Supporting Lightweight Nodes in Permissionless Blockchains, Cheaply and Generically

Yuan Lu, Qiang Tang, Guiling Wang
New Jersey Institute of Technology, Newark, NJ 07102, USA
{yl768, qiang, gwang}@njit.edu

***Abstract*—We for the first time conduct a systematic study on lightweight nodes of permissionless blockchains, in the setting where full nodes and lightweight nodes are rational. Under such a game-theoretic model, we design a light client protocol that is the extremely "cheap", i.e., the cost of lightweight nodes to obtain a record in the blockchain is a small constant, and "generic", i.e., the protocol can be deployed in any blockchain supporting Turing-complete smart contracts. We prove a strong notion of the refinement of Nash equilibrium can be realized, such that: (1) for any full nodes who store the complete blockchain, their best strategy is always to faithfully relay the readings from the blockchain to the lightweight nodes; (2) for any lightweight nodes who rely on full nodes to "read" the blockchain, their dominating strategy is always to correctly instruct the blockchain what they receive from the full nodes, so essentially they will pay the relaying nodes as promised. We design a basic protocol for one relaying node and the advanced version for two non-colluding relays, which can be generalized to any n relays ($n \geq 2$) as long as at most $n-1$ relays collude.**

I. Introduction

The emerging blockchain technology, in particular the permissionless blockchain, as a nice resultant of cryptography, distributed consensus and economic incentive, enables an append-only digital ledger to be jointly maintained and consistently replicated across Internet peers. As such, attractive decentralized ecosystems can be envisioned to reduce the reliance on undesired third-parties, and various promising decentralized applications (DApps), such as novel financial instruments [1], storage outsourcing [2, 3], crowdsourcing [4, 5], digital good market [6] and so on, can be realized atop.

Generally, a blockchain can be considered as an abstracted layer (e.g. ideal public ledger), and users of higher layer applications can simply interact with it to read and write [7, 8]. This basic abstraction of *reading ledger* [1] implicitly requires a DApp user runs a so-called blockchain *full node* [9, 10] that keeps a complete blockchain replica, so the user can read the ledger locally; in addition, the full node also has to devote bandwidth and computing power to download and verify all *new* blocks, as it has to update the local replica accordingly to catch up the consensus agreed among the whole network.

With the rapid popularization of blockchain and DApps, many user clients unavoidably become browser extensions [11, 12], smartphones [4] or IoT nodes [13], which have restricted resources to store, download or verify the whole blockchain. As such, a huge demand of blockchain's *lightweight nodes* (which also known as lightweight clients or light clients) has risen sharply. For example, it is highly desirable that a retail cashier can use her mobile cryptocurrency wallet to verify the settlement of a transaction sent to her for a particular purchase [14]. This demand calls for a protocol specifically designed for lightweight clients, through which one can maintain high-security assurance to "read" the blockchain correctly without running the expensive consensus protocol. What's more, such a desirable protocol is also promising to resolve the critical problem of efficient "cross-chain" communication between two different chains (or among multiple chains): the peers of a blockchain can use few (on-chain) resources to collectively maintain a light client of an external blockchain, such that the applications atop the first chain can read the external chain efficiently [15–18]. Additionally, even for the resourceful DApps users, they have few motivations to set up full nodes to maintain the ledger, and might also look for the light client protocol, as their main purpose is only to use DApps.

The challenges of designing a light client friendly protocol reside in the fact that without a complete replica of the blockchain, a light client essentially has to rely on some full nodes to forward the readings from the blockchain. Many of the existing systems simply rely on some *trusted* relay nodes [19], where two threats emerge: (i) a dishonest relay, who particularly has monetary transactions with the light clients, may relay false result to cheat the light clients for money; (ii) different from being miners to earn rewards, the full nodes have little incentive to participate in such a relay service [20]. Current efforts on lightweight protocols focus on providing succinct cryptographic proofs to convince the light client of the correctness of the forwarded blockchain readings. The general idea is that a light client keeps small amount of consensus-dependent metadata bound to the correct branch of the blockchain (e.g. the proof-of-works of the longest chain in Nakamoto consensuses), and it then can verify a cryptographic proof attesting the existence of a particular blockchain record with using these metadata [1, 15, 21, 22].

The existing approaches have several inherent drawbacks. First, most of them require the light clients to store and keep updating some metadata to validate the forwarded blockchain reading, which is not realistic in resource-starved environments such as IoT sensors and the multi-chain scenarios [21]. Even worse, the required metadata are usually *consensus-dependent*,

[1]Remark that writing valid messages into the blockchain is trivial, as one can always gossip with some full nodes to diffuse its messages to the whole blockchain network (a.k.a. network diffuse functionality [7]). The blockchain liveness will further ensure the appearance of the messages in the ledger.

meaning that these approaches are always specifically designed for their underlying consensus protocols. For example, in proof-of-stake (PoS) type of consensuses, one of few current candidates [22] is particularly designed for Algorand [23], and is not suitable for the blockchains utilizing other PoS protocols [24, 25], which inevitably creates prohibitive obstacles in many interesting use-cases such as multi-chain clients.

"*How to realize a lightweight protocol that is generic as well as friendly to extremely resource-starved environments*" is still a valuable *open problem*.

Our contributions Given the inherent insufficiencies of the state-of-the-art solutions, we take a different path, and conduct a systematic study on the above open problem in game-theoretic settings to solve it via mechanism design. Assuming light clients and relaying full nodes are rational, we leverage smart contract to implement a simple incentive mechanism, such that being honest is their best choice, i.e., for their highest utilities, the relay full nodes must faithfully relay blockchain readings to lightweight nodes and lightweight nodes must pay the relay full nodes accordingly.

In greater detail, our protocol first comes with a one-time setup phase: through a trusted full node (which is reliable because it can be user's personal computer temporarily connecting the blockchain network), a light client publishes a dedicatedly designed incentive smart contract and verifies that some relay full nodes, who are interested in providing the service of relaying blockchain records, have made initial deposits in this contract. After the on-chain setup, the user can log off the trusted full node, and his/her light client can work independently, and repeatably ask the relay full nodes to forward the results of querying the blockchain whenever it needs. In each chain query, the light client first posts the specifications of her query to the smart contract via a blockchain transaction, which can be done easily because writing to the blockchain is trivial for any Internet node including the light client.[1] Via the same transaction, the light client also places a deposit in the contract to promise certain payment for the relay nodes. Later, the relay nodes see the above transaction in their local replicas, and are incentivized by the promised payment to forward the corresponding query results to the light client off-chain. After receiving the query results off-chain, the light client further reports them to the contract via another blockchain transaction; otherwise, gets a fine. Finally, the smart contract can verify the (in)correctness of each relay full node, and pays (or confiscates the initial deposit) accordingly. The amount of the required deposits, payments and economic punishments for cheating is finely tuned to achieve a desired Nash equilibrium to guarantee: (i) any rational full node will relay the blockchain readings correctly; and (ii) any rational lightweight node will report the blockchain to pay the full nodes as promised.

Our basic protocol relies on a single relay node, and requires this relay to place a deposit larger than the potential loss of the light client when cheated, which is inherent but not desirable. To remove this restriction, we further design an improved version of the protocol, using two non-colluding relay nodes to "audit" each other. We further extend the protocol to accommodate the scenarios where are $n > 2$ relays with a coalition size up to $n - 1$ nodes.

