# Blink: An Optimal Proof of Proof-of-Work


## Lukas Aumayr
TU Wien, lukas.aumayr@tuwien.ac.at

## Zeta Avarikioti
TU Wien and Common Prefix, georgia.avarikioti@tuwien.ac.at

## Matteo Maffei
TU Wien and CDL-BOT, matteo.maffei@tuwien.ac.at

## Giulia Scaffino
TU Wien and Common Prefix and CDL-BOT, giulia.scaffino@tuwien.ac.at

## Dionysis Zindros
Stanford University and Common Prefix, dionyziz@gmail.com



## Abstract

Designing light clients for Proof-of-Work blockchains has been a foundational problem since Nakamoto's SPV construction in the Bitcoin paper. Over the years, communication was reduced from $O(C)$ down to $O(\text{polylog}(C))$ in the system's lifetime $C$. We present Blink, the first *provably secure* $O(1)$ *light client that does not require a trusted setup.*



**2012 ACM Subject Classification** Security and privacy → Distributed systems security

**Keywords and phrases** PoW Blockchains, Light Client, Proof of Proof-of-Work


## 1 Introduction

It is impractical for a blockchain user, such as a wallet, to download and verify the whole chain due to communication, computation, and storage constraints. In the seminal Bitcoin white paper [29], Satoshi Nakamoto predicted this need for efficiency and designed a *light client* called the Simplified Payment Verification (SPV) protocol, which decouples the download of the execution layer data (transactions) from the consensus layer data (block headers). An SPV client retrieves all block headers and verifies them according to the longest chain consensus rule. This process requires communication that grows linearly with the systems' lifetime as the header chain grows at a roughly linear rate.

Several subsequent works optimized this concept, introducing *superlight clients* whose communication complexity is only polylogarithmic (*succinct*) in the lifetime of the system [25, 21, 22, 12]. Nevertheless, these protocols are not out-of-the-box compatible with Bitcoin but instead require a consensus fork.

Designing a client with constant communication complexity has remained an elusive goal over the past dozen years. This paper fills this gap.

**Contributions.** In this work, we present Blink, a novel *interactive PoW light client with constant communication complexity*. In a nutshell, the Blink client connects to a set of full nodes, one of which is honest. The client locally samples a random value $\eta$, includes it in a transaction $\mathsf{Tx}_\eta$, and sends it to the full nodes. For instance, $\mathsf{Tx}_\eta$ can simply be a payment to a vendor's fresh address, which was sampled with high entropy. Then, Blink waits for $\mathsf{Tx}_\eta$ to be included in a (high-entropy) block and confirmed. The full nodes respond to the client with a proof $\pi$ consisting of $2k+1$ consecutive blocks, with the high-entropy block in the middle and $k$ blocks before and after it (see Figure 1); $k$ is the security parameter [18], e.g., the conventional 6 confirmation blocks in Bitcoin. Importantly, full nodes do not send the full header chain to the client. The constant-sized proof $\pi$ ensures that *the first block in the* *proof is stable in the chain* and, therefore, it can be considered as a checkpoint or, in other words, as a new genesis block $\mathcal{G}'$.

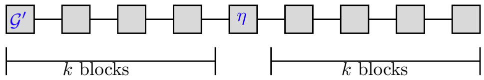


**Figure 1** Structure of the Blink's proof $\pi$. The proof $\pi$ consists of $2k+1$ consecutive blocks, with the block including the entropy $\eta$ in the middle, and $k$ blocks before and after it. The first block $\mathcal{G}'$ in $\pi$ is stable in the chain and acts as a new genesis block.

We highlight that *Blink does not require any trusted setup*, and we prove it secure under an honest majority of computational power, i.e., against less than 1/2 adversaries. We analyze security in the static PoW model introduced by the Bitcoin Backbone [18], and we adopt the light client state security definitions introduced in [32]. In this model, we refine the problem of Proofs of Proofs-of-Work [25] and prove that Blink has optimal communication cost, building the *first provably secure Optimal Proof of Proof-of-Work (OPoPoW) without trusted setup*.

Furthermore, Blink is a powerful tool that can be leveraged to develop a plethora of applications with enhanced efficiency compared to state-of-the-art protocols. Specifically, we present the following applications, all with constant communication costs:

1. Bootstrapping PoW blockchain clients, full nodes, and miners
2. Active payment verification
3. Past payment and ledger state verification
4. Bridging PoW blockchains

In further detail, Blink naturally provides a *bootstrapping method*: an SPV client or a light miner[1] broadcasts $\mathsf{Tx}_\eta$, and upon receiving $\pi$, they can efficiently select the tip of the current longest chain $\mathcal{G}'$, and start running their protocol on top of it. As a result, the bandwidth cost of bootstrapping is reduced from linear to constant with respect to the lifetime of the system.

Second, Blink allows *trustless and efficient verification of on-chain payments*. In particular, upon identifying the new genesis $\mathcal{G}'$, an SPV protocol is executed on top of $\mathcal{G}'$, finalizing the payment as soon as $k$ confirmation blocks appear on top of $\mathsf{Tx}_\eta$ in the longest chain. Hence, the payment protocol has the same latency as a standard SPV client, but only constant communication complexity as at most $3k+1$ blocks are relayed in total for finalizing a payment.

Third, assuming block headers include a commitment to the state of the ledger (e.g., Ethereum PoW and ZCash) or include an efficient way of verifying the history of transactions (i.e., ancestry proofs such as block interlinking in the form of Merkle Mountain Ranges [4, 3] or vector commitments [13]), the client enables the extraction of any historical state of the ledger from $\mathcal{G}'$ (including the current one). This means that users can use the Blink-based payment protocol to *read any past transaction as well as any historical state of a smart contract*. This approach reduces the communication overhead from polylogarithmic, as seen in state-of-the-art protocols [25, 12], to constant (in the system's lifetime).

Finally, Blink can serve as a building block for optimistic bridges, where $\pi$ is used as a fraud proof. This way, Blink enables the *first trustless, secure PoW bridge with constant*

[1] Light miners do not validate transactions included in the chain before they booted up thus, to be sure they start mining on the correct tip of the chain, they need to run an efficient protocol to identify the current longest chain [23]. They start fully verifying transactions after bootstrapping.

*communication* for relaying a transaction from a source to a destination chain. We also prove that a recent work claiming a trustless constant-size bridge construction [31] is, in fact, insecure.

We provide a *proof-of-concept implementation* of Blink, and evaluate its communication cost for the conventional confirmation block value $k = 6$ and the block height at the time of writing. We underscore that Blink improves on all previous light client solutions in terms of bandwidth: SPV requires 67.3MB, NIPoPoWs requires 10KB, FlyClient requires 5KB, ZK ZeroSync requires 197KB, whereas Blink requires only 1.6KB. All of the solutions have the same latency as they all have to wait for $k$ confirmation blocks.

**Related Work.** The description of Nakamoto's SPV client appears already in the paper that introduced Bitcoin [29]. A series of optimizations followed. The first succinct construction was the interactive *Proofs of Proof-of-Work protocol* [22] with polylogarithmic communication costs. Later work removed this interactivity and achieved security against 1/2 adversaries but succinctness only in the optimistic setting (against no adversaries) [25]. This construction was subsequently optimized [21], made practical [15], and redesigned with backwards compatibility in mind [26]. The optimistic setting limitation was alleviated in a follow-up work, achieving succinctness against all adversaries up to a 1/3 threshold [24]. An alternative construction was also proposed, enabling security and succinctness against a 1/2 adversary, and adding support for variable difficulty [12]. All these solutions require polylogarithmic communication, whereas Blink requires only constant.

Recently, generic (recursive) zero-knowledge (ZK) techniques were utilized to build constant communication light clients [11, 5, 33]. However, these approaches incur prohibitively high computational costs (or necessitate specialized blockchain deployments [5, 33] utilizing ZK-friendly cryptographic primitives [20]) and additionally require a trusted setup to generate and prove verification keys (which can only be removed if polylogarithmic communication is acceptable). Contrarily, Blink does not impose high communication costs nor a trusted setup.

To develop a constant communication light client without a trusted setup, the idea of using only a small segment of the chain near the tip was proposed [2]. However, the proposed construction was shown to be susceptible to pre-mining attacks and thus insecure [31]. Recently, another construction was introduced called Glimpse [31], combining the idea of [2] with the injection of a *high-entropy* transaction (which was originally introduced in [37, Chapter 5] but for a different purpose) to prove the provided segment of the chain is "fresh" and not pre-mined. Nevertheless, Glimpse remains insecure as we show in this work. We also leverage these ideas to design Blink, the first provably secure light client with constant communication that does not require a trusted setup.

Finally, a similar quest for proof of stake light clients has achieved polylogarithmic complexity in an interactive setting [10]. For a review of the long-standing light client problem, see [14]. Light clients are also a cornerstone for building trustless bridges between chains, a question that has been explored in a multitude of works [34, 36, 28, 19]. In this work, we demonstrate how Blink can be utilized to construct a trustless and efficient optimistic bridge.

**Comparison.** In Table 1, we compare the characteristics of existing light client protocols, including Blink. We denote by $C$ the lifetime of the system (informally, the length of the blockchain) and by $k$ the security parameter. According to the Bitcoin Backbone model, $k$ is the *common prefix* parameter, which is constant for a protocol execution, albeit with the trade-off of logarithmically increasing the probability of failure in the lifetime of the system.

We first observe that Glimpse [31] achieves $O(k)$ communication but it is not secure in

| | SPV[29] | KLS[22],NIPoPoW [25] FlyClient[12], Mining LogSpace[24] | ZK Clients[33, 5, 11] | Glimpse [31] | **Blink** |
|---|---|---|---|---|---|
| Communication Complexity | $\mathcal{O}(C)$ | $O(k \text{ polylog}(C))$ | $\mathcal{O}(1)$ | $\mathcal{O}(k)$ | $\mathcal{O}(k)$ |
| No Trusted Setup | ✓ | ✓ | ✗ | ✓ | ✓ |
| Adv. Resilience | 1/2 | 1/2 | 1/2 | ✗(?) | 1/2 |

**Table 1** Comparison of light client solutions

the honest majority assumption (as shown in this work); its exact resilience, if any, remains unknown. ZK clients, on the other hand, achieve $O(1)$ communication but necessitate a trusted setup; unlike Blink in which such assumption is not necessary. We further expose a trade-off between communication overhead and interactivity: prior state-of-the-art PoPoWs are non-interactive but require $O(k \text{ polylog}(C))$ communication [25, 12, 22, 24]. Contrarily, Blink only requires $O(k)$ communication but needs one round of interaction.

# 2 Protocol Design

In this section, we introduce Blink, the first provably secure, optimal PoW light client that does not require a trusted setup. We begin with a high-level overview of our protocol's objectives and introduce a protocol abstraction that embodies these goals. Next, we present Blink, describing in simple terms the rationale behind its design and security. Throughout this work, we will use the term block to mean a block header.

## 2.1 Optimal Proof of Proof-of-Work Client

A client protocol is an interactive protocol between a set of provers $P \in \mathcal{P}$ maintaining a ledger, and a verifier $V$, i.e., the client. If the provers convince $V$ about the current state of the ledger without asking $V$ to download the whole ledger or execute all the transactions, then the client is a *light client*. In particular, if the verifier only receives a *constant* amount of data independently of the ledger's lifetime, then the light client protocol has *optimal* communication.

A client is convinced about the state of a ledger or, simply, of a blockchain, if it receives a block $\mathsf{B}$ fulfilling the following properties: (a) $\mathsf{B}$ is safe, i.e., it will never be reverted in the view of an honest node; (b) $\mathsf{B}$ is live, i.e., $\mathsf{B}$ was created recently and therefore the client has an up-to-date view of the state of the blockchain.

Our goal is to design a client protocol that, with only constant communication complexity, satisfies the security notions defined in (a) and (b). Figure 2 illustrates a client protocol abstraction that realizes our objectives. It showcases the interaction between the set of provers ($\mathcal{P}$) and the verifier ($V$) highlighting the pivotal components of our construction which ensure security: the *constant-sized proof* $\pi$ that $P$ sends to $V$ and the *extraction of block* $\mathsf{B}$ from $\pi$, allowing $V$ to read the current state of the ledger.

Throughout the remainder of this work, we will omit discussing the initiation step, as it remains the same. For simplicity, we treat ledgers extracted from blockchains only, i.e., we assume the ledger is generated as the output of a blockchain protocol (and not, e.g., a DAG protocol). We generalize the discussion in later sections.

