Received April 30, 2020, accepted June 14, 2020, date of publication June 30, 2020, date of current version July 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3006113

Security Properties of Light Clients on the Ethereum Blockchain

SANTERI PAAVOLAINEN[1] AND CHRISTOPHER CARR[2,3]
[1]Department of Communications and Networking, School of Electrical Engineering, Aalto University, 00076 Aalto, Finland
[2]Department of Information Security and Communication Technology, Norwegian University of Science and Technology, 7491 Trondheim, Norway
[3]Department of Accounting, Economics and Finance, University West of England, Bristol BS16 1ZG, U.K.

Corresponding author: Santeri Paavolainen (santeri.paavolainen@aalto.fi)

This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant 779984. The work of Christopher Carr was supported in part by the Norwegian University of Science and Technology, Trondheim, Norway.

ABSTRACT Ethereum is a decentralized blockchain, known as being the second most popular public blockchain after Bitcoin. Since Ethereum is decentralised the canonical state is determined by the Ethereum network participants via a consensus mechanism without a centralized coordinator. The network participants are required to evaluate every transaction starting from the genesis block, which requires a large amount of network, computing, and storage resources. This is impractical for many devices with either limited computing resources or intermittent network connectivity. To overcome this drawback Ethereum defines a light client protocol where the light client fetches the blockchain state from a node operating as a light protocol server. Light clients are unable to maintain blockchain state internally, and as a consequence can only perform partial validation on blocks. Thus they rely on the light server for full block validation and to provide the updated blockchain state. Light clients connect to multiple light servers to mitigate the risk of relying on a single potentially dishonest server. Ethereum light clients are known to suffer from a probabilistic security model, but they are widely assumed to be secure under *normal* operating conditions. In fact, the implicit security assumptions of light clients have not been formally characterised in the literature. We present and analyse the probabilistic security guarantees under three different adversarial scenarios. The results show that for any adversary that is able to manipulate the network, the security assurances provided by the light protocol are severely impacted, and in some cases entirely lost. These results clearly demonstrate that the assumption of normal operating conditions is insufficient to justify the security assumptions of light clients. Our work also provides insight to the security of light clients under different security parameters, allowing light client implementers to more accurately understand the potential security trade-offs.

INDEX TERMS Blockchain, ethereum, light client, light ethereum subprotocol, security.

I. INTRODUCTION

The Ethereum blockchain is a well-known second-generation blockchain technology [1]. In contrast to earlier blockchain technologies, such as Bitcoin [2], Ethereum has a far shorter block interval — the period between state transitions — and allows for expressive smart contracts. Smart contracts are programs whose program code and execution state are stored on the blockchain. Ethereum has *accounts* as explicit entities, in contrast with Bitcoin where transactions are referred to as unspent transaction outputs — called UTXOs. The two approaches are distinguished as *account-centric blockchain* and *transaction-centric blockchain* models by Ren and Erkin [3]. They observe that in both blockchain models, consensus requires that *all nodes* in the network can reliably acquire and compute the state transition function. This requirement for consensus becomes a critical issue when attempting to connect resource constrained devices to the Ethereum network.

The Ethereum network is a distributed set of computer-participants called *nodes*, which have significant resource requirements. A node containing the full block history requires hundreds of gigabytes of storage. Even a node that discards much of the historical data still needs gigabytes of available storage [4]. For devices with limited storage space this may already inhibit them from participating in the network consensus protocol.

In addition to storing the blockchain history and its current state, a node must validate and process incoming blocks. Such nodes are also called called *validating nodes* or *full nodes*.

The associate editor coordinating the review of this manuscript and approving it for publication was Wenbing Zhao.

For a node to apply the full consensus protocol, it needs to have sufficient bandwidth and computational capacity. We call a *constrained device* any device that lacks the resources to operate as a consensus-protocol following node. For example, most mobile devices would fall into the constrained device category, as well as most Internet of Things (IoT) devices including consumer and industrial embedded applications. Devices which may have the required storage and processing capacity, but are lacking in either network connectivity (intermittent connectivity) or availability of power (battery-powered devices including vehicles) are also considered to be constrained in this study.

To address the problem of constrained devices many blockchains define *light protocols* used by *light clients*. Bitcoin defines the Simple Payment Verification (SPV) [2] protocol and Ethereum has the Light Ethereum Subprotocol (LES) [5]. These offload parts of the blockchain consensus and state management protocol to *light protocol servers*. In general, light clients are able to quickly identify the current canonical chain, retrieve and validate block headers, and query the light servers for further information such as transactions and blockchain state. While these light protocols are designed for space-efficient and secure data retrieval, they have the potential to introduce new attack vectors.

Light clients are reliant on light servers to both process blocks and keep them informed of the current state. Unlike full nodes, light clients cannot determine if a block it receives is based on, or describes operations that would result in an invalid blockchain state. They are effectively stateless and rely only on information from block headers to determine the true chain. This creates a potential attack vector for light clients, making them more vulnerable to entities who may wish to deceive them. In reality, some implementations do store blockchain data so that it is not a stateless process. However, this behavior is not part of the Ethereum specification and is not a required behavior of a conforming light client. Moreover, even storing some previous state may not prove a reliable countermeasure. Consequently light clients are susceptible to attacks where a malicious entity gains a substantial but not necessarily a majority of computational power even for a short period of time. This vulnerability is compounded when availability of local full nodes is decreased, or a greater percentage are malicious, such as in network partitioning attacks. Notably, the network may be partitioned even under normal conditions without any action specifically targeting the network. For example, the Internet may suffer from routing problems [6] or national security apparatus may temporarily block Internet access in a large region [7], [8].

It has been previously shown that Ethereum light clients achieve only probabilistic security assurances.[1] However, the overall security is considered to be high under normal operating assumptions, where "normal" in this case means that there is complete availability to each node. Al-Bassam *et al.* [11] write that light clients operate "well under normal circumstances", and would suffer security degradation only if a majority of the consensus controlling nodes (i.e. miners and full nodes) collude. Leiba *et al.* [12] similarly state that light clients are secure against dishonest servers when the majority of miners are honest and the light client is able to connect to at least one honest node. Many other works have noted these assumptions or demonstrated the weaker security assurances offered for light clients [13]–[16], but have not formally characterised the security of light clients to the extent our work provides.

[1] The consensus protocol itself has a probabilistic finality property, see [9], [10], but for simplification we describe it as fully deterministic in the absence of hostile activities.

Despite the general understanding of light client's security requirements, there is no comprehensive and formal description of the properties of light clients on an Ethereum network. This article addresses that omission. We describe the behavior of the Light Ethereum Subprotocol (LES), describe formally the common security assumptions, and evaluate the probability of a successful adversarial injection of an incorrect block under three different attack scenarios. The focus is solely on the Ethereum network, however many aspects of this analysis are valid for other account-centric blockchains.

This work takes a deep look at Ethereum light clients and their potential for malicious compromise. The contributions this article makes are:

- A formal definition of adversary's goals and capabilities, and the different attack scenarios we model (Section IV).
- A definition of different Markov processes to model the adversary's probability of success under different attack scenarios (Section V).
- Results assessing the security of a light client under the different attack scenarios under different relevant security parameters (Section VI).

We also discuss related work in Section II, with Section III covering background information on the Ethereum blockchain and light clients. Conclusions and discussion can be found in Section VII. Additionally Appendix A provides examples of how the adversary could exploit a light client, and Appendix B provides a detailed description of the Markov process matrix construction.

II. RELATED WORK

The interplay between honest miners and a miner attempting in some manner to subvert the blockchain mining process has previously been addressed by at least Eyal and Sirer [17], Nayak *et al.* [18], Sapirsthein *et al.* [19], and Gervais *et al.* [20]. Their work focuses on strategies that can be employed by a miner to maximize their mining rewards through strategic withholding of mined blocks, and on how to potentially exploit this benefit thereafter. While the different mining strategies described in the papers may break the intended purpose of mining rewards—i.e. to incentivize decentralized miners to reach rapid consensus—miners in these papers do nonetheless follow the consensus protocol correctly and attempt to get the mined blocks to be accepted as part of the blockchain network. This is in contrast to our work, where the adversary never has the intention to generate blocks that the network would accept.

One can distinguish between attempts to manipulate the blockchain consensus mechanism to one's benefit—as above—and attacks against a specific user or groups of users while they transact on the blockchain. This includes attacks such as the double-spend attack, Finney attack, and others—see [21], [22] for a summary. It is also possible to target blockchain nodes directly, for example using eclipse attacks [23]–[25] to manipulate and even isolate blockchain nodes from other nodes. While most of the results related to eclipse attacks against blockchain nodes apply only to Bitcoin, it is interesting that Marcus *et al.* specifically observe that Ethereum network nodes are more susceptible to eclipse attacks than Bitcoin nodes [24].

The challenges of constrained devices to blockchain integration is manifold and complex field. Some approaches look for mechanisms that allow the light client to reduce the amount of data that has to be transmitted. Kiayias *et al.* introduce Non-Interactive Proofs of Proof-of-Work (NIPoPoWs) [26] that allow up to 90% reduction in the block headers that need to be transmitted over the network. Bünz *et al.* further extend the proposed mechanism to light clients on both Bitcoin and Ethereum [27]. Danzi *et al.* measure bandwidth requirements of blockchain clients, and propose radio link layer aggregation [14] and multicasting schemes [28] to reduce bandwidth requirements. Other similar approaches have been taken by Palai *et al.* [15], Pustišek *et al.* [29], for example.

While integrating IoT devices with popular public blockchains such as Ethereum and Bitcoin is often desirable, another approach is to turn the problem around, and make the integration easier by designing the blockchain technology itself from start to be more IoT-friendly. Using alternative Sybil protection mechanisms such as staking in Tendermint [30] can change the dynamics of the validation process, making it easier for IoT devices even when this has not been the primary goal of the blockchain design. Blum and Bocek have proposed a blockchain specifically designed for light clients [31]. If public blockchain interaction is required, another approach is to use multiple blockchains, and segregate constrained devices into the more IoT-friendly blockchain, straddling the two blockchains using interledger techniques [32].

Al-Bassam *et al.* accurately identify the security problems posed for light clients, and consider a situation where the majority of the consensus is colluding against light clients [11]. Their approach is to have nodes employ a gossip protocol, distributing *fraud proofs*, i.e. proofs that are easily verifiable by a constrained device to demonstrate a block to be malicious, allowing light clients to avoid exposure to invalid block state even without the need to fully validate all blocks. In their case the fraud proofs are assumed to be identified by honest nodes. This is in contrast to our scheme, where the invalid blocks are not sent to the honest network (thus, no honest node could generate a fraud proof against them). We do not assume a dishonest majority, either.

While not specifically a protocol problem, Gruber *et al.* remark that "full nodes only have little incentives to (correctly) serve lightweight clients" [33]. This seems like a grave omission in the light client model—miners and full nodes themselves gain benefit from propagating blocks and headers to other *full nodes*, but they gain nothing from providing light protocol services. This leaves the light servers open for other incentives which may not be detrimental to light clients.

There are potential mitigation methods that are applicable to light clients in particular. For example, Marcus *et al.* describe countermeasures an Ethereum light client can take to protect itself against adversary attempting to eclipse it from the honest network [24]. Paavolainen and Nikander [34] propose a mechanism where the owner of an IoT device sends periodic decentralized beacons to the device attesting the owner's view of the canonical blockchain, allowing the device to determine a trusted blockchain state independently of the light protocol servers' intent.

III. ETHEREUM BLOCKCHAIN

Ethereum is a public and decentralized blockchain technology, operationally deployed as a peer network[2] [1]. Any node following the defined peer-to-peer protocol can join the network, retrieve historical blocks and transactions, submit transactions to the network, and if desired, act as a miner and propose new blocks to the network. Since the Ethereum network uses a cryptographic proof-of-work as a Sybil attack protection measure, this requires miners to invest resources in solving the cryptographic puzzle. To aid understanding, Table 1 lists the notation used within this section, and throughout.

A. BLOCK STRUCTURE

The block header of Ethereum is described in Fig. 1, with the fields that are referred to in the text in bold. Ommers (or uncle blocks), transactions, and receipts are not part of the block header, but are considered part of the block, and are transmitted separately. Unlike other fields, the block state is only inferred from the block, and is not explicitly defined, or transported, as part of the block. The majority of the fields are not relevant to this article, and we refer the reader to the Ethereum documentation [1] for further information. As in many other blockchains, the block header refers to the previous block by the hash of that block's header, termed the *parent hash* in Ethereum. We refer to the block header as B, and for a specific block n as B_n. The block header is often referred by the hash of the block header, $h_n = hash(B_h)$. Since any change in the information contained within the

[2]There are different networks using the same technology, such as Ropsten and Rinkeby test networks. "The Ethereum network" refers to the "main" network recognized by the cryptocurrency identifier ETH. This network is sometimes called the "homestead" network. Confusingly enough, "Ethereum" refers also to the technology in general.

