. This requirement prevents miners from withholding their number and tampering with the process. Without this requirement, a miner could wait until all other miners have submitted their numbers, then generate a number that excludes or selects a certain miner. The enormous amount of time needed to invert (reverse-engineer) the hashes created by the LWM algorithm makes this situation highly unlikely in Scrybe. Scrybe also requires each miner to include their hash to protect against deniability. Another strength of the LWM algorithm is the ability to function correctly as long as there is at least one good miner (the $\sum_j r_j \mathrm{mod} N$ remains random). As long as the transaction is in one good miner's pool, the transaction will be mined, but it will take longer for the transaction to be added to the blockchain if there are a large number of bad miners. Therefore, to guarantee the miners in Scrybe are able to reach consensus, we assume that the number of malicious miners is less than $\frac{1}{3}$ of the total number of miners. This assumption relies on Lamports Byzantine Fault Tolerance (BFT) (Lamport et al., 2019) results, where it is impossible to reach consensus if the number of faulty nodes, f is not less than $\frac{1}{3}$ of the total miners, n. This also assumes that the percentage of miners participating in the mining process is greater than 50% of the total number of miners. For the sake of simplicity, this version covers the aspects that are relevant to the scope of this paper. The scope of this paper includes the analysis of the system's availability against sabotage by malicious miners as a form of denial of services (DoS) attacks (Narayanan et al., 2016). The scope excludes the analysis of the nodes' ability to converge to a decision despite the presence of arbitrary errors, and excludes the analysis of the ability of LWM to reach consensus despite the presence of malicious miners[3], these issues are covered by Bhat, 2020). The LWM algorithm code (Brooks et al., 2018; Guin et al., 2018) is as follows:

4. Security verification for scrybe

Established techniques are applied here to analyze LWM's behavior and its ability to counteract known attacks. The presented analysis proves that the lightweight mining (LWM) protocol is secure and that it prevents relevant attacks such as malicious miner denial of service (DoS) attacks. According to

[3] This paper considers possible attacks in the system that are caused by miners delaying responses or/and forging messages. Timing issues and other aspects are not presented here because 1) they are not relevant to the problem we are discussing, 2) these issues depend on entirely different tools of verification, and 3), the page limit.

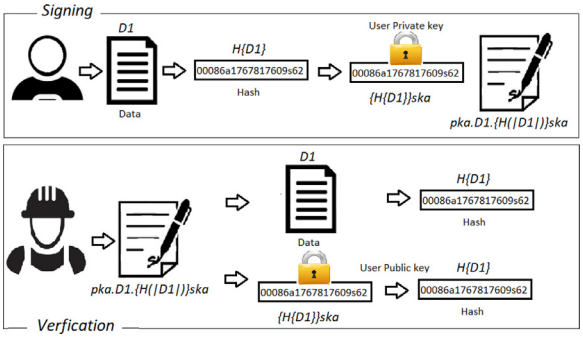

Fig. 1 – Digital signature diagram, adapted from (Dig-Signature, 2019).

Narayanan's work (Narayanan et al., 2016), DoS issues in mining protocols remain an open issue. The systems work, but there is not a theoretical understanding as to why they do so. Established tools are used here to give support for LWM guaranteeing that transactions eventually arrive on the chain. Digital signatures are leveraged in the LWM protocol to serve two purposes: integrity, and non-repudiation. Communicating Sequential Processes (CSP) is used to describe system behaviors and to check for synchronization within a number of concurrent processes. Queuing theory is used with the LWM protocol to show the algorithm's availability under malicious miner DoS attacks. (Other adversarial behavior (Cachin and Vukolić, 2017), such as the Sybil attack, are out of the scope of this paper (Narayanan et al., 2016)).

4.1. *Integrity*

A system's integrity is guaranteed when any corruption or tampering of its data can always be detected (Bishop, 2003). The LWM protocol in Scrybe uses digital signatures to assure system integrity and ensure that messages are not altered in transit. These messages can include financial transactions, software distributions, or any other message where there is a need to detect data tampering or forgery. A digital signature is implemented in Scrybe using a combination of the Rivest-Shamir-Adleman (RSA) algorithm (Katz and Lindell, 2014) and the SHA-256 cryptography hash function (Delfs et al., 2002). Digital signature use is illustrated in Fig. 1.

4.2. *Non-repudiation*

In the LWM protocol, when a client sends a transaction D1 to a miner and the miner sends a block B1 (where B1 = Sign1, Sign2,..., SignN) to the blockchain, all parties obtain the evidence from the digital signature that the transaction was sent by the client and the block was sent by the miner, as illustrated in Fig. 1. The miner can prove that the processed message was sent by the client by presenting $\{[D1]\}_{sk_a}$, which is the message signed by the client's private key (Ryan et al., 2001). Therefore, neither the sender nor the receiver of a message can deny transmitting or receiving the messages. Because digital signatures make deniability impossible, they provide high confidence authentication.

Table 2 – Format of messages.

$T_i, B_i ::=$	Messages (T: Transactions, B: Blocks)
i ($\in$ *User*)	Agent identities
sk_a ($\in$ *Key*)	Secret keys
$[T_1]_{sk_a}$	Signing of message T_1 with agent *a*'s secret key
$[< Sign_1, \cdots, Sign_N >]_{sk_b}$	Block: a group of *N* transactions signed with agent *b*'s secret key

4.3. *Availability*

When malicious miners drop valid transactions or erase transactions from their respective blocks, the system's availability is put at risk (Narayanan et al., 2016; Ryan et al., 2001). However, this risk is mitigated because blockchains are distributed systems. Clients submit transactions to all the miners in the blockchain, so any honest miner will include the transactions when they create a new block. Subsequently, miners that decide not to add a block to the blockchain will have a shorter chain, which will not match the copy that all miners have reached consensus on the network. Queuing theory (Kovalenko, 1968) is used to predict how much of a network an intruder can control. This is discussed further in Section 5.2.

4.4. *Communications sequential processes (CSP) modeling*

Communicating sequential processes (CSP) is a mathematical framework for describing and analyzing system behavior of multiple agents communicating by passing messages back and forth (Hoare, 1978). CSP describes two different classes: events and processes. Events represent communications, and processes represent behaviors. Each process is a set of guarded commands, where the guard is a condition that must be true for the command to execute, and each command execution is a transition to a new state. To achieve synchronization within numerous concurrent processes, the input and output must be synchronized (i.e., when one process is ready to send an output the other is ready to receive the input). If either one is not ready to communicate, the process is put in a wait queue. The system enters a non-deterministic state if more than one guard condition can be satisfied simultaneously (Hoare, 1978). Ryan et al. (2001) used CSP modeling to formalize security properties, which we use to model the LWM security protocol. Each message that could be sent by anyone in the system is treated as a command. Messages in this analysis are of two types, transactions and blocks, that have specific structures to include encryption and signing. To apply the CSP framework to analyze the LWM protocol, we must first define all the possible messages and their respective formats. These are defined in Table 2.