## 2.2 Blink Client

The ultimate goal of an OPoPoW client is to identify a recent, correct block of the ledger, by only receiving a constant-sized amount of data from the set of provers. Towards this, we

Initiation

**1.** $V$ connects to a set of nodes in $\mathcal{P}$

Proof Construction

**2.** Nodes in $\mathcal{P}$ may interact with $V$
**3.** Nodes in $\mathcal{P}$ send to $V$ a (set of) *constant-size* proof(s) $\pi$ that includes a block $\mathsf{B}$ that is safe and live

Block Extraction

**4.** $V$ verifies $\pi$
**5.** $V$ extracts $\mathsf{B}$ from $\pi$ and terminates

**Figure 2** Abstraction of an OPoPoW client protocol

start with a naive client construction; we identify security threats and propose solutions until we converge to a secure client protocol.

**A Naive Construction.** Let us start analyzing one of the simplest constructions one might think of. The provers give the last $k+1$ consecutive blocks in their longest chain to the client, who in turn verifies the validity of these blocks and accepts the first (the oldest) block in the proof as safe and live. We recall that in PoW blockchains, blocks are considered final after they have $k$ confirmation blocks, where $k$ is the safety parameter, e.g., in Bitcoin folklore blocks are considered final after 6 confirmations. According to [18], $k$ is a constant for a protocol execution, which, however, implies that the blockchain's security bounds degrade logarithmically with its lifetime. Since the client checks their validity, all blocks in the proof fulfill the PoW difficulty requirements. Trivially, this construction is broken: adversarial provers can have pre-mined $k+1$ fake blocks stored somewhere, and when the client boots up, they provide the client with a block that is either not part of the longest chain or is outdated. Upon receiving different $k+1$ blocks from honest and adversarial provers, the client cannot identify the correct chain with higher probability than random guessing.

**Preventing Upfront Mining Attacks.** To prevent this upfront mining attack, the client $V$ can locally sample a random string $\eta$ and give it to the provers along with a time window $T$, within which $V$ accepts a proof $\pi$ [31]. Then, provers can then broadcast an *entropy transaction* $\mathsf{Tx}_\eta$ which embeds $\eta$ to the blockchain network, and wait for it to be included in a block. We will call this block $\dot{\mathsf{B}}$ and since it contains $\eta$, this is a high-entropy block. Before the timeout $T$ expires, if $k$ blocks are built on top of $\dot{\mathsf{B}}$, $P$ sends to $V$ a proof $\pi$ consisting of $\dot{\mathsf{B}}$ followed by its $k$ confirmation blocks. Finally, $V$ accepts $\dot{\mathsf{B}}$. Figure 3 illustrates the protocol presented in [31]; $\lambda$ is a security parameter. While randomizing the proof $\pi$ solves the upfront mining attack, it does not lead us to a secure client protocol.

Indeed, we observe that an adversary $\mathcal{A}$ has a probability $\frac{t}{n} < \frac{1}{2}$ to be elected as PoW block proposer, with $n$ the total number of participants in the PoW game, out of which $t$ are controlled by the adversary. This means that $\mathcal{A}$ has a non-negligible probability to censor $\mathsf{Tx}_\eta$ in the first $k-1$ blocks after $\mathsf{Tx}_\eta$ was broadcast. If $T$ is such that fewer than $2k$ consecutive blocks are produced in $T$ with overwhelming probability, the adversary can violate the liveness of the client with probability $\frac{t}{n}$, because honest parties cannot produce a valid proof of $k+1$ blocks within time $T$. Figure 4 demonstrates this attack.

**Preventing Liveness Attacks.** To protect the client from this liveness attack, one could take different directions: (A) remove the time window $T$ (alternatively, increase it such that at least $2k$ blocks are produced in $T$ with overwhelming probability), or (B) accept proofs of length less than $k$. In (A), $V$ accepts the first proof $\pi$ it receives, with $\pi$ consisting of $\dot{\mathsf{B}}$ and

Proof Construction

5. $V$ samples $\eta \xleftarrow{\$} \{0,1\}^\lambda$
6. $V$ selects a time $T$ in the future that corresponds to the expected creation time of $k+1$ blocks
7. $V$ sends $\eta$ and $T$ to every $P \in \mathcal{P}$ along with a request to return a light client proof $\pi$ of length $k+1$ conditioned to $\eta$, within time $T$
8. $\mathcal{P}$ construct an entropy transaction $\mathsf{Tx}_\eta$ containing $\eta$ and broadcast it to the blockchain network
9. As soon as a party $P \in \mathcal{P}$ has a proof $\pi$ consisting of a block $\dot{\mathsf{B}}$ containing $\mathsf{Tx}_\eta$ with $k$ confirmations blocks, $P$ sends $\pi$ to $V$

Block Extraction

10. $V$ accepts $\pi$ if it was received within time $T$, $\dot{\mathsf{B}}$ contains $\mathsf{Tx}_\eta$ and has $k$ confirmation blocks on top of it
11. $V$ extracts $\dot{\mathsf{B}}$ from $\pi$ and terminates

**Figure 3** Naive client protocol [31].

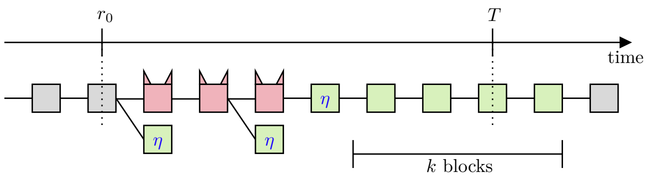


**Figure 4** Consider $k = 4$. The light client boots at round $r_0$ and broadcasts the entropy $\eta$. With significant probability, the adversary can censor $\mathsf{Tx}_\eta$ in the first $k-1$ blocks after $\mathsf{Tx}_\eta$ was broadcast. It results that honest parties might not find a proof $\pi$ of sufficient length by the timeout $T$.

at least $k$ confirmation blocks on top of it. In (B), $V$ accepts the proof $\pi$ that, by $T$, has the most confirmation blocks on top of $\dot{\mathsf{B}}$. In Figure 5, we present the client protocol for the (A) and (B) variants. While both these attempts safeguard the liveness of the client, the client's safety is broken: $V$ might accept a block that is not part of any honest party's chain.

We now describe the safety attack for (A), but a similar logic applies to (B) as well. After $V$ broadcasts $\mathsf{Tx}_\eta$, honest parties immediately include it on-chain, while the adversary $\mathcal{A}$ starts mining on a private chain that censors $\mathsf{Tx}_\eta$. $\mathcal{A}$ can mine $k-l$ blocks in its private chain, with $0 < l < k-1$, while honest parties only mine $\dot{\mathsf{B}}$ with at most $k-l-2$ confirmations. This can happen with non-negligible probability, as we are considering subchains with fewer than $k$ blocks [18], with $k$ being the safety security parameter. Then, $\mathcal{A}$ broadcasts their private chain, causing all honest parties to switch to the adversarial chain due to the longest chain rule. Honest parties subsequently include $\mathsf{Tx}_\eta$ in their new longer chain and keep mining on top of it. In the meantime, $\mathcal{A}$ starts privately mining on top of the abandoned chain that included $\mathsf{Tx}_\eta$ early on. Now, to create a valid proof, $\mathcal{A}$ only needs to privately mine $l+2 < k+1$ blocks, while honest parties need to mine $k+1$ blocks. As a result, $\mathcal{A}$ can generate a valid proof faster than honest parties, and trick the client to accept a proof consisting of blocks that will not be part of the honest chain, thereby breaking security. Figure 6 illustrates this attack.

**The Blink Proof.** Before detailing Blink, we observe that the safety attack in Figure 6 relies on $\mathcal{A}$ privately mining in order to delay the inclusion of $\mathsf{Tx}_\eta$ in the main chain. However, such censoring can only succeed for a limited time, specifically less than $k$ consecutive blocks, as extending a private chain beyond this would lead to a safety violation [18]. In other words, $\mathcal{A}$ may create up to $k-1$ blocks faster than honest parties (with non-negligible probability) but not more than that: any honest majority will create $k$ blocks faster than any minority

Proof Construction

**5.** $V$ samples $\eta \xleftarrow{\$} \{0,1\}^\lambda$
**6.** $V$ sends $\eta$ to every $P \in \mathcal{P}$ along with a request to return a light client proof $\pi$ of length $k+1$ conditioned to $\eta$
**7.** $\mathcal{P}$ construct an entropy transaction $\mathsf{Tx}_\eta$ containing $\eta$ and broadcast it to the blockchain network
**8.** As soon as a party $P \in \mathcal{P}$ has a proof $\pi$ consisting of a block $\dot{\mathsf{B}}$ containing $\mathsf{Tx}_\eta$ with $k$ confirmations blocks, $P$ sends $\pi$ to $V$

Block Extraction

**9.** $V$ accepts the first $\pi$ it receives where $\dot{\mathsf{B}}$ contains $\mathsf{Tx}_\eta$ and has $k$ confirmation blocks on top of it
**10.** $V$ extracts $\dot{\mathsf{B}}$ from $\pi$ and terminates

Proof Construction

**5.** $V$ samples $\eta \xleftarrow{\$} \{0,1\}^\lambda$
**6.** $V$ selects a time $T$ in the future
**7.** $V$ sends $\eta$ to every $P \in \mathcal{P}$ along with a request to return a light client proof $\pi$ conditioned to $\mathsf{Tx}_\eta$ within time $T$
**8.** $\mathcal{P}$ construct an entropy transaction $\mathsf{Tx}_\eta$ containing $\eta$ and broadcast it to the blockchain network
**9.** At time $T$, each party $P \in \mathcal{P}$ sends to $V$ its $\pi$, consisting of $\dot{\mathsf{B}}$ containing $\mathsf{Tx}_\eta$ along with all the subsequent confirmation blocks that $P$ is aware of

Block Extraction

**10.** $V$ accepts the $\pi$ that has the most confirmation blocks on top of $\dot{\mathsf{B}}$ containing $\mathsf{Tx}_\eta$ and was received within time $T$
**11.** $V$ extracts $\dot{\mathsf{B}}$ from $\pi$ and terminates

**Figure 5** Insecure attempts (A) and (B), top and bottom respectively.

adversary, rendering the attack in Figure 6 infeasible.

The client can securely accept a block of the blockchain, if they can identify it as a *safe block*, i.e., a block that is already $k$ deep in at least one honest party's chain [18]. Furthermore, the safe block also needs to be *live*, i.e., recent enough to be sufficiently close to the tip of the chain.

We know that the adversary can only censor $\mathsf{Tx}_\eta$ for $k-1$ blocks and it takes $k$ additional blocks for $\mathsf{Tx}_\eta$ to become safe (Figure 6). Therefore, we *modify* $\pi$ *to be of length* $2k+1$ *and to specifically contain* $\mathsf{Tx}_\eta$ *in the middle block* $\dot{\mathsf{B}}$*, i.e., at position* $k+1$, as depicted in Figure 1. However, if the client accepts $\dot{\mathsf{B}}$ of the first valid proof received, safety is again violated by the same attack described before: $\mathcal{A}$ can create $\pi$ before honest parties by using the $k+(k-l-1)$ blocks from the abandoned honest chain and mining $l+1$ new blocks; meanwhile, honest parties must mine $k > l+1$ new blocks. Nonetheless, $\pi$ now being of length $2k+1$, it necessarily contains a safe block, i.e., a block that is at least $k$ deep in the chain to be stable for all honest parties; this is true even if the $\pi$ the client receives comes from $\mathcal{A}$. In particular, $\pi$ contains at least a block that was safe even before $\mathsf{Tx}_\eta$ was broadcast: the honest subchain starting from the block $\dot{\mathsf{B}}$ included early on is at most of length $k-2$ and at least of length 1, thus the first block in $\pi$ is was already part of the honest parties' stable chain (cf. Figure 7). Naturally, the first block in $\pi$ is attached to genesis, otherwise honest parties would not have extended it. This holds true regardless of the strategy $\mathcal{A}$ follows: Honest parties only abandon their chain if they see a longer one.