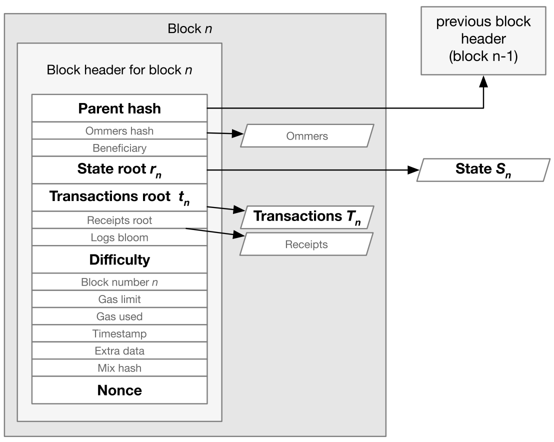

FIGURE 1. Ethereum block header diagram.

block header will result in a change of its hash output, this creates a hash chain from any block n to the starting block, called the *genesis block*, block 0.

A block contains several data structures that are referred to from the block header, but are not part of the block header itself. For example, the list of transactions T_n is transferred separately, and is referred to from the header by the hash of the root node of the transaction trie. The trie is deterministically populated from the transactions list, and consequently $t_n = hash(T_n)$ authenticates the set of transactions T_n. Therefore the list of transactions in a block cannot be altered, as any such change would result in a different value for the transaction root hash.

The block header also contains a hash of the system state $r_n = hash(S_n)$, which is is the root hash of the Merkle-Patricia trie of all account states. The system state S_n is not explicitly transmitted between (regular) Ethereum network nodes, and is instead maintained separately by each participating node. Thus the state S_n is considered implicit, and not considered to be directly part of the block n.

B. MINING

A cryptographic puzzle is easy to verify, but difficult to solve. The Ethereum blockchain uses a cryptographic puzzle as a protection against Sybil attacks, called a proof-of-work puzzle. The puzzle requires the hash of the *block header* to meet specific requirements. A miner can generate variations in the block header by changing the order of transactions, or more commonly, by changing the *nonce* field in the block header. Once a miner can generate a block header that meets the requirements, it broadcasts the block header through the peer-to-peer network of participating nodes.

More formally, the goal of the miner is to be able to generate a provisional block B'_{n+1}, distribute it to the network, and have it become part of the honest blockchain as block B_{n+1}. We use $'$ to indicate that the value is provisional and the corresponding block has not yet been accepted as part of the honest chain.

The miner starts by determining the current head of the chain B_n, i.e. finding the block that along the whole chain

TABLE 1. Notation used in this article. See Sections III–IV for details.

Symbol	Description
B_n, $\tilde{B}_m$	The block header for block n in the honest chain, and for block m in the adversarial chain, respectively
T_n, $\tilde{T}_m$	The transactions of honest block n and adversarial block m
S_n, $\tilde{S}_m$	The blockchain state including account information, smart contracts etc. in the honest chain block n and adversarial chain block m
B, $\tilde{B}$, T, ...	Reference to a general honest block, adversarial block, transactions, etc.
B', T', S'	Provisional block (transactions, state) that has not yet become part of the chain
$hash(\cdots)$	A cryptographic hash of some data or field in the Ethereum protocol—the specific hash mechanism is dependent on the context, but for simplicity, we do not make these differences explicit
$h_n = hash(B_n)$	Hash of the block header B_n
$r_n = hash(S_n)$	Root hash of state S_n from block n
$t_n = hash(T_n)$	Root hash of transaction trie T_n
$\Pi(S, T)$, $\tilde{\Pi}(\tilde{S}, \tilde{T})$	The state transition function, where Π follows Ethereum's function correctly whereas $\tilde{\Pi}$ may not
λ	The transition rate in the Markov process for the adversary successfully mining a block
$\mu = (1-\gamma)\mu_0$	The transition rate for the visible portion $1-\gamma$ of the complete honest network μ_0
γ	Amount of partitioned honest miners inaccessible from light client
$\mu_0 = 1$	The total mining power of all honest miners, normalized to unity
$\mathcal{V}$, $\mathcal{V}_h$, $\mathcal{V}_s$	The full validation, header validation and state validation functions, respectively
ΔW	Difference between the amount of "work" in adversarial and the honest chain
k	The block depth parameter used by a light client

to the genesis block B_0 accumulates the largest value for the work function[3] W. The next step is to determine the set of transactions T'_{n+1} to include in the block. This set of transactions is then used to update the blockchain state as described in (1) (slightly simplified—see [1] and [35] for a complete description). This defines a transition from an earlier state S_n as a function of the state transfer function Π, and executing the set of transactions T'_{n+1}, resulting in a new state S'_{n+1}:

$$S'_{n+1} = \Pi(S_n, T'_{n+1}). \tag{1}$$

The miner encodes the necessary information of the new state S'_{n+1} and transactions T'_{n+1} as r'_{n+1} and t'_{n+1}, includes them in the block header B'_{n+1}, and checks if the hash of the block header $hash(B'_{n+1})$ is a valid solution to the cryptographic puzzle. If it is not, the miner will modify the nonce in the block header, and repeat until it either succeeds, or it

[3]While Bitcoin selects the chain with most blocks, Ethereum has a more complex selection policy based on the total amount of work included in the chain, including work in proposed, but eventually abandoned blocks aka uncle blocks or ommers.

detects a competing block header B_{n+1} broadcast by another miner.

If the miner succeeds, it will broadcast its block header B'_{n+1} to the network. If there are no competing block headers, or the miner's block "wins" the competition, it will become committed to the chain, so $B_{n+1} = B'_{n+1}$. At this point it can also be said that at block $n + 1$ the network's state is $S_{n+1} = S'_{n+1}$ as originally calculated by the miner, thus forming a consensus on the state of the network.

C. MINING NETWORK

We define the network that is correctly following the Ethereum consensus rules, as described by the Ethereum protocol, as the *honest network*. This portion of the network consists of *honest miners*, *honest validating nodes*, and *honest light protocol servers*. In contrast to the adversary, as defined in Section IV-A, the honest network is not attempting to subvert any node on the network.

While the probability of successfully mining a block for any single miner is very small, the total probability of mining success over a large number of miners may be modelled using an exponential distribution [17], [18], [36]. Therefore we can identify the honest network's capacity to produce new, valid blocks as an exponential distribution $Exp(\mu_0)$, with the mean interval between two successfully mined blocks as μ_0^{-1}. For simplicity, we set $\mu_0 = 1$, and fix the time scale so that the time expressed is abstracted to $[t] = [\frac{1}{\mu_0}] = 1$.

The mining network may be *partitioned* — split into at least two distinct and mutually exclusive networks. We define the amount of partitioning as $\gamma \in [0, 1]$ from the viewpoint of some arbitrary light client, so that in normal operating conditions when no partitioning occurs $\gamma = 0$. Without loss of generality we define a partitioned network as a network split into two parts where: $\gamma\mu_0$ hashing power is currently unreachable from the from the light client, and $\mu = (1-\gamma)\mu_0$ is in the portion of the network that is reachable.

D. BLOCK VALIDATION

Any node receiving a new block header can directly verify that the block header itself is valid. We denote this verification process as $\mathcal{V}_h(B_n)$. This process ensures both the structural correctness of the block header (length, field values, etc.), and more importantly checks that the hash of the block header meets the requirements of the cryptographic puzzle. Since the block header contains references to auxiliary information not included in the block, such as the hash of the transaction trie t_n, a node can verify that a list of transactions is part of a block. The transaction list validation is an another validation that a node may perform, but for the purposes here, we can assume that $\mathcal{V}_h$ represents all validations that can be performed based on the block header and associated data with modest resource requirements.

A node can determine that the system state referenced in the block header as r_n is correct by evaluating the blockchain state transition function itself $S'_n = \Pi(S_{n-1}, T_n)$, and

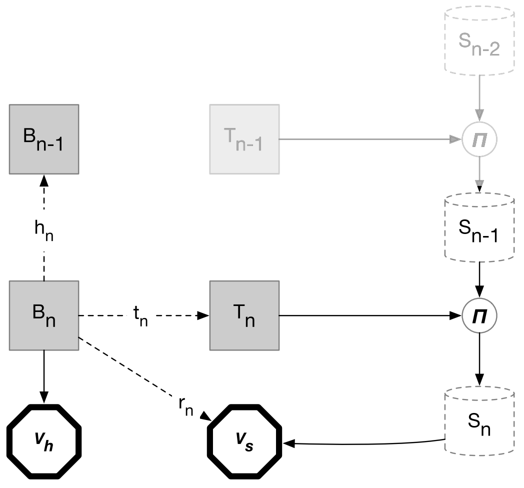

FIGURE 2. The information used in block validation for the block header validator $\mathcal{V}_h$ and the block state validator $\mathcal{V}_s$.

verifying that $hash(S'_n) = r_n$. We denote this verfication as $\mathcal{V}_s(S_{n-1}, B_n, T_n, \ldots)$, and the full set of checks as $\mathcal{V} \equiv \mathcal{V}_h \wedge \mathcal{V}_s$, as shown in Fig. 2.

The state S_n for any n is not explicitly transmitted as part of the Ethereum protocol—transmitting it would defeat the whole goal of decentralized security model where each node relies only on the state it has independently validated. The inclusion of the root hash of the Merkle-Patricia trie construction of the current blockchain state, r_n in the block header does, however, allow any node to verify that they share the same view of the blockchain state as the mining node. As described earlier, the state information S_n required to compute r_n can be several gigabytes in size, and requires processing power and network bandwidth to maintain. Consequently, along with miners and validating nodes there is a third network participant, commonly called a *light node* or a *light client*, that is unable to perform state validation $\mathcal{V}_s$. In summary, the three nodes types are:

Miners that are able to generate new blocks B_{n+1} that pass the full set of validations $\mathcal{V}$. To do this, they validate other incoming blocks to identify the canonical chain and maintain the state S.

Validating nodes perform all validations $\mathcal{V}$ on blocks arriving from other nodes and reject invalid blocks. Validating nodes maintain the state S.

Light clients communicate with other nodes to receive information on new blocks. They will query light protocol servers for any blockchain state they need. Light clients can perform header validation $\mathcal{V}_h$, but not state validation $\mathcal{V}_s$.

While miners and validating nodes form a peer-to-peer network where all nodes are peers (miners are different only in internal behavior, not external), light clients rely on some of the nodes to act as a intermediary to the light client. This client-server model defines a protocol differing from the normal peer-to-peer protocol, a *light client protocol*, described below.

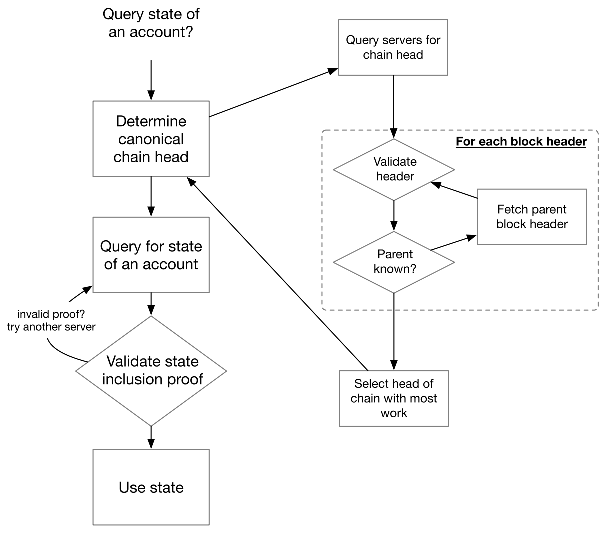
Query state of an account?
Determine canonical chain head
Query for state of an account
invalid proof? try another server
Validate state inclusion proof
Use state
Query servers for chain head
For each block header
Validate header
Fetch parent block header
Parent known?
Select head of chain with most work

FIGURE 3. **Overview of the behavior of a light client using the Light Ethereum Subprotocol.**

E. LIGHT CLIENT PROTOCOL

Light nodes can still perform $\mathcal{V}_h$ validation. This allows them to identify the *canonical chain* i.e. the chain with the largest amount of work that forms a unbroken chain of blocks to the genesis block B_0. This chain is verified by the light client to follow the consensus protocol for $\mathcal{V}_h$ validation. Additionally, while the light client is unable to calculate S_n on its own, it can still use the inclusion of the state root r_n to validate a *proof of inclusion* for any account in the full blockchain state. The proof of inclusion is a subset of the full Merkle-Patricia state trie, with a path starting from the root and descending to the subset of state requested by the client. The light node can retrace the path up to the root node and verify that the root hash matches r_n from the target block. In practice, this means that assuming the client has the canonical block header, it can authenticate all state responses from light servers, and reject any responses that do not match the established consensus on the canonical chain.