We model the participants using the LWM protocol in CSP and considered four agents: the client (initiator), the miner (responder), the server (in which the blockchain is stored), and the intruder (actor compromising the process). In this description, we use send channels as outputs and receive channels as inputs. Agents of the LWM protocol are described below:

- The process-local environment opens a session with agent *b*. Client *a* initiates the communications by sending a signed transaction, and running the protocol in CSP as follows:

$$Client(a) = env?b : Agent \rightarrow send.a.b.[T_1]_{sk_a} \rightarrow \\ \square sk_a \in Key \\ a, b \in User$$

- The selected miner, "miner *b*," receives the transactions in its pool and creates and sends a block. This miner runs the protocol in CSP as follows:

$$Miner(b) = \begin{pmatrix} receive.a.b.[T_1]_{sk_a} \rightarrow \\ send.b.s.[Sign_1, Sign_2, \cdots, Sign_N]_{sk_b} \rightarrow \end{pmatrix} \\ \square sk_a, sk_b \in Key \\ a, b, s \in User$$

- Server *s* receives the block and adds it to the blockchain copy that it stores. The server runs the protocol in CSP as follows:

$$Server(s) = \begin{pmatrix} receive.b.s.[Sign_1, Sign_2, \cdots, Sign_N]_{sk_b} \rightarrow \\ add.[B_1]_{sk_b} \rightarrow STOP) \end{pmatrix} \\ \square sk_a, sk_b \in Key$$

- The intruder y runs the protocol in CSP as follows:

$$Intruder(Y) = learn?m : messages \rightarrow Intruder(close(Y \cup \{m\})) \\ \square say?m : Y \cap messages \rightarrow Intruder(Y)$$

Y represents the subset of *facts* that the intruder knows. All the messages that might be generated or accepted by trustworthy agents or by server processes are represented in a subset of the *facts* called *messages*. Additionally, *close(Y)* represents the subset of all the *facts* that are possible under the rules of encryption. Described in CSP, the LWM protocol runs as follows:

1. The client sends a signed transaction (provenance metadata) T_1 to the miners. The signature on the transaction works as evidence that the message was sent by the client. For example, the transaction sent to the miner *b* will be as follows:

$$Message \quad 1 \quad a \rightarrow b : [T_1]_{sk_a}$$

$$evidence.b.[T_1]_{sk_a} \Rightarrow a \qquad sent([T_1]_{sk_a})$$

We model the behavior of an arbitrary user $Agent_a$ as follows:

$$Agent_a(Q) = \\ \square_{b \in Agent, m \in Q} \quad send.a.b.m \rightarrow Agent_a(Q) \\ \square_{m \in Q} \quad evidence.a.m \rightarrow Agent_a(Q)$$

Any user, such as $Agent_a$, with the information in Q can send any of this information in Q and can use it as evidence.

Table 3 – Run between client and intruder.

Externally	Agent	Which Sees	Intruder
take.A.I.m	A	*Send.A.I.m*	$learn.[T_1]_{sk_a}$ $modify.[T'_1]_{sk_a}$
fake.I.B.m	B	*Receive.I.B.m*	$say.[T_1]_{sk_a}$

2. The miners verify that message T_1 was not tampered with in transit and ensure that the client signed it using their private key. Miners will drop the transaction if the verification fails.
3. The miners keep all valid transactions in a pool and generate a candidate block. Each block is composed of these components: The miner ID, the block ID, the timestamp, the signature of the miner, a list of the transactions, all the random values generated by miners and their hashes, both hashes of the current block and the previous block, and the root hash of the Merkle tree. The selected miner then signs the block using its private key and adds it to the blockchain. Assuming *b* is chosen as the miner, *b* sends the following message to all the servers:

$$Message \quad 2 \quad b \rightarrow s1 : [Sign_1, Sign_2, \cdots, Sign_N]_{sk_b}$$

4. The blockchain (server) verifies the block it has received from the miner, and only adds verified blocks to the chain.

4.4.1. *Intruder capabilities described in CSP prove integrity and non-repudiation*

An intruder may be a legitimate user with an identity and public and private keys. The other nodes including the server will communicate with the intruder as a legitimate user. The intruder will be given certain initial knowledge that he or she uses to deduce further information about the traffic he observes in the medium. Because of the properties of the encryption, it is infeasible for the intruder to forge the private and/or public keys Ryan et al. (2001). Therefore, it is difficult for any intruder to tamper with messages while they are in transit. An intruder could cause a type of denial of service (DoS) attack by killing the message, thus preventing it from being delivered. However, because all data are in clear text in this model, there is no negative impact associated with intruders having their own copy of the data.

In this network, the events *say.m* and *learn.m* that are executed by the intruder are the replacements for the *receive.a.b.m* and *send.a.b.m* events that are executed by the agents, which connect the network. The *receive* and *say* channels connect to form the *fake* channel, and the *send* and *learn* channels connect to form the *take* channel. This is represented in Fig. 2. The dotted lines represent renaming. The trace that can happen between the client and intruder is shown in Table 3. It shows the two component processes with their renaming, what the intruder might deduce, and how the intruder might attempt to change the data being transmitted. It should be noted that adding more clients, miners, and blockchains does not change the flow because the process collapses to what is given in Table 3. The intruder who intercepted the transaction might attempt to tamper with the data T_1 producing T'_1 before sending it to the miner. However, this implies that $T_1 \neq T'_1$. so the verification fails and the miner drops the transaction as defined in the protocol.

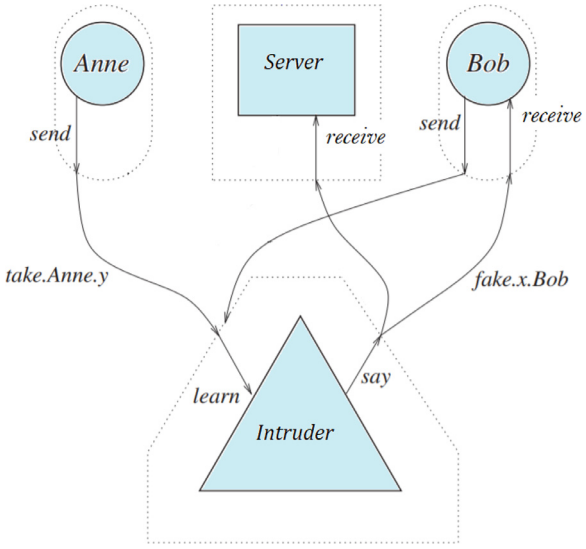

Fig. 2 – Network with the communications including the intruder, adapted from Ryan et al. (2001).

5. LWM vs. malicious miner denial of service (DoS) attack

In this section, we use CSP to describe the intruder's behavior causing a DoS attack, and offer a queue model analysis to show the average waiting time for the victim client transactions to be added to the blockchain.

5.1. *DoS attack described in CSP*

A malicious miner may exclude all transactions originating from a particular client from any block that the miner proposes. This is shown in Table 4.

Since the victim client's transaction (T_1) was not added to block(s) proposed by miner(s), the client will have to wait until an honest miner proposes a block, see Table 5, which will contain the client's transaction.