We note that for any $\pi$ coming from an honest party, any block before the entropy block

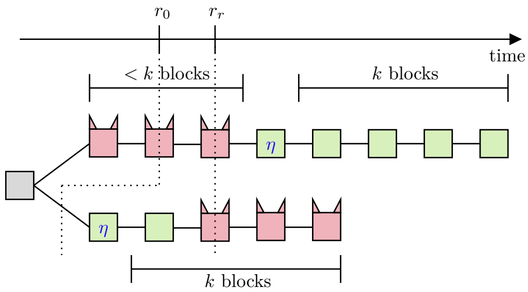


**Figure 6** Consider $k = 4$ and $l = 1$. The client broadcasts $\eta$ at $r_0$. $\mathcal{A}$ privately mines a subchain of 3 blocks censoring $\mathsf{Tx}_\eta$, while honest parties include $\mathsf{Tx}_\eta$ and only mine 2 blocks overall. At $r_r$, $\mathcal{A}$ releases the private chain, which is adopted by honest parties as per the longest chain selection rule. Honest parties now need to mine 5 blocks to find a valid $\pi$. Contrarily, $\mathcal{A}$ needs to only mine 3 blocks. Hence, $\mathcal{A}$ finds $\pi$ first.

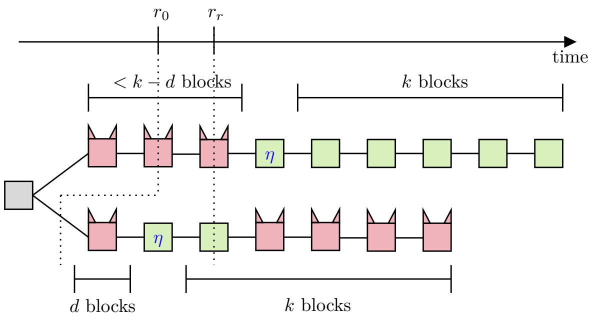


**Figure 7** Consider $k = 5$. As in Figure 6, except that $\mathcal{A}$ censors $\mathsf{Tx}_\eta$ by $d$ blocks also on the lower branch, such that $d \leq k - 1$ and s.t. the overall number of adversarial blocks before $\mathsf{Tx}_\eta$ on all branches is smaller than $k$. This shows why it is not sufficient to take less than $k$ blocks before $\mathsf{Tx}_\eta$.

$\dot{\mathsf{B}}$ is safe, as there are at least $k + 1$ confirmations. Thereby, *the first block* $\mathsf{B}$ *of any* $2k + 1$ *proof* $\pi$ *is always safe, i.e., it has at least* $k$ *confirmations in the view of an honest party.* As a result, the client can safely *accept the first block in the first valid* $\pi$ *it receives.*

**Blink Protocol.** In Figure 8 we showcase the pseudocode of the Blink protocol, while in Algorithm 1 we put forth the algorithm run by the Blink client, employing Algorithm 2; similarly, in Algorithm 3 we present the code run by provers. We use $m \dashrightarrow A$ to indicate that message $m$ is sent to party A and $m \dashleftarrow A$ to indicate that message $m$ is received from party A.

We observe that the client reads a proof of length $2k+1$, which is constant in the system's lifetime, and accepts a block that is $2k$ blocks old, incurring a waiting time of $k$ blocks, similarly to an SPV. *Blink is the first PoW light client protocol that achieves optimal proof size with only at most one round of communication between provers and verifier.*

# 3 Applications

In this section, we showcase how Blink can be used for different applications, ranging from verification of payments and state verification to bootstrapping and bridging.

## 3.1 Payment Verification

Consider a vendor that wants to check whether a particular buyer has made a payment for the purchase of a good. The vendor will only ship the goods after the client's payment has

Proof Construction

5. $V$ samples $\eta \xleftarrow{\$} \{0,1\}^\lambda$
6. $V$ sends $\eta$ to every $P \in \mathcal{P}$
7. $\mathcal{P}$ construct an entropy transaction $\mathsf{Tx}_\eta$ containing $\eta$ and broadcast it to the blockchain network
8. As soon as a party $P \in \mathcal{P}$ has $k$ confirmation blocks on top of the block $\dot{\mathsf{B}}$ containing $\mathsf{Tx}_\eta$, $P$ sends to $V$ $\pi$ consisting of $\dot{\mathsf{B}}$ with $k$ blocks before and $k$ blocks after it

Block Extraction

9. $V$ accepts the first $\pi$ it receives consisting of $2k+1$ consecutive well-formed blocks where the middle block contains $\eta$, i.e., $\pi[k] = \dot{\mathsf{B}}$
10. $V$ extracts the first block of proof $\pi$, i.e., $\mathsf{B} := \pi[0]$, and terminates

**Figure 8** Pseudocode of the Blink protocol

**Algorithm 1** The algorithm ran by the verifier $V$, i.e., the Blink client. We split the proof $\pi$ into $(\pi_0, \pi_1)$, with $\pi_0$ allowing to identify a stable and recent block of the blockchain, i.e., the new genesis $\mathcal{G}'$, and $\pi_1$ being the Merkle proof that verifies inclusion of $\eta$ into the middle block of $\pi_0$.

**function** $\textsc{Verifier}_{\mathcal{G}}()$<br>
    $\eta \leftarrow\!\!-\!\!- \{0,1\}^\lambda$<br>
    **for** $P \in \mathcal{P}$ **do**<br>
        $\eta \dashrightarrow P$<br>
        **while** True **do**<br>
            $\pi \dashleftarrow P$ ▷ Only constant amount of data downloaded<br>
            $(\pi_0, \pi_1) = \pi$<br>
            **if** $\textsc{Valid}_{\mathcal{G}}(\pi, \eta)$ **then**<br>
                **return** $\pi_0[0]$<br>
            **end if**<br>
        **end while**<br>
    **end for**<br>
**end function**

been verified. The Blink protocol, as described in Figure 8, only gives security for the first block in the proof and not, in particular, for the block containing $\mathsf{Tx}_\eta$: indeed, the proof $\pi$ accepted by the client might come from the adversary and, thus, the entropy block might not belong to the stable chain. In the payment setting, however, it is desirable to define *security of the stable entropy block* $\mathsf{B}_\eta$: $\mathsf{Tx}_\eta$ is the transaction of the payment to the vendor, with $\eta$ now being an address freshly sampled at random by the vendor. Assume the buyer has paid the correct amount to the vendor's new address. To argue about the finality of the payment, i.e., the finality of $\mathsf{Tx}_\eta$, we recall the strong security guarantee that Blink achieves: Blink allows us to define a recent, trustlessly identified, stable block. This block behaves as a secure checkpoint or, in other words, as a new genesis $\mathcal{G}'$: it is in the stable chain of honest parties, i.e., it will never be reverted, and the consensus rules applied to $\mathcal{G}'$ are consistent to the consensus rules applied to the genesis block $\mathcal{G}$. We now show how to extend the Blink protocol to verify payments. Upon accepting a proof $\pi$ and identifying $\mathcal{G}'$, the client can send $\mathcal{G}'$ to all provers, and provers start sending to the client all the blocks descending from $\mathcal{G}'$. The client now maintains the longest chain descending from $\mathcal{G}'$, essentially running an SPV algorithm with $\mathcal{G}'$ as a starting point. When $\mathsf{Tx}_\eta$ is in a block that is $k$-deep in the longest chain (this will happen, at most, $3k$ consecutive blocks on top of $\mathcal{G}'$), the client considers the payment final and terminates.

With one additional round of communication, Blink can now verify payments with a constant-sized proof. We observe that the client latency is the same one of a standard SPV

**Algorithm 2** The algorithm ran by $V$ to check the validity of the blocks in the proof. Let $x$ be the root of the transaction Merkle tree in a block, and $s$ be its parent hash.

**function** $\textsc{Valid}_{\mathcal{G}}(\pi, \eta)$
  $(\pi_0, \pi_1) \leftarrow \pi$
  **if** $|\pi_0| < k + 1$ **then**
    **return** False
  **end if**
  **if** $\neg\textsc{MerkleVerify}(\pi_1, \eta) \vee \pi_1.root \neq \pi_0[k+1].x$ **then**
    **return** False
  **end if**
  $h = \pi_0[0].s$
  **for** B $\in \pi_0$ **do**
    **if** B.s $\neq h$ **then** ▷ Ancestry failure
      **return** False
    **end if**
    $\mathrm{h} = H(\mathsf{B})$
    **if** $\mathrm{h} \geq \mathrm{T}$ **then** ▷ Hardcoded target T, static setting
      **return** False ▷ PoW failure
    **end if**
    **return** $\mathcal{G} = \pi_0[0] \vee |\pi_0| = 2k + 1$
  **end for**
**end function**

**Algorithm 3** The algorithm ran by the provers $P \in \mathcal{P}$.

**function** $\textsc{Prover}()$
  $\eta \dashleftarrow V$
  $\mathsf{Tx}_\eta \dashleftarrow \textsc{MakeTx}(\eta)$
  $\mathsf{Tx}_\eta \dashrightarrow \textsc{Network}$ ▷ Wait for $\mathsf{Tx}_\eta$ to be $k$-confirmed
  $\pi_0 \dashleftarrow \mathcal{C}[-(2k+1){:}]$ ▷ By Common Prefix, $\mathsf{Tx} \in \mathcal{C}[-(2k+1){:}]$
  $\pi_1 \dashleftarrow \textsc{MerkleProve}(\mathcal{C}[k+1], \eta)$
  $\pi \leftarrow (\pi_0, \pi_1)$
  $\pi \dashrightarrow V$
**end function**

client, i.e., $k$ blocks when there is no adversarial attack, and $2k$ when under attack. In Figure 9 we show the pseudocode for the Blink-based protocol for payment verification. We note that the Blink construction can be used out-of-the-box to verify payments in the Bitcoin Backbone protocol in the static difficulty setting. We refer the reader to Section 6 for variable difficulty and practical deployment.

## 3.2 Bootstrapping via Blink

In blockchains, there is an interplay between different types of parties: *consensus nodes*, *full nodes*, and *clients*. *Consensus nodes*, also called miners, receive transactions from the network (environment) and execute a distributed protocol that outputs a ledger, i.e., a finite, ordered sequence of transactions identical for all nodes. *Full nodes* do not participate in the distributed ledger protocol; instead, they receive the ledger from consensus nodes, execute transactions to verify their validity, and maintain the ledger. Finally, *clients* connect to full nodes to retrieve a specific state element from the ledger, e.g., an account balance. Bootstrapping these nodes usually requires a lot of time (from several hours to several days) and resources because, starting from genesis, they need to download and execute all the transactions in the ledger (full and consensus nodes) or verify all the blocks in the ledger

Proof Construction

5. $V$ samples $\eta \xleftarrow{\$} \{0,1\}^\lambda$
6. $V$ sends $\eta$ to every $P \in \mathcal{P}$
7. $\mathcal{P}$ construct an entropy transaction $\mathsf{Tx}_\eta$ containing $\eta$ and broadcast it to the blockchain network
8. As soon as a party $P \in \mathcal{P}$ has $k$ confirmation blocks on top of the block $\dot{\mathsf{B}}$ containing $\mathsf{Tx}_\eta$, $P$ sends to $V$ $\pi$ consisting of $\dot{\mathsf{B}}$ with $k$ blocks before and $k$ blocks after it
9. $V$ accepts the first $\pi$ it receives consisting of $2k+1$ consecutive well-formed blocks where the middle block contains $\eta$, i.e., $\pi[k] = \dot{\mathsf{B}}$
10. Upon accepting $\pi$, $V$ extracts the new genesis $\mathcal{G}' := \pi[0]$ and sends $\mathcal{G}'$ to all $P \in \mathcal{P}$
11. Each $P \in \mathcal{P}$ keeps sending all the blocks descending from $\mathcal{G}'$ in their chain

State Extraction

12. $V$ maintains the longest chain $\mathcal{C}$ descending from $\mathcal{G}'$
13. When $\mathsf{Tx}_\eta$ is $k$ deep in $\mathcal{C}$, $V$ extracts the state from the block including $\mathsf{Tx}_\eta$ and terminates

**Figure 9** Pseudocode of the payment verification with Blink

(SPV-based clients).

In Section 3.1 we used Blink to identify a recent stable block that behaves as a new genesis $\mathcal{G}'$ and, commencing from this block, our client started running an SPV protocol, i.e., the one often run by (light) clients. Blink can thus serve as an efficient bootstrapping protocol that allows the identification of a new stable block $\mathcal{G}'$ and, from that block (e.g., using the state commitment therein), runs the protocol of a consensus, full, or light node. In this way, nodes do not have to execute the entire transaction history or download past blocks but start executing only from transactions $2k$ blocks in the past.

