Generic Superlight Client for Permissionless Blockchains

Yuan Lu[1](✉), Qiang Tang[1,2], and Guiling Wang[1]

[1] New Jersey Institute of Technology, Newark, NJ 07102, USA
{yl768,qiang,gwang}@njit.edu

[2] JDD-NJIT-ISCAS Joint Blockchain Lab, Newark, USA

Abstract. We initiate a systematic study on the light-client protocol of permissionless blockchains, in the setting where full nodes and light clients are rational. In the game-theoretic model, we design a superlight-client protocol to enable a light client to employ some relaying full nodes (e.g., two or one) to read the blockchain. The protocol is "generic", i.e., it can be deployed disregarding underlying consensuses, and it is also "superlight", i.e., the computational cost of the light client to predicate the (non)existence of a transaction in the blockchain becomes a small constant. Since our protocol resolves a fundamental challenge of broadening the usage of blockchain technology, it captures a wide variety of important use-cases such as multi-chain wallets, DApp browsers and more.

Keywords: Blockchain · Light client · Game-theoretic security

1 Introduction

The blockchain can be abstracted as a global ledger [22,38] that can be read and written by the users of higher level applications [16,17,54]. Nevertheless, the basic abstraction of *reading the ledger*[1] implicitly requires a so-called personal *full node* [8,25] to execute consensus and maintain a local blockchain replica.

However, with the rapid popularity of blockchain, an increasing number of users become merely caring about the high-level applications such as cryptocurrencies instead of maintaining personal full nodes [28]. Let alone many users are resource-starved, say browser extensions and smartphones [13,55,57] that have too limited resources to stay on-line and execute the underlying consensus.

Thus an urgent demand of blockchain's *lightweight clients* or superlight clients [10,12,33,36], rises up. Consider a quintessential scenario: Alice is the cashier of a pizza store; a customer Bob tells her ₿1,000 has been paid for some pizzas, via a bitcoin transaction with txid 0xa1075d..., and claims the transaction is already

[1] Writing in the blockchain is trivial, as one can gossip with some full nodes to diffuse its messages to the entire blockchain network (a.k.a., network diffuse functionality [4,22]). Then the blockchain's liveness ensures the inclusion of the messages [22].

L. Chen et al. (Eds.): ESORICS 2020, LNCS 12309, pp. 713–733, 2020.
https://doi.org/10.1007/978-3-030-59013-0_35

in the blockchain; then, Alice needs to check that, by activating a lite wallet app installed in her mobile phone. That is to say, Alice, and many typical blockchain users, need *stay off-line* to opt out of the consensus, and can still wake up any time to "read" the blockchain with high security and low computational cost.

1.1 Insufficiencies of Prior Art

The fundamental challenge of designing superlight clients stems from a fact: the records in the blockchain are de facto "authenticated" by the latest chain agreed across the whole blockchain network (a.k.a., the main-chain) [34,48,51]. So without a replica of main-chain at hand, the client has to rely on some other *full nodes* to forward the main-chain's records. That said, the protocol of blockchains' superlight clients must deal with the probably distrustful full nodes that might forward fake blockchain readings.

Some Ad-Hoc Attempts. A few proposals attempt to prevent the client being cheated, by relying on heavyweight assumptions. For example, a few proposals [14,19] assume a diverse list of known full nodes to serve as relays to forward blockchain readings. These relays are usually some "mining" pools and a few so-called blockchain "explorers", so the client can count on the honest-majority of these known relays to read the chain. But for many real-world permissionless blockchains, this assumption is too heavy to hold with high-confidence. Say Cardano [1], a top-10 blockchain by market capitalization in 2020, has very few "explorers" on the run; even worse, the naive idea of recruiting "mining" pools as relays is more elusive, as most of them would not participate without moderate incentives [28]. It becomes unclear how to identify an honest-majority set of known relays for each permissionless blockchain in the wild. As such, these ad-hoc solutions become unreliable for the elusive heavy assumptions.

Cryptographic Approaches. To design the light-client protocol against malicious relay full nodes, a few cryptographic approaches are proposed [10,33,48,59].

– *Straight Use of SPV is Problematic.* The most straightforward way to instantiating the idea is to let the client keep track of the suffix of the main-chain, and then check the existence of transactions by verifying SPV proofs [48], but the naive approach causes at least one major issue: the client has to frequently be on-line to track the growth of main-chain. Otherwise, when the client wakes up from a deep sleep, it needs to at least verify the block headers of the main-chain linearly. Such bootstrapping can be costly, considering the main-chain is ever-growing, say the headers of Ethereum is growing at a pace of $\sim$1 GB per year. As result, in many critical use-cases such as web browsers and/or mobile phones, the idea of straightly using SPV proofs becomes unrealistic.

– *PoW-Specific Results.* For PoW chains, some existing superlight clients such as FlyClient and NiPoPoW [10,33] circumvent the problem of SPV proofs. These ideas notice the main-chain is essentially "authenticated" by its *few suffix blocks*, and then develop some PoW-specific techniques to allow the *suffix blocks* be proven to a client at only sublinear cost. But they come with one major limit, namely, need to verify PoWs to discover the correct suffix, and therefore cannot fit the promising class of proof-of-stake (PoS) consensuses [14,24,34].

– *Superlight Client for PoS is Still Unavailable.* For PoS chains, it is yet unclear how to realize an actual superlight client that can go off-line to completely opt out of consensus. The major issue is lacking an efficient way of proving the suffix of PoS chains to an off-line client, because the validity of the suffix blocks relies on the signatures of stakeholders, whose validities further depend on the recent stake distributions, which further are authenticated by the blockchain itself [12, 14,24,34]. Some recent efforts [23,39,44] allow the *always-online* clients to use minimal space to track the suffix of PoS chains, without maintaining the stake distributions. But for enabling off-line, there only exist few fast-bootstrapping proposals for *full nodes* [6,41], which still require to download and verify a linear portion of the PoS chain, thus incurring substantial cost [15,24].

Demands of "Consensus-Oblivious" Light Client. Most existing light-client protocols are highly specialized for particular consensuses (e.g., PoW). This not only prevents us adapting them to instantiate actual superlight clients for the PoS chains, but also hinders their easy deployment and user experience in many important use-cases in reality. A typical scenario is the multi-chain wallet that supports various cryptocurrencies atop different chains; each chain is even running a distinct consensus. Bearing the high-specialization of existing light-client protocols [10,23,33,39,44], the multi-chain wallet needs to instantiate different light-client protocols for distinct consensuses and ends up to contain many independent "sub-wallets", which not only is burdensome for the client users but also challenges the developers to correctly implement all sub-wallets. In contrast, with a generic protocol fitting all, we can simply tune some parameters at best.

Generic Solution in Another Different Setting. The cryptographic setting seems to be an inherent obstacle-ridden path to the generic light-client protocol, as it usually needs the full nodes to prove the main-chain's suffix validated by the consensus rules [10,23,33,39]. Even if one puts forth a "consensus-oblivious" solution in this setting, it likely has to convert all those "suffix proofs" into a generic statement of verifiable computation (VC) [44], which is unclear how can be practical, considering VC itself is not fully practical yet for complicated statements [10] (see the full paper [42] for a thorough review on pertinent topics).

To meet the urgent demand of the generic light-client protocol, we explicitly deviate from the cryptographic setting and focus on the game-theoretic approach, in light of many successful studies such as rational multi-party computation [3,7,21,26,27,29,32,37] and rational verifiable computation [16,52,56]. In the rational setting, we can expect a consensus-independent incentive mechanism to assist a simple light-client protocol, so all rational protocol participants (i.e., the relay nodes and the client) would follow the protocol for selfishness.