Consider the case where a malicious miner (*F*) decides to drop a block generated by an honest miner. To model the Malicious $_{miner}$ process, we assume the set *facts* includes the initial knowledge of all agent names, public keys that belong to other agents, the hashing algorithm, and all blocks. The malicious miner always gets the block being transmitted, and either drops the block or adds it to the chain.

$$\begin{aligned} Malicious_{miner}(F) = {} & learn?m : blocks \rightarrow Malicious_{miner}(induce(F \cup \{m\})) \\ & \Box drop?m : F \cap blocks \rightarrow Malicious_{miner}(F) \\ & \Box add?m : F \cap blocks \rightarrow Malicious_{miner}(F) \end{aligned}$$

Table 4 – Case: Miner performing a DOS attack on a client.

Externally	Agent	Which Sees	Miner
$take.A.B.m$	A	$Send.A.B.m$	$learn.[T_1]_{sk_a}$ $drop.[T_1]_{sk_a}$
$fake.B.S_1.m$	S_1	$Receive.B.S_1.m$	$say.[< Sign_2, \cdots, Sign_N >]_{sk_b} >$

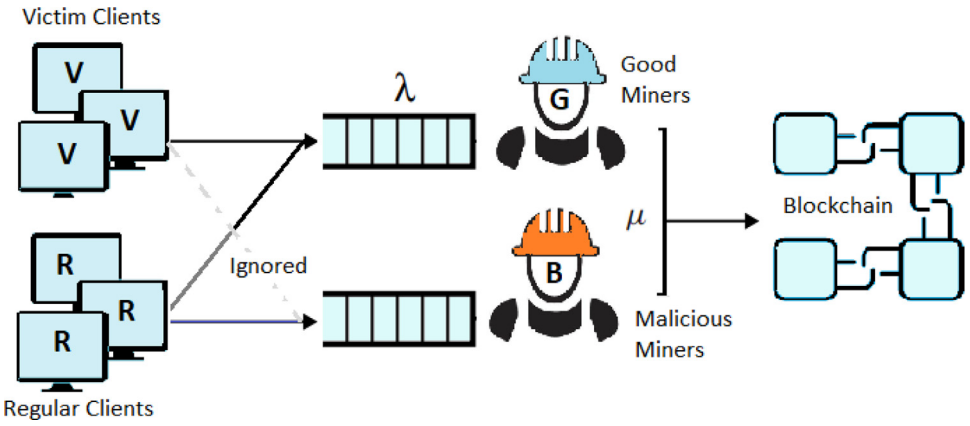

Fig. 3 – The queue model logic.

5.2. *Queuing theory analysis*

The victim's transactions eventually arrive at the good miners' pool (Hambolu, 2018), which is proved using a Petri Net (Salimifard and Wright, 2001). However, in this paper, we are also interested in the average waiting time before a client's transaction is added to the blockchain. To measure this, we utilized queuing theory. There are two types of clients: "regular clients," whose transactions are mined by any miner, and "victim clients," whose transactions are mined only by good miners. We assume that all transactions from regular clients and victim clients will end up in the miner's pool. When a good miner is selected, the transactions follow the first-come-first-serve (FCFS) rule. When a malicious miner is selected, victim client transactions are ignored, and only regular client transactions are served. Victim client transactions continue to wait in the good miners' pool until one of the good miners is selected to mine the next block. We assume in our analysis that the next miner selection process will start after the current miner adds its block to the blockchain. A queue model is adopted to estimate the average waiting time for the victim client transaction to be added to the blockchain.

It is intuitive to conclude that when the number of malicious miners increases, the waiting time for victim clients' transactions will also increase. However, the expected waiting time is of interest. Scrybe is a single server model where only one miner can be active at a time. As described in Fig. 3, we observe the average waiting time for the victim client transactions to be processed.

5.2.1. *Theoretical analysis*

Scrybe follows a birth-death Markov process (Meyer et al., 1972), which is a stochastic process that has a Markov property, also called a memoryless property. This means that one can only predict the future state based on the present state. A birth-death process is a continuous-time Markov chain (Meyer et al., 1972) where the state variable increases or decreases by one, which represents the increase and decrease of the number of transactions. In our analysis, we use the queuing theory model as a single service model, where only one miner can be active at one time. The transaction arrivals and departures follow the Markov process and the population is infinite. In the $M/M/1 : \infty/FCFS$ model, the $\mathbf{M}/M/1$ denotes the transactions that arrive with rate λ and follow the Poisson distribution (Saaty, 1961), the $M/\mathbf{M}/1$: denotes the service times with rate μ that are assumed to follow the exponential distribution (Saaty, 1961), and the $M/M/\mathbf{1}$ denotes one single service at a time. The capacity population of the system is assumed to be infinite and the queue protocol is first come first serve (FCFS). Since we are looking for the expected average waiting time for the victim transactions to wait until they are added to the blockchain, we use the system average waiting time formula as it is known in any basic queue model (Saaty, 1961), given in $\mathbf{W}$ as follows:

$$\mathbf{W} = \frac{\lambda}{\mu(\mu - \lambda)} + \frac{1}{\mu}, \text{ where } \mu > \lambda. \tag{1}$$

In this analysis, we denote the number of regular clients as $\mathbf{R}_{clients}$, the number of victim clients as $\mathbf{V}_{clients}$, the number of good miners as $\mathbf{G}_{miners}$, and the number of malicious miners as $\mathbf{M}_{miners}$. The malicious miners ignore the victim client transactions in their pool and either include other client transactions or include nothing in the block that is being mined. This will cause a delay before the victim client transactions are served, so we are interested in finding the average waiting time for both regular and victim client transactions in the good min-

Table 5 – Case: Honest miner proposes a block that includes the client's transaction.

Externally	Agent	Which Sees	Miner
$take.A.C.m$	A	$Send.A.C.m$	$learn.[T_1]_{sk_a}$ $add.[T_1]_{sk_a}$
$fake.C.S_1.m$	S_1	$Receive.C.S_1.m$	$say.[< Sign_1, Sign_{N+1} \cdots, Sign_{N+N} >]_{sk_c}$