When a light node uses another node for state management and retrieval, it is called a *light client*. It operates a *light protocol* to communicate with the full node, and is known as a *light server*. This mechanism is general to other types of blockchains, such as SPV for Bitcoin. For Ethereum it is known as the Light Ethereum Subprotocol (LES) [5].

An overview of the process of a light client using LES protocol is shown in Fig. 3. If the client wants to identify the balance of a specific account, it first identifies the canonical chain by performing header validation on all block headers it receives, and identifies the chain with the largest amount of work. Next, the client queries the server for the target account information at some predetermined block depth k. If the canonical blockchain head is B_n, then the client is interested in the state from block $n-k$. The server replies with the specific account state data, and the proof of inclusion as the required hashes of the Merkle-Patricia tree. The client calculates the hash of the retrieved subset of the state data, and ascends up the inclusion proof until it reaches its own root hash r'_{n-k}, which it then compares against the r_{n-k} from the block header B_{n-k}.

Since the value of k is controlled by the client, it can be tuned to different values depending on the client's requirements. For deciding on a suitable k value the general guidance is to consider overall security requirements and risks [37]. For example, if the light client needs to quickly act on a state change (such as a payment transaction), then it has to use a lower k value. Similarly, if the client does not have to react to rapid changes it can safely use a higher k value. As an upper bound $k = 30$ seems to be a reasonable choice since some cryptocurrency exchanges use it before processing real-world money transfers based on deposits on the Ethereum blockchain [38].

A light client is assumed to contact a large number of light servers, potentially chosen at random. The number N of servers the client connects to is determined by the client. The client may receive information about different chain heads from different servers. This may happen if there are forks in the blockchain for instance. However, since the client can independently determine the total amount of work each chain represents, it can be assumed to be able to determine the canonical chain. It is expected that the honest chain — in the long run — will always contain the most amount of work. Consequently the common assumption is that the canonical chain, representing the largest amount of work is also the honest chain. Note that the client *does not determine the canonical chain by voting*—if it receives just one copy of the chain with the most work, it will consider that chain to be the canonical chain even if all other servers it communicates with provide an alternate chain.

F. ETHEREUM NETWORK CHARACTERISTICS

For evaluation purposes we need to note some characteristic features of the Ethereum blockchain network that is often also referred to as the *homestead* network. The mean block interval in this network varies slightly (μ_0^{-1} in our notation), although it is commonly approximated as 15 seconds.

The capacity of the Ethereum network to determine whether a particular block header meets the required properties of the cryptographic puzzle is called *hashrate*. Since hashrate is a way of describing the speed of block creation, sometimes the terms *mining power*, *hashing power* etc. are used to express the same idea. The main Ethereum network—as of writing of this article—has an aggregate hashrate[4] of 180×10^{12} per second (terahashes/s, TH/s). For comparison, a high-end GPU card[5] attains a hash rate of 88 MH/s representing less than one millionth of the hashrate of the full network. Since the likelihood of successfully finding a new block for a single GPU-based miner is very low, most miners join so-called *mining pools* which aggregate its

[4]https://www.etherchain.org/charts/topMiners
[5]https://www.hashrates.com/gpus/

members' hashrate and divide mining rewards in proportion to the contribution of each member. At the time of writing the largest single Ethereum mining pool controls about 30% of the total hashrate, the 10th biggest pool has 1.4% of the total mining power, and the top 10 mining pools hold in aggregate over 80% of all hashing power.

IV. THREAT MODEL AND ATTACK SCENARIOS

Next we consider the general threat model and three different attack scenarios against which we evaluate the vulnerability of a light client on the Ethereum network. First we describe the adversary and its goals and capabilities, explaining what the adversary needs to accomplish in relation to the honest network to succeed. Then, three different attack scenarios are described that provide more specific context on the behavior of the light client and the adversary.

A. ADVERSARY

We assume an adversary can mine blocks at a mean rate λ^{-1}, i.e. its relative mining power in relation to the honest network is λ/μ_0. The adversary also controls a set of dishonest light protocol servers which are not affected by any network partition $\gamma > 0$. For a summary of the configuration regarding a light client's network environment please see Fig. 4. In summary, the adversary controls an *adversarial network* consisting of *dishonest miners* and *dishonest light protocol servers*.

In our model, the adversary is targeting a specific *target node* that exclusively uses the light protocol to interact with the Ethereum network. The specific goal of this adversary is to inject an *adversarial block* to the target, and have the target consider the injected block part of the canonical chain. Consequently, the target may change its behavior in a manner that is exploitable by the adversary. For a more detailed discussion and examples on how the adversary can use an injected adversarial block to their advantage, see Appendix A.

B. ADVERSARIAL CHAIN

The adversary generates a fork in the blockchain—the adversarial chain—that does not have to conform to the Ethereum state transfer rules. Therefore the adversary is permitted to generate blocks that fail state validation. The invalid blocks would be rejected immediately by the honest network, but this does not concern the adversary as it does not have any need to distribute the invalid blocks to the honest network. Instead, the adversary wants to inject the invalid blocks to the targeted light client so that it will use the *adversarial state* $\tilde{S}_n$. This state is not consistent with the state transformation rules of the honest network:

$$\tilde{S}_n = \tilde{\Pi}(S_{n-1}, \tilde{T}_n) \wedge \tilde{S}_n \neq \Pi(S_{n-1}, \tilde{T}_n). \tag{2}$$

The state is encoded in the adversarial block $\tilde{B}_n$ that, due to (2), would fail the state validation $\mathcal{V}_s$. This deviation is however, undetectable by the light client as by definition, it is unable to perform state validation. The adversarial block

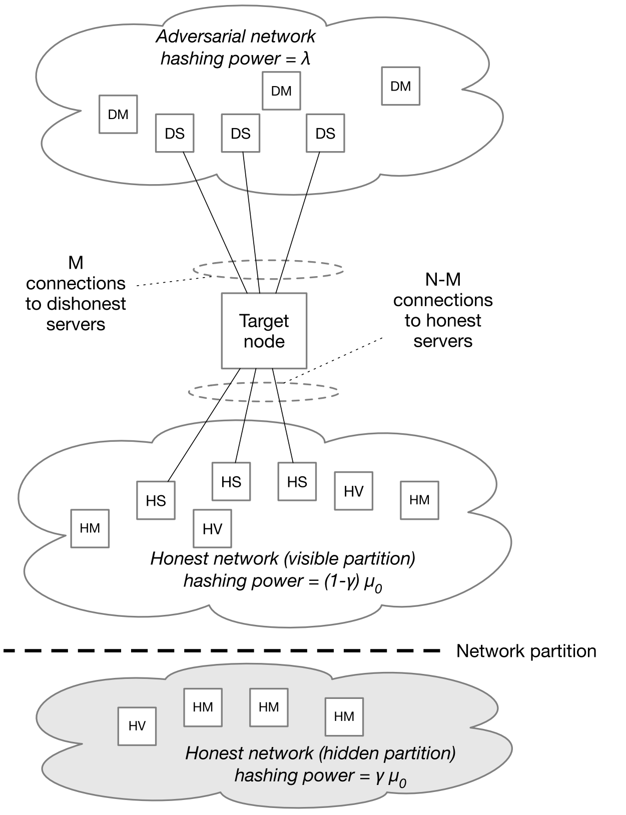

FIGURE 4. Scenario with adversarial and honest network, and a target node. A target node with honest and adversarial networks, including partitioning. DM = dishonest miner, DS = dishonest server, HM = honest miner, HV = honest validating node, HS = honest server.

must, however, still pass the $\mathcal{V}_h$ validation at the client, i.e. it must contain a valid proof-of-work.

The *adversarial chain* is the sequence of blocks mined by the adversary $\left(\tilde{B}_{n-m}, \ldots, \tilde{B}_n\right)$. The adversarial chain consists of blocks that would not be accepted to the honest chain as they fail state validation. We similarly identify adversarial transactions of block n as $\tilde{T}_n$, the adversarial state $\tilde{S}_n$, and the resulting state root hash $\tilde{r}_n$. In our analysis the state transfer function $\tilde{\Pi}$ does not necessarily have to follow the defined Ethereum state transfer rules, i.e. $\exists S, T : \Pi(S, T) \neq \tilde{\Pi}(S, T)$.

For the adversary to successfully manipulate the client's view of the network state it must generate a chain that the light node would accept as the canonical chain and reply to the client node's state requests with extracts from the adversarial state $\tilde{S}$. This implies the following three requirements:

R1. The client must connect to a light protocol server that the adversary controls;

R2. An adversary must be able to convince the client to recognize the adversarial chain as the canonical chain. This requires that the work function of the adversarial chain is larger than that of the honest chain;

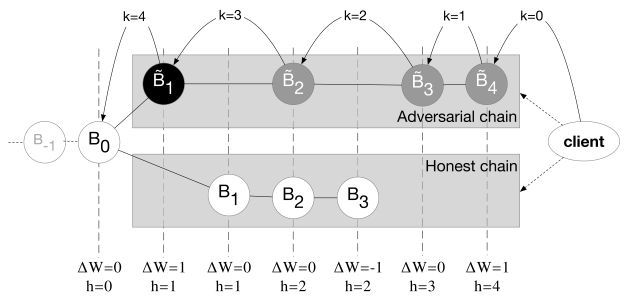

FIGURE 5. Conflict between honest and adversarial chains with respect to the client. B_0 is the block the adversary forks its chain from (not necessarily the genesis block).

R3. The adversarial chain must be deep enough so that the client with depth parameter k will use an adversarial block $\tilde{B}_{n-k}$ instead of an honest block B_{n-k}.

This process is demonstrated in Fig. 5 where the adversary has forked the honest chain at block $n = 0$. The vertical lines represent samples in time. The ΔW is the difference of work between the honest and adversarial chains; h is the height of the adversarial chain from the fork. The lines from the client represent the block the client would be using at different block depth k values. When the client is presented with both chains, it would select the adversarial chain if $\Delta W > 0$ and select the honest chain when $\Delta W \leq 0$.

As an example, the adversary in Fig. 5 has initially succeeded in mining a block $\tilde{B}_1$. This results in the adversary having a parallel chain of height $h = 1$ and the difference in the work function of the two chains is favourable to the adversary with $\Delta W = 1$. Since the adversarial chain contains more *work* than the honest chain, a light client would accept the adversarial chain. However, when the honest chain mines B_1, the work difference between the two chains decreases to zero and the adversary cannot be certain that its chain would be selected. In our model we conservatively assume that ties are always resolved to the honest network.

Moreover, the adversary would not necessarily succeed at block $\tilde{B}_1$ if the block depth parameter k of the client is over zero. For example, with the final state in the diagram where the adversarial chain is at $\tilde{B}_4$ and the honest chain is at B_3 i.e. $\Delta W = 1$ and $h = 4$, the client using $k = 4$ would bypass all of the adversarial blocks and use state from the honest block B_0, with the adversary consequently failing. Conversely, if $k < h \wedge \Delta W > 0$ the adversary would be able to provide adversarial state $\tilde{S}$ from its own block and succeed. Our goal is to determine the probability of a a success for an adversary.

C. ATTACK SCENARIOS

While the goal of the adversary is to inject its own block to the target node, the general context of the attack affects the timing and constraints of the attack. IoT devices and other types of light clients vary significantly in their capabilities and how they are deployed. How the device operates on the blockchain affects the type of attacks an adversary can initiate. We have decided to condense these variations into three different attack scenarios A1–A3 that we believe are useful abstractions of real-world attacks.

A1. The target node performs one read-only operation on the blockchain at unpredictable time t_0, whereby it reads state from the blockchain at depth k. The goal of the attacker is to ensure the state that is used is obtained from an adversarial block.

A2. The target node connects to the network at time t_0 that is not known to the attacker *a priori*, and disconnects at time $t_1 = t_0 + \Delta t$. Unlike in the A1 scenario, the adversary needs to inject state that is dependent on information available at t_0, thus forcing the adversary to generate a fresh fork. The adversary has Δt time to successfully construct an adversarial chain that the target accepts.

A3. Similar to A2, but the client does not have a natural timeout (t_1 is unbounded). Hence the adversary has indefinite time Δt for the injection attack.

For all scenarios A1–A3, we assume the adversary needs to inject only one invalid state that is independent of later actions by the adversary or the target node. We define this as *1-inject attack*.

Since these attack scenarios are general, let us describe potential examples where they would be applicable and the questions we want to answer. The A1 scenario is a stationary scenario, where the adversary needs to inject information known *a priori*, while the client connect time t_0 is unknown. Upon establishing a connection, the client will perform only a single state check, i.e. Δt is zero or very close to it. A simple but practical example of this case would be someone attempting to falsely convince someone else that they have a 1-million ether balance. The adversary has to constantly maintain an adversarial chain that it has to present at any time that client chooses. This leads to the first question:

Q1. What is the probability of the adversary of having a successful adversarial chain at any randomly chosen point in time?