## 3.3 State Verification

In this work, we demonstrated how to convince a light client about the state of a ledger, incurring only constant communication overhead. As specified in Section 2.1, Blink operates on the premise that each block embeds a constant-sized commitment to the current state of the ledger. Commitments come in different flavors (Merkle tree-based commitments, accumulators, vector commitments), and they are used to download and verify the UTXO set or account balances after the block, including it, has been successfully executed.

Having a chain with state commitments enables Blink to be used to verify more than just payments: *Blink allows to verify account balances and read the current state of on-chain contracts*. For a discussion on chains that have state commitments and how to introduce them to systems like Bitcoin, see Section 6.

## 3.4 Historical Transaction Verification

While it is uncommon to verify very old transactions, it might be necessary for some applications to verify, e.g., a few weeks old transactions. In these cases, once Blink identifies the new genesis $\mathcal{G}'$, one could travel back the chain block by block until hitting the block containing the transaction to be verified. While Blink has constant communication, this is a naive approach for checking past transactions that comes with a linear overhead: the older the transaction, the more blocks the client has to download. More advanced techniques called *proof of ancestry*, achieve better performances in proving that a block is an ancestor of another block: these include using Merkle Mountain Ranges (MMRs), i.e., extensions of

Merkle trees that allow for efficient appends in logarithmic openings, or vector commitments with constant opening. It follows that Blink allows to succinctly synchronize with the current state of the ledger and, from there, using a proof of ancestry, to travel back the transaction history until verifying the desired old transaction. When verifying historical transactions, the communication of Blink remains constant but can be combined with a linear, logarithmic, or constant proof of ancestry.

### 3.5 Bridging with Blink

After more than 15 years of research and work from academia and industry alike, the blockchain space has grown in a variety of 100+ chains, each presenting different and unique features in terms of consensus, privacy, throughput, applications, and programmability. To leverage these diverse opportunities and to enhance users' flexibility in the crypto world, light clients have recently become a pivotal component for bridges as well, allowing them to efficiently and securely read the state of a chain within new resource-constrained environments: blockchain themselves.

Successful bridges move a high volume of transactions: ideally, at least one transaction per block. In this case, every block that includes a cross-chain transaction must be relayed by the bridge from the source to the destination chain, in an SPV-like fashion. However, contrarily to an SPV client, the on-chain costs of the bridge can be minimized by avoiding verifying blocks by default. Instead, blocks can be optimistically accepted and only verified on-demand, i.e., in case a dispute is raised. This is what an *optimistic bridge* does. We demonstrate how to *use Blink for creating succinct fraud proofs to resolve disputes.*

Consider a PoW source blockchain $\mathcal{C}_S$ including state commitments in its blocks and allowing for efficient ancestry proofs. *Relayers* of the bridge can optimistically relay a stable block $\mathsf{B}$ from $\mathcal{C}_S$ to the destination blockchain $\mathcal{C}_D$, by submitting $\mathsf{B}$ along with a random string $\eta_R$ they sampled to the smart contract, where the bridge is deployed. Should a *challenger* notice misbehavior, they have a time window to start a challenge in which they pinpoint the contested block $\mathsf{B}$ and they reveal a random string $\eta_C$ to the bridge contract. The challenger proceeds to publish a transaction $\mathsf{Tx}_\eta$ on $\mathcal{C}_S$, which includes $\eta := \eta_R \oplus \eta_C$ (where $\oplus$ is bit-wise xor). Both parties need to contribute with a random string to prevent each of them from cheating, i.e., pre-mining a fake proof. The bridge contract will accept, from anyone, the first valid proof $\pi$ containing $\mathsf{Tx}_\eta$, and via ancestry proof it can verify whether or not $\mathsf{B}$ is an ancestor of the first block in $\pi$ by checking the block height. If it is not, $\mathsf{B}$ is removed from the bridge contract. Honest behavior can be incentivized through collateral that is slashed or redistributed in case of misbehavior.

## 4 Model

### 4.1 Notation

The bracket notation $[n]$ refers to the set $\{1, \ldots, n\}$ for a natural number $n$. $A[i]$ denotes the $i$-th element (starting from 0) of a sequence $A$, while negative indices like $A[-i]$ refer to the $i$-th element from the end. $A[i:j]$ represents the subsequence of $A$ from index $i$ (inclusive) to $j$ (exclusive), while $A[i:]$ and $A[:j]$ represent the subsequences from $i$ onwards and up to $j$, respectively. The notation $|A|$ denotes the size of the sequence $A$. The symbols $A \preceq B$ and $A \prec Y$ indicate that $A$ is a prefix or a strict prefix of $B$ or $Y$, respectively.

We denote with $\mathcal{C}_r^{\bigcap} := \bigcap_{P \in \mathcal{H}} \mathcal{C}_r^P$ the intersection of the view of all honest parties' chains at round $r$. Similarly, we denote with $\mathcal{C}_r^{\bigcup} := \bigcup_{P \in \mathcal{H}} \mathcal{C}_r^P$ the union of the chains of all honest parties, that yields a blocktree. For simplicity, we extend our slicing notation that chops off the last $k$ elements of a sequence, i.e., $[: -k]$, to trees as well. For trees, it works as follows. For every leaf in a tree, select that leaf and the $k-1$ preceding nodes. Then, for every leaf, remove all selected nodes. The slicing notation for trees will be helpful later on, when distinguishing between a stable chain in the view of all honest parties and a stable chain in the view of at least one honest party. It follows that $\mathcal{C}_r^{\bigcap}[: -k]$ is the intersection of the view of the blockchain of all honest parties at round $r$, pruned of the last $k$ blocks; likewise, $\mathcal{C}_r^{\bigcup}[: -k]$ is the union of the view of the blockchain of all honest parties at round $r$, pruned of the last $k$ blocks. In Lemma 42 (Appendix A.1), we prove that $\mathcal{C}_r^{\bigcup}[:-k] = \bigcup_{P \in \mathcal{H}} \mathcal{C}_r^P[:-k]$.

We say a block *extends* another block, if the former has the latter as ancestor and has a higher block height. We say a block *descends* from another block, if the former extends the latter or they are the same block. Finally, two blocks are *parallel* when they have the same height.

## 4.2 Ledger Model

We assume a synchronous network, i.e., all honest parties are guaranteed to receive messages sent by honest within a known delay. We consider the protocol execution to proceed in discrete rounds.

▶ **Definition 1** (Ledger). *A* ledger *is a sequence of transactions.*

▶ **Definition 2** (Distributed Ledger Protocol). *A* distributed ledger protocol *is an Interactive Turing Machine which exposes the following methods:*

- execute*: Executes a single round of the protocol, during which the machine can communicate with the network.*
- write *(*Tx*): Takes transaction* Tx *as input.*
- read *(): Outputs a ledger.*

A distributed protocol that returns a total order of the input transactions for all consensus nodes, satisfies two key properties: safety and liveness. The notation $\mathcal{L}_r^P$ denotes the output of read () invoked on party $P$ at the end of round $r$.

▶ **Definition 3** (Safety). *A distributed ledger protocol is* safe *if:*

- *(Self-consistency) For any honest party $P$ and any rounds $r_1 \leq r_2$, it holds that $\mathcal{L}_{r_1}^P \preceq \mathcal{L}_{r_2}^P$.*
- *(View-consistency) For any honest parties $P_1, P_2$ and any round $r$, it holds that either $\mathcal{L}_r^{P_1} \preceq \mathcal{L}_r^{P_2}$ or $\mathcal{L}_r^{P_2} \preceq \mathcal{L}_r^{P_1}$.*

▶ **Definition 4** (Liveness). *A distributed ledger protocol is $u$-*live *if all transactions written to any honest party at round $r$, appear in the ledgers of all honest parties by round $r+u$.*

The ledger uniquely defines the system's current state. An empty ledger is equivalent to a constant genesis state, denoted as $st_0$. To ascertain the state of a non-empty ledger, transactions from the ledger are sequentially applied to the state, starting from the genesis state. This transaction application to the existing state is encapsulated by a transition function $\delta$. For a given ledger $\mathcal{L} = \{tx_1, \ldots, tx_n\}$, the state of the system is $\delta(\ldots\delta(\delta(st_0, tx_1), tx_2)\ldots, tx_n)$.

We use the shorthand notation $\delta^*$ to apply a sequence of transactions $tx = \{tx_1, \ldots, tx_n\}$ to a state. Specifically, $\delta^*(st_0, tx) = \delta(\ldots\delta(\delta(st_0, tx_1), tx_2)\ldots, tx_n)$.

**Prover-Verifier Model.** A client protocol is an interactive protocol between the client, acting as verifier $V$, and a non-empty set of full nodes, acting as provers $P \in \mathcal{P}$. We focus on a client $V$ that bootstraps on the network for the first time and it is only aware of the genesis state.

We assume that the client is honest and connects to at least one honest prover, in accordance with the standard non-eclipsing assumption. While honest parties adhere to the correct protocol execution, the adversary can execute any probabilistic polynomial-time algorithm.

We can now define state security for client protocols, as originally introduced in [32]. Assuming safety, we use $\mathcal{L}_r^{\bigcup}$ to denote the longest among all the ledgers kept by honest parties at the end of round $r$, and $\mathcal{L}_r^{\bigcap}$ to denote the shortest among them.

▶ **Definition 5** (Ledger Client State Security [32]). *An interactive Prover-Verifier protocol* $\Pi(P, V)$ *is* state secure *with safety parameter* $v$*, if the state commitment* $\langle \mathsf{st} \rangle$ *output by* $V$ *at the end of the protocol execution at round* $r$ *satisfies safety and liveness as defined below.*

*There exists a ledger* $\mathcal{L}$ *such that* $\langle \delta^*(\mathsf{st}_0, \mathcal{L}) \rangle = \langle \mathsf{st} \rangle$*, and* $\forall r' \geq r + v$*:*

***Safety:*** $\mathcal{L}$ *is a prefix of* $\mathcal{L}_{r'}^{\bigcup}$*.*

***Liveness:*** $\mathcal{L}_r^{\bigcap}$ *is a prefix of* $\mathcal{L}$*.*

When a client gets knowledge of the state of the ledger without downloading the entire ledger or executing all transactions, it is a *light client*. Ideally, a light client learns the desired state element by downloading asymptotically less data than a full node. We measure the performance of a client protocol by defining the *communication cost* for the verifier. In other words, for a specific client protocol we measure the data received by the verifier in the proof construction ($\pi$) phase.

▶ **Definition 6** (Client Communication Cost). *We define* $\mathsf{cost}(\mathcal{E}, V)$ *to be the communication cost (in bits) of an execution* $\mathcal{E}$ *of a protocol* $\Pi(\mathcal{P}, V)$ *for party* $V$*.*

We say that a client protocol has *optimal communication cost* if $\mathsf{cost}(\mathcal{E}, V) = O(1)$, i.e., the verifier receives only a constant amount of data. In particular, we will show later that Blink is a light client with optimal communication cost $\mathsf{cost}(Blink) = O(k) = O(1)$, where $k$ is the safety security parameter that is constant for a protocol execution [18].

▶ **Definition 7** (Optimal Proof-of Proof-of-Work Protocol (OPoPoW)). *A light client protocol is an* Optimal Proof-of Proof-of-Work *protocol when it is secure (Definition 5) and has optimal communication cost (Definition 6).*

## 4.3 PoW Blockchain Model

A blockchain protocol is a distributed ledger protocol that operates typically as follows: Consensus nodes receive and broadcast chains composed of blocks. Each node $P$ maintains a view of the blockchain, denoted by $C^P$, which invariably starts with the genesis block $G$. Nodes verify these chains by ensuring they comply with the validity and consensus rules. These chains include fixed-size transactions arranged in a specific order. Every node interprets its chain to produce a transaction sequence, i.e., to output its ledger. Moreover, a consensus node receives new, unconfirmed transactions from the network, and attempts to add them to its ledger by proposing a new block that includes them. The nodes' local views the ledger can vary from node to node because of the network latency. Honest nodes adhere to the consensus protocol, while adversarial nodes may diverge from it. Nevertheless, under specific assumptions, a blockchain protocol may guarantee that the local chains of different parties satisfy the two key properties of ledgers, namely safety and liveness, albeit typically in a probabilistic manner.