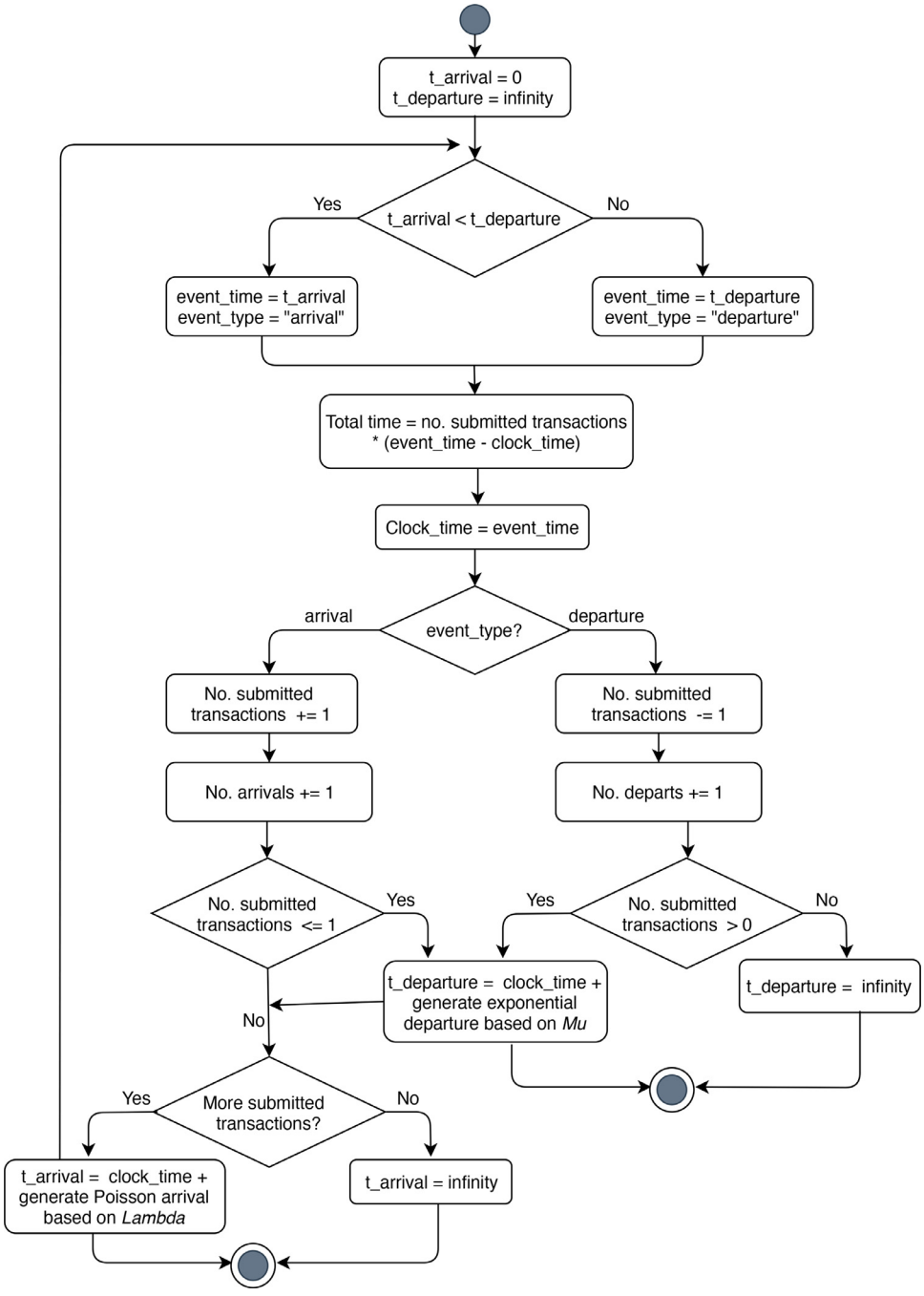

Fig. 4 – Queue model algorithm, adapted from Dis, 2019.

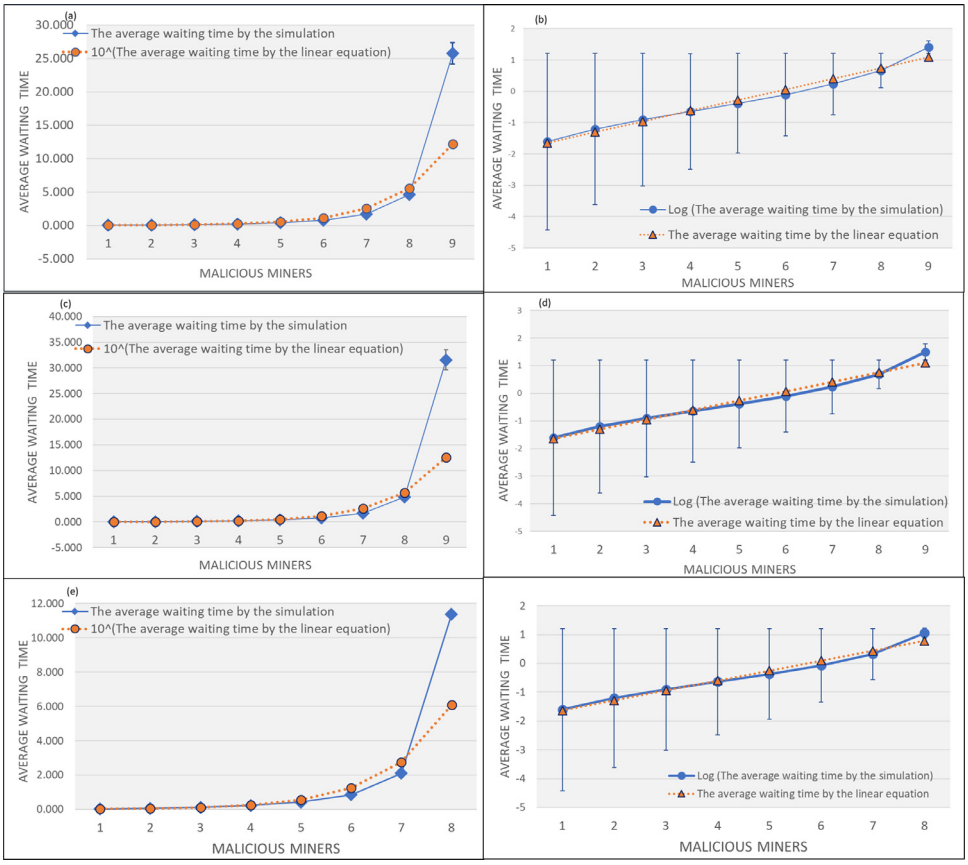

Fig. 5 – Average waiting time vs. number of malicious miners with:(a) and (b): 0.1% victim client transactions, (c) and (d): 50% victim client transactions, (e) and (f): 98% victim client transactions.

ers' pool. We assume that all clients submit their transactions at the same rate, so we set an arbitrary number for the client transaction generation rate as gen_{rate} that is 6000 transactions per second. Additionally, we assume that all miners mine at the same rate, so we set the average mining rate as **r** equals five transactions per second. Using the above conditions, the average arrival rate $\lambda_{G_{miners}}$ for the good miners' pool is given in the following equation:

$$\lambda_{G_{miners}} = gen_{rate} \times (V_{clients} + R_{clients} \times \frac{G_{miners}}{(G_{miners} + M_{miners})}) \quad (2)$$

The average service rate $\mu_{G_{miners}}$ is the reciprocal of the expected average waiting time. We find $\mu_{G_{miners}}$ as follows:

- Each miner has the same chance of being selected, so the probability that the selected miner is good is given in $\mathbf{P}_{G_{miners}}$ as follows:

$$\mathbf{P}_{G_{miners}} = \frac{G_{miners}}{(G_{miners} + M_{miners})} \quad (3)$$

- Thus, the probability that a malicious miner is selected, denoted by $\mathbf{P}'_1$, is the probability of the victim client transactions being delayed, which is as follows:

$$\mathbf{P}'_1 = 1 - P_{G_{miners}} \quad (4)$$

- Accordingly, the probability of a victim transaction being delayed twice in a row is denoted by $\mathbf{P}'_2$ as follows:

$$\mathbf{P}'_2 = (1 - P_{G_{miners}})^2 \quad (5)$$

- Therefore, the probability of a victim transaction being delayed n times in a row is denoted by $\mathbf{P}'_n$ as follows:

$$\mathbf{P}'_n = (1 - P_{G_{miners}})^n \quad (6)$$

Given that the average mining time for any transaction to be mined is $\frac{1}{r} = 0.2$ and using the expected value formula (Ross, 2014), which is the probability of a transaction to be delayed multiplied by the number of delays multiplied by the amount of time for each delay.