We remark that our protocol and incentive design can be applied to any blockchain supporting Turing-complete smart contracts, regardless of the underlying consensus protocol. In addition, the protocol is extremely efficient, since the computational cost of the light clients is a small constant.

Typical application scenarios Our protocol naturally supports a wide variety of applications, as it resolves the fundamental issue preventing low-capacity users from using blockchain.

Mobile wallet for multiple chains. Our protocol can be leveraged to implement a mobile wallet, particularly to support multiple blockchains, which can even run over different types of consensus protocols. The mobile wallet allows a lightweight node user to confirm the cryptocurrency transactions of various chains at a small and constant computational cost.

Decentralized public forum or blog. A public blog/forum DApp is another natural application scenario of our protocol. For example, a light client of Steemit [26] can get rid of any Web server, which is currently required to retrieve and display the blockchain contents. When surfing a decentralized forum without a Web server, users might frequently search contents by their keywords and jump there, which essentially requires the users to query the blockchain in many ways to obtain various blockchain records. All types of these chain queries will be supported by our light client protocol efficiently.

Marketplace for tokenized asset. To search a desirable cryptokitty [12] (which is an simplified example of tokenized digital asset) to buy, a mobile user usually searches in a marketplace hosted by a Web server, relying which to faithfully search the blockchains to find a correct kitty. Our protocol can naturally support a distributed marketplace by allowing a mobile user to efficiently "read" the ledger to figure out the kitties for-sale without relying a Web server.

II. Related Work

Lightweight node protocols The SPV client is the first light client protocol for Proof-of-Work (PoW) blockchain, proposed as early as Bitcoin [1]. Following the protocol, users need to download, verify and store a chain of block headers, and then can verify the existence of any transaction, with the help of other full nodes. The main weaknesses of SPV client is that the block headers to download, verify and store will grow linearly with the number of the blocks in the chain, which nowadays corresponds 40 MB in Bitcoin and more than 2 GB in Ethereum. For this reason, the concept of Proofs of PoW (PoPoW) was formulated [15, 27], through which one can store only the genesis block header, and then verify the existence of any transaction at a communication cost sub-linear to the length of PoW chain. All above schemes cannot be applied to proof-of-stake (PoS) blockchains, because in a PoS blockchain, the validity of a block relies on the signature(s) of stakeholder(s), whose validities further depend on the distribution of all “stakes” recorded in the blockchain ledger. Some fast bootstrap methods such as [22] was proposed for PoS light clients, but only works for a particular PoS chain [23]; worse still, the bootstrapping is still at a substantial cost which is unaffordable to most resource-constrained devices.

Vitalik Buterin [28] proposed to avoid forks in PoS chain by using incentives: a selected committee member will be punished, if she proposes two different blocks in one epoch. In this way, the lightweight node was claimed to be supported, as a light client who receives a fake block different from the correct one in the chain can send the malicious block back to the blockchain network to punish the creator of the cheating block. However, as the committees rotate periodically, it is unclear whether the protocol still works if the light client is querying about some “ancient” blocks that were generated before the relay node becomes a committee. Another recent attempt in [20] focused on implementing a prototype to provide incentives for relay full nodes, but it is not clear how to ensure the full nodes not forward fake blockchain readings.

Another line of studies propose the concept of stateless client [29, 30]. Essentially, a stateless client can validate new blocks without storing the aggregated states of the ledger. But it is still insufficient to help resource-starved users, because (i) a stateless client has to download all new blocks which consumes much bandwidth, and (ii) a stateless client cannot read ancient records in the blockchain.

Outsourced computations Reading from blockchain can be viewed as a special computation taking the ledger as input. Thus the general techniques of outsourcing computations may be adopted to support light clients. However, these general techniques do not perform well in the particular use-case.

Verifiable computation allows a prover to produce a cryptographic proof to convince a verifier that the output is obtained through correctly executing a pre-defined computation [31]. Recent constructive developments of zk-SNARKs [32] enables very efficient verification with the price that proving takes tremendous time and memory. For heavy tasks such as to prove the length (and even more complex properties) of a blockchain, it is usually infeasible in practice [21].

Attestation via trusted hardwares has attracted many attentions recently [33, 34]. However, recent Foreshadow attacks [35, 36] put SGX's attestation keys in danger of leakage, and potentially allow am adversary to forge attestations, which challenges the fundamental assumption of trusted hardwares.

Outsourced computation via incentive games have been discussed before [37, 38]. Some recent studies even consider the blockchain to facilitate such games to outsource computations [39, 40]. All these studies assume an implicit game mediator (e.g. the blockchain) who can speak to and listen to all involving parties, including the requester, the workers and the arbiter. However, in our setting, we have to resolve a special issue that there is no such a game mediator, since the blockchain, as a potential candidate of the mediator, can not speak to the light nodes.

III. Preliminary

Blockchain The permissionless blockchain can instantiate a public ledger [7], which is essentially an ever-growing linearized transaction log maintained by a network of untrusted Internet nodes collectively. An honest node in such a network, also known as an honest full node, keeps a blockchain replica consistent with other honest ones', by following a set of pre-defined rules called consensus protocol. In general, we can view a blockchain network as an ideal public ledger that possesses two critical properties:

- *Persistence* can enforce the convergence of the local replicas of all honest nodes, eventually.
- *Liveness* ensures any transaction to appear in the replicas of all honest nodes, eventually.

The two properties ensure that: (i) A full node, who joins and executes the consensus protocol, can “read” the blockchain ledger from its local replica and also “write” any valid transaction into the chain by broadcasting; (ii) Any node, who may not participate in the consensus protocol, is able to “write” valid transactions into the blockchain. In practice, the writing can be realized by gossiping with some full nodes to “broadcast” the transactions to all full nodes, which is known as the well understood network diffusion functionality [7, 8].

Smart contract Shortly after the emergence of blockchain, it was realized that this fantastic primitive can achieve much more beyond a bookkeeping ledger, as the transactions can also contain versatile program codes (a.k.a. smart contracts) besides monetary transfers. Especially when the same execution environment is shared within the network through the underlying consensus rules, the same executing results of these codes can be enforced by the whole blockchain network collectively [41]. More detailedly, the functionalities of a blockchain supporting Turing-complete scripts can be modeled as follows [8]:

- *Address*. By default, the sender of a transaction in the blockchain is referred to a pseudonym, a.k.a. address. Such an address is bound to a public key, and the security of digital signatures can further ensure that one cannot send messages in the name of a blockchain address, unless she owns the corresponding secret key.
  Also, a smart contract deployed in the blockchain can be referred by a unique address, such that the contract can hold cryptocurrency balance there. One can also call the contract to execute (to trigger a pre-specified condition to disperse the balance in the contract), through writing a transaction pointing to this unique address.
- *Reliable message delivery*. Any Internet node can leverage the blockchain to faithfully deliver messages to any blockchain full node in the form of transactions, which is immediately inhered from the *persistence* and *liveness* of blockchain. The only exception is that the blockchain cannot “send” messages to an Internet node who does not join the consensus, e.g., a lightweight node.
- *Correct computation*. The blockchain can be seen as a transactions-driven consensus computer, whose internal states are kept by each full node [42]. Moreover, these full

nodes (including miners) will persistently receive newly proposed block, and execute the computations (i.e. smart contracts) defined by the current states, with taking the block's transactions as inputs. Then the full nodes will update their blockchain's states according to the computing results. Moreover, consistent state updates are enforced among the whole blockchain network collectively.