To model the proof-of-work setting, the *q-bounded synchronous setting* defined in [18] can be leveraged. The protocol is analyzed in the static model, where the number of consensus nodes $n$ remains fixed throughout the protocol execution, albeit not known to the nodes themselves. Furthermore, each of them is assumed to have an equal computational power (flat model). The protocol proceeds in synchronous communication rounds. We highlight that the static model implies *static difficulty*, i.e., the PoW difficulty remains the same throughout the protocol execution. The limited capability of the nodes to generate PoW solutions is captured by their restricted access to the hash function $H(\cdot)$ modeled as a Random Oracle; each node is allowed $q$ queries per round. The adversary controls up to $t < \frac{n}{2}$ nodes, meaning they are allowed $t \cdot q$ queries per round. The adversary can insert messages, manipulate their order, and launch Sybil attacks, creating seemingly honest messages. However, the adversary cannot censor honest parties' messages, ensuring that all honest parties receive honestly broadcast messages.

The Bitcoin Backbone model [18] identifies three security properties of a blockchain: *common prefix*, *chain quality*, and *chain growth*. Informally, common prefix dictates that at any point in time, any two honest parties' chains after pruning the last $k$ blocks are either the same or one is a prefix of the other. Chain growth expresses that the blockchain makes progress at least at the pace at which the honest parties produce blocks. Finally, chain quality captures the ratio of honestly produced blocks in the system in any long enough chunk of the chain. The formal definitions can be found in Appendix A.1. A blockchain protocol satisfying common prefix, chain quality, and chain growth also maintains a secure ledger, as per Definition 3 and Definition 4, under the so-called *$k$-deep confirmation rule*. This rule states that all nodes consider a block safe when it is part of their local chain pruned by the last $k$ blocks. As expected, both safety and liveness hold probabilistically.

**Chain Client Security.** As a blockchain defines a specific distributed ledger protocol, full nodes, and clients function as described above. Inheriting the same interactive model, we now define the client security for blockchain protocols. To do so, we first define the notion of admissible blocks as a stepping stone.

▶ **Definition 8** (($u, k$)-Admissible Block at $r$). *Parameterized by $u \in \mathbb{N}$ and $k \in \mathbb{N}$, we call admissible block at $r$ any block $B$ observed at round $r$ fulfilling the following properties:*

- ***Safety:*** $\mathsf{B} \in \mathcal{C}^{\bigcup}_{r+u}[: -k]$
- ***Liveness:*** $\mathsf{B} \notin \mathcal{C}^{\bigcap}_{r}[: -k]$

In our definition of $(u, k)$-admissible blocks at $r$, the parameters $u$ and $k$ are free parameters. In our proofs, it turns out that this admissibility holds if $u$ is the "wait time" parameter of liveness, and $k$ is the "depth" parameter of safety/persistence of [18]. Thus, for readability we omit $(u, k)$ and mean admissibility in the round in which the client terminates, if not stated otherwise.

▶ **Definition 9** (Chain Client Security). *An interactive Prover-Verifier protocol $(P, V)$ for clients is* secure *if any block* $\mathsf{B}$ *output by the Verifier at the end of the protocol execution at round $r^*$ is* admissible *for some round $r \leq r^*$.*

In other words, the client accepts a block $\mathsf{B}$ at round $r^*$, if for some round $r \leq r^*$ the following holds: $\mathsf{B}$ is seen as stable by at least one honest party at round $r + u$ (safety), and $\mathsf{B}$ is not yet seen by all parties at round $r$ (liveness).

**State Commitments.** We consider PoW blockchains in which block headers include state commitments, denoted by $\langle st \rangle$. State commitments are a succinct representation of the state of the ledger, and they are assumed to be of constant size. In the account model of, e.g., Ethereum, an example of state commitment is the Merkle root of account balances; in the UTXO model of, e.g., Bitcoin, an example is the Merkle root of the Sparse Merkle Tree where the value of each leaf corresponds to a UTXO of the UTXO set [29, 32]. Equipped with this functionality, client protocols satisfying Definition 9 also satisfy Definition 5. We stress however that state commitments are necessary in Blink only for the extraction of the ledger's state but not for the secure proof creation.

## 5 Analysis

In this section, we present the main theorems and formal analysis of our paper. We start by giving a high-level overview on the proof strategy, followed by the formal proofs. Due to space constraints, some preliminary definitions and lemmas used in the proofs are deferred to Appendix A.

**Analysis Overview.** The main theorem we prove in this paper is as follows.

▶ **Theorem 10.** *Blink achieves ledger client state security (Definition 5).*

Towards proving Theorem 10, we start proving the admissibility of $\pi[0]$. We identify a special type of block, which we call *convergence event at a round* $r$ (Definition 49). A convergence event is an honestly produced block that has (by round $r$) no parallel block that is acceptable. We call a block an *acceptable block* (Definition 44) if it is valid and there is at least one honest party who might potentially switch to a chain including it. These convergence event blocks have some interesting properties. In particular, (i) all acceptable blocks at some round $r$ need to descend from all convergence events at round $r$ with smaller block height (Lemma 52); (ii) a block that is a convergence event $\tilde{\mathsf{B}}$ in a round in which there exists a valid block $\hat{\mathsf{B}}$ with a height of at least $k$ more than $\tilde{\mathsf{B}}$ (even if $\hat{\mathsf{B}}$ is only known to the adversary), $\tilde{\mathsf{B}}$ is destined to become stable for all honest parties (Lemma 53); (iii) the nearest ancestral convergence event to any block is always fewer than $k$ blocks away (Theorem 54). Note that these desirable properties hold regardless of our construction, and might be of independent interest.

Towards proving the safety of $\pi[0]$, we show that $\pi[k:]$ always extends a so-called anchor block $\tilde{\mathsf{B}}$, which is the nearest convergence event at the time that $\pi$ is found and sent to the client (Theorem 16). Since $\pi[k]$ is fewer than $k$ blocks away from its nearest ancestral convergence event (Theorem 54), we know that $\tilde{\mathsf{B}} \in \pi$. Also, $\tilde{\mathsf{B}}$ will become stable (Lemma 53), and thus $\pi[0]$ is safe. Intuitively, liveness holds since $\pi[k]$ is fresh as it contains the newly sampled $\eta$ and $\pi[0]$ is exactly $k$ blocks away and thus also new; we formally prove this in Theorem 21.

Towards chain client safety, we start arguing about the first proof $\pi$ of length $2k+1$ the client accepts at round $r^*+1$, with $\pi[k]$ containing $\eta$. As a first step, we say that $\pi$ must extend an *anchor block* $\tilde{\mathsf{B}}$ (Anchor, Theorem 16). In turn, $\tilde{\mathsf{B}}$ extends a block $\mathsf{B}'$ which is stable for all honest parties already at round $r_0$. Intuitively, this holds because when $\eta$ is broadcast, honest parties will only produce blocks extending $\tilde{\mathsf{B}}$. As a result of honest majority, a proof extending the anchor is found first.

Liveness and safety yield that $\pi[0]$ is an admissible block at round $r^*$ (Theorem 21) and we prove that the longest chain rule applied to the genesis block is consistent with the longest chain rule applied to $\pi[0]$ (New Genesis, Lemma 22). Finally, we show that, eventually, all honest parties will have an admissible block including $\mathsf{Tx}_\eta$ and that such a block is close to $\pi[0]$. It follows that running a (succinct) SPV algorithm on top of $\mathcal{G}' = \pi[0]$ will guarantee to Blink admissibility of a block $\mathsf{B}_\eta$ including $\mathsf{Tx}_\eta$, when $\mathsf{B}_\eta$ is buried $k$ blocks deep in the longest chain. We prove that $\mathsf{Tx}_\eta$ becomes stable for all honest parties (Lemma 26) after $u$ rounds and that its distance to $\mathcal{G}'$ is upper-bounded by $3k$ blocks (Lemma 27).

To conclude, we prove by reduction that any client construction that fulfilling the chain client security definition, having state commitments, also fulfills the ledger client state security definition (Corollary 24).

▶ **Theorem 11.** *Blink has optimal* communication cost*, i.e.,* $O(k)$*.*

The *communication cost* (Definition 6) measures the bits sent/received by $V$ during an execution $\mathcal{E}$ of a protocol $\Pi(\mathcal{P}, V)$. For each $P$, to which $V$ is connected, there is the following overhead. To identify the new genesis block, $V$ sends $\eta$ which has a size of $O(1)$ and receives (at most) one proof consisting of $2k + 1$ blocks for each $P \in \mathcal{P}$. This makes for a total size of $O(k)$. To predicate security of the block including $\eta$, the client sends the new genesis $\mathcal{G}'$ to all full nodes and it keeps receiving blocks descending from $\mathcal{G}'$, until the entropy block is $k$ deep in the longest chain - this will happen after, at most, $3k$ blocks from $\mathcal{G}'$. This makes for a total size of $O(k)$.

Note that light client constructions connect to a subset of all full nodes. Depending on how many nodes the light client connects to, the overhead increases. This is true for other light client constructions as well. Regardless, the *communication cost* of *Blink* is $O(1)$, i.e., constant in the chain length $\mathcal{C}$. From Theorems 10 and 11 it follows, that *Blink* is an Optimal Proof-of Proof-of-Work Protocol (OPoPoW, Definition 7).

## 5.1 Safety and Liveness of Blink

We model time to proceed in discrete rounds. Our network model stipulates that messages sent in a round $r$ reach the recipient in round $r + 1$. Like other nodes, the client can send and receive messages.

Consider a client booting up at round $r_0 - 1$ and broadcasting the entropy $\eta$. $\eta$ is received by the blockchain nodes at round $r_0$. We say the proof $\pi$ is generated at round $r^*$ and received by the client at round $r^* + 1$. Upon receiving the proof, the client sends $\pi[0]$ to full nodes and waits for $\mathsf{Tx}_\eta$ to become stable. Finally, the client terminates when $\mathsf{Tx}_\eta$ is stable in the chain of honest parties, i.e., at round $r^{**} \geq r^* + 3$.

Should the blockchain have fewer than $k$ blocks at round $r_0$, a proof with fewer than $k$ blocks before $\eta$ is valid if its first block is the genesis block. However, if the chain is shorter than $k$ blocks, the chain itself is already succinct and a light client is not needed.

Consider the blocktree of the execution at round $r_0$. We define $\mathsf{B}' \in \mathcal{C}^{\bigcap}_{r_0}$ as the block with the greatest height which is a convergence event at $r_0$.

▶ **Lemma 12.** $\mathsf{B}'$ *exists.*

**Proof.** The genesis block satisfies the definition of $\mathsf{B}'$. ◀

We denote the round in which $\mathsf{B}'$ was produced as $r'$, with $r' < r_0$. From Lemma 53, we know that all honest blocks produced after $r'$ extend $\mathsf{B}'$.

Now, consider the blocktree of the execution at round $r^*$. We define $\tilde{\mathsf{B}}$ as the block with the greatest height that descends from $\mathsf{B}'$, was mined before $r_0$, and it is a convergence event at $r^*$. Because this block is similar to the blocks named $\tilde{\mathsf{B}}$ in Theorems 54 and 55, we re-use the name $\tilde{\mathsf{B}}$. We say $\tilde{\mathsf{B}}$ is produced at round $\tilde{r}$, with $r' \leq \tilde{r} < r_0$. We define $\tilde{S} := \{\tilde{r}, \dots, r^*\}$.

▶ **Lemma 13.** $\tilde{\mathsf{B}}$ *exists.*

**Proof.** $\mathsf{B}'$ satisfies the definition of $\tilde{\mathsf{B}}$. ◀

▶ **Lemma 14.** *Acceptable blocks produced in* $\tilde{S}$ *descend from* $\tilde{\mathsf{B}}$.

**Proof.** This follows directly from Lemmas 51 and 52 (Appendix A.2).
◀

As a consequence of Lemma 14 and Observation 46, all honest blocks produced in $\tilde{S}$ descend from $\tilde{\mathsf{B}}$.

▶ **Lemma 15.** *All honest blocks produced in uniquely successful rounds within* $\{\tilde{r}+1,\ldots, r_0\}$ *have a parallel acceptable (by* $r^*$*) adversarial block.*

**Proof.** Because of the maximality (in terms of height) of $\tilde{\mathsf{B}}$, all blocks extending $\tilde{\mathsf{B}}$ and mined in uniquely successful rounds before $r_0$ have a parallel, acceptable adversarial block. ◀

For a set of consecutive rounds $S$, let $X(S)$ be honest queries, i.e., rounds in which at least one honest node found a block, $Y(S)$ be uniquely successful honest queries, i.e., rounds in which exactly one honest node found a block, and $Z(S)$ be adversarial queries, i.e., rounds in which the adversary found a block. We denote with $|X(S)|$, $|Y(S)|$, and $|Z(S)|$ the number of successful queries in $X(S)$, $Y(S)$, and $Z(S)$. These sets are defined in [18] or Appendix A.1.