- The expected value when there are multiple probabilities of delays, where t is the number of delays, is given in Eq. (7), which simplifies to Eq. (11):

$$\mathbf{E}_{(t)} = \frac{1}{r} \sum_{n=1}^{\infty} n \times (1 - P_{G_{miners}})^n, \text{ where } 0 < P_{G_{miners}} < 1 \quad (7)$$

Divided both sides by $(1 - P_{G_{miners}})$, will get:

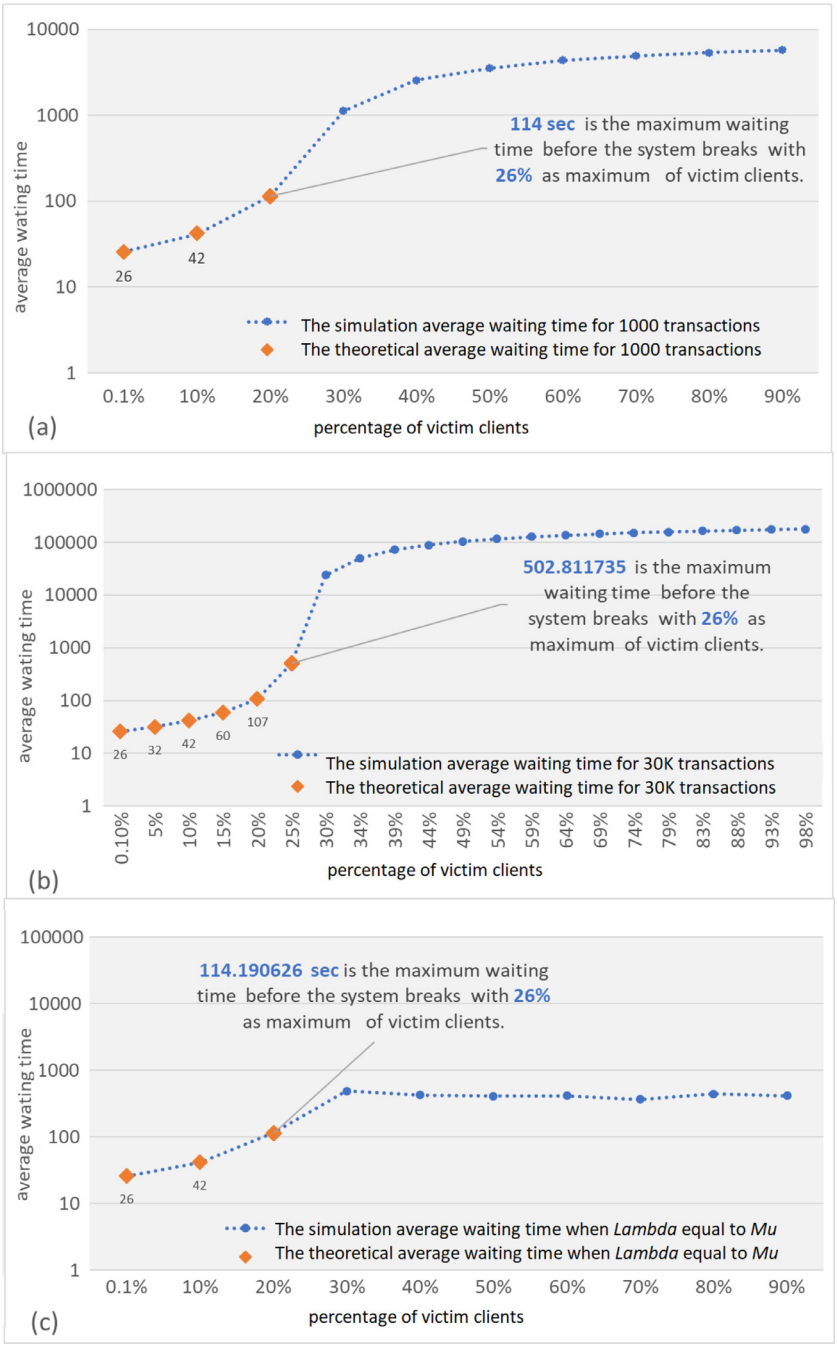

Fig. 6 – The maximum average waiting time and percentage of victim clients before the system breaks down with 90% malicious miners: (a) for 1000 transactions, (b) for 30,000 transactions, (c) for 1000 transactions and fixed λ to 0.055556 where it is equal to μ.

$$\frac{E_{(t)}}{1-P_{G_{miners}}} = \frac{1}{r}\sum_{n=1}^{\infty} n \times (1-P_{G_{miners}})^{n-1}, \text{ where } 0 < P_{G_{miners}} < 1 \tag{8}$$

Using the variant (Taylor, 1955; Wikipedia contributors, 2020b) for the following geometric series:

$$\sum_{n=1}^{\infty} n \times x^{n-1} = \frac{1}{(1-x)^2}, \text{ for } |x| < 1 \tag{9}$$

And substituting into Eq. 8:

$$\frac{E_{(t)}}{1-P_{G_{miners}}} = \frac{1}{r \times (-P_{G_{miners}})^2} \tag{10}$$

Thus,

$$\mathbf{E}_{(t)} = \frac{1-P_{G_{miners}}}{r \times (P_{G_{miners}})^2} \tag{11}$$

Table 6 – Simulation parameters.

Parameter	Symbol	Values
Transactions	T	1K, 30K
Malicious miners	M_{miners}	1 to 10
Good miners	G_{miners}	$\|10 - M_{miners}\|$
Victim clients	$V_{clients}$	1 to 1000
Regular clients	$R_{clients}$	$\|1000 - V_{clients}\|$
Iterations	i	1 to 45
Mining rate	r	5 BPS
Confidence interval percent	CI	95%
Transactions generation rate	gen_{rate}	6000 TPS

- The average time for a transaction to be served, denoted $\mu_{G_{miners}}$, is $\frac{1}{E(t)}$ as in Eq. (12),
  where $\mu_{G_{miners}}$ is the mean service time elapsed before the victim client transaction is added into the blockchain.

$$\mu_{G_{miners}} = \frac{r \times (P_{G_{miners}})^2}{1 - P_{G_{miners}}} \tag{12}$$