A battery-powered device provides an example of the A2 scenario. The device periodically connects to the network remaining connected until the battery completely drains. If the battery is charged by solar power, the next reconnect time can not be accurately predicted. This scenario is also applicable to other situations with a natural timeout where the attack fails if no adversarial block is injected within the time limit Δt. A merchant waiting for commitment on an Ethereum payment would become suspicious after an excessive delay, for example. Specifically, in the A2 scenario an adversary is not able to "pre-prepare" the attack, and they must start creation of the adversarial chain at t_0. This could occur because the adversary needs to inject a specific identifier to the blockchain state, which must be first obtained from the client. This raises the second question:

Q2. What is the adversary's probability of 1-inject success within Δt time?

Finally, if the client either does not have a timeout at all ($t \to \infty$), or the adversary is able to control t_1 by choosing the time of attack when they have a successful adversarial chain, the adversary will *eventually* always succeed. For example, if the target is a smart lock which reads the list of allowed key card identifiers from a smart contract, the attacker can physically wait near the target, and attempt a break-in only when an acceptable adversarial chain has been achieved. The relevant question is not the probability of success, but of the time required, leading to the third question:

Q3. What is the expected time needed for the adversary to gain a specific success probability?

For all three scenarios the adversary may be able to achieve $\gamma > 0$ by exploiting naturally occuring Internet outages. We can assume that some honest mining capacity remains, however not all: $0 < \gamma < 1$. If an adversary is able to perform active attacks, they may be able to completely isolate a client from all honest servers resulting in $\gamma = 0$.

In practice, isolating miners from each other is difficult due to their large geographic dispersion. An adversary with physical or proximate access to the target may be more successful in isolating the client from honest servers. Nonetheless, we conclude that an attacker may be able to either exploit a previously partitioned Ethereum network, or to purposefully generate a partition to their benefit.

V. ANALYSIS

With the formalism of light client behavior and the adversary's goals defined, the next step is to develop a model based on these definitions that allows us to evaluate the security of the light client under different operating conditions. This is defined as a Markov process whose construction depends on the client's depth k parameter and the attack scenario.

A. CONNECTIONS TO HONEST AND DISHONEST SERVERS

Meeting requirement R1 is dependent on whether a light node connects to a server controlled by an adversary. In this step the client will pick out a random set of N nodes it will connect to. While this process is technically selection without replacement, we assume the overall population is sufficiently large that we can describe the whole source population using a continuous variable $f = \frac{M_0}{N_0+M_0}$, which represents the proportion of dishonest light server nodes, within the population of dishonest nodes M_0 and honest nodes N_0, that the light client may connect to. Therefore, the probability of a light client connecting to M dishonest nodes follows the Bernoulli trial distribution $B(N, f)$,

$$P(M \text{ dishonest nodes}) = \binom{N}{M} f^M (1-f)^{N-M}. \qquad (3)$$

We can identify three different limiting cases that are relevant to later analysis of an adversarial success probability.

C1. The client connects to only honest nodes ($M = 0$).

C2. The client connects only to dishonest nodes ($M = N$).

C3. The client connects to at least one honest and dishonest nodes ($0 < M < N$).

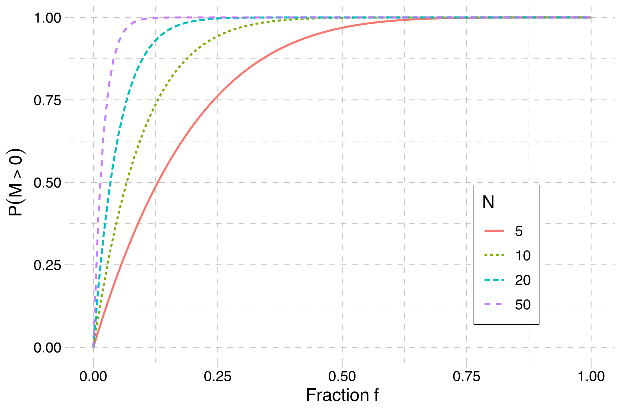

FIGURE 6. Probability of a client connecting to at least one dishonest light server as a function of the proportion of dishonest servers *f* in the server population, and the number *N* of connections the client establishes.

For simplicity, we observe that C1 and C2 are degenerate cases of the full analysis model (developed later) and do not have to be considered separately:

1) If $M = 0$, the adversary has no possibility of success, as we assume honest servers are also validating nodes and would refuse to propagate the adversarial chain to the target node (effectively $\lambda = 0$).

2) If $M = N$, this is functionally equivalent to $\gamma = 1 \Rightarrow \mu = 0$, as any new blocks from the honest network will not reach the target node.

If N is sufficiently large a light client is highly likely to connect to at least one dishonest server even with modest f:

$$P(M > 0) = 1 - P(M = 0) = 1 - (1-f)^N. \qquad (4)$$

This is shown in Fig. 6. It is also possible that an adversary using other methods such as network proximity or active connection manipulation to artificially boost $P(M > 0)$ above the value from (4). **Consequently we assume the target node connects to at least one honest and to at least one dishonest node (case C3).**

B. NON-CAPTIVE MODEL (A1)

We describe the interplay between the honest network and the adversary using a continuous-time Markov process. Markov processes have been previously used for blockchain miner analysis by Eyal and Sirer [17] to describe selfish mining strategy, while Nayak *et al.* [18] extend the selfish mining model with two-phase Markov process to evaluate alternative mining strategies.

In our model, the adversary has to meet two specific criteria regarding the adversarial chain. The adversarial chain must have *more work than the honest* chain to meet the R2 requirement. Also, the adversary must be able to generate a sufficiently *deep* chain to meet the R3 requirement. This latter requirement is dependent on the client's choice of k, the block depth parameter.

These criteria are tracked in the Markov process states as $(h, \Delta W)$ parameters, where h is the *phase* of the model, and

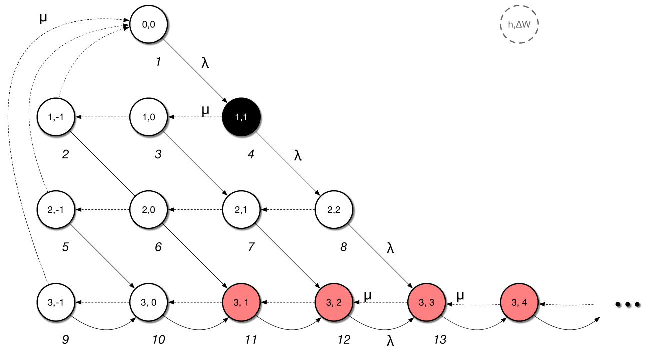

FIGURE 7. Markov process for a light client with block depth $k = 2$, applicable for attack scenario A1. This model has an infinite birth-death process tracking the ΔW value.

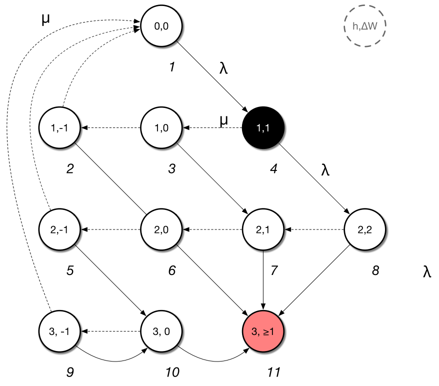

FIGURE 8. Markov process for a light client with block depth $k = 2$, applicable for attack scenarios A2 and A3. This model includes a captive state.

ΔW is the work difference between adversarial and honest chains. We encode the depth of the adversarial chain in the phase parameter $h = 0, 1, \ldots$ up to level $h = k + 1$ which subsumes all chain depths that are sufficiently deep to meet R3 ($h > k$). The work difference ΔW is the difference between work in the adversarial chain and work in the honest chain, and consequently R2 is met when $\Delta W > 0$.

The model includes two different processes: λ-process for adversarial mining and μ-process for honest network miners.[6] The *transition rates* for these processes are λ and μ respectively and are state- and time-independent. (Network partitioning is incorporated via the $\mu = (1-\gamma)\mu_0$ definition.) Fig. 7 displays an example with $k = 2$ on how the states can be arranged in a meaningful way to describe the system model starting from the initial state $(0, 0)$. The model has four different types of state transitions. The simplest one is λ-transition, i.e. **the adversary successfully mines a block**, when the adversarial chain is not yet deep enough to satisfy R2 requirement as described below (the notation is in the format *transition* : *condition* $\Rightarrow$ *newstate*, with the current state defined by h and ΔW).

1) $\lambda : h \leq k \Rightarrow (h + 1, \Delta W + 1)$

Thus, when the system is on $k + 1$ phase, an λ-transition remains on the current phase, and increases ΔW:

2) $\lambda : h = k + 1 \Rightarrow (h, \Delta W + 1)$

When the **honest network mines a block**, our model *resets* to the initial state if the work difference grows so large that more λ-transitions would be needed than from the initial state. If this is not yet the case, it will simply reduce ΔW (move one step left in the figure):

3) $\mu : \Delta W = 1 - k \Rightarrow (0, 0)$
4) $\mu : \Delta W > 1 - k \Rightarrow (h, \Delta W - 1)$

From Fig. 7 we can characterise some additional features. The black circle in the figure represents the state where the adversary mines the first block that fails $\mathcal{V}_s$ validation. The red circles represent states where the adversary is able to convince the light node to accept invalid state, where $h > k \wedge \Delta W > 0$. The solid arrows represent transitions where the adversary successfully mines a block (λ-transition) and dashed arrows (leading from right to left), are those where the honest network successfully mines a block — a μ-transition. While this model is bounded at the left by $\Delta W \geq 1 - k$, it may have indefinitely large ΔW in the $k + 1$ phase. This is described by an infinite birth-death process from the last state with inbound λ-transition from $(k+1, k+1)$ state.

The model is formally defined as a $\mathbf{Q} = q_{i,j}$ transition matrix where $q_{i,j}$ is the transition intensity from state i to state j. More specifically for $k = 2$ the matrix $\mathbf{Q_2}$ is ($\Lambda = -\lambda - \mu$ in the diagonal) (5), as shown at the bottom of the next page.

Please see Appendix B for details on how to succinctly describe the $\mathbf{Q}$ matrix by using generator matrices for each phase of the model.

C. CAPTIVE MODEL (A2 & A3)

The model in Section V-B describes the probabilities of the adversary succeeding in injecting an invalid block to a light client at *any specific time*. It is not, however, suitable for analysing the cumulative probability, i.e. the likelihood of an adversary succeeding *at, or before, a specific time*.

Unlike the non-captive model, the *captive model* as described in Fig. 8 is finite and has an absorbing state. The absorption state $(k + 1, \geq 1)$ applies for all success transitions. This allows us to evaluate the success probability of scenarios A2 and A3 as a function of time as now the model captures the *cumulative probability* of success for any time at or before given t.

The transitions in the captive model are mostly similar to the non-captive model for $h < k$, but differ for phases k and $k+1$. In the captive model when the **adversary mines a block** the simplest transition is a straightforward transition to a non-success state in the next phase:

1) $\lambda : h \leq k \wedge \Delta W < 0 \Rightarrow (h + 1, \Delta W + 1)$

[6] We prefer to use λ and μ instead of α, β used by other authors [17], [18]. The underlying process closely resembles a two-dimensional birth-death process, and the literature more commonly uses λ and μ in this context.

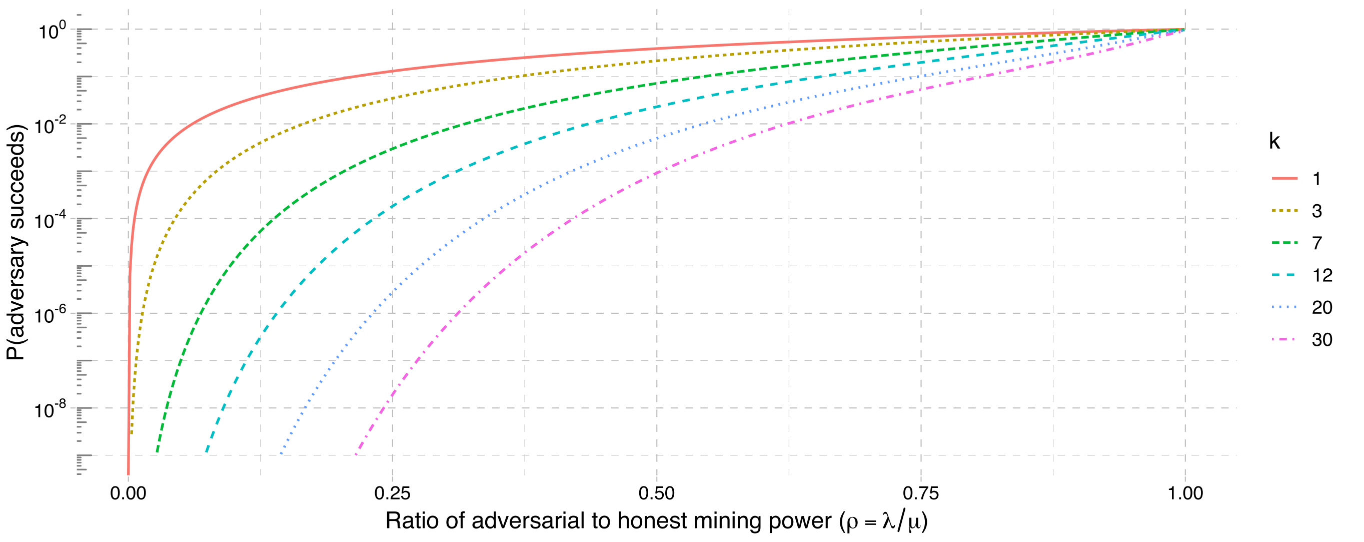

FIGURE 9. Probability of success by the attacker using non-captive model as a function of $\rho = \lambda/\mu$ ratio at the equilibrium state e.g. when $t \to \infty$, and for different k values. It is possible to read different γ values from the graph by the relation of $\mu = (1-\gamma)\mu_0$.