Following that, this paper would present: *a systematic treatment to the light-client problem of permissionless blockchains in the game-theoretic setting.*

1.2 Our Results

In the game theoretic setting the players are rational, we design a superlight protocol to enable a light client to recruit several relay full nodes (e.g., one[2] or two) to securely evaluate a general class of predicates about the blockchain.

Contributions. To summarize, our technical contributions are:

- Our light-client protocol can be bootstrapped in the rational setting, efficiently and generically. First, the protocol is superlight, in the sense that the client can go off-line and wake up any time to *evaluate* a general class of chain predicates at a tiny constant computationally cost; as long as the truthness or falseness of these chain predicates is reducible to few transactions' inclusion in the blockchain.
- Moreover, our generic protocol gets rid of the dependency on consensuses and can be deployed in nearly any permissionless blockchain (e.g., Turing-complete blockchains [11,58]) without even velvet forks [60], thus supporting the promising PoS type of consensuses.
- We conduct a systematic study to understand whether, or to what extent, the light-client protocol is secure in the rational setting (without trusted third-parties). We make non-trivial analyses of the incomplete-information extensive game induced by our light-client protocol and conduct a comprehensive study to understand how to design the incentives to achieve security in different scenarios, from the standard setting of non-cooperative full nodes to the pessimistic setting of colluding full nodes.
- Our protocol enables the rational client to *evaluate* non-existence of a given transaction besides the existence, i.e., the rational client can be convinced by the rational full nodes that a given transaction is *not* in the blockchain. That provides a simple way to performing non-existence "proof". In contrast, relevant studies in the cryptographic setting either give up non-existence proof [36,48] or have to heavily modify the blockchain's data structure [9,47].

Solution in a Nutshell. Assuming the light client and relay nodes are rational, we leverage the smart contract to facilitate a simple yet still useful incentive mechanism, so being honest becomes the best rational choice. From high-level, the light-client protocol proceeds simply as following:

- *Setup*. The client and relay node(s) place their initial deposits in an "arbiter" smart contract. An incentive mechanism can use these deposits to facilitate rewards/punishments and deter deviations from the light-client protocol.
- *Repeated queries*. After setup, the client can repeatedly query the relays to forward blockchain readings (up to k times). Each query proceeds as:
  1. *Request*. The client firstly specifies the details of the predicate to query in the arbiter contract, which can be done since writing in the contract is trivial for the network diffuse functionality (see footnote 1).

[2] Note that the case of one relay can model the pessimistic scenario that all recruited full nodes are colluding to form a single coalition.

2. *Response.* Once the relays see the specifications of the chain predicate in the arbiter contract, they are incentivized to evaluate the predicate and forward the ground truth to the client off-chain.
3. *Feedback.* Then the client decides an output, according to what it receives from the relays. Besides, the client shall report what it receives to the arbiter contract; otherwise, gets a fine.
4. *Payout.* Finally, the contract verifies whether the relays are honest, according to the feedback from the client, and then facilitates an incentive mechanism to reward (or punish) the relays. The incentive mechanisms leverage the deposits (placed by the client and relays) to ensure "following the protocol" to be a desired equilibrium, such that the rational relays and client would not deviate during any stage of the protocol.

Challenges and Techniques. The instantiation of the above idea is challenged by the limit of the "handicapped" arbiter contract. In particular, the arbiter contract cannot directly verify the non-existence of transactions, even if it can verify any transaction's existence [5,36]. Thus, for a chain predicate whose trueness (resp. falseness) is reducible to the existence (resp. non-existence) of some transactions, the arbiter contract can only verify its trueness but not its falseness. This allows the relays to adopt a malicious strategy: "always forward unverifiable bogus disregarding the actual ground truth", because doing so would not be caught by the contract and thus the relays are still paid.

To circumvent the limit of the arbiter contract, we squeeze the most of its "handicapped" verifiability to finely tune the incentive mechanism, such that "flooding fake unverifiable claims" become irrational. Following that, any deviations from the protocol are further deterred, from the standard setting of non-cooperative relays to the extremely hostile case of colluding relays:

- If two *non-cooperative relays* (e.g., two competing mining pools in practice) can be identified and recruited, we leverage the natural tension between these *two* selfish relays to "audit" each other. As such, fooling the client is deterred, because a selfish relay is incentivized to report the other's (unverifiable) bogus claim, by producing a proof attesting the opposite of the fake claim.
- In the extremely adversarial scenario where any two recruited relays can *form a coalition*, the setting becomes rather pessimistic, as the client is essentially requesting an unknown knowledge from a *single* party. Nevertheless, the incentive can still be slightly tuned to function as follows:
  1. The first tuning does *not* rely on any extra assumption. The adjustment is to let the arbiter contract assign a higher payoff to a proved claim while make a lower payoff to an unprovable claim. So the best strategy of the *only* relay is to forward the actual ground truth, as long as the malicious benefit attained by fooling the client is smaller than the maximal reward promised by the client.

     Though this result has limited applicabilities, for example, cannot handle valuable queries, it still captures a variety of meaningful real-world use-cases, in particular, many DApp browsers, where the relay is not rather interested in cheating the client.

2. The second tuning relies on another moderate rationality assumption, that is: at least one selfish *public* full node (in the entire blockchain network) can keep on monitoring the internal states of the arbiter contract at a tiny cost and *will not cooperate* with the recruited relay.
   Thus whenever the recruited relay forwards an *unprovable* bogus claim to the client, our design incentivizes the *selfish* public full node to "audit" the relay by proving the opposite side is the actual ground truth, which deters the recruited relay from flooding unprovable bogus.

Application Scenarios. Our protocol supports a wide variety of applications, as it solves a fundamental issue preventing low-capacity users using blockchain:

- *Decentralized application browser.* The DApp browser is a natural application scenario. For example, a lightweight browser for CryptoKitties [13] can get rid of a trusted Web server. When surfing the DApp via a distrustful Web server, the users need to verify whether the content rendered by the server is correct, which can be done through our light-client protocol efficiently.
- *Mobile wallet for multiple cryptocurrencies.* Our protocol can be leveraged to implement a super-light mobile wallet to verify the (non)existence of cryptocurrency transactions. In particular, it can keep track of multiple coins atop different blockchains running over diverse types of consensuses.

2 Warm-Up: Game-Theoretic Security

The game-theoretic analysis of an interactive protocol starts by defining an extensive game [16,31,50] to model the strategies (i.e., probabilistic interactive Turing machines) of each party in the protocol. A utility function would assign every party a certain payoff, for each possible execution induced by the strategies of all parties. Then the security is argued by the properties of the extensive game, for example, its Nash equilibrium [29] or other refined equilibria [16,30,31,37,50].

Here we give a simple interactive "protocol" to exemplify the game-theoretic setting (c.f., our full paper [42] for rigor definitions of game-theory preliminaries). Consider the oversimplified "light-client protocol" that be described as the extensive game as shown in Fig. 1: *Alice is a cashier of a pizza store; her client asks a full node (i.e., relay) to check a transaction's (non)existence, and simply terminates to output what is forwarded by the relay.*

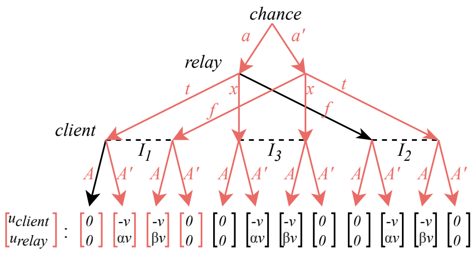

Fig. 1. The extensive game of an oversimplified light-client "protocol". The utility function is an example to clarify the *insecurity* of such a trivial idea.