5.2.2. *Simulation analysis*

The previously described theoretical approach is implemented as a discrete event simulation (DES) (Cassandras and Lafortune, 2009) in Python. It simulates the queue model using both calculated $\lambda_{G_{miners}}$ and $\mu_{G_{miners}}$ based on the logic above, and the simulation time is advanced based on the event that is closest in occurrence (see Fig. 4). We ran 45 iterations of the simulation with 1000 transactions, 10 miners, and 1000 clients. The results converged after 45 runs of each scenario. We designed the simulation for several scenarios based on different numbers of malicious miners and different percentages of victim clients, simulations parameters which are used as inputs are shown in Table 6. We calculated the average waiting time by summing the difference between departure time and arrival time for all transactions then dividing by the number of transactions. We adopted the 95% confidence interval (Maria, 1997) to find the accuracy and the uncertainty of the estimated average waiting time for our samples. This is calculated for each scenario (1000 transactions) and then averaged out for all 45 iterations. This is shown mathematically in Eqs. (13)-(15), and (16), where i represents a scenario for 1000 transactions that is repeated 45 times for a specific number of malicious miners and a specific percentage of victim client transactions.

$$\mathbf{X}_i = Mean_{1000(Transactions)} \tag{13}$$

$$\mathbf{Mean}_i = \frac{\sum_{i=1}^{45}(\mathbf{X}_i)}{45} \tag{14}$$

$$\sigma_i = \sqrt{\frac{\sum_{i=1}^{45}(\mathbf{X}_i - \mathbf{Mean}_i)^2}{45}} \tag{15}$$

$$\mathbf{95\%\ CI}_i = \frac{1.96 \times \sigma_i}{3\sqrt{5}} \tag{16}$$

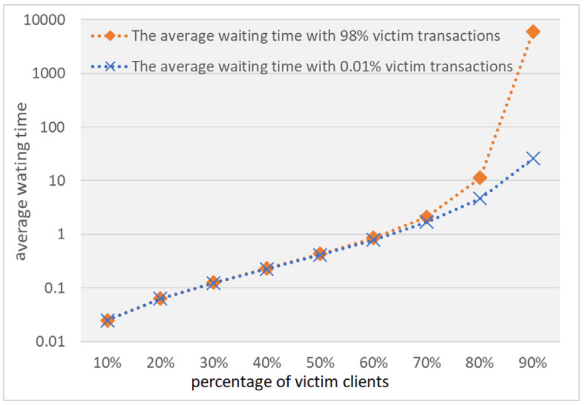

Fig. 7 – Number of victim clients transactions impact on the average waiting time.

5.2.3. *Results*

The average waiting time calculated in the queue model simulation, which is denoted by S_{Wait}, approximately matches the theoretical average waiting time, which is denoted by Th_{Wait}, with 95% of the 45 means within the confidence interval. The average waiting time for the victim client transactions increased exponentially as the percentage of malicious miners increased. Table 7 shows a sample of the simulation results for one scenario (1,000 transactions and 491 victim clients). We applied several procedures to verify the simulation results for each scenario; they are summarized as follows:

- Calculate the log_{10} of the average waiting time resulted from the simulation.
- Apply linear regression (Seber and Lee, 2012) to generate an equation that predicts the average waiting time with different numbers of malicious miners.

$$\mathbf{y}_i = b + a x_i, \text{ where } x_i \text{ represent the number of malicious miners} \tag{17}$$

- Calculate values based on above developed formula.
- Calculate the anti-log which is $10^{\mathbf{y}_i}$, where $\mathbf{y}_i$ is the generated value from Eq. (17).

Points generated from the equation resulting from the data linear regression are plotted in Fig. 5 along with the simulation data. The graphs are done in both linear and log scales.

Table 7 – Simulation results sample.

Inputs		Generated by simulation			Outputs		
$V_{clients}$	M_{miners}	$P_{G_{miners}}$	$\lambda_{G_{miners}}$	$\mu_{G_{miners}}$	S_{Wait}	Th_{Wait}	95% CI_{size}
491	1	0.9	0.158	40.500	0.025	0.025	0.002
	2	0.8	0.150	16.000	0.063	0.063	0.004
	3	0.7	0.141	8.167	0.125	0.125	0.008
	4	0.6	0.133	4.500	0.229	0.229	0.014
	5	0.5	0.124	2.500	0.420	0.421	0.026
	6	0.4	0.116	1.333	0.824	0.821	0.051
	7	0.3	0.107	0.643	1.854	1.867	0.115
	8	0.2	0.099	0.250	6.573	6.614	0.408
	9	0.1	0.090	0.056	3530.911	–28.768	126.611

Table 8 – Simulation results sample.

Inputs		Generated by simulation			Outputs		
$V_{clients}$	M_{miners}	$P_{G_{miners}}$	$\lambda_{G_{miners}}$	$\mu_{G_{miners}}$	S_{Wait}	Th_{Wait}	95% CI_{size}
491	1	0.9	0.158	40.500	0.025	0.025	0.002
	2	0.8	0.150	16.000	0.063	0.063	0.004
	3	0.7	0.141	8.167	0.125	0.125	0.008
	4	0.6	0.133	4.500	0.229	0.229	0.014
	5	0.5	0.124	2.500	0.420	0.421	0.026
	6	0.4	0.116	1.333	0.824	0.821	0.051
	7	0.3	0.107	0.643	1.854	1.867	0.115
	8	0.2	0.099	0.250	6.573	6.614	0.408
	9	0.1	0.090	0.056	3530.911	–28.768	126.611

Algorithm 1: Lightweight Mining Algorithm (LWM).

Initialize $N \leftarrow$ The number of miners;
for *each miner* $m_i, 0 \leq i < N$ **do**
m_i generates a random number r_i;m_i broadcasts the SHA-3 hash of r_i, denoted by $H(r_i)$;Once m_i has collected all N hashes $\{H(r_0), H(r_1), \cdots, H(r_{N-1})\}$, m_i broadcasts r_i.Once m_i has collected all N random numbers $\{r_0, r_1, \cdots, r_{N-1}\}$, m_i calculates $l = \sum_j r_j \mod N$.m_l is the selected miner to create the next block from the collected transactions.(Without loss of generality, we map $m_i = i, 0 \leq i < N$ as a simple rank ordering for the registered miners.)

Both show a reasonable match between both sets of data up to eight malicious miners, at which point they begin to diverge. When there were nine malicious miners and the system and simulation broke down, the results did not match. To show singularity, we ran the simulation for several cases with the scenario of nine malicious miners to find when the system breaks down. This allowed us to discover the maximum waiting time that the victim client should wait to be served, and what percentage of miners can be malicious before the system breaks down. The cases are as follows: running the simulation for 1000 transactions, 30,000 transactions, and fixed λ to be 0.055556, which is close to μ. The analysis shows that the system breaks down when 20% - 30% of the clients are victims. Using linear interpolation, we found that the system singularity occurs when the system has 26% victim clients and nine malicious miners (see Fig. 6). The average waiting time increases exponentially when the number of malicious miners increases, but 90% of miners must be malicious for the system to break down. Increasing the percentage of targeted clients has almost no impact on the average waiting time when the number of malicious miners is less than 80% (see Fig. 7). Targeting more clients when the system has only one good miner has a large impact on the average waiting time.