Remark we require that the blocks' hashes can be read by smart contracts from the blockchain's internal states (i.e. available global variables) [43], which can be instantiated by the proposal of Andrew Miller [44] in practice.

Game, strategy, equilibrium and practical mechanism A game Γ in normal form consists of: (1) a set of all players denoted by $\mathbf{W} = \{W_1, \ldots, W_n\}$; (2) a space of players' pure strategies, denoted by $\mathbf{S} = \mathbf{S_1} \times \cdots \times \mathbf{S_n}$, in which $\mathbf{S_i}$ denotes player W_i's pure strategies, while a strategy σ_i of W_i can be chosen from his strategy space (possibly by a randomized way), and we call $\vec{\sigma} = \{\sigma_1, \ldots, \sigma_n\}$ a joint strategy of the players; (3) a utility function that defines the utility of each player under each joint strategy in $\mathbf{S}$. If there is a special joint strategy under which no player can expect better utility through unilaterally changing his strategy, it is also known as a Nash equilibrium. Standard Nash equilibria assume each player makes decision independently. More generally, we consider the collusion of a subset of players, i.e., a coalition. A rational coalition, in this paper, is assumed to maximize the sum of the utilities of all its members. Through the paper, we also apply a particular refinement of Nash equilibria, which survives iterated elimination of weakly-dominated strategies [45, 46], as a rational player likely will not use a strategy if there is a better alternative. In case the game Γ is dominance-solvable with having such a refined equilibrium $\vec{\sigma}$ in game Γ, we call $(\Gamma, \vec{\sigma})$ a *practical mechanism*, which is a widely-accepted security definition in game-theoretic settings [45, 46].

IV. Problem Formalization

A light client protocol involves a lightweight node and one or multiple full nodes. The lightweight node relies on the full nodes as relays to "read" the blockchain, and the full nodes receives the corresponding payments, if they forward correct chain readings. In our settings, all the participants are *rational*, i.e., they always choose a strategy to maximize their utilities.

Lightweight nodes have limited computing, communication and storage resources, so they can neither participate in the consensus protocol nor keeps a replica of the whole blockchain. They can always write valid transactions into the blockchain, via gossiping with some full nodes. The transactions will eventually be disseminated to the whole blockchain network, and then be written into the ledger, as same as any other transactions sent by any full node. This is guaranteed by the well abstracted underlying network diffusion functionality of the blockchain [7, 8]. When a lightweight node reads the blockchain, it relies on some full nodes to forward. To incentivize the full nodes to relay blockchain readings, it also promises some payments for the relays.

Full nodes have a copy of the up-to-date blockchain ledger by participating in the consensus protocol. When receiving a light client's request to forward a blockchain reading, a full node has several choices. It can do nothing, if it makes no benefit to be a relay but only consumes its own resources. It may also be motivated to send a forged query result for self-interest. For example, when a light client sells some products for cryptocurrencies and a full node is a buyer, the full node may choose to cheat the light client that a nonexistent cryptocurrency transaction is settled. So the full node can take the product for free, if there is no mechanism to punish cheating. Our game aims to provide effective incentive and punishment mechanism, and as a result, a rational full node chooses to be honest and collaborative for its best interest.

Parameters The parameters of our game theory model are illustrated as follows:

- p: The payment that the lightweight node agrees to pay for a correct blockchain reading result.
- m: How much a full node earns, if it successfully fools the lightweight node to believe a fake blockchain reading.

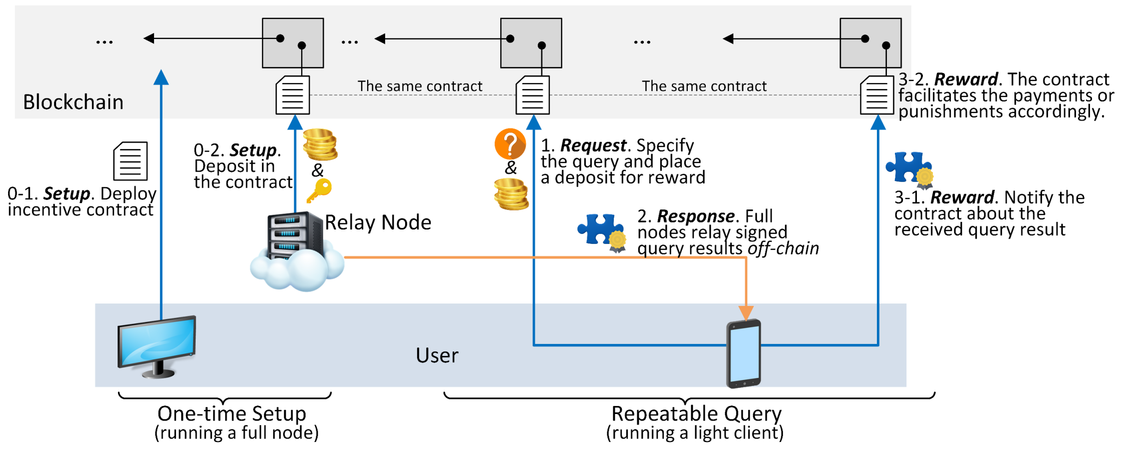
Note that when $m > p$, our game must be dedicatedly designed, since in this case, being a cheating relay node gets higher payoff than being an honest. Our game must ensure a relay node gets a penalty higher than the payoff of cheating, so a rational relay node will faithfully forward the chain readings.

Security goals We aim to design a light client protocol in the above game-theory settings to realize a *practical mechanism* [45, 46] that ensures two security goals:

- The full nodes always forward the correct results of reading the blockchain to lightweight nodes, so lightweight nodes can "query" the blockchain ledger correctly.
- The lightweight nodes always pay the relay full nodes correctly, according to a pre-specified immutable incentive policy, only if these relay full nodes faithfully forward the blockchain readings.

The above *practical mechanism* corresponds to a Nash equilibrium which survives iterative elimination of weakly dominant strategies (IEWDS), under which any deviation from our security goals will be irrational and then be deterred.

Other assumptions We apply the standard ideal public ledger model of blockchain [7, 8], which captures: (1) a transaction sent to it will eventually be included by the blockchain, i.e., the message delivery via the blockchain is partially asynchronous; (2) the smart contract can be any Turing-complete script, e.g., it can verify a Merkle tree proof to attest a leaf is committed in a Merkle tree root [17, 18]. Also, we assume that a light client and a full node can establish a secure off-chain communication channel on demand. In practice, this can be done through either a third-party name service or through "broadcasting" encrypted network addresses via the blockchain [47]. Also, the off-chain communications are partially asynchronous, i.e., a message sent off-chain delivers eventually, e.g., even if a full node temporarily crashes down, it can eventually be back online and deliver messages to the light client.



Blockchain
The same contract
The same contract
3-2. ***Reward***. The contract facilitates the payments or punishments accordingly.
0-1. ***Setup***. Deploy incentive contract
0-2. ***Setup***. Deposit in the contract
&
Relay Node
1. ***Request***. Specify the query and place a deposit for reward
&
2. ***Response***. Full nodes relay signed query results *off-chain*
3-1. ***Reward***. Notify the contract about the received query result
User
One-time Setup
(running a full node)
Repeatable Query
(running a light client)

Fig. 1. The schematic diagram of the protocol. The user runs a full node in the setup phase, and later logs off the full node to work independently as a lightweight client. Blue line indicates to write into the blockchain, while orange line represents the off-chain communication.