Strategy, Action, History, and Information Set. Let the above *oversimplified* "protocol" proceed in synchronous round. In each round, the parties will execute its *strategy*, i.e., a probabilistic polynomial-time ITM in our context, to produce and feed a string to the protocol, a.k.a., take an *action*. During the course of the protocol, a sequence of actions would be made, and we say it is a *history* by the convention of game theory literature; moreover, when a party acts, it might have learned some (incomplete) information from earlier actions taken by other parties, so the notion of *information sets* are introduced to characterize what has and has not been learned by each party (see [42] for formal definitions). In greater detail, the *oversimplified* "light-client protocol" is interpreted as Fig. 1:

1. *Round 1 (chance acts).* A definitional virtual party called *chance* sets the ground truth, namely, it determines $True$ or $False$ to represent whether the transaction exists (denoted by a or a' respectively). To capture the uncertainty of the ground truth, the chance acts arbitrarily.
2. *Round 2 (relay acts).* Then, the relay is activated to forward $True$ or $False$ to the light client, which states whether the transaction exists or not. Note the strategy chosen by the relay is an ITM that can produce arbitrary strings in this round, we need to map the strings into the admissible actions, namely, t, f and x. For definiteness, we let the string of ground truth be interpreted as the action t, the string of the opposite of ground truth be interpreted as the action f, and all other strings (including abort) be interpreted as x.
3. *Round 3 (client acts).* Finally, the client outputs $True$ (denoted by A) or $False$ (denoted by A') to represent whether the transaction exists or not, according to the (incomplete) information acquired from the protocol. Note the client knows how the relay acts, but cannot directly infer the action of *chance*. So it faces three distinct information sets I_1, I_2 and I_3, which respectively represent the client receives $True$, $False$ and others in Round 2. The client cannot distinguish the histories inside each information set.

Utility Function. After the protocol terminates, its game reaches a so-called *terminal history*. A well-defined *utility function* specifies the economic outcome of each party, for each terminal history induced by the extensive game.

In practice, the utility function is determined by some economic factors of the parties and the protocol itself [16,29]. For example, the rationale behind the utility function in Fig. 1 can be understood as: (i) the relay is motivated to fool the client to believe the nonexistence of an existing transaction, because this literally "censors" Alice to harm her business by a loss of $\$v$, which also brings a malicious benefit $\$\alpha \cdot v$ to the relay; (ii) the relay also prefers to fool the client to believe the existence of a non-existing transaction, so the relay gets free pizzas valued by $\$\beta \cdot v$, which causes Alice lose $\$v$ (i.e., the amount supposed to be transacted to purchase pizzas), (iii) after all, the oversimplified protocol itself does not facilitate any punishment/reward, so will not affect the utility function.

Security via Equilibrium. Putting the game structure and the utility function together, we can argue the (in)security due to the equilibria in the game. In particular, we can adopt the strong notion of *sequential equilibrium* for extensive games [16,30,31,50] to demonstrate that the rational parties would not deviate, at each stage during the execution of the protocol. As a negative lesson, the oversimplified "light-client game" in Fig. 1 is *insecure*, as the relay can unilaterally deviate to fool the client for higher utility. In contrast, if the protocol is *secure* in game-theoretic settings, its game shall realize desired equilibrium, s.t., rational parties would not diverge for highest utilities.

3 Preliminaries

Blockchain Addressing. A blockchain (e.g., denoted by C) is a chain of block (headers). Each block commits a list of payload (e.g., transactions). Notation-wise, we use Python bracket $\mathsf{C}[t]$ to address the block (header) at the height t of the chain C, and $\mathsf{C}[0]$ represents the genesis block. W.l.o.g., a block $\mathsf{C}[t]$ is defined as a tuple of $(h_{t-1}, nonce, \mathsf{root})$, where h_{t-1} is the hash of the block $\mathsf{C}[t-1]$, $nonce$ is the valid PoX (e.g., the correct preimage in PoW), and root is Merkle tree root of payload. By $\mathsf{C}[t].\mathsf{root}$ it denotes the Merkle root of block $\mathsf{C}[t]$.

Payload & Merkle Tree. Let $\mathsf{TX}_t := \langle \mathsf{tx}_1, \mathsf{tx}_2, \cdots, \mathsf{tx}_n \rangle$ denote a sequence of transactions that is the payload of the block $\mathsf{C}[t]$. Recall TX_t is included by the block $\mathsf{C}[t]$ through Merkle tree [48,58], which is an authenticated data structure scheme of three algorithms ($\mathsf{BuildMT}, \mathsf{GenMTP}, \mathsf{VrfyMTP}$). $\mathsf{BuildMT}$ inputs $\mathsf{TX}_t = \langle \mathsf{tx}_1, \cdots, \mathsf{tx}_n \rangle$ and outputs a Merkle tree MT with root. GenMTP takes the tree MT (built for TX_t) and a transaction $\mathsf{tx} \in \mathsf{TX}_t$ as input, and outputs a proof π_j for the inclusion of tx in TX_t at the position j. $\mathsf{VrfyMTP}$ inputs π_j, root and tx and outputs either 1 or 0. The Merkle tree scheme satisfies: (i) *Correctness.* $\Pr[\mathsf{VrfyMTP}(\mathsf{MT.root}, \mathsf{tx}, \pi_i) = 1 \mid \pi_i \leftarrow \mathsf{GenMTP}(\mathsf{MT}, \mathsf{tx}),\ \mathsf{MT} \leftarrow \mathsf{BuildMT}(\mathsf{TX})] = 1$; (ii) *Security.* for $\forall$ P.P.T. $\mathcal{A}$, $\Pr[\mathsf{VrfyMTP}(\mathsf{MT.root}, \mathsf{tx}, \pi_i) = 1 \wedge \mathsf{tx} \neq \mathsf{TX}[i] \mid \pi_i \leftarrow \mathcal{A}(1^\lambda, \mathsf{MT}, \mathsf{tx}),\ \mathsf{MT} \leftarrow \mathsf{BuildMT}(\mathsf{TX})] \leq negl(\lambda)$. Note that by $nelg(\cdot)$ we denote a negligible function through the paper. The construction of the Merkle tree scheme is deferred to the full version [42].

Smart Contract. Essentially, a smart contract [11,58] can be abstracted as an ideal functionality with the access to a global ledger subroutine, so it can faithfully instruct the ledger to freeze "coins" and then correctly facilitate conditional payments [35,38]. This paper explicitly adopts the widely-used notations invented by Kosba *et al.* [38] to describe the smart contract, for example:

- The contract can access the global time clock T, which can be seen as an equivalent notion of the height of the latest blockchain.
- The contract can access a global dictionary ledger for conditional payments.
- We slightly enhance their notations to allow the contract to access a global dictionary $\mathsf{blockhashes}$. Each item $\mathsf{blockhashes}[t]$ is the hash of the block $\mathsf{C}[t]$.[3] Note the local blockchain replicas of all full nodes would be consistent to $\mathsf{blockhashes}$ (within one clock) according to the global ledger model [35].

[3] Remark that the above modeling requires the block hashes can be read by smart contracts from the blockchain's internal states (e.g. available global variables) [20]. In Ethereum, this currently can be realized via the proposal of Andrew Miller [45] and will be incorporated due to the already-planned Ethereum enhancement EIP-210 [2].