6. Conclusions

This paper offered a novel approach to evaluate the availability of the Scrybe blockchain, as well as its reliability, using proven analytical methods for assessing system security. We presented the reliability and availability analysis of the lightweight mining (LWM) protocol for secure provenance that is used in the Scrybe blockchain. The analysis showed that the LWM protocol is reliable and that it possesses the ability to protect against certain classes of DoS and delay attacks. Digital signatures were employed in the protocol to prove integrity and non-repudiation of messages. We described the system behavior via communication sequential processes (CSP) to check for synchronization within a number of concurrent processes.

Queuing theory was employed to identify the average waiting time for client blockchain transactions being targeted by malicious miners. This tested the Scrybe blockchain's ability to make progress despite the presence of malicious miners or resistant servers. We showed the synchronization behav-

ior within a number of concurrent messages running in the system among clients and miners. Using the CSP protocol, we also identified an intruder's capabilities and possible threats in the system. Additionally, we designed our queue model as a discrete-event simulation (DES) such that the simulation time is advanced based on the event that is closest in occurrence and the simulation can directly advance to the occurrence time of the next event. This design allowed the simulation to run for longer periods of time with more transactions because of this ability to jump ahead. We chose a 95% confidence interval to guarantee accuracy as well as the uncertainty of the estimated average waiting times for our samples to lie within 5%. We applied linear regression to generate the equation that predicts the average waiting time with different numbers of malicious miners. To validate the results, we calculated the antilog value, which is equal to 10^n, where n is the generated value by the linear equation, to verify the validity of results and we found a strong degree of similarity. Overall, we demonstrated that the lightweight mining (LWM) protocol described in this paper and employed in the Scrybe blockchain is both secure and reliable. Hence, Scrybe provides a sound basis for a data-provenance blockchain. We showed that the LWM protocol guarantees integrity and non-repudiation of messages using digital signatures. Via CSP, we proved that the LWM protocol is reliable against malicious miner DoS attacks; CSP was used to show the synchronization within a number of concurrent processes. We showed the ability of LWM to continue functioning with almost 95% of the miners acting maliciously in the system. Further, we identified the average waiting time that a victim client's transactions take to be added to the blockchain. We proved that targeting successively more victim clients has no impact on the average waiting time of addition of a block to the blockchain but the number of malicious miners does in fact impact the waiting time.

As future work, we note that the methodology described here to evaluate the availability could be applicable to other blockchain-based systems in which the consensus protocol is probabilistic in nature. For such systems, our methodology can be modified to show the reliability and robustness of the system against attacks like those described in this paper. For instance, we can apply this methodology to permissionless blockchains such as Ethereum (Wood et al., 2014). It is a transaction-based state-machine (Idelberger et al., 2016); the system processes the next step based on the inputs it reads. So, we need to show the synchronization within concurrent messages in order to check for deadlock occurrences and/or non-deterministic status. In the LWM protocol, the probability that a good miner is selected is equal among good miners; in Ethereum, the probability differs based on each miner's stake (Wood et al., 2014). The blockchain protocols Hyperledger Fabric, Sawtooth Lake, Ripple (Schwartz et al., 2014) are likely to be amenable to our analysis as well. We will describe such analyses in future communications.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Amani Altarawneh: Methodology, Software, Formal analysis, Validation, Writing - original draft, Writing - review & editing. **Fei Sun:** Methodology, Software, Formal analysis. **Richard R. Brooks:** Conceptualization, Supervision, Formal analysis, Writing - review & editing, Visualization, Funding acquisition. **Owulakemi Hambolu:** Formal analysis. **Lu Yu:** Conceptualization, Writing - review & editing. **Anthony Skjellum:** Conceptualization, Supervision, Formal analysis, Writing - review & editing, Visualization, Funding acquisition.

Acknowledgments

This work was also supported in part by the National Science Foundation under grants Nos. numbers CCF-1562659, CCF-1562306, CCF-1617690, CCF-1822191, CCF-1821431. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Altarawneh A, Herschberg T, Medury S, Kandah F, Skjellum A. Buterin's scalability trilemma viewed through a state-change-based classification for common consensus algorithms. In: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). IEEE; 2020. p. 0727–36.

Baliga A. Understanding blockchain consensus models. In: Persistent. semanticscholar.org; 2017. p. 1–18.

Becker G. Merkle signature schemes, merkle trees and their cryptanalysis. Ruhr-University Bochum, Tech. Rep 2008.

Benchoufi, M., Porcher, R., Ravaud, P., 2017. traceability of consent [version 3; referees: 1 approved, 2].

Benningfield, A., 2015. Hyperledger supply chain traceability: anti counterfeiting. [Online, Accessed:08/15/2018].

Bhat N B. Security analysis of a blockchain network. Online 2020.

Bishop M. What is computer security? IEEE Security & Privacy 2003;1(1):67–9.

bitshares.org,. Delegated proof-of-stake consensus. [Online, Accessed: Nov/17/2019].

Block.one, 2018. EOS.IO Technical White Paper v2. [Online, Accessed: 10/27/2019].

Brooks RR, Wang K, Yu L, Oakley J, Skjellum A, Obeid JS, Lenert L, Worley C. Scrybe: a blockchain ledger for clinical trials. In: IEEE Blockchain in Clinical Trials Forum: Whiteboard challenge winner; 2018. p. 1–2.

Cachin C. State machine replication with byzantine faults. In: Replication. Springer; 2010. p. 169–84.

Cachin C, Vukolić M. Blockchain consensus protocols in the wild. arXiv preprint arXiv:1707.01873 2017.

Cassandras CG, Lafortune S. Introduction to discrete event systems. Springer Science & Business Media; 2009.

Castro M, Liskov B. Practical byzantine fault tolerance. In: Seltzer MI, Leach PJ, editors. In: Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22–25, 1999. USENIX Association; 1999. p. 173–86.

Delfs H, Knebl H, Knebl H, 2. Springer; 2002.