When the system is in phase $k+1$, but still far away from the success state it will keep in the same phase:

2) $\lambda : h = k+1 \wedge \Delta W < 0 \Rightarrow (h, \Delta W + 1)$

There is only one success state, so all transitions that would in non-captive model end up in $h > k \wedge \Delta W > 0$ state end up in the same, single absorbing success state:

3) $\lambda : h \geq k \wedge \Delta W \geq 0 \Rightarrow (k+1, \geq 1)$

When the **honest network mines a block** the transitions are similar as in non-captive model. The system may reset to (0, 0) state, or stay in phase but decrease the work difference. Note however, that there is no μ-transition out of the success state.

4) $\mu : \Delta W = 1 - k \Rightarrow (0, 0)$

5) $\mu : h < k \wedge \Delta W \leq 0 \Rightarrow (h, \Delta W - 1)$

The captive model is finite, and its size is solely determined by the k parameter. As it has an absorbing state that can be reached from all other states, eventually the probability of being in state $\pi_{k+1,\geq 1} \to 1$ as $t \to \infty$. Thus, if $\lambda > 0$ and given enough time, the adversary will always succeed in the captive model.

D. NUMERICAL APPROXIMATION

The underlying **non-captive Markov process** (Section V-B) is irreducible and infinite. The process does have a limiting distribution if $\lambda < \mu$ and this can be solved analytically by noting that the birth-death process has a recurrence relation and can be replaced by a single term. However, there are limits—in practice, an analytical solution using symbolic mathematics software seems to be out of reach for $k > 6$ for calculating the limiting distribution, and for $k > 2$ for a time-dependent (derivative) solution.

Since we are interested in both large $k > 10$ values as well as determining the time evolution of the adversarial success probability this necessitates the use of numerical methods. This requires truncation of the $\mathbf{Q}$ matrix to a finite size and approximation of $t \to \infty$ to a finite value to determine the equilibrium probability distribution.

$$\mathbf{Q_2} = \left[\begin{array}{cccccccccccccccc}
-\lambda & \cdot & \cdot & \lambda & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\
\mu & \Lambda & \cdot & \cdot & \cdot & \lambda & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\
\cdot & \mu & \Lambda & \cdot & \cdot & \cdot & \lambda & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \mu & \Lambda & \cdot & \cdot & \cdot & \lambda & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\
\mu & \cdot & \cdot & \cdot & \Lambda & \cdot & \cdot & \cdot & \cdot & \lambda & \cdot & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & \mu & \Lambda & \cdot & \cdot & \cdot & \cdot & \lambda & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \mu & \Lambda & \cdot & \cdot & \cdot & \cdot & \lambda & \cdot & \cdot & \cdots \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu & \Lambda & \cdot & \cdot & \cdot & \cdot & \lambda & \cdot & \\
\mu & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \Lambda & \lambda & \cdot & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu & \Lambda & \lambda & \cdot & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu & \Lambda & \lambda & \cdot & \cdot & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu & \Lambda & \lambda & \cdot & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \mu & \Lambda & \lambda & \\
 & & & & & & & \vdots & & & & & & & \ddots
\end{array}\right] \tag{5}$$

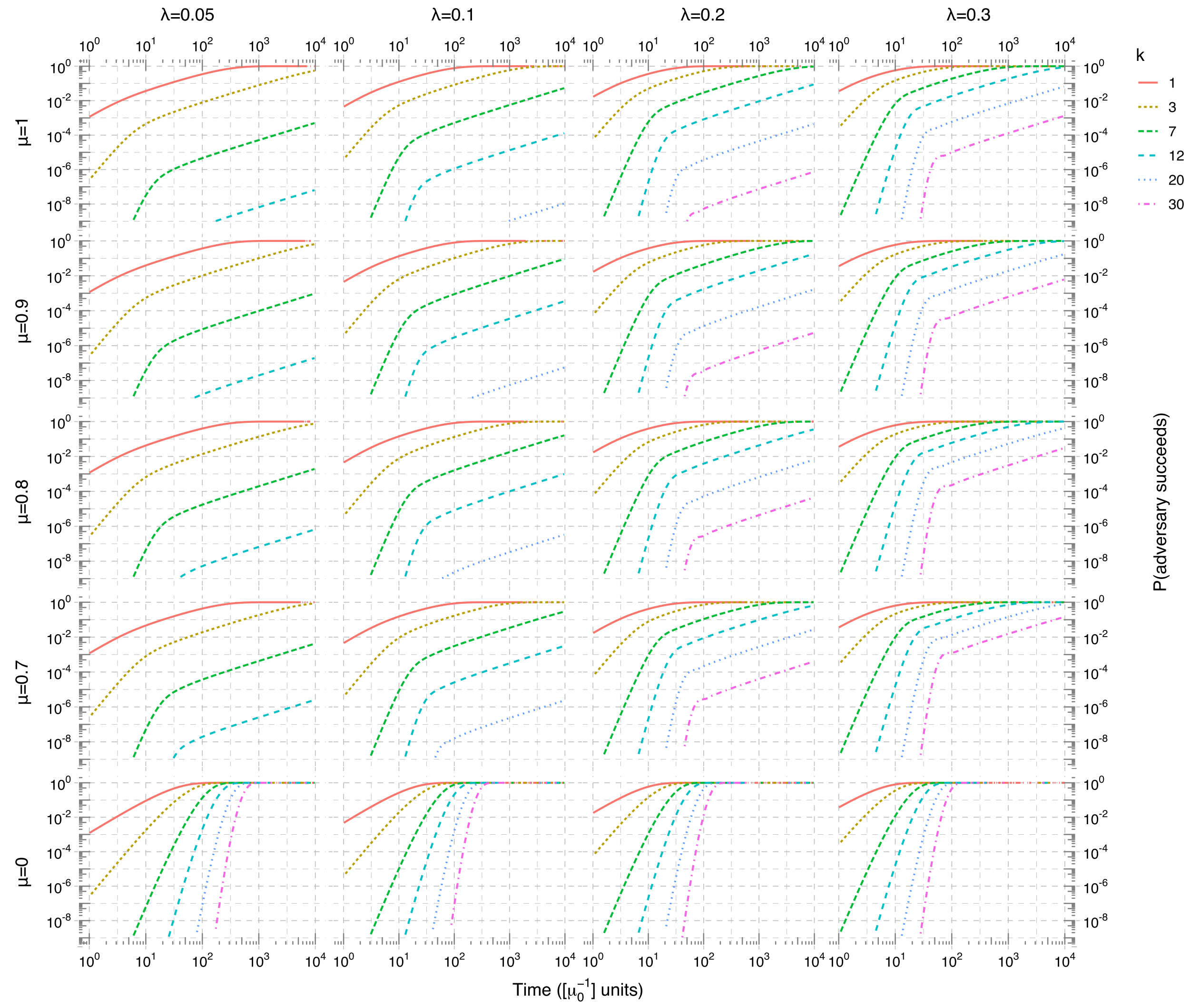

FIGURE 10. **The time development of attacker's success probability using captive model for different λ ratios along the horizontal axis, and for different μ values along the vertical graphs axis (corresponding to different γ partitioning values, e.g. $\mu = (1 - \gamma)\mu_0$). The individual lines correspond to $k = \{3, 7, 13, 21, 30\}$ from top to bottom. The time units are normalized to μ_0^{-1} e.g. value of one represents the block interval in the original honest portion of the network. Please note that both axes in the graphs are logarithmic.**

We truncate the matrix using a b parameter (length of the birth-death chain) by dynamically increasing it from an initial value until the absolute change of the probability of the last birth-death process state is below a threshold (10^{-6}), or we hit a maximum size for the birth-death process ($b_{\max} = 1000$). We justify the use of a maximum truncation length by noting that this limit only occurs when $\lambda \approx \mu$ which we consider to be an unlikely long-term equilibrium due to the dynamic nature of the honest mining pool, and thus less relevant to our analysis. In practice we see smooth behavior in the results for the region around $\lambda \approx \mu$.

By using a truncated **Q** matrix the time evolution of the state probabilities $\bar{\pi} = (\pi_1, \ldots, \pi_n)$ is from initial state $\bar{\pi}_0 = (1, 0, \ldots)$ in $[t] = [\mu_0^{-1}]$ units:

$$\bar{\pi}(t) = \bar{\pi}_0 e^{\mathbf{Q}t}. \tag{6}$$

This can be used to compute the time evolution of the probability distribution. Finally, to calculate the equilibrium distribution we approximate $t \to \infty$ as $t = 10^7 / \max(\lambda, \mu)$. This was empirically determined to be sufficiently large, as using larger t values would have less than 10^{-4} change in the absolute result.

The Markov process with a **captive state** (aka absorbing state, Section V-C) is finite and does not need to be truncated. Therefore the captive model matrix size is solely determined by the parameter k. However, due to the absorbing state the model can take require a large time value to reach a steady state, we use an accurate binary search only for small t values, and for larger values provide only an approximate lower bound.

VI. RESULTS

The results are presented for the three different questions posed in Section IV-C.

Q1. What is the adversary's probability of having a successful adversarial chain at any random point in time? The equilibrium probabilities of the non-captive model are shown in Fig. 9.

TABLE 2. The time t (in μ_0^{-1} units) that is required to reach specific probability p for success in injecting at least one adversarial block to the target node. Results are shown with two significant digits. For $t > 10^7$ the required time is approximate only. The wall-clock time is approximately 25 minutes for $100\mu_0^{-1}$, 4 hours for $1000\mu_0^{-1}$, almost 2 days for 10^4, 17 days for 10^5, half year for 10^6 and almost 5 years for 10^7 block intervals. All results are shown with two digits of accuracy.

		$t\ (\mu_0^{-1})$								
		$k=7$			$k=12$			$k=30$		
λ	μ	$p=50\%$	$p=90\%$	$p=99\%$	$p=50\%$	$p=90\%$	$p=99\%$	$p=50\%$	$p=90\%$	$p=99\%$
0.05	1	$>14\times10^6$	$>60\times10^6$	$>0.1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.05	0.95	9.8×10^6	$>40\times10^6$	$>80\times10^6$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.05	0.8	3.5×10^6	$>12\times10^6$	$>40\times10^6$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.05	0.7	1.6×10^6	5.3×10^6	$>12\times10^6$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.05	0.5	240,000	810,000	1.6×10^6	$>0.12\times10^9$	$>0.4\times10^9$	$>0.8\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.05	0.2	3,600	12,000	24,000	62,000	210,000	410,000	$>0.8\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.05	0	150	240	320	250	360	460	610	770	910
0.1	1	120,000	410,000	810,000	$>60\times10^6$	$>0.18\times10^9$	$>0.4\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.1	0.95	93,000	310,000	620,000	$>40\times10^6$	$>0.12\times10^9$	$>0.4\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.1	0.8	38,000	130,000	250,000	6.8×10^6	$>40\times10^6$	$>60\times10^6$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.1	0.7	20,000	65,000	130,000	2.1×10^6	7.1×10^6	$>16\times10^6$	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.1	0.5	4,300	14,000	29,000	150,000	490,000	980,000	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.1	0.2	250	780	1,600	920	3,000	6,000	27,000	90,000	180,000
0.1	0	77	120	160	130	180	230	310	380	450
0.2	1	2,200	7,200	14,000	74,000	240,000	490,000	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.2	0.95	1,800	5,800	12,000	51,000	170,000	340,000	$\gg1\times10^9$	$\gg1\times10^9$	$\gg1\times10^9$
0.2	0.8	900	3,000	5,900	15,000	51,000	100,000	$>0.16\times10^9$	$>0.6\times10^9$	$\gg1\times10^9$
0.2	0.7	560	1,900	3,700	6,700	22,000	44,000	$>18\times10^6$	$>60\times10^6$	$>0.12\times10^9$
0.2	0.5	210	670	1,300	1,100	3,700	7,400	150,000	490,000	980,000
0.2	0.2	42	120	240	68	240	520	160	860	2,300
0.2	0	38	59	80	63	89	110	150	190	230
0.3	1	320	1,000	2,100	3,300	11,000	22,000	5.2×10^6	$>18\times10^6$	$>40\times10^6$
0.3	0.95	270	890	1,800	2,500	8,200	16,000	2.4×10^6	7.9×10^6	$>16\times10^6$
0.3	0.8	160	530	1,100	1,000	3,400	6,700	220,000	730,000	1.5×10^6
0.3	0.7	120	370	740	560	1,800	3,700	44,000	150,000	290,000
0.3	0.5	57	180	360	170	550	1,100	2,100	6,900	14,000
0.3	0.2	26	45	97	42	64	140	100	130	190
0.3	0	26	39	53	42	59	76	100	130	150

The equilibrium model is time-independent, and consequently only the ratio $\rho = \lambda/\mu$ has an impact. The graph shows $\rho \in [0, 1.5]$ and various k values. The overall result is as expected—a larger k parameter gives lower success probability for the adversary (i.e. higher security for the light client), and the ρ value has to be substantial before the instantaneous success probability is at $p = 0.1$ level even for a low $k = 1$ value.