- The contract would not send its internal states to the light client, which captures the client opts out of consensus. However, the client can send messages to the contract, due to the well abstracted network diffusion functionality.

4 Problem Formulation

This section rigorously defines the light-client problem in the rational model.

4.1 Defining the Readings from the Blockchain

The basic aim is to allow the resource-starved clients to evaluate the falseness or trueness about some statements over the blockchain [33]. This subsection would define these statements about blockchain as chain predicates.

Chain Predicate. We focus on a general class of chain predicates whose trueness can be induced by up to l transactions' inclusions in the chain, such as "whether a transaction tx is in the chain". Formally, the chain predicate is in the form of:

$$\mathsf{P}^{\ell}(\mathsf{C}[0:N]) = \begin{cases} \mathit{False}, \text{ otherwise} \\ \mathit{True}, \exists \mathsf{C}' \subset \mathsf{C}[0:N] \text{ s.t. } \mathsf{D}^{\ell}(\mathsf{C}') = \mathit{True} \end{cases}$$

where C' is a subset of the blockchain $\mathsf{C}[0:N]$, and $\mathsf{D}^{\ell}(\cdot)$ is a computable predicate taking C' as input and can be expressed as:

$$\mathsf{D}^{\ell}(\mathsf{C}') = \begin{cases} \mathit{True}, \exists\ \{\mathsf{tx}_i\} \text{ that } |\{\mathsf{tx}_i\}| \le \ell : \\ \qquad f(\{\mathsf{tx}_i\}) = 1\ \wedge\ \forall\ \mathsf{tx}_i \in \{\mathsf{tx}_i\}, \\ \qquad\qquad \exists\ \mathsf{C}[t] \in \mathsf{C}' \text{ and P.P.T. computable } \pi_i \text{ s.t.} \\ \qquad\qquad\qquad \mathsf{VrfyMTP}(\mathsf{C}[t].\mathsf{root}, \mathsf{H}(\mathsf{tx}_i), \pi_i) = 1 \\ \mathit{False}, \text{ otherwise} \end{cases}$$

where $f(\{\mathsf{tx}_i\}) = 1$ captures that $\{\mathsf{tx}_i\}$ satisfies a certain relationship, e.g., "the hash of each tx_i equals a specified identifier txid_i", or "each tx_i can pass the membership test of a given bloom filter", or "the overall inflow of $\{\mathsf{tx}_i\}$ is greater than a given value". We let P_N^{ℓ} be short for $\mathsf{P}^{\ell}(\mathsf{C}[0:N])$.

Examples of Chain Predicate. The definition of chain predicate captures a wide range of blockchain "readings". For any predicate under this category, its trueness can be succinctly attested by up to ℓ transactions' inclusion in the chain. For $\ell = 1$, some concrete examples are:

- "The transaction tx is included in $\mathsf{C}[0:N]$", the trueness of which can be attested by tx's inclusion in the chain $\mathsf{C}[0:N]$.
- "tx is included in $\mathsf{C}[0:N]$ and satisfies a given bloom filter $f(\cdot)$", which is reducible to tx's inclusion in the chain and the value of $f(\mathsf{tx})$.
- "A set of transactions $\{\mathsf{tx}_j\}$ are included in $\mathsf{C}[0:N]$", whose trueness can be attested by all tx_j's inclusions in the chain.

Limits of Chain Predicate. A chain predicate is a binary question, whose trueness is reducible to the inclusion of some transactions. The actual meaning of a chain predicate depends on how to concretely specify it. A "meaningful" chain predicate might need certain specifications from an external party outside the system. For example, the cashier of a pizza store can specify a transaction to evaluate its (non)existence, only if the customer tells the txid.

"Handicapped" Verifiability of Chain Predicate. We focus on the chain predicate in form of P_N^ℓ, namely, whose trueness is provable.[4] Such "handicapped" verifiability can be well abstracted through a tuple of two algorithms ($\mathsf{evaluate}, \mathsf{validateTrue}$) along with their properties:

- $\mathsf{evaluate}(\mathsf{P}_N^\ell) \rightarrow \sigma$ or $\perp$: The algorithm takes the replica of the blockchain as auxiliary input and outputs σ or $\perp$, where σ is a proof for $\mathsf{P}_N^\ell = True$, and $\perp$ represents its falseness; note the proof σ here includes: a set of transactions $\{\mathsf{tx}_i\}$, a set of Merkle proofs $\{\pi_i\}$, and a set of blocks C';
- $\mathsf{validateTrue}(\sigma, \mathsf{P}_N^\ell) \rightarrow 0$ or 1: This algorithm takes $\mathsf{blockhashes}$ as auxiliary input and outputs 1 (accept) or 0 (reject) depending on whether σ is deemed to be a valid proof for $\mathsf{P}_N^\ell = True$; note the validation parses σ as $(\{\mathsf{tx}_i\}, \{\pi_i\}, \mathsf{C}')$ and verifies: (i) C' is included by $\mathsf{blockhashes}[t]$ where $t \leq N$; (ii) each tx_i is committed by a block in C' due to Merkle proof π_i; (iii) $f(\{\mathsf{tx}_i\}) = 1$ where $f(\cdot)$ is the specification of the chain predicate.

The above algorithms satisfy: (i) *Correctness.* For any chain predicate P_N^ℓ, there is $\Pr[\mathsf{validateTrue}(\mathsf{evaluate}(\mathsf{P}_N^\ell), \mathsf{P}_N^\ell) = 1 \mid \mathsf{P}_N^\ell = true] = 1$, and (ii) *Verifiability.* for any P.P.T. $\mathcal{A}$ and P_N^ℓ, there is $\Pr[\mathsf{validateTrue}(\sigma \leftarrow \mathcal{A}(\mathsf{P}^\ell), \mathsf{P}_N^\ell) = 1 \mid \mathsf{P}_N^\ell = false] \leq negl(\lambda)$, where $\mathsf{evaluate}$ implicitly takes the blockchain replica as input, and $\mathsf{validateTrue}$ implicitly inputs $\mathsf{blockhashes}$. In the remaining of the paper, $\mathsf{evaluate}$ can be seen as a black-box callable by any full nodes, and $\mathsf{validateTrue}$ is a subroutine that can be invoked by any smart contract.

4.2 System and Adversary Model

The system explicitly consists of a lightweight client, some relay(s) and an arbiter contract. The messages among them can deliver within a known delay ΔT. In details,

The Rational Lightweight Client. $\mathcal{LW}$ is abstracted as:

- It is **rational** and selfish; moreover, it is computationally bounded, i.e., it can only take an action computable in probabilistic polynomial-time;
- It opts out of consensus; to capture this, we assume:
  - The client can *send* messages to the contract due to the network diffusion functionality [22,38];

[4] Remark that in the full paper [42], we define another class of chain predicates whose falseness is provable instead of trueness, which can captured by our protocol as well, though we omit detailed discussions here for presentation simplicity.

  – The client cannot receive messages from the contract except a short setup phase, which can be done in practice because the client user can temporarily boost a personal full node by fast-bootstrapping protocols.