V. Protocol: One Relay Node

Intuition Our protocol guarantees a relay full node gets the payment as promised by a light client, if forwarding the correct blockchain reading, or gets punished if cheating the light client by relaying a fake result. To ensure the relay full node gets punished if cheating, the protocol requires it to place a deposit d in a dedicatedly designed smart contract, which confiscates the deposit if it cheats. To ensure the relay full node gets paid if forwarding correctly, the game requires the light client to place a deposit in the contract for each query. The incentive smart contract is the "arbiter" to evaluate whether the relay node correctly forwards or cheats, and subsequently determines whether to pay or to punish it.

Note that the relay full node forwards the blockchain reading to the light client off-chain, which implies that the contract has to rely on the light client to report what it receives from the relay full node; otherwise, the contract has no idea what the relay full node really did, and therefore cannot facilitate the correct payment/punishment. At first glance, the light client can claim it receives nothing or a fake result from the relay full node, so it can avoid paying. To enforce a light client faithfully report to the contract whatever it receives, our protocol requires: (1) the contract will forfeit the light client's deposit, if the light client does not report anything; (2) the full node shall sign the reading before sending it to the light client, so the contract later can verify the result is indeed from the full node by checking the digital signature.

To instantiate the above idea, our protocol first runs a setup phase, and then the light client can repeatedly ask the full node to forward blockchain readings. The whole procedure is briefly illustrated in Fig.1, and the details are presented below.

A. Protocol details

One-time setup phase In the setup phase, the user of a lightweight node LWN runs a full node, and announces a smart contract codifying a dedicatedly designed incentive mechanism. After the contract is deployed, a relay full node, denoted by RFN, chooses to join the protocol by depositing an amount of d in the contract. Its corresponding public key, denoted by pk, is also sent and recorded by the smart contract. The setup phase is closed after RFN joins with making the deposit; at the same time, LWN records RFN's pk, and logs off the full node mode.

In practice, the setup can be done when LWN user is running a full node in her PC, and her light client (e.g. a smartphone) connects to the PC to sync. Note that many fast bootstrap methods [15, 22, 48] can help our setup phase, through which a LWN user can quickly launch a full node catching up the up-to-date consensus.

Repeatable query phase Once the setup phase is done, LWN can request RFN to relay chain readings repeatedly. The details in each query are presented as follows:

1) *Request*. For each query, LWN firstly sends a transaction to the preset smart contract, which encapsulates detailed specifications of the query along with a deposit (denoted by $p + e$, where p is the promised payment, and e is a positive number) for this query. We denote the query Q. A typical example query could be asking the current balance of a given blockchain address. [2]
2) *Response*. RFN can learn the query Q and the settlement of the deposit $(p + e)$ from its local blockchain replica. It conducts the query over the blockchain and obtains the result denoted by R (which for example can be the current balance of an account given by Q). When

[2]The queries supported by our protocol are restricted by the ground-truths that can be verified by a smart contract. In practice, a Turing complete smart contract can efficiently verify any transaction or any authenticated state of the blockchain. For example, in Ethereum (PoW) or Cardano (PoS), a smart contract can trivially check the current balance of a given blockchain address [43]. To facilitate more complicated queries (e.g. the existence of a transaction or other record), one can let the contract compute a few hashes to verify a so-called Merkle tree proof [1]. Note that this paper does not focus on implementing these trivial contracts.

RFN relays R to LWN off-chain, [3] it also attaches the digital signature σ generated for R using the secret key corresponding to its public key pk. Upon receiving the forwarded R, LWN checks the validity of σ using pk, and only accepts R with a valid σ.

We remark that LWN verifies the signature of each query result relayed by RWN, and omit this step in later context for presentation simplicity. When we mention "LWN receives a query result R", implicitly LWN receives R with a valid signature, though R can either be correct or wrong.

3) *Reward.* Upon the light client LWN receives the forwarded blockchain reading, it will report the smart contract (R, σ) by sending another transaction to the contract; or it simply does nothing if receiving nothing [4]. Then the smart contract will first verify σ under pk, and then checks the equity between R and the ground truth of query Q.[2] As such, the incentive mechanism pre-specified in the smart contract can be enforced according to the correctness of R. In details, the clauses of the incentive mechanism are pre-defined as follows:
   - **Clause 1**. When LWN reports a correct R (which also is correctly signed by RFN), e is returned to LWN, and p is paid to RFN;
   - **Clause 2**. When LWN reports a validly signed false R, $e + p$ is returned to LWN, and the deposit d of RFN is sent to LWN, if d is still placed in the contract;
   - **Clause 3**. Otherwise, LWN is reporting the contract nothing, and the contract just holds the deposit $p + e$, until LWN reports.

B. Game and analysis

In this subsection, we analyze the conditions under which our protocol design can achieve the security goals presented earlier. Particularly, we focus on the repeatable query phase, when the lightweight node LWN sequentially asks the relay node to forward blockchain readings for k times. [5] We denote such a game of k repeated queries as Γ_1. Briefly speaking, in each of the k queries, the relay full node RFN has three possible actions: (1) relay the correct query result to LWN, denoted by c; (2) relay a forged query result to cheat LWN for benefit m, denoted by f; and (3) send LWN nothing, denoted n. Correspondingly, when RFN takes the action c or f, LWN has two possible actions in response: (1) report to the smart contract what it receives, either it is some results or nothing, denoted by a; (2) do not report to the smart contract, denoted by n. When RFN takes the action n during the query, LWN has only one action: report the smart contract what it receives, i.e., nothing, which can also be denoted by a.

We prove that when the deposit d placed by the participating RFN is large enough, we can achieve the security goals in the game Γ_1, i.e., a rational RFN always chooses the strategy of "sticking with forwarding correct blockchain readings during all k queries", and a rational LWN chooses the strategy of "always reporting the blockchain whatever it receives from RFN in all queries". The formal security analysis is shown as follows.

Lemma 1. *In Γ_1, the strictly dominant strategy of LWN is always to report the incentive contract whatever received from RFN in all k queries, i.e., a rational LWN will follow this strategy and not deviate from it in Γ_1.*

Proof. Given any pure strategy of RFN, denoted by $\vec{s} = (s_1, \ldots, s_k)$. No matter what $\vec{s}$ is, the lightweight node will have less payoff, if s/he does not use the pure strategy $\vec{\sigma} = (\sigma_1 = a, \ldots, \sigma_k = a)$. Consider the situations that LWN deviate from the strategy $(a, \ldots, a)$ in l stages, where $l < k$. (Note that LWN cannot deviate from a in a stage where RFN relays nothing.) We have the following observations: (1) if in these stages, RFN relays all correct query results, then the deviations in these stages will cause LWN suffer from a utility loss of el; (2) if in these stages, RFN relays at least one false query result, then the deviations in these stages still will cause LWN suffer from a utility loss, that has a lower bound of el and a upper bound of $el + d$. So the dominating strategy of LWN is $\vec{\sigma} = (\sigma_1 = a, \ldots, \sigma_k = a)$. [6] □

Lemma 2. *In Γ_1, conditioned that the i-th stage is the first time that RFN takes the action of f, i.e., relays a false query result, then its strictly best strategy under these conditions should be "send correct query results before i-th stage, and send false query result after i-th stage", and the utility of RFN will be $(i-1)p + (k-i+1)m - d$.*

Proof. Under that condition that the i-th stage is the first time when RFN deviates by sending a false query result, RFN can take actions c or n before i-th stage, while it can choose c, f or n after i-th stage. Consider the benefit of cheating RFN (i.e.,m), is larger than the payment for each query (i.e. p), and also consider Lemma.1, it is clear to see that the best strategy is to take c before i-th stage and take f after. It is trivial to show the best utility is a function of i as $(i-1)p + (k-i+1)m - d$. □

Theorem 1. *Under the condition that $kp > km - d$, there is a practical mechanism via Γ_1 such that: (1) the lightweight*

[3] Note that we assume the off-chain communication can be established on demand in the paper, which in practice can be done through a name service or "broadcasting" encrypted network addresses through the blockchain [47].