The literature does not generally give details on what is considered a "normal environment" for light clients to operate securely. We assume this means $\lambda \ll \mu \Rightarrow \rho \ll 1$ and $k \geq 3$. For example, consider an attacker controlling mining power equivalent to 5% of the honest network ($\lambda = 0.05$, $\mu = 1 \Rightarrow \rho = 0.05$). The probability p of success in the non-captive model's equilibrium limit is $p = 0{,}16 \cdot 10^{-3}$ for $k = 3$, and $p = 0{,}11 \cdot 10^{-6}$ for $k = 7$. The probability diminishes rapidly for larger k values ($p(k = 30) = 0{,}27 \cdot 10^{-24}$). These results are in line with the view that light clients are secure under the assumption of an honest majority of computing power and a passive attacker. This corresponds to a scenario where the timing of the activity is controlled by the client, such as checking whether a very recent transaction—whose timing cannot be influenced by the attacker—is deep enough in the chain.

We also consider a situation where the attacker is able to subvert a major mining pool on the honest network. For example, subverting 30% of the honest network equals to $\lambda = 0.3$, $\mu = 0.7 \Rightarrow \rho = 3/7 \approx 0.43$. In this case the equilibrium probabilities would be significantly different: $p = 0{,}15$ for $k = 3$, $p = 0{,}038$ for $k = 7$, $p = 9 \cdot 10^{-3}$ for $k = 12$, and $p = 1{,}2 \cdot 10^{-3}$ for $k = 20$. This result must be interpreted as meaning that an adversary would have approximately 4% probability of successfully injecting an invalid state to the light client with $k = 7$ when the adversary is attempting to maintain an adversarial chain with previously known invalid state.

Q2. What is the adversary's probability of 1-inject success within Δt time? The dynamic captive model takes the time evolution of the success probability into account, and provides a view into situations where the adversary has only a limited amount of time to perform the attack to perform a 1-inject attack. In this scenario, the attacker starts mining an adversarial chain at a specific point in time, and has Δt time to succeed in the attack. This can be a situation where the adversary has to interact with the client, or when the client generates a transaction, and the adversary has Δt time until the client checks for the status of the transaction. As can be seen in Fig. 10, the success probability increases as Δt increases.

Using the same adversary as in earlier results, we will first look at a modest mining power that is well within reach of a well-resourced adversary, $\lambda = 0.05$, $\mu = 1$. The adversary's success probability increases over time, and for a client with $k = 7$ block depth, $p(t = 20) = 0{,}46 \cdot 10^{-6}$, $p(t = 240) = 12 \cdot 10^{-6}$, $p(t = 5760) = 0{,}3 \cdot 10^{-3}$, and $p(t = 2102400) = 0{,}1$ (these correspond to 5 minutes, 1 hour, 1 day, and 1 year respectively). Under this scenario, an adversary with a modest $\lambda = 0.05$ mining power has a 10% probability of success *at least once within a year* against a light client with block depth $k = 7$. The probablity increases rapidly for lower k, for example $p(k = 3, t = 240) = 0{,}019$ representing a significant probability of success over just one hour.

If we take the adversary capable of subverting a 30% mining pool, it can gain substantial success probability even if able to operate the mining pool for only one hour ($t = 240$). For low k values the success probability is very high, for example $p(k = 3) = 1$. Even for a relatively high $k = 12$, the adversary has over 25% probability of successful state injection against a light client when leveraging the subverted miner.

Q3. What is the expected time needed for the adversary to gain a specific success probability? We have summarised a selection of k, λ and μ values for $p = \{0.5, 0.9, 0.99\}$ in Table 2. The results show the same general pattern as observed above: the time required for success increase as k increases. Conversely, the required time decreases as λ/μ ratio decreases. Therefore, achieving even a 50% success probability in unfavourable conditions ($\lambda = 0.05, k = 7$) requires over six years.

The situation changes drastically, again, for an adversary capable of subverting a mining pool. Targeting a light node with $k = 7$, the adversary can achieve 50% success rate in 120 block intervals (30 minutes), and 99% success probability in less than 3 hours (740 block intervals). Even for a very high value $k = 30$ the corresponding 50% success chance occurs in a week ($t = 44\,000$), and 99% probability in just over 2 months ($t = 290\,000$).

VII. CONCLUSIONS

Our results reject many of the common assumptions on light client security. While light clients may be considered secure against invalid block injections from adversaries under "common" scenarios, these are optimistic scenarios—with a short attack window, an honest majority, and no network partitioning. However, the situation changes drastically as one moves away from these optimistic assumptions. Even with only modest hashing power, an adversary has a statistically significant probability of success under partial network isolation. We also see that if an adversary is able to subvert an existing mining pool on the Ethereum network, its success probability increases substantially.

The security of light clients is further eroded when we consider adversaries who able to manipulate the operating environment of the target node. The adversary may have access to the device, and even without tampering with the device itself, it could manipulate its network accessibility or power availability. GSM and WiFi jammers are relatively easy to obtain and deploy, allowing a malicious entity the ability to control availability of the network that a client is connected to. It has already been shown that the security assurances of an eclipsed light client are severely reduced. Our model (at $\gamma = 1$) confirms that.

We also demonstrate that even without isolating the client, or partitioning the network, a patient adversary that can mount 1-injection attack can gain significant success probability if willing to wait (for days or months). The naive IoT-based lock example from Appendix A-C could be attacked by a patient adversary—they set up a person on-site who waits for a signal when the attack is successful, who then dashes over to the lock with a fake key card.

A secure system cannot rely on optimistic situations, and must be secure under an active adversary. We believe there is a clear need for a formal blockchain model that would not only consider the adversary's mining power, but take into account partitioning and eclipsing scenarios.

For constrained devices our recommendation would be not to use light client protocols for any devices which are not under trusted human supervision or interaction, e.g. any off-site or industrial applications in particular. If interaction with public blockchains is required, we would recommend to look into ledger-to-ledger bridging approaches [39], or employ a trusted third party to attest and pin down the blockchain state [34]. For other light clients we see at $k \geq 12$ the time for 1-inject success under somewhat "reasonable but powerful adversary" assumptions is several hundred block intervals or more. If a human is in the loop, this delay of several hours would probably raise healthy suspicion. In general, we believe a light client should choose a k value *significantly larger than for a validating node for the same use case*.

It is interesting to note that one might naively assume that a light client is more secure against eclipse attacks (connecting to only dishonest servers) with large values of N. However, as N increases the client is more and more likely to connect to at least one dishonest server. Since the adversary in our model requires only one connection to dishonest server to be able to complete, this means that as N increases, in some circumstances the light node becomes increasingly susceptible to attack.

We identify potential mitigation strategies and areas of further research. The most pressing mitigation would be to ensure all blockchain-related operations come with a hard timeout, after which the client should abort the operation and require retry, or fall back to a safe mode requiring manual (human) intervention. If this is not feasible, the device could use heuristics to determine when a partition occurs, and change to a safe operating mode until normal Ethereum connectivity is restored. We do also recognise that if these *safe mode* protocols are not designed correctly they could also become a potential attack vector.

Ethereum blockchain is not the only applicable blockchain technology, and even other variants and proposed protocol changes within the ''Ethereum family'' may provide different level of security for light clients. For example, a proof-of-stake mechanism would change the adversarial model significantly and in our opinion whether a proof-of-stake mechanism would be more or less secure to a light client is not immediately obvious. We see this also as a potentially useful area of further research.

While we discuss the threat of 1-inject attacks, further elaboration of the scenarios with potentially an analysis of existing smart contracts on whether they can be securely operated by light clients under 1-inject threat model or not could be beneficial. Similarly, analysing vulnerability of a light client to 2-inject attacks under different operating regiments would potentially allow identification of secure operating regimes under more complex adversarial light client interactions.

Finally, the analysis is limited by our assumptions. For example, Ethereum allows slow changes to the difficulty of the cryptographic puzzle in response to changes in the network hashing power. This means that a partitioned network will over time adapt so that it will again reach a similar block interval as the full network. We do not take this into account. While we believe that this assumption has very little effect, it is a notable divergence from the Ethereum protocol. Similarly, we believe that the adversary's strategy as defined in Section IV-A is most likely sub-optimal. This does not invalidate our results, but does mean that our results represent a conservative *lower bound* on the adversary's success probability, and thus also the *upper bound* for light client security under the described attack.

APPENDIX A
EXAMPLES OF ATTACKS AGAINST A LIGHT CLIENT

Without a deep knowledge of Ethereum security model it is not perhaps immediately obvious what an adversary is able to achieve by injecting a block $\tilde{B}$ to the client. This section describes some of the potential types of attacks the adversary may be able to achieve under different scenarios.

Important: There are many different methods that can be employed at LES clients to mitigate many of the attacks below either completely, or decrease adversary's success probability significantly. These examples are intended only to demonstrate the mechanisms at an adversary's disposal if or when they succeed in injecting an invalid block $\tilde{B}$ into a client.

A. DOUBLE-SPENDING ATTACK

The classic double-spending attack is based on the idea that an adversary tricks the target into thinking they have received a payment from the adversary, while the adversary successfully manages to keep the payment to themselves. In this scenario we have the adversary as a buyer, and the target as a seller. The two parties agree on a payment for goods. The buyer generates a transaction for sending the required amount of cryptocurrency to the seller's account. The seller can look at the transaction and verify the amount and recipient, after which the seller can submit the transaction to the blockchain network (seller does not have to trust the buyer to submit the transaction). The seller will first verify that the transaction has been accepted to a mined block, and then wait for k blocks until handing out the merchandise.

The double-spending attack has been analysed extensively, see for example Karame *et al.* [40], and Liao and Katz [41]. In our model the adversary has no need for incorrect state $\tilde{S}$, as all that is required in the inclusion of the payment to the adversarial chain, and the ability to present and maintain this incorrect view, and present it to the buyer. The adversary would submit a conflicting transaction to the honest network just like in a normal double-spending attack.

Thus, while a double-spending attack is similarly feasible under our model, it can be performed by the adversary without the need to generate incorrect blocks $\tilde{B}$ if the transaction itself is acceptable to the honest network. If the transaction is not valid — for example, the payee is missing sufficient funds — then the method described in this article becomes relevant.

B. FAKE WEALTH

Since the state $\tilde{S}$ on the adversarial chain does not have to follow normal state transition rules, the adversary can simply set his or her own account balance to an arbitrary amount (say, 1 000 000 ethers, equivalent to several million USD). Under normal state transition rules this balance must be traceable through transactions to either blockchain rewards, or the genesis block wealth, and cannot be created out of thin air. These constraints do not, of course, apply to the adversarial state $\tilde{S}$.

Whether just showing off a fake balance to another device is actually useful is, though, questionable.

C. FOOLING AN IoT DEVICE

Let's assume there's a lock that is opened by a near-field communication device (NFC device) utilizing a secure challenge-response protocol. This allows the lock to securely establish the identity of the NFC device, while an eavesdropper is unable to clone the identity due to the presence of the challenge-response protocol.

Each lock is configured to allow only a specific set of identified NFC devices to open the lock. The set of allowed NFC identifiers is managed by a smart contract. The smart contract is periodically queried to provide a list of identifiers that, if demonstrated via NFC protocol, open the lock. The portion of a smart contract below demonstrates how the owner of the locks sets the allowed keys, and how the locks retrieve the set of keys that are allowed to open the lock.[7]

```
contract LockManager {
   address owner;
   mapping (uint => uint[]) public allowedKeys;

   ...

   function setAllowedKeys(
       uint lockId,
       uint[] memory fobIds)
     public
   {
       require(msg.sender == owner,
             ``Only owner can modify keys'');
       allowedKeys[lockId] = fobIds;
   }

   function getAllowedKeys(
       uint lockId)
     public view returns (uint[] memory)
   {
       return allowedKeys[lockId];
   }

   ...
}
```

The underlying assumption is that since the smart contract is secure, the set of allowed keys can only be changed by the contract owner. Thus, assuming the owner's secret key is secure, in Ethereum, any party using the method `getAllowedKeys` is guaranteed to retrieve only the values as set by the owner. No other party is able to generate a transaction that would be accepted by `setAllowedKeys`. Since the honest blockchain consists only of blocks that pass the both $\mathcal{V}_h$ and $\mathcal{V}_s$ validations, the security of the contract and its data are presumed.