The Rational Full Node. $\mathcal{R}_i$ is modeled as:

- It is **rational**. Also, the full node $\mathcal{R}_i$ might (or might not) cooperate with another full node $\mathcal{R}_j$. The **(non)-cooperation** of them is specified as:
  – The **cooperative** full nodes form a coalition to maximize the total utility, as they can share all information, coordinate all actions and transfer payoffs, etc. [49]; essentially, we follow the conventional notion to view the cooperative relays as **a single** party [7].
  – **Non-cooperative** full nodes maximize their own utilities independently in a selfish manner due to the standard non-cooperative game theory, which can be understood as that they are not allowed to choose some ITMs to communicate with each other [40];
- It can only take P.P.T. computable actions at any stage of the protocol;
- The full node runs the consensus, such that:
  – It stores the complete replica of the latest blockchain;
  – It can send/receive messages to/from the smart contract;
- It can send messages to the light client via an off-chain private channel.[5]

The Arbiter Contract. $\mathcal{G}_{ac}$ follows the standard abstraction of smart contracts [35,38], with a few slight extensions. First, it would not send any messages to the light client except during a short setup phase. Second, it can access a dictionary $\mathsf{blockhashes}$ [2,45], which contains the hashes of all blocks. The latter abstraction further allows the contract to invoke $\mathsf{validateTrue}$ to verify the proof attesting the trueness of any predicate P^{ℓ}_{N}, in case the predicate is actually true.

Economic Factors of Modeling. For the sake of completing our game-theoretic model, we clarify the economic parameters of the modeling as follows:

- v: The factor means the "value" attached to the chain predicate under query. If the client incorrectly evaluates the predicate, it loses v. For example, the cashier Alice is evaluating the (non)existence of a certain transaction; if Alice believes the existence of a non-existing transaction, she loses the amount to be transacted; if Alice believes the nonexistence of an existing transaction, her business is harmed by such censorship.
- $v_i(\mathsf{P}^{\ell}_{N}, \mathsf{C}) \rightarrow [0, v_i]$: This function characterizes the motivation of the relay $\mathcal{R}_i$ to cheat the light client. Namely, it represents the extra (malicious) utility that the relay $\mathcal{R}_i$ earns, if fooling the client to incorrectly evaluate the chain predicate. We consider $\sum_i v_i(\mathsf{P}^{\ell}_{N}, \mathsf{C}) \leq v$, so the overall malicious utilities attained by fooling the client is up to the "value" attached to the query.

[5] Such assumption can be granted if considering the client and the relays can set up private communication channels on demand. In practice, this can be done because (i) the client can "broadcast" its network address via the blockchain [43], or (ii) there is a trusted name service that tracks the network addresses of the relays.

- In addition, all communications and P.P.T. computations can be done costlessly (unless otherwise specified).

Remark. We describe an advanced modeling of economic factors to capture the cost of maintaining a personal full node by the client user in the full version [42]. Here we ignore this factor by assuming the client cannot access any personal full node once the protocol is set up. This simplified modeling still makes sense to allow us argue security of our protocol, conditioned on the setup phase is done.

4.3 Security Goal

The aim of the light-client protocol $\Pi_{\mathcal{LW}}$ in the above game-theoretic model is to allow a rational light client employ some rational relaying full nodes (e.g., one or two) to correctly *evaluate* the chain predicates, and these recruited relay nodes are correctly paid as pre-specified. In details, we require:

- Correctness. If all parties are honest, we require: (i) the relay nodes are correctly paid; (ii) the light client correctly evaluates some chain predicates under the category of $\mathsf{P}^{\ell}(\cdot)$, regarding the chain $\mathsf{C}[0:T]$ (i.e. the chain at the time of evaluating). Both requirements shall hold with probability 1.
- Security. We adopt a strong game-theoretic security notion of *sequential equilibrium* [16,30,31] for incomplete-information extensive games. Consider an extensive-form game Γ that models the light-client protocol $\Pi_{\mathcal{LW}}$, and let $(\mathbf{Z_{bad}}, \mathbf{Z_{good}})$ as a partition of the terminal histories $\mathbf{Z}$ of the game Γ, where $\mathbf{Z_{good}}$ captures and only captures all protocol executions where no party deviates. Given a *ϵ-sequential equilibrium*[6] of Γ denoted by σ, the probability of reaching each terminal history $z \in \mathbf{Z}$ can be induced, which can be denoted by $\rho(\sigma, z)$.
  Our security goal requires: there is a ϵ-sequential equilibrium σ of Γ where ϵ is at most a negligible function $nelg(\lambda)$ in security parameter λ, such that under the ϵ-equilibrium σ, the game Γ terminates in $\mathbf{Z_{good}}$.

5 A Simple Light-Client Protocol

Here we present a simple light-client protocol, in which a client ($\mathcal{LW}$) employs two (or one) relays to evaluate the chain predicates P_N^{ℓ} (as defined in Sect. 4.1).

5.1 Arbiter Contract and High-Level of the Protocol

The light-client protocol centers around an arbiter smart contract $\mathcal{G}_{ac}$ as shown in Fig. 2. It begins with letting all parties place their initial deposits. Later, the client can ask the relays to forward some readings about the blockchain, and then feeds what it receives back to the contract. Once the contract hears the feedback from the client, it leverages the initial deposits to facilitate some proper incentive mechanism, thus preventing deviations by rewarding and/or punishing.

[6] Remark that due to the notion of *ϵ-sequential equilibrium*, the rational game players are not sensitive for any utility increments that are less than ϵ.

```
The arbiter contract $\mathcal{G}_{ac}$ for $m$ relays ($m = 1$ or $2$)
Init. Let state := INIT, deposits := {}, relays := {}, pubKeys := {}, ctr := 0, predicate := $\emptyset$,
      predicate.N := 0, $T_{end} := 0$
                              Setup phase
Create. On receiving the message (create, $k, p, e, d_L, d_F, \Delta T$) from $\mathcal{LW}$:
    - assert state = INIT and ledger[$\mathcal{LW}$] $\geq \$k \cdot d_L$
    - store $k, p, e, r, d_L, d_F$, and $\Delta T$ as internal states
    - ledger[$\mathcal{LW}$] := ledger[$\mathcal{LW}$] $- \$k \cdot d_L$
    - ctr := $k$ and state := CREATED
    - send (deployed, $k, p, e, d_L, d_F, \Delta T$) to all
Join. On receiving (join, $pk_i$) from $\mathcal{R}_i$ for first time:
    - assert state = CREATED and ledger[$\mathcal{R}_i$] $\geq \$k \cdot d_F$
    - ledger[$\mathcal{R}_i$] := ledger[$\mathcal{R}_i$] $- \$k \cdot d_F$
    - pubKeys := pubKeys $\cup$ ($\mathcal{R}_i, pk_i$)
    - state := READY, if |pubKeys| = $m$
                              Queries phase
Request. On receiving (request, $\mathsf{P}^\ell$) from $\mathcal{LW}$:
    - assert state = READY and ledger[$\mathcal{LW}$] $\geq \$(p + e)$
    - ledger[$\mathcal{LW}$] := ledger[$\mathcal{LW}$] $- \$(p + e)$
    - predicate := $\mathsf{P}^\ell_T$ //Note $T$ is the current chain height
    - $T_{end} := T + \Delta T$
    - send (quering, ctr, predicate) to each full node registered in pubKeys
    - state := QUERYING
Feedback. On receiving (feedback, responses) from $\mathcal{LW}$ for first time:
    - assert state = QUERYING
    - store responses for the current ctr
Timer. Upon $T \geq T_{end}$ and state := QUERYING:
    - call Incentive(responses, predicate) subroutine
    - let ctr := ctr $- 1$
    - if ctr $> 0$ then state := READY
    - else state := EXPIRED
```

Fig. 2. The contract $\mathcal{G}_{ac}$ by pseudocode notations in [38]. The Incentive subroutine is decoupled from the protocol and will be presented separately in the later section.