[4] Remark that our protocol actually works under partial asynchronous network. Such a network condition will ensure that: if the relay node indeed forwards, the lightweight node will receive eventually; if the relay full node does not relay, the lightweight node might wait, but our analysis later will show this case can be deterred, because a rational full node always forwards the blockchain readings eventually, which terminates our protocol for sure.

[5] Remark k is a-priori known by LWN and RFN, which can be understood as the incentive contract only allows up to k times queries for each one-time setup. Also, to ensure the lightweight node eventually queries k times, a deposit shall be placed in the contract during the setup phase, which is refundable only if k queries are requested.

[6] More precisely, $(\sigma_1 = a, \ldots, \sigma_k = a)$ shall be denoted as $(\sigma_1 = aa, \ldots, \sigma_k = aa)$ by convention of game theory literature, where aa indicates a pure strategy "taking action a when RFN takes c or f as well as taking action a when RFN takes n". Remark that we use a as an abbreviation for "taking action a no matter what action RFN takes" through this paper.

node always receives the correct query result forwarded by the full node, and (2) the full node always receives what the lightweight node promises to pay.

Proof. Lying on Lemma.1 and Lemma.2, it is clearly true. □

Intuitively, Theorem.1 tells us that a light client can repeatedly ask the full node to relay the blockchain readings after a one-time on-chain setup. When the potential loss of the light client gets fooled is no greater than the deposit placed by the relay node during the setup phase, our protocol is effective to help the light client to query blockchain records that are not that "valuable".

Computational efficiency. To make a query of reading the blockchain, a lightweight node only needs to store one public key, send two blockchain transactions [7], instantiate a secure channel between the relay full node, and verify a signature. The involved computation and communication cost is very low. From the on-chain perspective, the only workload is to execute a simple incentive smart contract, where the payment/punishment clauses only require few light operations (e.g., to get the current balance of an account in Ethereum [43]). [8]

VI. Protocol: Two Non-Colluding Relay Nodes

When there is only one relay node, to achieve the security goals, the deposit required for this relay node could be too high for a relay node to be comfortable to participate in the protocol, or a light client has to send some queries that will not cause substantial loss, e.g., ask the existence of transactions that are smaller than a certain amount or are inherently low valuable (e.g. a blog or forum thread recorded by the blockchain). In this section, we propose an advanced version of the protocol which employs two non-colluding relay nodes to "audit" each other, so as to remove the restriction of the basic protocol.

Intuition We introduce a protocol utilizing two non-colluding full nodes as relays, denoted by RFN_1 and RFN_2, respectively. If and only if the same reading is forwarded by both RFN_1 and RFN_2, the light node LWN believes the blockchain reading. In this way, LWN leverage RFN_1 and RFN_2 to "audit" each other, and the deposits required of these two relays can be greatly reduced.

There is one subtlety in this setting: the two relay nodes could be non-colluding, but there may exist some trivial strategy that both of them outputs some constant, say 0, as the query result. Since the reading from both relay nodes are the same, the lightweight node can be fooled and believes it is a correct reading. This raises an alarm of the applicability of the protocol involving two non-colluding relay nodes: if there exist some trivial common constants, the relay nodes might choose them to fool the lightweight node, as they get benefits larger than the deposits placed.

We argue for many use-cases, the two non-colluding relays will not share any common information on how to cheat LWN for money, and advocate that checking such a condition is necessary before applying this second protocol. In this way, forwarding the correct chain reading will still be the dominating strategy for each relay node. These conditions may not be satisfied for arbitrary query over the blockchain, but can be satisfied in a wide range of lightweight node's queries per se. For example, LWN asks for "recent transactions sent to me" (which is one major query for wallet applications), and RFN_1 wants to cheat LWN of the existence of some fake transactions sent by Eve (denoted by T_1), while RFN_2 wants to cheat LWN of the existence of some non-existing transactions signed by $Mallory$ (denoted by T_2). In this case, $\mathsf{T}_1 \cap \mathsf{T}_2$ can be seen as empty set as the two relay nodes do not share an blockchain account. Take another example: LWN might ask for "the IDs of the kitties on sale that have some particular attributes" (which is a keyword query of the cryptokitties marketplace), and RFN_1 wants to cheat LWN at a set of its kitties (denoted by S_1), while RFN_2 wants to cheat LWN at a different set of kitties (denoted S_2). In that example, we argue that $\mathsf{S}_1 \cap \mathsf{S}_2 = \emptyset$ since the non-colluding RFN_1 and RFN_2 do not share the ownership of any kitty.

To ensure the applicability of the protocol, a general technique can be applied to slightly modify the queries: when the lightweight node LWN wants a particular reading from the blockchain, the query can be "*to fetch a block header containing my reading result, along with a Merkle tree proof to attest its existence in this block header*". The relay full nodes cannot work out the same fake query result to cheat LWN for money, without coordination. As illustrated in Fig.2, due to the collision resistance of cryptographic hash function, the two relays sharing no common information on how to cheat LWN for money cannot counterfeit the same block headers. The only exception can be that they forge two same empty blocks, but an empty block cannot cheat anything for their self-interests nor can make any benefit.

Remark that if a query falls out of the above wide range of use-cases, a lightweight node can simply employ the basic version of the protocol.

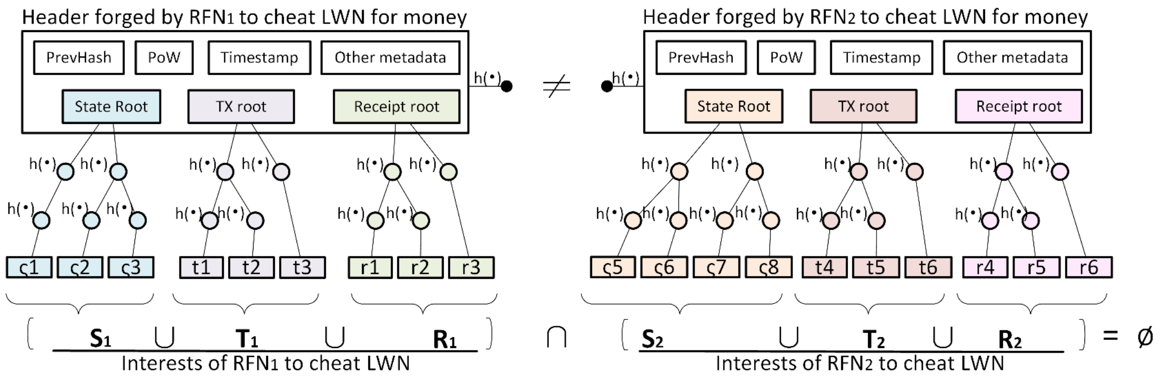
A. Protocol

During the setup phase, the protocol runs similarly to the case of one relay node. The only difference is the incentive contract published by the lightweight node user is dedicatedly designed to recruit two relay full nodes to "audit" each other. Each of the two relay nodes shall place a deposit d in the preset smart contract. Later in the section, we will show the required amount of d can be significantly reduced.