However, a lock relying on LES protocol can be fooled into accepting a key $\tilde{u}$ ("unlocker") not part of the set of allowed keys U_l for the specific lock l. This can be accomplished by two different ways if the adversary is able to fool the lock to accepting an invalid block $\tilde{B}$ as a valid representation of the smart contract state.

The first mechanism is to modify the smart contract's *storage*, where values of all of the smart contract's variables are stored. The adversary can determine the location of the `allowedKeys` mapping in the storage, and calculate the storage location of the key array for lock l. Thus, the adversary is able to generate a block $\tilde{B}_n$ where the state of correct execution $\tilde{S}_n = \tilde{\pi}(S_{n-1}, T)$ differs from the "correct" S_n only by having $\tilde{u} \in \tilde{U}_l$ in $\tilde{S}_n$ (and thus, in $\tilde{B}_n$ accepted by the lock), while in the "correct" smart contract state $\tilde{u} \notin U_l$. Thus, when later the lock updates its set of allowed keys, it will retrieve block $\tilde{B}$, verify its $\mathcal{V}_h$ validity, and retrieve $\tilde{U}_l$. At this point is it trivial for the adversary to use its own key $\tilde{u}$ to open the lock.

The second mechanism does not need to alter the smart contract storage at all—instead it will simply overwrite the whole smart contract (this is possible in Ethereum because the smart contract *address*, while unique and permanent, does not authenticate the smart contract *code*). In short, the adversary changes the smart contract's code to one which always returns the adversary's key:

```
contract LockManager {
   function getAllowedKeys(
       uint /*unused lockId*/)
     public view returns (uint[] memory)
   {
       uint256[] memory keys = new uint256[](1);
       keys[0] = <adversary's key id>;
       return keys;
   }

   ...
}
```

The smart contract code is referenced via its account data, which in turn contains hash of the smart contract bytecode (called `codeHash`). In the Ethereum model a smart contract's code is immutable—under correct Π transition rules there is no valid execution that changes an account's `codeHash`. The actual bytecode M is referenced by its hash $hash(M)$, meaning that under honest network assumption the client is able to verify the correctness of the retrieved contract code by hashing it and comparing to the account's `codeHash` value. However, the adversary is able to modify the smart contract account, and change the `codeHash` value to $hash(\tilde{M})$ value with $\tilde{M}$ being, for example, the version of the code always returning adversary's own key identifier.

APPENDIX B
GENERATING THE Q MATRIX

We are using *generator matrices*—matrices as parts of matrices—to help structure the $\mathbf{Q}$ matrix construction. The use of generator matrices allows us to describe each *phase* of the Markov process as a "single" row of backward, local, and forward matrices for the phase. These "meta-rows" are not uniform in size, and depend on parameters of the model.

A. GENERAL CONSTRUCTION

Section V-B describes the Markov process for the adversarial, non-captive model, and its transition matrix $\mathbf{Q}$, where a $\mathbf{Q_2}$

[7]The authors most definitely do not think it is a good idea to use blockchain to specify the set of allowed keys.

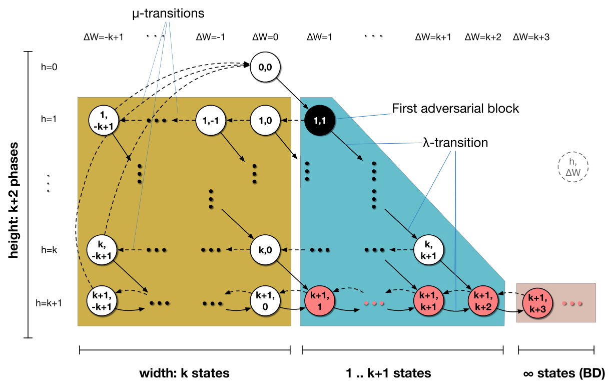

FIGURE 11. Generic description of the non-captive Markov process. The black circle represents the first $\tilde{B}$ block mined by the adversary, and red circles represent states where the adversary is able to inject invalid state to the client (e.g. succeeds). There are an infinite number of success states in non-captive model, which can be modeled as a simple birth-death process.

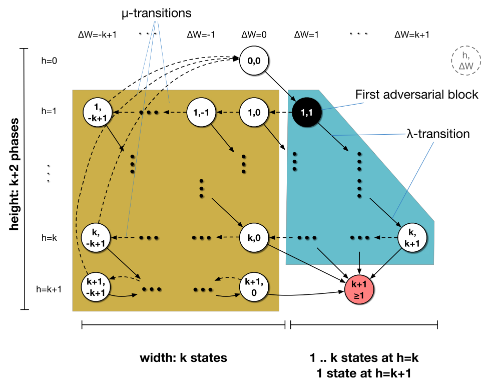

FIGURE 12. The captive process is similar to the non-captive model except for success states, which are replaced with a single $(k+1, \geq 1)$ state that has no outbound μ-transitions. The captive model is always finite in size. Note that for $k = 1$ the regular rectangular shape of the $\Delta W \leq 0$ states breaks—to see how this is left as an intellectual exercise for the reader.

matrix for the non-captive model was given as an example. While the matrix may look daunting at first, here we show that it has a recurring structure thanks to the phase-type construction. The use of generator matrices has been described in the literature, for example, by Harchol-Balter [42]. We employ this technique to describe a generic generator structure for *any* $\mathbf{Q_k}$:

$$\mathbf{Q_k} = \begin{bmatrix} \mathbf{L_0} & \mathbf{F_0} & & & & & \\ \mathbf{B_1} & \mathbf{L_1} & \mathbf{F_1} & & & & \\ & \ddots & & \ddots & \ddots & & \\ \cdots\cdots & \mathbf{B_h} & \cdots\cdots & & \mathbf{L_h} & \mathbf{F_h} & \\ & & \ddots & & & \ddots & \ddots \\ \cdots\cdots & & \mathbf{B_{k+1}} & \cdots\cdots & & & \mathbf{L_{k+1}} \end{bmatrix} \tag{7}$$

Each phase of the model is described by a single generator row in the above matrix. All the elements in the $\mathbf{Q}$ matrix $\mathbf{G}$, $\mathbf{L}$, and $\mathbf{F}$ are themselves matrices of varying size $(m \times n)$. We refer to these are the *backward generator*, the *local generator*, and the *forward generator* matrix respectively. The number of states $m(h, k)$ for each phase h is dependent on the depth parameter k for $h \in \{0, \ldots, k\}$ for of a non-captive model portion can expressed as a recurrence relation (see also Fig. 12):

$$m(0, k) = 1 \tag{8}$$
$$m(h, k) = k + h \quad h \leq k. \tag{9}$$

Since the value of ΔW can grow without bounds, the size of the $\mathbf{L_{k+1}}$ matrix is theoretically infinite, although in practice it is truncated at some point. We define the truncation length of the $\Delta W > k + 2$ portion as b and use it to define m at phase $k + 1$:

$$m(k + 1, k) \to \infty \tag{10}$$
$$m_b(k + 1, k) = 2k + b + 1. \tag{11}$$

The captive model $m^c(h, k)$ is slightly different on the $h = k+1$ phase (there is a corner case for $k = 1$ with only one state on $h = k + 1$ phase instead of two states one might expect):

$$m^c(h, k) = m(h, k) \quad h < k + 1 \tag{12}$$
$$m^c(2, 1) = 1 \tag{13}$$
$$m^c(k + 1, k) = k + 1 \quad k \neq 1. \tag{14}$$

We define the total number of states are $M_b(k) = \sum m(h, k)$ and $M^c(k) = \sum m^c(h, k)$ as the total number of core states for the two models:

$$M_b(k) = 3/2k^2 + 5/2k + b + 2 \tag{15}$$
$$M^c(1) = 4 \tag{16}$$
$$M^c(k) = 3/2k^2 + 2/2k + 2 \quad k \neq 1. \tag{17}$$

B. CORE BACKWARD MATRIX

The number of columns n_B in the backward generator matrix $\mathbf{B}_h$ for non-captive models can be expressed as recurrence relation where each phase's backward generator matrix is the same width as previous phase's one plus the number of columns in the local generator matrix n_L of previous phase:

$$n_B(0, k) = 0 \tag{18}$$
$$n_B(h, k) = n_B(h - 1, k) + n_L(h - 1, k) \quad h \leq k + 1. \tag{19}$$

For the captive model the number of backward generator matrix columns n^c_B is almost identical, apart from the lack of birth-death process phase:

$$n^c_B(h, k) = n_B(h, k) \quad h \leq k + 1. \tag{20}$$

The backward matrix $\mathbf{B}_h$, size $m(h, k) \times n_B(h, k)$, has only a single transition to the first state always from the leftmost e.g. first state in the phase for all core phases. That is, the backward transition is always to the first phase—state 1.

$$\mathbf{B}_0 = [] \tag{21}$$

$$\mathbf{B}_h = \begin{bmatrix} \mu & 0 & \cdots \\ 0 & 0 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad 0 < h \leq k + 1 \tag{22}$$

The captive model backward generator matrix is same as non-captive model's.

$$\mathbf{B}_h^c = \mathbf{B}_h \tag{23}$$

C. LOCAL MATRIX

The size of a local matrix for both non-captive $n_L(h, k)$ and captive $n_L^c(h, k)$ is always the same, e.g. the number of states in the phase:

$$n_L(h, k) = m(h, k) \tag{24}$$

$$n_L^c(h, k) = m^c(h, k) \tag{25}$$

For non-captive model in all phases $0 < h < k+1$, the local generator matrix is always similar with μ-transitions moving to the previous state in the same phase, and the diagonal being balanced with the λ-transition in the forward matrix:

$$\mathbf{L}_0 = \begin{bmatrix} -\lambda \end{bmatrix} \tag{26}$$

$$\mathbf{L}_h = \begin{bmatrix} -\lambda-\mu & 0 & \cdots & \cdots \\ \mu & -\lambda-\mu & 0 & \cdots \\ 0 & \mu & -\lambda-\mu & \cdots \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \quad 0 < h < k+1 \tag{27}$$

While the first row may appear to be unbalanced, it has a reset μ-transition that is part of the backward generator matrix $\mathbf{B}_h$.

Since the $k+1$ phase contains the birth-death process from $\Delta W > k+2$ onwards, the local generator matrix for it will also have λ-transitions most of the time:

$$\mathbf{L}_{k+1}^* = \begin{bmatrix} -\lambda-\mu & \lambda & 0 & \cdots & \cdots & \cdots \\ \mu & -\lambda-\mu & \lambda & 0 & \cdots & \cdots \\ 0 & \mu & -\lambda-\mu & \lambda & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \tag{28}$$

If $b \to \infty$ this does not stop, but if b is finite then at some point the last state cannot have a λ transition and needs to be balanced:

$$\mathbf{L}_{k+1} = \begin{bmatrix} -\lambda-\mu & \lambda & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ \cdots & \cdots & \cdots & \mu & -\mu \end{bmatrix} \tag{29}$$

For the captive model, the local generator matrix is identical except for $k+1$ phase where the very last state has no transitions (it is absorbing state).

$$\mathbf{L}_h^c = \mathbf{L}_h \quad h \le k \tag{30}$$

$$\mathbf{L}_{k+1}^c = \begin{bmatrix} -\lambda-\mu & \lambda & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \tag{31}$$

D. FORWARD MATRIX

The size of the forward generator matrix depends on the number of states in the next phase.

$$n_F(h, k) = n_L(h+1, k) \quad h < k+1 \tag{32}$$

$$n_F(k+1, k) = b \tag{33}$$

$$n_F(k+2, k) = 0 \tag{34}$$

$$n_F^c(h, k) = n_F(h, k) \quad h < k \tag{35}$$

$$n_F^c(k, k) = m_L^c(k+1, k) \tag{36}$$

$$n_F^c(k+1, k) = 0. \tag{37}$$

The generator matrix is easiest to describe consisting of a zero matrix, and a diagonal λ matrix:

$$\mathbf{F}_h = \begin{bmatrix} \mathbf{0}_{m(h,k),n_L(h+1,k)-n_L(h,k)} & \lambda \mathbf{I}_{m(h,k)} \end{bmatrix} \quad h \le k \tag{38}$$

or put more descriptively,

$$\mathbf{F}_0 = \begin{bmatrix} 0 \cdots \lambda \end{bmatrix} \tag{39}$$

$$\mathbf{F}_h = \begin{bmatrix} 0 & \lambda & 0 & \cdots & \cdots \\ 0 & 0 & \lambda & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \cdots & \cdots & \cdot & \cdots & \lambda \end{bmatrix} \quad 0 < h \le k \tag{40}$$

For the captive model the forward generator matrix is similar up to the previous-to-last phase which needs to transition either to the non-absorbing state (if the target state has $\Delta W \le 0$), or to the sole absorbing state otherwise.

$$\mathbf{F}_h^c = \mathbf{F}_h \quad h < k \tag{41}$$

$$\mathbf{F}_k^c = \begin{bmatrix} 0 & \lambda & \cdots & \cdots \\ \vdots & \ddots & \lambda & \cdots \\ \vdots & \ddots & \ddots & \lambda \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \lambda \end{bmatrix} \tag{42}$$

Neither non-captive or captive model have further phases than $k+1$, and consequently there is no forward matrix for phase $k+1$.