For security in the rational setting, the incentive mechanism must be powerful enough to precisely punish misbehaviors (and reward honesty). Our main principle to realize such powerful incentive is letting the arbiter contract to learn as much as possible regarding how the protocol is actually executed off-chain, so it can precisely punish and then deter any deviations.

Nevertheless, the contract has "handicapped" abilities. So we have to carefully design the protocol to circumvent its limits, for the convenience of designing powerful incentive mechanisms to deter deviations later.

First, the contract $\mathcal{G}_{ac}$ does not know what the relay nodes forward to the light client off-chain. So the contract $\mathcal{G}_{ac}$ has to rely on the client to figure out what the relays did. At the first glance, the client might cheat the contract, by claiming that it receives nothing from the relays or even forging the relays' messages, in order to avoid paying. To deal with the issue, we require that: (i) the relays authenticate what they forward to the client by digital signatures, so the contract later can verify whether a message was originally sent from the relays, by checking the attached signatures; (ii) the contract requires the light client to deposit an amount of $\$e$ for each query, which is returned to the client, only if the client reports some forwarded blockchain readings signed by the relays.

```
The light-client protocol $\Pi_{\mathcal{LW}}$ (where are $m$ relays)
———— Setup phase ————
• Protocol for the light client $\mathcal{LW}$:
  Create. On instantiating a protocol instance:
    - decide $k, p, e, d_L, d_F, \Delta T$ and let $\mathsf{ctr}_{lw} := k$
    - send $(\mathsf{create}, k, p, e, d_L, d_F, \Delta T)$ to $\mathcal{G}_{ac}$
  Off-line. On receiving $(\mathsf{initialized}, \mathsf{pubKeys})$ from $\mathcal{G}_{ac}$:
    - record $\mathsf{pubKeys}$ and disconnect the trusted full node
.....................................................................
• Protocol for the relay $\mathcal{R}_i$:
  Join. On receiving $(\mathsf{deployed}, k, p, e, d_L, d_F, \Delta T)$ from $\mathcal{G}_{ac}$:
    - generate a key pair $(sk_i, pk_i)$ for signature scheme
    - send $(\mathsf{join}, pk_i)$ to $\mathcal{G}_{ac}$
———— Queries phase ————
• Protocol for the light client $\mathcal{LW}$:
  Request. On receiving a message (from the higher level app) to evaluate the predicate $\mathsf{P}^{\ell}$:
    - $T_{feed} := T + 2\Delta T$, and send $(\mathsf{request}, \mathsf{P}^{\ell})$ to $\mathcal{G}_{ac}$
  Evaluate. On receiving $(\mathsf{response}, \mathsf{ctr}_i, \mathsf{result}_i, \mathsf{sig}_i)$ from the relay $\mathcal{R}_i$:
    - assert $T \leq T_{feed}$ and $\mathsf{ctr}_i = \mathsf{ctr}_{lw}$
    - assert $\mathsf{vrfySig}(\langle \mathsf{result}_i, \mathsf{ctr} \rangle, \mathsf{sig}_i, pk_i) = 1$
    - $\mathsf{responses} := \mathsf{responses} \cup (\mathsf{result}_i, \mathsf{sig}_i)$
    - if $|\mathsf{responses}| = m$ then
        output $b \in \{True, False\}$, if $\mathsf{responses}$ claim $b$
  Feedback. Upon the global clock $T = T_{feed}$:
    - $\mathsf{ctr}_{lw} := \mathsf{ctr}_{lw} - 1$
    - send $(\mathsf{feedback}, \mathsf{responses})$ to $\mathcal{G}_{ac}$
.....................................................................
• Protocol for the relay $\mathcal{R}_i$:
  Respond. On receiving $(\mathsf{quering}, \mathsf{ctr}, \mathsf{P}^{\ell}_N)$ from $\mathcal{G}_{ac}$:
    - $\mathsf{result}_i := \mathsf{evaluate}(\mathsf{P}^{\ell}_N)$
    - $\mathsf{sig}_i := \mathsf{sign}(\langle \mathsf{result}_i, \mathsf{ctr} \rangle, sk_i)$
    - send $(\mathsf{response}, \mathsf{ctr}, \mathsf{result}_i, \mathsf{sig}_i)$ to $\mathcal{LW}$
```

Fig. 3. The light-client protocol $\Pi_{\mathcal{LW}}$ among honest relay node(s) and the light client.

Second, the contract has a "handicapped" verifiability, which allows it to efficiently verify a claim of $\mathsf{P}^{\ell}_N = True$, if being give a succinct proof σ. To leverage the property, the protocol is designed to let the relays attach the corresponding proof σ when claiming the provable trueness. Again, this design is a simple yet still useful way to allow the contract "learn" more about the protocol execution, which later enables a powerful mechanism to precisely punish deviations.

5.2 The Light-Client Protocol

In the presence of the contract $\mathcal{G}_{ac}$, our light-client protocol can be formally described as Fig. 3. The protocol starts with a setup phase, during which the relay(s) and client make initial deposits. Then the client can work independently and request the relays to help it evaluate up to k chain predicates, repeatedly.

Setup Phase. As shown in Fig. 3, the user of a lightweight client $\mathcal{LW}$ connects to a trusted full node in the setup phase, and announces an "arbiter" smart contract $\mathcal{G}_{ac}$. After the contract $\mathcal{G}_{ac}$ is deployed, some relay full nodes (e.g. one or two) are recruited to join the protocol by depositing an amount of $\$k \cdot d_F$ in the contract. The public keys of the relay(s) are also recorded by contract $\mathcal{G}_{ac}$.

Once the setup phase is done, each relay full node places the initial deposits $\$k \cdot d_F$ and the light client deposit $\$k \cdot d_L$, which will be used to deter their deviations from the protocol. At the same time, $\mathcal{LW}$ records the public keys of the relay(s), and then disconnects the trusted full node to work independently.

In practice, the setup can be done by using many fast bootstrap methods [33,41,53], which allows the user to efficiently launch a personal trusted full node in the PC. So the light client (e.g. a smart-phone) can connect to the PC to sync. Remark that, besides the cryptographic security parameter λ, the protocol is specified with some other parameters:

- k: The protocol is expired, after the client requests the relay(s) to evaluate some chain predicates for k times.
- $k \cdot d_L$: This is the deposit placed by the client to initialize the protocol.
- $k \cdot d_F$: The initial deposit of a full node to join the protocol as a relay node.
- p: Later in each query, the client shall place this amount to cover the well-deserved payment of the relay(s).
- e: Later in each query, the client shall place this deposit e in addition to p.

Repeatable Query Phase. Once the setup is done, $\mathcal{LW}$ disconnects the trusted full node, and can ask the relay(s) to query some chain predicates repeatedly, as clarified in Fig. 3. It is immediate to see the *correctness* of the protocol: when all parties are honest, the relay(s) receive the payment pre-specified due to incentive mechanism in the contract, and the client always outputs the ground truth of chain predicate. However, the security would depend on the payoffs clauses facilitated by the incentive subroutine, which will be elaborated in later subsections as we intentionally decouple the protocol and the incentive design.

6 Adding Incentives for Security

Without a proper incentive subroutine, the simple light-client protocol is insecure to any extent, as the relays are always motivated to cheat the client. This section studies on how to squeeze most out of the "handicapped" abilities of the arbiter contract to design proper incentives to achieve desired equilibrium for security.

6.1 Basic incentive Mechanism and its Security

We firstly design the basic incentive subroutines that are only based the rationality of the light client and relays, and them analyze that these incentives make the "light-client game" secure to what extent.