▶ **Theorem 16** (Anchor). *In a typical execution, the block with* $\eta$ *and its* $k$ *subsequent blocks of the proof* $\pi$ *that the client accepts, i.e.,* $\pi[k:]$, *always extend* $\tilde{\mathsf{B}}$.

**Proof.** Let $Y(\tilde{S})$ be the set of honest uniquely successful queries within $\tilde{S}$, and $Z(\tilde{S})$ be the set of successful adversarial queries within $\tilde{S}$. Consider Figure 10 and let us define the following disjoint sets, $Y_1, Y_2$ and $Z_1, Z_2$, where $Y_1 \cup Y_2 = Y(\tilde{S})$ and $Z_1 \cup Z_2 = Z(\tilde{S})$.

1. The queries of $Z_1$ produce blocks that extend $\tilde{\mathsf{B}}$.
2. The queries of $Z_2$ produce blocks that do not extend $\tilde{\mathsf{B}}$.
3. The queries of $Y_1$ produce blocks parallel to (at least) one of the blocks in $Z_1$ acceptable at $r^*$.
4. The queries of $Y_2$ produce blocks not parallel to any of the blocks in $Z_1$ acceptable at $r^*$.

▷ Claim 17. If $|Y_2| = k+1$, at round $r^*+1$ the client has received a proof $\pi$ with the blocks $\pi[k:]$ extending $\tilde{\mathsf{B}}$.

This is true because in $Y_2$ there are no successful adversarial queries in $\tilde{S}$ producing blocks extending $\tilde{\mathsf{B}}$ and having parallel blocks. Furthermore, by definition of $\tilde{\mathsf{B}}$ and by causality, there cannot be successful adversarial queries outside of $\tilde{S}$ producing blocks extending $\tilde{\mathsf{B}}$. Yet, there can exist successful adversarial queries in $Z_2$ which produce blocks not extending $\tilde{\mathsf{B}}$.

▷ Claim 18. After $r_0$, honest parties do not extend blocks in $Z_2$.

Blocks in $Z_2$ do not extend $\tilde{\mathsf{B}}$, and thus are not acceptable by $r^*$. Therefore honest parties do not extend them within $\tilde{S}$.

After $r_0$, honest nodes will include $\eta$ in a block, if $\eta$ was not included before. It follows that the block in $Y_2$ with the smallest height descends from a block including $\eta$. The $k$ blocks produced by the remaining queries in $Y_2$ extend the block with $\eta$ by one block each, as they are uniquely successful and there are no parallel, adversarial acceptable by $r^*$ blocks.

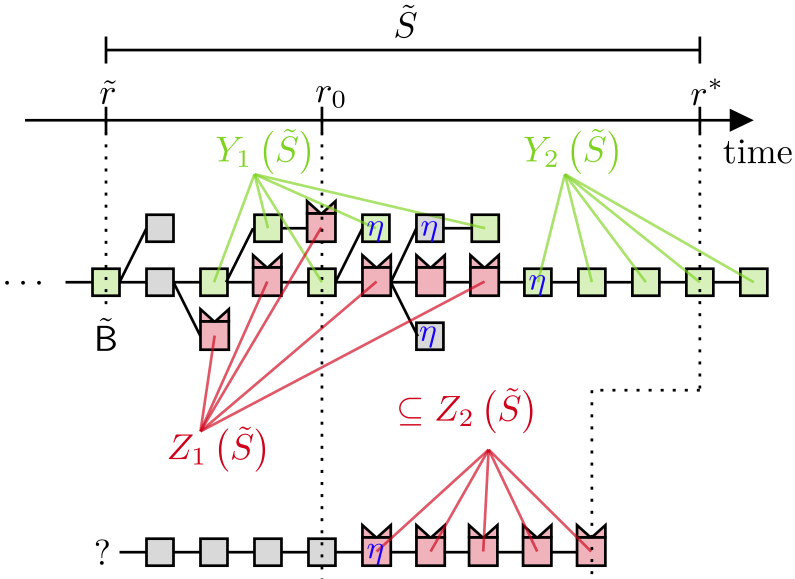


**Figure 10** This figure illustrates the proof of Theorem 16.

▷ Claim 19. Independently of $k$, the block in $Y_2$ with the greatest height has all other blocks of $Y_2$ as ancestors.

Towards a contradiction of Theorem 16, suppose that at round $r^* + 1$ the client accepts a proof $\pi$ which is generated at round $r^*$ and where $\pi[k :]$ does not extend $\tilde{\mathsf{B}}$. For the client to receive such a proof, the number of blocks produced between $r_0$ and $r^*$ not extending $\tilde{\mathsf{B}}$, thus in $Z_2$, has to be larger than or equal to $k+1$. Therefore, also $|Z_2| \geq |Y_2|$. $|Y_2|$ can grow at most of 1 per round: if $|Y_2|$ was of $k+1$ in a previous round $r_p < r^*$, the light client would have received the proof in $r_p + 1$, contradicting the minimality of $r^*$. Now we count these sets. We have that $|Z| = |Z_1| + |Z_2|$ and $|Y| = |Y_1| + |Y_2|$. By definition of $Y_1$, we know that $|Y_1| \leq |Z_1|$. It follows, that $|Z| = |Z_1| + |Z_2| \geq |Y_1| + |Y_2| = |Y|$. However, from Lemma 34 we know that $|\tilde{S}| \geq \lambda$ and thus, typicality bounds apply to this set of rounds. This means that $|Z| < |Y|$, which is a contradiction. This concludes the proof of Theorem 16. ◀

▶ **Lemma 20.** $\tilde{\mathsf{B}} \in \pi$.

**Proof.** Because the block containing $\eta$, $\pi[k]$ or $\dot{\mathsf{B}}$, which was produced in round $\dot{r}$, is acceptable and has a height larger than any block that was honestly produced before it, we know from Theorem 55 that the nearest convergence event at $\dot{r}$ has a height difference smaller than $k$ blocks. ◀

▶ **Theorem 21.** *In a typical execution, the first element $\pi[0]$ in the proof $\pi$ accepted by Blink client at round $r^*$ is an admissible block (cf. Definition 8).*

**Proof.** (Safety) From Lemma 20 we know that $\pi$ includes $\tilde{\mathsf{B}}$. From Theorem 54, we know that $\tilde{\mathsf{B}}$ is safe (i.e., $\tilde{\mathsf{B}} \in \mathcal{C}^{\bigcup}_{r_0+u}[: -k]$). Since $\pi[0]$ is either $\tilde{\mathsf{B}}$ or an ancestor of $\tilde{\mathsf{B}}$, $\pi[0]$ is safe as well, i.e., $\pi[0] \in \mathcal{C}^{\bigcup}_{r_0+u}[: -k]$ .

(Liveness) Let $l'$ be the height of $\mathsf{B}'$. Define $\mathsf{B}'' := \mathcal{C}^{\bigcap}_{r_0}[-k-1]$, and denote its height with height $l''$. Since $\mathsf{B}'$ is by definition either $\mathsf{B}''$ (if the latter is uniquely successful and has no adversarial blocks at the same height by round $r_0$) or else an earlier block, it follows that $l'' \geq l'$.

At round $r_0$, honest users each have a local chain with height of at least $l'' + k$, because $\mathsf{B}''$ is stable for all honest parties at round $r_0$. Since $\pi[k]$ includes $\eta$ it has to be mined after $r_0$, which is the round in which $\eta$ was released. This means, for the height $l_k$ of $\pi[k]$, it holds that $l_k > l'' + k$.

As $\pi[0]$, with height $l_0$, is $k$ blocks before $\pi[k]$, it holds that $l_0 = l_k - k$. Therefore $l_0 + k > l'' + k$, which means that $l_0 > l''$. However, since $\mathsf{B}''$ was the last block in the stable intersection at round $r_0$, this implies $\pi[0] \notin \mathcal{C}^{\bigcap}_{r_0}[: -k]$.

Therefore, at round $r^*$ when the client accepts the a proof $\pi$, $\pi[0]$ is an admissible block. ◀

We observe that, after $r_0$, every honest chain tip descends from $\pi[0]$. We refer to $\pi[0]$ as new genesis block $\mathcal{G}'$.

▶ **Lemma 22** (New Genesis). *The longest chain rule applied to the genesis block $\mathcal{G}$ is consistent with the longest chain rule applied to $\mathcal{G}'$, with $\mathcal{G}'$ being an admissible block.*

**Proof.** Suppose there exists a longest chain that contains $\mathcal{G}$ but does not contain $\mathcal{G}'$. From admissible safety, we know that $\mathcal{G}'$ is stable for at least one honest user $U$, i.e., $\mathcal{G}' \in \mathcal{C}^{\bigcap}_{r}[: -k]$. Since the longest chain does not contain $\mathcal{G}'$, honest users will adopt it in the next round, including the user $U$ who has reported $\mathcal{G}'$ as stable. This violates common prefix. ◀

▶ **Corollary 23** (Chain Client Security for Blink). *Blink is* chain client secure *according to Definition 9.*

Given a client protocol $\Pi$ which outputs a block $\mathsf{B}$, one can build another protocol $\Pi'$ that runs $\Pi$ and reports the state commitment in $\mathsf{B}$.[2]

▶ **Corollary 24.** *For any client protocol $\Pi$ that is chain client secure, the corresponding protocol $\Pi'$ constructed in the above manner is* ledger client state secure *(Definition 5).*

This follows from a simple reduction since $\Pi'$ merely reports the state commitment of $\mathsf{B}$. If the state commitment was such that $\Pi'$ is not ledger client state secure, the corresponding $\mathsf{B}$ cannot have been admissible. This concludes the proof of the main theorem Theorem 10, which is stated again here:

▶ **Theorem 25** (Ledger Client State Security for Blink). *Blink is* ledger client state secure *with the safety parameter $v$ (Definition 5) being the wait time parameter $u$ of liveness (Definition 8).*

## 5.2 Safety and Liveness of $\mathsf{B}_\eta := \pi[k]$

We now consider the case where Blink is used to verify a payment (or anything else that is in $\mathsf{B}_\eta$), and we show that the corresponding proof size remains constant. We recall that in this use-case, after adopting $\mathcal{G}'$ and sending it to the provers, the Blink client maintains the longest chain descending from $\mathcal{G}'$. We now show that the entropy block will be eventually stable for all honest parties at most $3k$ consecutive blocks away from $\mathcal{G}'$.

▶ **Lemma 26** (Stability of $\mathsf{Tx}_\eta$). *In a typical execution, a block $\mathsf{B}_\eta$ including $\mathsf{Tx}_\eta$ becomes stable for all honest parties at most at round $r_0 + u$, i.e., $\mathsf{B}_\eta \in \mathcal{C}^{\bigcap}_{r_0+u}[: -k]$.*

[2] For instance, this can easily be achieved for any blockchain protocol that has state commitments.

**Proof.** It follows from the ledger liveness in Definition 4. ◀

▶ **Lemma 27** (Vicinity of $\mathsf{Tx}_\eta$). *In a typical execution, a block* $\mathsf{B}_\eta$ *including* $\mathsf{Tx}_\eta$ *becomes stable for all honest parties at most* $3k$ *consecutive blocks away from* $\mathcal{G}'$.

**Proof.** Let $r_g$ be the round at which the new genesis block is produced. By construction, $k$ consecutive blocks are produced between $r_g$ and $r_0$. By Lemma 26, the entropy transaction $\mathsf{Tx}_\eta$ becomes stable for all honest parties at most at $r_s = r_0 + u$. By the liveness of the chain (chain quality and chain growth), at $r_s$, at most $2k - 1$ consecutive blocks are produced between $\mathcal{G}'$ and $\mathsf{B}_\eta$ (Corollary 38), and $\mathsf{Tx}_\eta$ is at least $k$ blocks deep in every honest party's chain. It follows that after at most $3k$ consecutive blocks are produced, $\mathsf{B}_\eta$ is stable for all honest parties. ◀

## 6 Practicality, Limitations, and Extensions of Blink

**State Commitments.** In Section 3, we presented an application of Blink to build a light client that can be convinced about the current state of a ledger with optimal communication cost. This way, we enable the confirmation of historical transactions in the ledger, tracing back to its genesis. However, this application operates on the premise that each block embeds a state commitment to the current ledger state. While several blockchains like ZCash, Nimiq, and Ethereum PoW uphold this premise, the most notable PoW blockchain, Bitcoin, does not incorporate state commitments in its block headers, even though there have been proposals [16]. NIPoPoWs, i.e., the polylogarithmic clients described in [25, 12], have the potential to be added retroactively via a velvet fork [27, 35]. The idea of introducing state commitments for Blink via velvet fork is appealing, however, its practical application is still undetermined.