Douceur JR. The sybil attack. In: International workshop on peer-to-peer systems. Springer; 2002. p. 251–60.
Gervais A, Karame GO, Wüst K, Glykantzis V, Ritzdorf H, Capkun S. On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. ACM; 2016. p. 3–16.
Guin U, Cui P, Skjellum A. Ensuring proof-of-authenticity of IoT edge devices using blockchain technology. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE; 2018. p. 1042–9.
Hambolu O. Maintaining Anonymity And Trust: Applications of Innovative Data Strusctures. Clemson University; 2018.
Hoare CAR. Communicating sequential processes. Commun ACM 1978;21(8):666–77.
Idelberger F, Governatori G, Riveret R, Sartor G. Evaluation of logic-based smart contracts for blockchain systems. In: International Symposium on Rules and Rule Markup Languages for the Semantic Web. Springer; 2016. p. 167–83.
Katz J, Lindell Y. Introduction to modern cryptography. Chapman and Hall/CRC; 2014.
Kovalenko BGIN. Introduction to queueing theory. Israel Program for Scientific Translation, Jerusalem; 1968.
Lamport L. The part-time parliament. ACM Transactions on Computer Systems (TOCS) 1998;16(2):133–69.
Lamport L, Shostak R, Pease M. The Byzantine Generals Problem. In: Concurrency: the Works of Leslie Lamport; 2019. p. 203–26.
Liang X, Shetty S, Tosh D, Kamhoua C, Kwiat K, Njilla L. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID); 2017. p. 468–77. doi:10.1109/CCGRID.2017.8.
Lin I-C, Liao T-C. A survey of blockchain security issues and challenges.. IJ Network Security 2017;19(5):653–9.
Maria A. Introduction to modeling and simulation, 29; 1997. p. 7–13.
Medury S, Skjellum A, Brooks RR, Yu L. Scraaps: X. 509 certificate revocation using the blockchain-based scrybe secure provenance system. In: 2018 13th International Conference on Malicious and Unwanted Software (MALWARE). IEEE; 2018. p. 145–52.
Meyer P, Smythe R, Walsh J. Birth and death of markov processes, 3; 1972. p. 295–305.
Mukhopadhyay U, Skjellum A, Hambolu O, Oakley J, Yu L, Brooks R. A brief survey of cryptocurrency systems. In: Privacy, Security and Trust (PST), 2016 14th Annual Conference on. IEEE; 2016. p. 745–52.
Introduction to discrete-event simulation and the simpy language. Accessed 2019.
Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system.
Narayanan A, Bonneau J, Felten E, Miller A, Goldfeder S. Bitcoin and Cryptocurrency Technologies: AComprehensive Introduction. Princeton University Press; 2016.
Ongaro, D., Ousterhout, J., 2013. In search of an understandable consensus algorithm (extended version).
Pirretti M, Zhu S, Vijaykrishnan N, McDaniel P, Kandemir M, Brooks R. The sleep deprivation attack in sensor networks: analysis and methods of defense. Int. J. Distrib. Sens. Netw. 2006;2(3):267–87.
Ross SM. Introduction to Probability Models. Academic press; 2014.
Ryan P, Schneider SA, Goldsmith M, Lowe G. The Modelling and Analysis of Security Protocols: The CSP Approach. Addison-Wesley Professional; 2001.
Saaty TL, 34203. McGraw-Hill New York; 1961.
Salimifard K, Wright M. Petri net-based modelling of workflow systems: an overview. Eur J Oper Res 2001;134(3):664–76.
Schwartz D, Youngs N, Britto A, et al. The ripple protocol consensus algorithm. Ripple Labs Inc White Paper 2014;5:8.
Seber GA, Lee AJ, 329. John Wiley & Sons; 2012.
Taylor, A. E., 1955. Advanced calculus.
Vukolić M. The quest for scalable blockchain fabric: proof-of-work vs. bft replication. In: International Workshop on Open Problems in Network Security. Springer; 2015. p. 112–25.
Wikipedia contributors, 2020a. Decentralized application — Wikipedia, the free encyclopedia. [Online; accessed 9-March-2020].
Electronic signature. Accessed 2019.
Wikipedia contributors, 2020b. Geometric series — Wikipedia, the free encyclopedia. [Online; accessed 9-March-2020].
Wood G, et al. Ethereum: a secure decentralised generalised transaction ledger. Ethereum project yellow paper 2014;151(2014):1–32.
Xu JJ. Are blockchains immune to all malicious attacks? Financial Innovation 2016;2(1):25.
Technopedia,. Data lineage. Online Accessed 2018.
Worley, C., Yu, L., Brooks, R. R., Oakley, J., Hambolu, O., Skjellum, A., Altarawneh, A., Obeid, J. S., Lenert, L., Wang, K., et al.,. Scrybe: a 2nd-generation blockchain technology with lightweight mining for secure provenance and related applications. 2018.
Zinder, A., 2018. Systems and methods of secure provenance for distributed transaction databases. US Patent 10,097,356.
Zookeeper: A distributed coordination service for distributed applications. Accessed 2017.

Amani Altarawneh received her BA from Mu'tah University in Computer Science, Alkarak, Jordan and MS in CS from Bridgewater State University in MA, USA. She is currently a Ph.D. candidate in Computing Engineering/Computer Science at The University of Tennessee at Chattanooga (UTC), Tennessee, USA. Her research interests include Cybersecurity, queuing theory, IoT and smart cities, as well as distributed systems.

Fei Sun received the B.S. degree in Electronic Information Science and Technology from Xiamen University, Fujian, China. She is pursuing the Ph.D. degree in Computer Engineering from Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina. Her research interests include queuing theory, cybersecurity, connected vehicle communication.

Dr. Richard R. Brooks got his BA from Johns Hopkins University in Mathematical Sciences and Ph.D. in Computer Science from Louisiana State University. He was head of the Penn State ARL distributed systems department for seven years and is now a professor of Computer Engineering at Clemson. Dr. Brooks' network security research projects have included funding from NSF (analyzing denial of service), DoE (authentication and authorization), BMW Corporation (penetration testing), NIST (standards definition), AFOSR (timing side-channels) and the US State Department (creating anonymous communications tools). He finds attacks that disable security measures by working at a different level of the protocol stack. His Internet freedom work involves interactions with at risk populations working for freedom of expression. He has current work that looks at using the blockchain to secure metadata, securing connected vehicles, and making the smart grid robust to attacks.

Dr. Owulakemi Hambolu is a Senior Software Engineer/ Security Engineer at Dell EMC. She received BEng in Electrical Engineering from Ahmadu Bello University, Nigeria, MS in Computer Engineering from Clemson University, USA and Doctorate from Clemson

University, USA. Her research interests include Blockchain, cloud security solutions, and Security Development Lifecycle.

Dr. Lu Yu received the B.S. degree in Information Engineering and the M.S. degree in Control Theory and Control Engineering from Xian Jiaotong University, Xian, China, and the Ph.D. degree in Electrical Engineering from Clemson University, Clemson, South Carolina. She is currently a Research Assistant Professor with the Holcombe Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina. Her research interests include cybersecurity, privacy-preserving data mining, blockchain technology, and game theory.

Dr. Anthony (Tony) Skjellum studied at Caltech (BS, MS, PhD). His Ph.D. work emphasized portable, parallel software for large-scale dynamic simulation, with a specific emphasis on message-passing systems, parallel nonlinear and linear solvers, and massive parallelism. From 1990-93, he was a computer scientist at LLNL focusing on performance-portable message passing and portable parallel math libraries. From 1993–2003, he was on the faculty in Computer Science at Mississippi State University, where his group coinvented the MPICH implementation of the Message Passing Interface (MPI) together with colleagues at Argonne National Laboratory. From 2003–2013, he was professor and chair at the University of Alabama at Birmingham, Dept. of Computer and Information Sciences. In 2014, he joined Auburn University as Lead Cyber Scientist and led R&D in cyber and High-Performance Computing for over three years. In Summer 2017, he joined the University of Tennessee at Chattanooga as Professor of Computer Science, Chair of Excellence, and Director, SimCenter, where he continues work in HPC (emphasizing MPI, scalable libraries, and heterogeneous computing) and Cybersecurity (with strong emphases on IoT and blockchain technologies). He is a senior member of ACM, IEEE, ASEE, and AIChE, and an Associate Member of the American Academy of Forensic Science (AAFS), Digital & Multimedia Sciences Division.