Basic incentive for Two Relays. If two non-cooperative relays can be recruited, the incentive subroutine takes the feedback message from the client as input, and then facilitates the incentives following hereunder general principles:

- It firstly verifies whether the feedback sent from the client indeed encapsulates some responses that were originally sent from $\mathcal{R}_1$ and/or $\mathcal{R}_2$. If feedback contains two validly signed responses, return $\$e$ to the client; If feedback contain one validly signed response, return $\$e/2$ to the client.

- If a relay claims $\mathsf{P}_N^\ell = True$ with attaching an invalid proof σ, its deposit for this query (i.e. $\$d_F$) is confiscated and would not receive any payment.
- When a relay sends a response message containing $\perp$ to claim $\mathsf{P}_N^\ell = False$, there is no succinct proof attesting the claim. The incentive subroutine checks whether the other relay full node provides a proof attesting $\mathsf{P}_N^\ell = True$. If the other relay proves $\mathsf{P}_N^\ell = True$, the cheating relay loses its deposit this query (i.e. $\$d_F$) and would not receive any payment. For the other relay that falsifies the cheating claim of $\mathsf{P}_N^\ell = False$, the incentive subroutine assigns it some extra bonuses (e.g. doubled payment).
- After each query, if the contract does not notice a full node is misbehaving (i.e., no fake proof for truthness or fake claim of falseness), it would pay the node $\$p/2$ as the basic reward (for the honest full node). In addition, the contract returns a portion of the client's initial deposit (i.e. $\$d_L$). Moreover, the contract returns a portion of each relay's initial deposit (i.e. $\$d_F$), if the incentive subroutine does not observe the relay cheats during this query.

The rationale behind the above incentive clauses is straightforward. First, during any query, the rational light client will always report to the contract whatever the relays actually forward, since the failure of doing so always causes strictly less utility, no matter the strategy of the relay full nodes; Second, since the two relay full nodes are non-cooperative, they would be incentivized to audit each other, such that the attempt of cheating the client is deterred.

Basic incentive for One Relay. When any two recruited relays might collude, the situation turns to be pessimistic, as the light client is now requesting an unknown information from only a single distrustful coalition. To argue security in this rather pessimistic case, we consider only one relay in the protocol. To deal with the pessimistic case, we tune the incentive subroutine by incorporating the next major tuning (different from the incentive for two relays):

- If the relay claims $\mathsf{P}_N^\ell = False$, its deposit $\$d_F$ is returned but paid less than $\$p$, namely, $\$(p - r)$ where $\$r \in [0, p]$ is an incentive parameter.
- Other payoff rules follow those designed for two non-cooperative relays.

The ideas behind the above incentive clauses are letting the only relay node to "audit" itself, which means: the relay can expect a higher payment as long as it presents a verifaible claim instead of an unverifiable claim.

Analysis and Security Theorems of Basic incentive. To demonstrate the above delicately tuned incentive clauses are implementable, we describe them in the conventional pseudocode notations [38] in the full version [42]. In addition, due to page limit, we omit the detailed structure of the extensive games induced by the light-client protocol along with the utility functions induced by the incentive clauses, c.f., the full paper [42] for the detailed analysis about these extensive games. Here are the main security theorems (of the basic incentives):

Theorem 1. *If two non-cooperative relays are recruited, there exists a ϵ-sequential equilibrium in the extensive game of the light-client protocol, under* *which no rational party deviates from the protocol except with negligible probability, conditioned on $d_F + p/2 > v_i$ and $d_L > (p + e)$.*

Theorem 1 states that: if there are two non-cooperative relays, the sufficient conditions of security are: (i) the initial deposit d_F of relay node is greater than its malicious benefit v_i that can be obtained by fooling the client; (ii) the initial deposit d_L of the client is greater than the payment p plus another small parameter e. The above conclusion hints us how to safely set up the light-client protocol to instantiate a cryptocurrency wallet in practice, that is: let the light client and the relays finely tune and specify their initial deposits, so the client can query the (non)existence of any transaction, as long as the transacted amount of the transaction is not greater than the initial deposit placed by the relay nodes.

Theorem 2. *If only one relay is recruited, there is a ϵ-sequential equilibrium in the light-client protocol's extensive game, under which no rational party deviates except with negligible probability, when $d_F + p - r > v_i$, $r > v_i$, and $d_L > (p + e)$.*

Theorem 2 reveals that: even in an extremely hostile scenario where only one single relay exists, deviations are still prevented when fooling the light client to believe the non-existence of an existing transaction does not yield better payoff than honestly proving the existence. The statement presents a feasibility region of our protocol that at least captures many important DApps (e.g., decentralized messaging apps) in practice, namely: fooling the client is not very financially beneficial for the relay, and only brings a payoff v_i to the relay.

6.2 Augmented incentive and its Security

In the pessimistic scenario of only one recruited relay full node, we can introduce an extra rationality assumption that: at least one public full node (denoted by $\mathcal{PFN}$) can monitor the internal states of the arbiter contract at a tiny cost (say zero for the convenience of analysis) and does not cooperate with the only recruited relay. This extra rationality assumption can boost an incentive mechanism to deter the relay and client from deviating from the light-client protocol.

Augmented incentive for One Relay. The tuning of the incentive mechanism stems from the observation that: if there is *any* public full node that does not cooperate with the recruited relay (and monitor the internal states of the arbiter contract), it can stand out to audit a fake claim about $\mathsf{P}_N^{\ell} = False$ by producing a proof attesting $\mathsf{P}_N^{\ell} = True$. Thus, we slightly tune the incentive subroutine by adding merely few lines of pseudocode (see [42] for details), which can be summarized as:

- When the recruited relay node forwards a response that claims $\mathsf{P}_N^{\ell} = False$, the incentive subroutine shall wait few clock periods (e.g., one). During the waiting time, the public full node is allowed to send a proof attesting $\mathsf{P}_N^{\ell} = True$ to falsify a fake claim of $\mathsf{P}_N^{\ell} = False$; in this case, the initial deposit d_F of the cheating relay is confiscated and paid to the public full node.
- Other payoff rules are same to the basic incentive mechanism.

Analysis and Security Theorems of Augmented incentive. The formal instantiation of the above augmented incentive mechanism along with the detailed security analysis are deferred to the full version [42] due to page limit. The main security theorem about the augmented incentive mechanism is:

Theorem 3. *Given the augmented incentive mechanism, a ϵ-sequential equilibrium exists in the light-client protocol's extensive game, where no rational party deviates except with negligible probability when $d_F > v_i$, $d_L > (p+e)$ and a non-cooperative public full node that can "monitor" the arbiter contract costlessly.*

The economics behind Theorem 3 can be understood similarly as Theorem 1.

7 Discussions

Feasibility. We also shed light on the concrete implementation of the protocol in the full paper [42]. Our experiments atop Ethereum indicate that a non-optimized basic instantiation has been arguably practical. In particular, to query a transaction's (non)existence, the on-chain handling fee (which characterizes the on-chain feasibility and excludes the incentives to pay) is less than half US dollar at the time of writing (see [42] for details).

Future Outlook. As this is the first study that formally discusses the light clients of permissionless blockchains in game-theoretic settings, the area remains largely unexplored, and a few potential studies can be conducted for more realistic instantiations. For example, many crypto-economic protocols (e.g., PoS blockchains [14,24,34] and payment channels [18,46] already introduce locked deposits, and it becomes enticing to explore the composability of using the same collateral in the light-client protocol and other crypto-economic protocols, without scarifying the securities of all protocols.