Upon the completion of the setup, the lightweight node can repeatedly request the relays to forward blockchain readings off-chain. For each query, the protocol runs in a way similar to the case of one relay node: The lightweight node specifies the query and deposit $p + e$ to the contract via a blockchain

[7]Remark that sending a transaction to the blockchain can be done through a few gossiping messages to some randomly chosen blockchain nodes, whose cost is very low [49].

[8]To support any reading from the blockchain, the smart contract might need to verify a so-called Merkle tree proof with computing some hash values [16–18], which is still cheap. We defer the formal discussion on that to Section.VI.



Header forged by RFN1 to cheat LWN for money
PrevHash
PoW
Timestamp
Other metadata
State Root
TX root
Receipt root
h(•)
ς1 ς2 ς3 t1 t2 t3 r1 r2 r3
S1 ∪ T1 ∪ R1
Interests of RFN1 to cheat LWN
≠
∩
= Ø
Header forged by RFN2 to cheat LWN for money
ς5 ς6 ς7 ς8 t4 t5 t6 r4 r5 r6
S2 ∪ T2 ∪ R2
Interests of RFN2 to cheat LWN

Fig. 2. The schematic diagram to show two forged block headers including no conflict interest on how to cheat the lightweight node for money. Remark that we borrow Ethereum's block header structure where are three Merkle tree roots. Transactions, some aggregated states, and receipts (a.k.a. logs) [41] are committed to the three Merkle roots, respectively. The aggregated states can be derived through all historic transactions recorded in the ledger.

transaction. Then RFN_1 and RFN_2 relay signed query results R_1 and R_2 to LWN off-chain, respectively. [3] Finally, the lightweight node reports the contract what it receives from the two relay nodes. The three major differences from the basic protocol are listed as follows:

First, we require relay nodes send the query result R with a block header H that shall include R, plus a Merkle tree proof π to attest the existence of R in H, [9] as opposed to the case of one relay where only chain reading R is sent. As discussed earlier in this section, if the non-colluding nodes have no common interest of cheating LWN for money, they cannot predicate a common fake block header without coordination.

Second, the contract verifies the correctness of a blockchain reading in a more complicated way. Note that the contract receives a tuple $(\mathsf{H}, \mathsf{R}, \pi, \sigma_i)$ from LWN, where H is a block header, R is the blockchain reading, π is Merkle tree proof attesting that R is in H, and σ_i is the signature of RFN_i for $(\mathsf{H}, \mathsf{R}, \pi)$ $(i \in [1, 2])$. The contract first verifies σ_i using relays' public keys to ensure LWN faithfully reports what it receives. If failing to verify the signature, the contract simply reverts. Otherwise, the contract further: (1) computes the hash of H, and checks that hash is same to the correct one provided by the blockchain's globally available variables [43, 44]; (2) verifies that Merkle tree proof π correctly attests that R is committed in H. If both conditions satisfy, the contract realizes that RFN_i forwards a correct blockchain reading to LWN; otherwise, the contract knows RFN_i forwards a false result.

Last, the incentive mechanism pre-defined in the smart contract is dedicatedly designed for the case of two full node. The clauses of the incentive contract are:

- **Clause 1**. when LWN reports two correct results, e is returned to LWN, and each full node gets $p/2$;
- **Clause 2**. when LWN reports two false results, $e+p$ is returned to LWN, and the deposits of both full nodes are sent to LWN, if both deposits are still in the contract;
- **Clause 3**. when LWN reports a correct and a false, $e + p/2$ will be returned to LWN, $p/2$ will be paid to the honest full node; if deposit d of the cheating full node is still in the contract, LWN gets $d/2$; for the other $d/2$, if the deposit d of this honest full node is also placed in the contract, the honest full node gets the other $d/2$, and otherwise the $d/2$ of the cheating full node's deposit is burnt [10];
- **Clause 4**. if LWN reports a correct result only, $e/2$ is returned to LWN, $p/2$ is paid to the honest full node, and the remaining deposit of LWN $(p/2+e/2)$ is burnt;
- **Clause 5**. if LWN reports only a false result, $e/2+p/2$ is returned to LWN, the remaining deposit $p/2 + e/2$ of LWN is burnt, and the deposit of the cheating full node will be taken, half of which is paid to LWN and the other half is burnt;
- **Clause 6**. otherwise, LWN is reporting nothing; the contract holds the deposit $p + e$ until LWN reports.

Note that if the contract detects a full node sends a fake result to the lightweight node, the full node might get paid if acts honestly in next queries, but it loses the chance to earn the potentially confiscated deposit of the other full node.

B. Game and analysis

In this analysis, we particularly focus on the case LWN repeatedly requests k blockchain readings [5] and denote the game representing these k queries as Γ_2. In short, we prove that the desired security can be realized in the game Γ_2, i.e., a rational LWN chooses the strategy of "always reporting the blockchain whatever results relayed by the full nodes during k queries", and rational full nodes always choose the strategy of "forwarding the correct blockchain readings during all k queries". The detailed discussions are as follows.

[9]Remark that Merkle tree is a basic data structure to organize records (e.g. transactions or states) in blockchains [1, 41]. A so-called Merkle tree proof can be used to attest a record is committed to the Merkle tree's root; the verification of such a proof only requires the Merkle root and costs a few hashes [1, 17].

[10]Remark cryptocurrencies can be "burnt" by many ways in practice, e.g., sending them to a "randomly" chosen blockchain address [50], or sending them to a special contract where no conditional payment clause is specified.

Lemma 3. *In Γ_2, there is a strictly dominant strategy of the lightweight node LWN, that is to report the smart contract whatever it receives from the two relay nodes in all queries.*

Proof. No matter what strategies used by the two full nodes in the k queries, LWN has worse utility, if it arbitrarily deviates from using the strategy $(a, \dots, a)$, which can be clearly seen from the utilities determined by the clauses of the incentive contract. Actually, the loss of utility caused by any deviation includes (1) the loss of the prepaid e or part of e and (2) the loss of the possible chance of getting the confiscated deposit of the full node(s). □

Lemma 4. *In Γ_2, conditioned on that RFN_1 and RFN_2 share no common information to cheat LWN and that deposit d of relay node is non-negative, there is a strictly dominant strategy of RFN_1 to always relay the correct query results to LWN during all the k queries, no matter what a strategy is used by RFN_2.*