**Multiple Clients.** Blink addresses the problem of one light client connecting to multiple full nodes and asking for the current state of the chain. In case we have multiple such requests, it is possible to compress the different entropy transactions using standard techniques. For example, multiple random strings can be ordered in a Merkle tree, and only the Merkle root is published on-chain within the entropy transaction. For this to be safe, each light client instance needs to have a Merkle proof of inclusion of its randomness in the tree.

**Entropy Transaction Fees.** Blink incurs on-chain fees which can be paid by light clients within entropy transactions. These fees can be paid in the form, for instance, of an anyone-can-spend output. The way the light client pays the on-chain cost for the entropy transaction can also be addressed in other ways on the application level: For instance, dedicated contracts or untrusted services can be designed such that clients' costs are mitigated.

**Interactivity.** Blink demands one round of interactivity between the client and the full nodes, unlike its predecessors that operate non-interactively [12, 25, 22]. This is the trade-off we incur for achieving a constant-sized proof instead of a polylogarithmic one as in [12, 25, 22]. We could remove the interactivity by introducing additional assumptions, for example: (i) a trusted committee service operates the client, similarly to the service provided by Chainlink for oracles, (ii) a random beacon acts as global entropy source and provides a service for Blink clients. However, both solutions come with drawbacks, i.e., centralization or a strong non-practical cryptographic primitive, respectively. It remains an open question whether designing a non-interactive light client with constant communication is possible without extra assumptions.

**Variable Difficulty.** Blink is analyzed in the static setting [18], i.e., the PoW difficulty remains the same throughout the protocol execution. In practice, Bitcoin uses a variable difficulty recalculation. Blink can still be used safely if we assume that parties agree on a difficulty beforehand, look it up on a trusted service (e.g., some blockchain explorer), or make some assumptions on the computational power of a potential adversary. Ideally, however, we can design a construction that is secure in the variable difficulty setting [17]. This challenge can be overcome by utilizing difficulty balloons to measure the current difficulty in a succinct fashion [37]. This approach, which is not unlike ours, utilizes entropy proofs to estimate (within some error) the current PoW difficulty of the network, by which point we can apply Blink as is. However, we anticipate that such an approach would only be secure under a weaker adversary that controls up to 1/3 of the computational power of the system. To provide an intuition behind this threshold, consider an adversary $t < 1/2$ that acts as follows: while measuring the difficulty, the adversary can abstain, thus creating a false sense of how many blocks she can produce in any given set of rounds. Thereby, she can take advantage of this false estimation to mine privately the required proof thereby violating the safety of Blink. We estimate that this adversarial advantage may be mitigated if honest nodes can produce double as many PoWs as the adversary.

Another approach would be to modify our light client construction by changing the selection rule for the proof: now the client would choose the proof with the most work after the intersection of all proofs within a given time window. We conjecture such an approach may alleviate the possible attack vectors of a minority adversary ($t < 1/2$), and we plan to explore it in future work.

## 7 Evaluation

We evaluate the feasibility of Blink by measuring its proof size and its waiting time for Bitcoin. A Proof-of-Concept implementation of Blink can be found at [8] and all entropy transactions broadcast during this evaluation can be inspected at this Bitcoin address [1].

Our client uses the python bitcoin-utils library [7] to create the entropy transactions, and the python request HTTP library [9] to communicate via RPC APIs [6].

**Experimental Setup.** We deployed two mainnet Bitcoin full nodes running Bitcoin Core 25.0 and acting as untrusted provers: one was operated in-house on our own hardware (Central Europe) and the other one on a Vultr virtual machine (UK). We use two different deployments to emulate more realistic network conditions. The nodes maintain the entire history of all transactions of the ledger and they allow us to broadcast transactions to the Bitcoin network as well as to retrieve blocks, transactions, and Merkle proofs.

We ran our custom implementation client on commodity hardware. The client begins by sampling uniformly at random a 160-bit string $\eta$ and creating the entropy transaction $\mathsf{Tx}_\eta$ by placing $\eta$ in an `OP_RETURN` output. The size of $\mathsf{Tx}_\eta$ is 222 bytes. Then, the client connects to the two Bitcoin full nodes, broadcasts $\mathsf{Tx}_\eta$, and waits for it to be $k$-confirmed (we set $k = 6$ according to Bitcoin folklore). When one of the two full nodes reports $\mathsf{Tx}_\eta$ $k$-deep, the client downloads and verifies the Blink proof $\pi$ of size $2k + 1$ block headers, i.e., it checks blocks' parent-child relation and the PoW inequality.

**Proof Size.** We measure all the data received by the client from the full node that first reports $\mathsf{Tx}_\eta$ with $k$ confirmations. This data amounts to 7728 bytes (7360 for $\pi_0$, and 368 for $\pi_1$, Algorithm 1).

The 7728 bytes of network data transmission required is due to the use of the inefficient JSON format and to the available standard RPC endpoints of the bitcoind full node. Using an optimized data transmission that avoids superfluous data, the total amount of data transmitted over the network can be brought down to 1646 bytes per prover connection (1040

| Full node | SPV[29] | KLS[22], NIPoPoW [25], Mining LogSpace[24] | FlyClient[12] | ZK ZeroSync Client [30] | **Blink** |
|---|---|---|---|---|---|
| 684GB | 67.3MB | 10KB | ~5KB | 197KB | 1.6KB |

**Table 2** Comparison of light client solutions for Bitcoin mainnet at height 841368, using the parameter $k = 6$

bytes for the 13 block headers of 80 bytes each, 384 bytes for the Merkle inclusion proof consisting of 12 sibling SHA256 hashes of 256 bits each, and 222 bytes for the transaction $\mathsf{Tx}_\eta$). In Table 2, for height 841368, we compare this to a full node that requires 684GB, an SPV client that requires 67.3MB, NIPoPoW and FlyClient clients that require 10.0KB and ~5KB, respectively, and to a PoW ZK-STARK-based client (ZeroSync[30]) that requires 197KB. We note that *the differences between these clients will be more pronounced as the blockchain grows.* We further note that proving Bitcoin's state with ZeroSync costs 4k USD (one-time cost), whereas Blink only incurs the cost of running a full node, e.g., $\sim 15$ USD a day.

**Waiting Time.** We measure the time it takes the client algorithm to run, averaging it over 10 runs. We broadcast the entropy transaction with a high-priority fee, which allows $\mathsf{Tx}_\eta$ to be included in the next 1 or 2 blocks. The average waiting time of the client to accept a proof is 59 minutes, with a standard deviation of 17 minutes. This is in accordance with the Bitcoin folklore belief of 6 blocks per hour. Any node that waits for 6 confirmations incurs the same waiting time, regardless of whether it is a full node or a light client. However, full nodes and SPV clients need to download a linear amount of data in the system's lifetime, while Blink requires only constant data in the chain's length to be downloaded.

## 8 Conclusion

This work presents Blink, the first Optimal Proof of Proof-of-Work client with constant communication complexity and without trusted setup. Blink allows to securely identify a state of the ledger which is safe and live by solely downloading a proof of $2k + 1$ consecutive blocks. We showcase how Blink can be leveraged in several different applications, ranging from verification of payments and state verification to bootstrapping and bridging. We prove Blink secure in the Bitcoin Backbone model against an adversary with minority computational power. Finally, we implemented Blink to verify its feasibility and we measured its proof size (experimental 7.7KB, 1.6 KB theoretical) and waiting time ($59 \pm 17$ minutes).

## Acknowledgments

The authors thank Joachim Neu and Kostis Karantias for the helpful discussions in the early phase of the work. The work was partially supported by CoBloX Labs, by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement 771527-BROWSEC), by the Austrian Science Fund (FWF) through the SFB SpyCode project F8510-N and F8512-N, and the project CoRaF (grant agreement ESP 68-N), by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association through the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT), and by the WWTF through the project 10.47379/ICT22045.

## References


1 Bitcoin 137WhkasQG5zE1zpHZZijkGkSiEW2mo4qy Address . https://blockstream.info/address/137WhkasQG5zE1zpHZZijkGkSiEW2mo4qy.

2 How to validate Bitcoin payments in Ethereum (for only 700k gas!), 2018. https://medium.com/summa-technology/cross-chain-auction-technical-f16710bfe69f.

3 Merkle Mountain Ranges, 2018. https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md.

4 Merkle Mountain Ranges (MMR), 2018. https://docs.grin.mw/wiki/chain-state/merkle-mountain-range/.

5 Mina docs, 2023. https://docs.minaprotocol.com/about-mina.

6 Bitcoin RPC APIs, Chain Query, 2024. https://chainquery.com/bitcoin-cli.

7 Bitcoin utils, 2024. https://pypi.org/project/bitcoin-utils/.

8 OPoPoW Blink Client Implementation, 2024. https://anonymous.4open.science/r/OPoPoW-Blink-Client/README.md.

9 Python Request Library, 2024. https://pypi.org/project/requests/.

10 Shresth Agrawal, Joachim Neu, Ertem Nusret Tas, and Dionysis Zindros. Proofs of proof-of-stake with sublinear complexity, 2023. arXiv:2209.08673.

11 Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale, 2020. https://eprint.iacr.org/2020/352.pdf.

12 Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 928–946, 2020. doi:10.1109/SP40000.2020.00049.

13 Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public-Key Cryptography – PKC 2013*. Springer Berlin Heidelberg, 2013.

14 Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. Sok: Blockchain light clients. In *IACR Cryptology ePrint Archive*, 2021. URL: https://api.semanticscholar.org/CorpusID:245908572.

15 Stelios Daveas, Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. A Gas-Efficient Superlight Bitcoin Client in Solidity. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 132–144, 2020.

16 Thaddeus Dryja. Utreexo: A dynamic hash-based accumulator optimized for the Bitcoin UTXO set. 2019. URL: https://eprint.iacr.org/2019/611.pdf.

17 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*. Springer International Publishing, 2017.

18 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Journal of the ACM (to appear)*, 2024.

19 Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. Cryptology ePrint Archive, Paper 2018/1239, 2018. https://eprint.iacr.org/2018/1239. URL: https://eprint.iacr.org/2018/1239.

20 Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 519–535. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

21 Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Compact storage of superblocks for nipopow applications. In Panos Pardalos, Ilias Kotsireas, Yike Guo, and William Knottenbelt, editors, *Mathematical Research for Blockchain Economy*. Springer International Publishing, 2020.

22 Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan,

Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2016.
23 Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. Mining in logarithmic space. In *CCS*, pages 3487–3501. ACM, 2021.
24 Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. Mining in Logarithmic Space. CCS '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3460120.3484784.
25 Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*. Springer International Publishing, 2020.
26 Aggelos Kiayias, Andrianna Polydouri, and Dionysis Zindros. The velvet path to superlight blockchain clients. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 205–218, 2021.
27 Aggelos Kiayias, Andrianna Polydouri, and Dionysis Zindros. The velvet path to superlight blockchain clients. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, AFT '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3479722.3480999.
28 Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *IACR Cryptology ePrint Archive*, 2019. URL: https://api.semanticscholar.org/CorpusID:53243925.
29 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. http://bitcoin.org/bitcoin.pdf.
30 Linus Robin, George Lukas, Milson Andrew, and Steffens Tino. Zerosync - stark proofs for bitcoin, 2024. https://github.com/ZeroSync/header_chain.
31 Giulia Scaffino, Lukas Aumayr, Zeta Avarikioti, and Matteo Maffei. Glimpse: On-demand PoW light client with constant-size storage for DeFi. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/scaffino.
32 Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light clients for lazy blockchains. Cryptology ePrint Archive, Paper 2022/384, 2022. URL: https://eprint.iacr.org/2022/384.
33 Psi Vesely, Kobi Gurkan, Michael Straka, Ariel Gabizon, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, and Eran Tromer. Plumo: An Ultralight Blockchain Client, 2023. https://celo.org/papers/plumo.
34 Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless cross-chain bridges made practical. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*, 2022. doi:10.1145/3548606.3560652.
35 A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottenbelt. A wild velvet fork appears! Inclusive blockchain protocol changes in practice. Berlin, Heidelberg, 2018. Springer-Verlag. doi:10.1007/978-3-662-58820-8_3.
36 Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. XCLAIM: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.
37 Dionysis Zindros. *Decentralized Blockchain Interoperability*. PhD thesis, University of Athens, Apr 2020.