ACKNOWLEDGMENT

The authors would like to thank Prof. Colin Boyd for the work he put into a draft of another earlier article, which eventually lead us to the topic considered in this article. They also would like to thank Signe-Anita Lindgrén for her help during the proofreading of the final version.

REFERENCES

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, Apr. 2014.

[2] S. Nakamoto. (2008). *Bitcoin: A Peer-To-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] Z. Ren and Z. Erkin, "VAPOR: A value-centric blockchain that is scale-out, decentralized, and flexible by design," in *Financial Cryptography and Data Security. FC* (Lecture Notes in Computer Science), vol. 11598, I. Goldberg and T. Moore, Eds. Cham, Switzerland: Springer, 2019, pp. 487–507. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-32101-7_29

[4] A. Schoedon. (Nov. 29, 2017). *The Ethereum-Blockchain Size Will not Exceed 1TB Anytime Soon*. Accessed: Jan. 11, 2018. [Online]. Available: https://dev.to/5chdn/the-ethereum-blockchain-size-will-not-exceed-1tb-anytime-soon-58a

[5] Ethereum. (May 2019). *Light Ethereum Subprotocol (LES)*. Accessed: Oct. 19, 2019. [Online]. Available: https://github.com/ethereum/devp2p/blob/master/caps/les.md

[6] M. J. Levy. (Jun. 27, 2019). *The Deep-Dive Into How Verizon and a BGP Optimizer Knocked Large Parts of the Internet Offline Monday*. Accessed: Nov. 7, 2019. [Online]. Available: https://blog.cloudflare.com/the-deep-dive-into-how-verizon-and-a-bgp-optimizer-knocked-large-parts-of-the-internet-offline-monday/

[7] NetBlocks. (Aug. 3, 2019). *Evidence of Internet Disruptions in Russia During Moscow Opposition Protests*. Accessed: Nov. 7, 2019. [Online]. Available: https://netblocks.org/reports/evidence-of-internet-disruptions-in-russia-during-moscow-opposition-protests-XADErzBg

[8] C. Hogg. (May 14, 2010). *China Restores Xinjiang Internet*. Accessed: Nov. 7, 2019. [Online]. Available: http://news.bbc.co.uk/2/hi/asia-pacific/8682145.stm

[9] V. Gramoli, ''From blockchain consensus back to Byzantine consensus,'' *Future Gener. Comput. Syst.*, vol. 107, pp. 760–769, Jun. 2020, doi: 10.1016/j.future.2017.09.023.

[10] C. Natoli and V. Gramoli, ''The balance attack or why forkable blockchains are ill-suited for consortium,'' in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2017, pp. 579–590, doi: 10.1109/DSN.2017.44.

[11] M. Al-Bassam, A. Sonnino, and V. Buterin, ''Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities,'' 2018, *arXiv:1809.09044*. [Online]. Available: http://arxiv.org/abs/1809.09044

[12] O. Leiba, Y. Yitzchak, R. Bitton, A. Nadler, and A. Shabtai, ''Incentivized delivery network of IoT software updates based on trustless proof-of-distribution,'' in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)*, Apr. 2018, pp. 29–39, doi: 10.1109/EuroSPW.2018.00011.

[13] L. da Costa, A. Neto, B. Pinheiro, W. Cordeiro, R. Araújo, and A. Abelém, ''Securing light clients in blockchain with DLCP,'' *Int. J. Netw. Manage.*, vol. 29, no. 3, p. e2055, 2019, doi: 10.1002/nem.2055.

[14] P. Danzi, A. E. Kalor, C. Stefanovic, and P. Popovski, ''Delay and communication tradeoffs for blockchain systems with lightweight IoT clients,'' *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2354–2365, Apr. 2019, doi: 10.1109/JIOT.2019.2906615.

[15] A. Palai, M. Vora, and A. Shah, ''Empowering light nodes in blockchains with block summarization,'' in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5, doi: 10.1109/NTMS.2018.8328735.

[16] N. Alexopoulos, S. M. Habib, and M. Mühlhäuser, ''Towards secure distributed trust management on a global scale: An analytical approach for applying distributed ledgers for authorization in the IoT,'' in *Proc. Workshop IoT Secur. Privacy (IoT S&P)*. New York, NY, USA: ACM, 2018, pp. 49–54, doi: 10.1145/3229565.3229569.

[17] I. Eyal and E. G. Sirer, ''Majority is not enough: Bitcoin mining is vulnerable,'' in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2014, pp. 436–454, doi: 10.1007/978-3-662-45472-5_28.

[18] K. Nayak, S. Kumar, A. Miller, and E. Shi, ''Stubborn mining: Generalizing selfish mining and combining with an eclipse attack,'' in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 305–320, doi: 10.1109/EuroSP.2016.32.

[19] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, ''Optimal selfish mining strategies in bitcoin,'' in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2016, pp. 515–532, doi: 10.1007/978-3-662-54970-4_30.

[20] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, ''On the security and performance of proof of work blockchains,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, Oct. 2016, pp. 3–16, doi: 10.1145/2976749.2978341.

[21] M. Saad, J. Spaulding, L. Njilla, C. A. Kamhoua, D. Nyang, and A. Mohaisen, ''Overview of attack surfaces in blockchain,'' in *Blockchain for Distributed Systems Security*. Hoboken, NJ, USA: Wiley, 2019, ch. 3, pp. 51–66. [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119519621, doi: 10.1002/9781119519621.

[22] J. Joshi and R. Mathew, ''A survey on attacks of bitcoin,'' in *Proc. Int. Conf. Comput. Netw. Big Data IoT (ICCBI)*, in Lecture Notes on Data Engineering and Communications Technologies, A. Pandian, T. Senjyu, S. M. S. Islam, and H. Wang, Eds. Cham, Switzerland: Springer, 2020, pp. 953–959, doi: 10.1007/978-3-030-24643-3_113.

[23] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, ''Eclipse attacks on bitcoin's peer-to-peer network,'' in *Proc. USENIX Secur. Symp.*, 2015, pp. 129–144.

[24] Y. Marcus, E. Heilman, and S. Goldberg, ''Low-resource eclipse attacks on Ethereum's peer-to-peer network,'' in *Proc. IACR Cryptol. ePrint Arch.*, 2018, vol. 2018, no. 236, pp. 1–15.

[25] A. E. Yves-Christian, B. Hammi, A. Serhrouchni, and H. Labiod, ''Total eclipse: How to completely isolate a bitcoin peer,'' in *Proc. 3rd Int. Conf. Secur. Smart Cities, Ind. Control Syst. Commun. (SSIC)*, Oct. 2018, pp. 1–7, doi: 10.1109/SSIC.2018.8556790.

[26] A. Kiayias, A. Miller, and D. Zindros, ''Non-interactive proofs of proof-of-work,'' Cryptol. ePrint Arch., Rep. 2017/963, 2017. [Online]. Available: http://eprint.iacr.org/2017/963

[27] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, ''Flyclient: Super-light clients for cryptocurrencies,'' Cryptol. ePrint Arch., Rep. 2019/226, 2019. [Online]. Available: http://eprint.iacr.org/2019/226

[28] P. Danzi, A. E. Kalor, C. Stefanovic, and P. Popovski, ''Repeat-authenticate scheme for multicasting of blockchain information in IoT systems,'' in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2019, pp. 1–7, doi: 10.1109/GCWkshps45667.2019.9024468.

[29] M. Pustišek, A. Umek, and A. Kos, ''Approaching the communication constraints of Ethereum-based decentralized applications,'' *Sensors*, vol. 19, no. 11, p. 2647, 2019, doi: 10.3390/s19112647.

[30] A. Amoordon and H. Rocha, ''Presenting tendermint: Idiosyncrasies, weaknesses, and good practices,'' in *Proc. IEEE Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Feb. 2019, pp. 44–49, doi: 10.1109/IWBOSE.2019.8666541.

[31] R. Blum and T. Bocek, ''Superlight–a permissionless, light-client only blockchain with self-contained proofs and BLS signatures,'' in *Proc. IFIP IEEE Symp. Integr. Netw. Service Manage.*, Apr. 2019, pp. 36–41.

[32] V. A. Siris, P. Nikander, S. Voulgaris, N. Fotiou, D. Lagutin, and G. C. Polyzos, ''Interledger approaches,'' *IEEE Access*, vol. 7, pp. 89948–89966, 2019, doi: 10.1109/ACCESS.2019.2926880.

[33] D. Gruber, W. Li, and G. Karame, ''Unifying lightweight blockchain client implementations,'' in *Proc. Workshop Decentralized IoT Secur. Standards*. San Diego, CA, USA: Internet Society, 2018, pp. 1–8, doi: 10.14722/diss.2018.23010.

[34] S. Paavolainen and P. Nikander, ''Decentralized beacons: Attesting the ground truth of blockchain state for constrained IoT devices,'' in *Proc. Global IoT Summit (GIoTS)*, Jun. 2019, pp. 1–6, doi: 10.1109/GIOTS.2019.8766432.

[35] E. Hildenbrandt, M. Saxena, X. Zhu, N. Rodrigues, P. Daian, D. Guth, and G. Rosu. (Aug. 1, 2017). *KEVM: A Complete Semantics of the Ethereum Virtual Machine*. Accessed: May 17, 2019. [Online]. Available: https://www.ideals.illinois.edu/handle/2142/97207

[36] Y. Sompolinsky and A. Zohar, ''Bitcoin's security model revisited,'' 2016, *arXiv:1605.09193*. [Online]. Available: http://arxiv.org/abs/1605.09193

[37] N. Carter. (Aug. 5, 2019). *It's the Settlement Assurances, Stupid*. Accessed: Nov. 7, 2019. [Online]. Available: https : / /medium.com/nic__carter/its-the-settlement-assurances-stupid-5dcd1c3f4e41

[38] Payward. (2019). *Cryptocurrency Deposit Processing Times*. Accessed: Nov. 7, 2019. [Online]. Available: http://support.kraken.com/hc/en-us/articles/203325283-Cryptocurrency-deposit-processing-times

[39] P. Nikander, J. Autiosalo, and S. Paavolainen, ''Interledger for the industrial Internet of Things,'' in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, vol. 1, Jul. 2019, pp. 908–915, doi: 10.1109/INDIN41052.2019.8972167.

[40] G. O. Karame, E. Androulaki, and S. Capkun, ''Double-spending fast payments in bitcoin,'' in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: ACM, 2012, pp. 906–917, doi: 10.1145/2382196.2382292.

[41] K. Liao and J. Katz, ''Incentivizing blockchain forks via whale transactions,'' in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, and M. Jakobsson, Eds. Cham, Switzerland: Springer, 2017, pp. 264–279. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-70278-0_17 doi: 10.1007/978-3-319-70278-0_17.

[42] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge, U.K.: Cambridge Univ. Press, 2013, doi: 10.1017/CBO9781139226424.

SANTERI PAAVOLAINEN received the M.Sc. degree in computer science from the University of Helsinki, Finland, in 2016, and the B.Sc. degree in engineering physics from the School of Science, Aalto University, Finland, in 2019, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering. He enrolled to University of Helsinki, Helsinki, Finland, in 1993, but soon after got sidetracked and worked in the software industry for over two decades. During this time, he has worked extensively on scalable Internet service design, cloud computing, and technology strategy development. Eventually, he heard the siren song of academia. His research interests in the IoT devices and distributed ledger (i.e., blockchain) integration.

CHRISTOPHER CARR received the master's degree (Hons.) in cryptography and communication technology from the Royal Holloway University of London, in 2012, and the Ph.D. degree in Trondheim from the Norwegian University of Science and Technology, in 2014. During his time as a Ph.D. candidate, he took part in a televised research dissemination competition, was awarded a scholarship at the Queensland University of Technology and won finding for a startup project that was subsequently established into its own company. He is currently working in blockchain technology, amongst other areas, with the Norwegian University of Science and Technology. He also works as a Research and a Teaching Fellow with the University West of England, Bristol.