Proof. As the rational LWN always uses its strictly dominant strategy (c.f. Lemma.3) to report the contract whatever the full nodes relay to it, when RFN_1 makes any deviation from the strategy of always sending the correct query result, it has a loss of its utility, even if RFN_2 uses an arbitrary strategy, which can be seen by checking the utilities determined by the incentive clauses. RFN_1 will have a utility loss for the following reasons: (1) since LWN does not deviate, any deviation of RFN_1 will cause the loss of deposit d; (2) any deviation causes RFN_1 to lose the opportunity of getting the confiscated deposit of RFN_2; (3) any deviation causes the loss of lightweight's payments; and (4) any deviation provides no help to get any additional earnings, because RFN_2 does not collude with it to forge the same query result to cheat LWN. □

Theorem 2. *There is a practical mechanism via Γ_2 such that: (1) a lightweight node always gets the correct results from both relay full nodes in k queries; (2) relay full nodes receives the promised payment in all queries, after relaying the correct results.*

Proof. Considering Lemma.3, Lemma.4, and that RFN_1 and RFN_2 are interchangeable notations, the theorem is proved automatically. □

Computational efficiency. The cost of a light client to "read" the blockchain is constant and essentially very small. The required operations include to store two public keys, to send the blockchain two transactions, to instantiate two secure channels to receive query results from two relays, to verify two signatures and to compute a few hashes to verify Merkle tree proof. For the on-chain efficiency, the only workload is to keep a simple incentive smart contract, where the incentive clauses only require some light operations, e.g., to verify a Merkle tree proof with computing a few hashes [17, 18].

C. Extension: n relay full nodes

The above protocol relies on an assumption that the two relay nodes are independent, which can be invalidated by a malicious node forging multiple identities in practice. Many well-known authentication methodologies [4, 51] can be employed to prevent an adversary from forging unlimited identities, and therefore can enforce the full nodes to authenticate in the smart contract when joining the protocol in the setup phase. However, there still is a realistic worry that an attacker can possess few valid identities to successfully authenticate few times to control few relays nodes (e.g. two relays) during the setup phase, which motivates us generalize our protocol to leverage n relays, where n can be a number larger than the identities controlled by the adversary.

We consider a more general setting: n full nodes join the protocol and $n-1$ out of them can arbitrarily coordinate (which is the optimal coalition if we do not allow a coalition consisting of all n relay nodes) to form a coalition denoted by $\mathbf{C}$,[11] and only one full node, denoted by RFN, does not collude with $\mathbf{C}$. We reasonably assume that $\mathbf{C}$ and RFN will share no common interest for a wide range of queries, e.g., when a lightweight node requests to "find the transactions sent to a give blockchain address" or "find the for-sale kitties with short cooldown duration", it is very likely that different malicious nodes will cheat the light client in different ways if without collusions.

The protocol generalized for n relay nodes is basically the same as the one of using two relays, except a generalized incentive mechanism. In the on-chain setup phase, the light client runs a trusted full node to publish an incentive smart contract where each relay full node shall make a deposit d. After that, the light client can repeatedly ask the full nodes to query the blockchain and forward the query result. For each query, the light client needs to send the blockchain a transaction to place a deposit of $p + e$ promising a payment of p for all n full nodes. The light client only believes in the correctness of a query result if all the n relay nodes forward the same. The generalized incentive mechanism for each query is the crux of the protocol using n relay nodes, and is illustrated as follows.

- **Clause 1**. In each query, if LWN reports i correct and j incorrect results ($i + j = n, i >= 0, j >= 0$), each of the i honest full nodes get p/n as the payment and LWN gets $j \times p/n + e$ back out of its deposit $p + e$. All unconfiscated deposits of the j cheating full nodes, denoted by D (which is less or equal to $j \times d$), is given to the honest full nodes (whose deposit are still unconfiscated) and the lightweight node, or is burnt. More specifically, each honest full node (whose deposit d is still in the contract) and LWN will get $D/(i+1)$. If there are honest full nodes whose deposits have been confiscated before, there is a reminder of D, which will be burnt.

[11]A coalition in our context is rational and always aim to maximize its total utilities of all its coalition members.

- **Clause 2**. In each query, if LWN reports i correct and j incorrect results ($0 < i + j = k < n, i >= 0, j >= 0$), each of the i honest full nodes gets p/n payment (which in total is $i \times p/n$), LWN gets $k \times e/n + j \times p/n$ returned out of its deposit $p + e$. The unconfiscated deposits D of all cheating full node is taken by the honest full nodes whose deposits are still hold, and the LWN, or will be burnt. More specifically, each honest full node (whose deposit d is still in the contract) and LWN will get $D/(n - j + 1)$, and the rest of D will be burnt.
- **Clause 3**. Otherwise, LWN reports nothing. The contract holds the deposit $p + e$, until LWN reports.

Note that if the contract verifies that a relay node forwards a fake result in a query, this full node will still get paid if being honest in the following queries, but cannot earn the potentially confiscated deposit of other cheating relays anymore.

The above incentive design is generalized to use n relay nodes. We consider the game where a lightweight node repeatedly asks n full nodes to forward blockchain readings for k times, and denote such a game as Γ_n. Note that the light client believes a blockchain reading, if it is forwarded by all n relays. The security analysis is intuitively similar to the case of two relay nodes. As the optimal coalition $\mathbf{C}$ which has $n-1$ full nodes can be conceptually viewed as first relay full node, and the only one relay full node who does not join the coalition $\mathbf{C}$ is essentially the second one. Specifically, we prove that a practical mechanism exists in the above stringent setting, such that our security goal can be realized to guarantee: (1) all full nodes will forward correct chain readings during all k queries, and (2) all full nodes will get correctly paid in the k queries. The detailed security analysis is as follows.

Lemma 5. *In Γ_n, the strictly dominant strategy of the lightweight node LWN is to report the smart contract whatever receiving from the n relay nodes in all k queries.*

Proof. No matter what strategies used by the full nodes, the lightweight suffers from less utility when it arbitrarily deviates from the strategy $(a, \ldots, a)$ because: (1) any deviation leads up to the loss of all or part of the prepaid e; (2) any deviation causes the possible loss of getting confiscated deposits of false-reporting full nodes. $\square$

Lemma 6. *In Γ_n where are a coalition $\mathbf{C}$ of $n - 1$ relays and one out of $\mathbf{C}$ relay RFN, conditioned on that RFN and the nodes in $\mathbf{C}$ share no common interest of cheating LWN and the deposit d is non-negative, there is a strictly dominant strategy of RFN to always relay the correct query results to LWN during all the k queries.*

Proof. If the relay nodes in $\mathbf{C}$ use arbitrary strategies and LWN uses its dominant strategy (c.f. Lemma.5), when RFN makes any deviation from the strategy of always sending the correct query result, it has a loss of its utility. This is because (1) any deviation causes the loss of deposit; (2) any deviation causes the loss of the opportunity of getting the confiscated deposit of the other full node; (3) any deviation causes the loss of lightweight's payments; and (4) any deviation does not bring any malicious earnings. $\square$

Lemma 7. *In Γ_n where are a coalition $\mathbf{C}$ of $n - 1$ relays and one out of $\mathbf{C}$ relay RFN, conditioned on that RFN and the nodes in $\mathbf{C}$ share no common interest of cheating LWN and the deposit d is non-negative, there will be a dominant strategy of the coalition $\mathbf{C}$, which is to enforce all coalition members to always relay correct query results to LWN.*

Proof. According to Lemma.5 and 6, the rational coalition $\mathbf{C}$ will not deviate from "sending the lightweight correct queries in k queries", because any deviation will cause: (1) the loss of payments; (2) the loss of all or part of deposits; (3) the loss of the opportunity to get the probably confiscated deposit of RFN; (4) no chance to fool LWN and earn extra benefits. $\square$

Theorem 3. *In Γ_n where are a coalition $\mathbf{C}$ of $n - 1$ relays and one out of $\mathbf{C}$ relay RFN, conditioned on that RFN and the nodes in $\mathbf{C}$ share no common interest of cheating LWN and the deposit d is non-negative, there is a practical mechanism such that: (1) all full nodes will relay the correct query results in all k queries; and (2) the lightweight node will always report the contract whatever it receives, such that all full nodes who faithfully forward will get the correct payments as pre-specified in the incentive contract.*

Proof. Considering Lemma.5, 6, and 7, the only joint strategy that can survive from the iterative deletion of dominated strategies is the one presented in the above theorem. $\square$

VII. Conclusion and Open Problems

We for the first time design and formally analyze a light client protocol of public blockchains that is secure in game theory model. Our design is the first lightweight blockchain protocol that is extremely friendly to the resource-starved environments as well as compatible with any blockchain regardless of the underlying consensus protocol.