Acknowledgment. We thank anonymous reviewers for valuable comments. Qiang is supported in part by JDDigits via the JDD-NJIT-ISCAS Joint Blockchain Lab and a Google Faculty Award.

References

1. Cardano. https://www.cardano.org/en/home/
2. Ethereum EIP-210. https://eips.ethereum.org/EIPS/eip-210
3. Abraham, I., Dolev, D., Gonen, R., Halpern, J.: Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In: Proceedings of ACM PODC 2006, pp. 53–62 (2006)
4. Babaioff, M., Dobzinski, S., Oren, S., Zohar, A.: On bitcoin and red balloons. In: Proceedings of ACM EC 2012, pp. 56–73 (2012)
5. Back, A., et al.: Enabling blockchain innovations with pegged sidechains (2014). http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains

6. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Proceedings of ACM CCS 2018, pp. 913–930 (2018)
7. Beimel, A., Groce, A., Katz, J., Orlov, I.: Fair computation with rational players (2011). https://eprint.iacr.org/2011/396
8. Bitcoin Core (2019). https://github.com/bitcoin/bitcoin
9. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 561–586. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_20
10. Bünznz, B., Kiffer, L., Luu, L., Zamani, M.: FlyClient: Super-light clients for cryptocurrencies. In: Proceedings of IEEE S&P 2020 (2020)
11. Buterin, V.: A next-generation smart contract and decentralized application platform (2014)
12. Buterin, V.: Light clients and proof of stake (2015). https://blog.ethereum.org/2015/01/10/light-clients-proof-stake/
13. CryptoKitties (2018). https://www.cryptokitties.co/
14. Daian, P., Pass, R., Shi, E.: Snow White: robustly reconfigurable consensus and applications to provably secure proof of stake. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 23–41. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_2
15. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 66–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_3
16. Dong, C., Wang, Y., Aldweesh, A., McCorry, P., van Moorsel, A.: Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In: Proceedings of ACM CCS 2017, pp. 211–227 (2017)
17. Dziembowski, S., Eckey, L., Faust, S.: FairsWap: how to fairly exchange digital goods. In: Proceedings of ACM CCS 2018, pp. 967–984 (2018)
18. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: virtual payment hubs over cryptocurrencies. In: Proceedings of IEEE S&P 2019, pp. 327–344 (2019)
19. Electrum (2011). http://docs.electrum.org/en/latest/
20. Ethereum Foundation: Solidity Global Variables (2018). https://solidity.readthedocs.io/en/develop/units-and-global-variables.html
21. Fuchsbauer, G., Katz, J., Naccache, D.: Efficient rational secret sharing in standard communication networks. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 419–436. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_25
22. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
23. Gaži, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. In: Proceedings of IEEE S&P 2019 (2019)
24. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 51–68 (2017)
25. Go Ethereum (2019). https://github.com/ethereum/go-ethereum

26. Gordon, S.D., Katz, J.: Rational secret sharing, revisited. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 229–241. Springer, Heidelberg (2006). https://doi.org/10.1007/11832072_16
27. Groce, A., Katz, J.: Fair computation with rational players. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 81–98. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_7
28. Gruber, D., Li, W., Karame, G.: Unifying lightweight blockchain client implementations. In: Workshop on Decentralized IoT Security and Standards (DISS) (2018)
29. Halpern, J., Teague, V.: Rational secret sharing and multiparty computation. In: Proceedings of ACM STOC 2004, pp. 623–632 (2004)
30. Halpern, J.Y., Pass, R.: Sequential equilibrium in computational games. ACM Trans. Econ. Comput. (TEAC) **7**(2), 1–19 (2019)
31. Halpern, J.Y., Pass, R., Seeman, L.: Computational extensive-form games. In: Proceedings of ACM EC 2016, pp. 681–698 (2016)
32. Izmalkov, S., Micali, S., Lepinski, M.: Rational secure computation and ideal mechanism design. In: Proceedings of IEEE FOCS 2005, pp. 585–594 (2005)
33. Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work (2017). https://eprint.iacr.org/2017/963.pdf
34. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
35. Kiayias, A., Zhou, H.-S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_25
36. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: Bracciali, A., Clark, J., Pintore, F., Rønne, P.B., Sala, M. (eds.) FC 2019. LNCS, vol. 11599, pp. 21–34. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_3
37. Kol, G., Naor, M.: Games for exchanging information. In: Proceedings of ACM STOC 2008, pp. 423–432 (2008)
38. Kosba, A., Miller, A., Shi, E., et al.: Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In: Proceedings of IEEE S&P 2016, pp. 839–858 (2016)
39. Kwon, J., Buchman, E.: Cosmos: a network of distributed ledgers (2017). https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md
40. Lepinksi, M., Micali, S., Shelat, A.: Collusion-free protocols. In: Proceedings of ACM STOC 2005, pp. 543–552 (2005)
41. Leung, D., Suhl, A., Gilad, Y., Zeldovich, N.: Vault: fast bootstrapping for cryptocurrencies. In: NDSS 2019 (2019)
42. Lu, Y., Tang, Q., Wang, G.: Generic superlight client for permissionless blockchains. arXiv preprint arXiv:2003.06552 (2020)
43. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of ACM CCS 2016, pp. 17–30 (2016)
44. Meckler1, I., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf
45. Miller, A.: Ethereum blockhash contract (2017). https://github.com/amiller/ethereum-blockhashes
46. Miller, A., Bentov, I., Kumaresan, R., McCorry, P.: Sprites and state channels: payment networks that go faster than lightning. In: Proceedings of FC (2019)

47. Miller, A.E., Hicks, M., Katz, J., Shi, E.: Authenticated data structures, generically. In: Proceedings of ACM POPL 2014, pp. 411–423 (2014)
48. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
49. Osborne, M., Rubinstein, A.: A Course in Game Theory (1994)
50. Park, S., Kwon, A., Fuchsbauer, G., Gaži, P., Alwen, J., Pietrzak, K.: SpaceMint: a cryptocurrency based on proofs of space. In: Proceedings of FC 2018, pp. 480–499 (2018)
51. Pass, R., Shi, E.: Rethinking large-scale consensus. In: 2017 IEEE 30th Computer Security Foundations Symposium (CSF), pp. 115–129. IEEE (2017)
52. Pham, V., Khouzani, M.H.R., Cid, C.: Optimal contracts for outsourced computation. In: Poovendran, R., Saad, W. (eds.) GameSec 2014. LNCS, vol. 8840, pp. 79–98. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12601-2_5
53. Poelstra, A.: Mimblewimble (2016). https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf
54. Protocol Labs: Filecoin: A Decentralized Storage Network (2017). https://filecoin.io/filecoin.pdf
55. Steemit (2016). https://steemit.com/
56. Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains (2017). https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf
57. Tomescu, A., Devadas, S.: Catena: efficient non-equivocation via bitcoin. In: Proceedings of IEEE S&P 2017, pp. 393–409 (2017)
58. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger (2014). https://ethereum.github.io/yellowpaper/paper.pdf
59. Xu, L., Chen, L., Gao, Z., Xu, S., Shi, W.: EPBC: efficient public blockchain client for lightweight users. In: Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, p. 1. ACM (2017)
60. Zamyatin, A., Stifter, N., Judmayer, A., Schindler, P., Weippl, E., Knottenbelt, W.J.: A wild velvet fork appears! inclusive blockchain protocol changes in practice. In: Proceedings of FC 2018, pp. 31–42 (2018)