Open problems Since this is the first work that formally discusses the lightweight clients of permissionless blockchains in game theoretic settings, the area remains largely unexplored. A few potential future works can be done. First, this paper does not consider the details of recruiting relay nodes (and assumes no Sybil attacker can control all relay nodes in the discussion about multiple relay nodes), otherwise the lightweight node has to enforce the relay nodes to place large enough deposit. So a Sybil-proof incentive mechanism seems to be an interesting extension to deter such Sybil attackers. Second, we currently rely on that there is at least one relay node who does not collude with others, otherwise the relays must place considerable deposits. Although one can realize this assumption by recruiting many more relays, a more enticing way is to design a mechanism to naturally deter the collusion. Last but not least, some reputation based mechanisms could be envisioned to ensure the long-term goodness of lightweight nodes and relay nodes, such that large deposits can be avoided.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proc. IEEE S&P 2014*, pp. 475–490.

[3] Protocol Labs, "Filecoin: A Decentralized Storage Network," 2017. [Online]. Available: https://filecoin.io/filecoin.pdf

[4] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 853–865.

[5] Y. Lu, Q. Tang, , and G. Wang, "On Enabling Machine Learning Tasks atop Public Blockchains: a Crowdsourcing Approach," in *Proc. IEEE ICDM Workshops (BlockSEA) 2018*.

[6] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proc. ACM CCS 2018*, pp. 967–984.

[7] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 2015, pp. 281–310.

[8] A. Kosba, A. Miller, E. Shi *et al.*, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE S&P 2016*, pp. 839–858.

[9] Bitcoin Core. [Online]. Available: https://github.com/bitcoin/bitcoin

[10] Go Ethereum. [Online]. Available: https://github.com/ethereum/go-ethereum

[11] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via bitcoin," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 393–409.

[12] CryptoKitties. [Online]. Available: https://www.cryptokitties.co/

[13] S. Underwood, "Blockchain beyond bitcoin," *Commun. ACM*, vol. 59, no. 11, pp. 15–17, Oct. 2016.

[14] Steve Fiorillo. How to Use Bitcoin for Purchases. [Online]. Available: https://www.thestreet.com/investing/bitcoin/what-can-you-buy-with-bitcoin-14556706

[15] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," 2017. [Online]. Available: https://eprint.iacr.org/2017/963.pdf

[16] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," 2014. [Online]. Available: http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains

[17] BTC Relay. [Online]. Available: https://github.com/ethereum/btcrelay

[18] A. Kiayias and D. Zindros, "Proof-of-work sidechains," 2018. [Online]. Available: https://eprint.iacr.org/2018/1048

[19] Metamask. [Online]. Available: https://metamask.io/

[20] G. K. Damian Gruber, Wenting Li, "Unifying lightweight blockchain client implementations," in *Workshop on Decentralized IoT Security and Standards (DISS)*, 2018.

[21] M. Z. Loi Luu, Benedikt Bnz. Flyclient super light clients for cryptocurrencies. [Online]. Available: https://scalingbitcoin.org/stanford2017/Day1/flyclientscalingbitcoin.pptx.pdf

[22] D. Leung, A. Suhl, Y. Gilad, and N. Zeldovich, "Vault: Fast bootstrapping for cryptocurrencies," in *The Network and Distributed System Security Symposium (NDSS) 2019*. Internet Society.

[23] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[24] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Advances in Cryptology – CRYPTO 2017*, pp. 357–388.

[25] P. Daian, R. Pass, and E. Shi, "Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake," in *International Conference on Financial Cryptography and Data Security*, 2019.

[26] Steemit. [Online]. Available: https://steemit.com/

[27] A. Kiayias, N. Lamprou, and A.-P. Stouka, "Proofs of proofs of work with sublinear complexity," in *International Conference on Financial Cryptography and Data Security*, 2016, pp. 61–78.

[28] V. Buterin. Light clients and proof of stake. [Online]. Available: https://blog.ethereum.org/2015/01/10/light-clients-proof-stake/

[29] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," 2018, https://eprint.iacr.org/2018/968.pdf.

[30] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," 2018, https://eprint.iacr.org/2018/1188.

[31] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Advances in Cryptology – CRYPTO 2010*, pp. 465–482.

[32] E. Ben-Sasson, A. Chiesa, D. Genkin *et al.*, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology – CRYPTO 2013*, pp. 90–108.

[33] V. Costan and S. Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://eprint.iacr.org/2016/086.

[34] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM CCS 2016*, pp. 270–282.

[35] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

[36] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom, "Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution," *Technical report*, 2018.

[37] V. Pham, M. H. R. Khouzani, and C. Cid, "Optimal contracts for outsourced computation," in *International Conference on Decision and Game Theory for Security*. Springer, 2014, pp. 79–98.

[38] A. Kupcu, "Incentivized outsourced computation resistant to malicious contractors," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 6, pp. 633–649, Nov 2017.

[39] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *Proc. ACM CCS 2017*, pp. 211–227.

[40] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," 2017. [Online]. Available: https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf

[41] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[42] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM CCS 2016*, 2016, pp. 254–269.

[43] Solidity Global Variables. [Online]. Available: https://solidity.readthedocs.io/en/develop/units-and-global-variables.html

[44] A. Miller. Blockhash contract. [Online]. Available: https://github.com/amiller/ethereum-blockhashes

[45] J. Halpern and V. Teague, "Rational secret sharing and multiparty computation," in *Proc. ACM STOC 2004*, pp. 623–632.

[46] S. D. Gordon and J. Katz, "Rational secret sharing, revisited," in *International Conference on Security and Cryptography for Networks*, 2006, pp. 229–241.

[47] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM CCS 2016*, pp. 17–30.

[48] A. Poelstra, "Mimblewimble," 2016, https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

[49] Bitcoin network. [Online]. Available: https://en.bitcoin.it/wiki/Network

[50] Ethereum Genesis Address: The Black Hole That Has Over 520 Million Worth Of Tokens. [Online]. Available: https://btcmanager.com/ethereum-genesis-address-black-hole-520-million-worth-tokens/

[51] M. H. Au, W. Susilo, and S.-M. Yiu, "Event-oriented k-times revocable-iff-linked group signatures," in *Australasian Conference on Information Security and Privacy 2006*, pp. 223–234.