# A Analysis

## A.1 Background from the Bitcoin Backbone [18]

We now introduce notation, definitions, theorems, and lemmas stated in [18] which will be necessary for our analysis.

The properties of blockchain protocols defined in the backbone model are presented below. Such properties are defined as predicates over the random variable $\text{view}^{t,n}_{\Pi,A,Z}$ by quantifying over all possible adversaries $A$ and environments $Z$ that are polynomially bounded. Note that blockchain protocols typically satisfy properties with a small probability of error in a security parameter $\kappa$ (or others). The probability space is determined by random queries to the random oracle functionality and by the private coins of all interactive Turing machine instances.

▶ **Definition 28** (Common Prefix Property [18]). *The common prefix property $Q_{cp}$ with parameter $k \in \mathbb{N}$ states that for any pair of honest players $P_1$, $P_2$ adopting the chains $C_1$, $C_2$ at rounds $r_1 \leq r_2$ in $view^{t,n}_{\Pi,A,Z}$ respectively, it holds that $C_1^{|k|} \preceq C_2$.*

▶ **Definition 29** (Chain Quality Property [18]). *The chain quality property $Q_{cq}$ with parameters $\mu \in \mathbb{R}$ and $\ell \in \mathbb{N}$ states that for any honest party $P$ with chain $C$ in $view^{t,n}_{\Pi,A,Z}$, it holds that for any $\ell$ consecutive blocks of $C$, the ratio of honest blocks is at least $\mu$.*

▶ **Definition 30** (Chain Growth Property [18]). *The chain growth property $Q_{cg}$ with parameters $\tau \in \mathbb{R}$ and $s \in \mathbb{N}$ states that for any honest party $P$ that has a chain $C$ in $view^{t,n}_{\Pi,A,Z}$, it holds that after any $s$ consecutive rounds, it adopts a chain that is at least $\tau \cdot s$ blocks longer than $C$.*

Closely following[18], we will call a query $q \in \mathbb{N}$ of a party successful if it returns a valid solution to the PoW. For each round $i$, $j \in [q]$, and $k \in [t]$, we define Boolean random variables $X_i$, $Y_i$, and $Z_{ijk}$ as follows. If at round $i$ an honest party obtains a PoW, then $X_i = 1$, otherwise $X_i = 0$. If at round $i$ exactly one honest party obtains a PoW, then $Y_i = 1$, otherwise $Y_i = 0$. Regarding the adversary, if at round $i$, the $j$-th query of the $k$-th corrupted party is successful, then $Z_{ijk} = 1$, otherwise $Z_{ijk} = 0$. Define also $Z_i = \sum_{k=1}^{t}\sum_{j=1}^{q} Z_{ijk}$. For a set of rounds $S$, let $X(S) = \sum_{r\in S} X_r$ and similarly define $Y(S)$ and $Z(S)$. Further, if $X_i = 1$, we call $i$ a successful round and if $Y_i = 1$, a uniquely successful round. We denote with $f$ the probability that at least one honest party succeeds in finding a PoW in a round.

▶ **Definition 31** (Typical Execution [18]). *An execution is $(\epsilon, \lambda)$-typical (or just typical), for $\epsilon \in (0, 1)$ and integer $\lambda \geq 2/f$, if, for any set $S$ of at least $\lambda$ consecutive rounds, the following hold.*

*(a)* $(1 - \epsilon)\mathbb{E}[X(S)] < X(S) < (1 + \epsilon)\mathbb{E}[X(S)]$ *and* $(1 - \epsilon)\mathbb{E}[Y(S)] < Y(S)$.

*(b)* $Z(S) < \mathbb{E}[Z(S)] + \epsilon\mathbb{E}[X(S)]$.

*(c) No insertions, no copies, and no predictions occurred.*

Let $n$ be the number of consensus nodes, out of which $t$ are controlled by the adversary. Let $Q$ be an upper bound on the number of computation or verification queries to the random oracle. Let $L$ be the total number of rounds in the execution, and $\lambda, \kappa$ security parameters. Finally, we denote with $\nu$ the min-entropy of the value that the miner attempts to insert in the chain.

▶ **Theorem 32** (Theorem 4.5 in [18]). *An execution is not typical with probability less than*

$$\epsilon_{\mathsf{typ}} = 4L^2 e^{-\Omega(\epsilon^2\lambda f)} + 3Q^2 2^{-\kappa} + [(n-t)L]^2 2^{-\nu}.$$

▶ **Lemma 33** (Lemma 4.6 in [18]). *The following hold for any set $S$ of at least $\lambda$ consecutive rounds in a typical execution. For $S = \{i : r < i < s\}$ and $S' = \{i : r \leq i \leq s\}$, $Z(S') < Y(S)$.*

▶ **Lemma 34** (Lemma 4.8 in [18], (aka Patience Lemma)). *In a typical execution, any $k \geq 2\lambda f$ consecutive blocks of a chain have been computed in more than $\frac{k}{2f}$ consecutive rounds.*

▶ **Lemma 35** (Lemma 4.1 in [18], (aka Pairing Lemma)). *Suppose the $k$-th block* B *of a chain $\mathcal{C}$ was computed by an honest party in a uniquely successful round. Then the $k$-th block a chain $\mathcal{C}'$ either is* B *or has been computed by the adversary.*

▶ **Lemma 36** (Lemma 4.2 in [18], (aka Chain Growth)). *Suppose that at round $r$ an honest party has a chain of length $l$. Then, by round $s \geq r$, every honest party has adopted a chain of length at least $l + \sum_{i=r}^{s-1} X_i$.*

▶ **Theorem 37** (Theorem 4.11 in [18], (aka Chain Quality)). *In a typical execution the chain quality property holds with parameters $\ell \geq 2\lambda f$ and*

$$\mu = 1 - \frac{1+f}{(1-f)(1-\epsilon)} \cdot \frac{t}{n-t} - \frac{(1+f)\epsilon}{1-\epsilon}$$

$$> 1 - \frac{1}{1-2\delta/3} \cdot \frac{t}{n-t} - \frac{\delta/3}{1-\delta/3} \xrightarrow{\delta \to 0} \frac{n-2t}{n-t}$$

▶ **Corollary 38** (Corollary 4.12 in [18]). *In a typical execution the following hold.*

- *Any $\lceil 2\lambda f \rceil$ consecutive blocks in the chain of an honest party contain at least one honest block.*
- *For any $\lambda$ consecutive rounds, the chain of an honest party contains an honest block computed in one of these rounds.*

In our analysis, we assume a typical execution in all proofs. We note that from Theorem 32 typical execution fails with negligible probability, resulting in our proofs holding with overwhelming probability.

## A.2 Preliminaries

In this section, we introduce some definitions, observations, and lemmas that will be used as building blocks in the formal analysis of Blink security (Section 5.1).

Let $H$ be a hash function modeled as a Random Oracle, and let $T$ be the target hash value used by parties for solving the PoW. Given a chain $\mathcal{C}$ and a block $b$ to be inserted in the chain, consider the hash $h = H(\mathcal{C}[-1], b)$ of these values, and let $ctr$ be a counter.

▶ **Definition 39** (PoW Inequality). *The PoW inequality holds if $H(ctr, h) < T$.*

If a $ctr$ fulfilling the PoW inequality is found, the chain $\mathcal{C}$ is extended by the block $b$ (which includes $ctr$). If no suitable $ctr$ is found, the chain remains unaltered.

▶ **Definition 40** (Valid Chain). *A chain $\mathcal{C}$ is (syntactically)* valid *if:*

- *$\mathcal{C} = \emptyset$, or*
- *$\mathcal{C}[:-1]$ is valid and the PoW inequality holds for $h = H(\mathcal{C}[-2], \mathcal{C}[-1])$.*

▶ **Definition 41** (Valid Block). *A block is* valid *if it belongs to a valid chain.*

▶ **Lemma 42.** *The following equality holds:*

$$\mathcal{C}_r^{\bigcup}[:-k] = \bigcup_{P \in \mathcal{H}} \mathcal{C}_r^P[:-k] \qquad (1)$$

**Proof.** We observe that $\mathcal{C}_r^{\bigcup}$ is a tree where each leaf $\mathcal{C}_r^P$ corresponds to the view of the chain of (at least) one honest party $P$ at some round $r$. $\mathcal{C}_r^{\bigcup}[:-k]$ is the result of taking $\mathcal{C}_r^{\bigcup}$ and removing the last $k$ blocks from each of the leaves of the tree. $\bigcup_{P\in\mathcal{H}}\mathcal{C}_r^P[:-k]$ is the result of taking all the chains of honest parties at round $r$, chopping off the last $k$ blocks and taking the union of these chains. By common prefix, honest parties' chains can only diverge by less than $k$ blocks; therefore, $\mathcal{C}_r^{\bigcup}[:-k]$ is a chain such that $\mathcal{C}_r^{\bigcup}[:-k] = \bigcup_{P\in\mathcal{H}}\mathcal{C}_r^P[:-k]$, with some honest parties being aware of all the blocks in it, and some others lagging behind. ◀

▶ **Definition 43** (Acceptable Chain at $r$). *A valid chain $\mathcal{C}$ is* acceptable at round $r$*, if*
- *$\mathcal{C} = \emptyset$, or*
- *$\mathcal{C}[:-1]$ is acceptable at $r$, and either $\mathcal{C} \preceq \mathcal{C}_r^{\bigcap}[:-k]$ or $\mathcal{C}_r^{\bigcap}[:-k] \preceq \mathcal{C}$ .*

An important notion we use is an *acceptable block*. Intuitively, an acceptable block is a block to which honest parties can switch to without violating common prefix. Honest nodes will never switch to chains containing non-acceptable blocks.

▶ **Definition 44** (Acceptable Block at $r$). *A block is* acceptable at $r$ *if it belongs to an acceptable chain at $r$.*

▶ **Observation 45.** *If a block is stable in an honest party's view, it is also acceptable.*

▶ **Observation 46.** *All honestly produced blocks are acceptable in the round in which they are produced.*

▶ **Observation 47.** *All honestly produced blocks only descend from blocks that are acceptable in the round in which the former are produced.*

▶ **Observation 48.** *Any block* $\mathsf{B}$ *produced in round $r_B$ and acceptable in round $r \geq r_B$, is also acceptable in all rounds in the set of consecutive rounds $\{r_B, \ldots, r\}$.*

▶ **Definition 49** (Convergence Event at $r$). *A block* $\mathsf{B}$ *is a* convergence event at round $r$ *if it is produced in a uniquely successful round $r_B$ and, by round $r \geq r_B$, it does not have a parallel acceptable block in any round in the set of consecutive rounds $\{r_B, \ldots, r\}$.*

▶ **Observation 50.** *A convergence event is always honestly produced.*

▶ **Lemma 51.** *If a block* $\mathsf{B}$ *produced in round $r_B$ is a convergence event at round $r$, it is a convergence event in all rounds in the set of consecutive rounds $\{r_B, \ldots, r\}$.*

**Proof.** By definition, there are no acceptable blocks at $\{r_B, \ldots, r\}$ parallel to $\mathsf{B}$. Therefore, $\mathsf{B}$ fulfills the definition of convergence event at all rounds $\{r_B, \ldots, r\}$. ◀

▶ **Lemma 52.** *An acceptable block* $\mathsf{B}$ *at $r$ must descend from all convergence events at $r$ with a height smaller or equal to* $\mathsf{B}$*'s height.*

**Proof.** Towards a contradiction, suppose there exists a convergence event $\hat{\mathsf{B}}$ at $r$, such that $\mathsf{B}$ does not descend from $\hat{\mathsf{B}}$. There must be a block $\mathsf{B}'$ parallel to $\hat{\mathsf{B}}$ from which $\mathsf{B}$ descends. Because $\mathsf{B}$ is acceptable at $r$, by definition, $\mathsf{B}'$ needs to be acceptable at $r$. However, both $\mathsf{B}'$ acceptable and $\hat{\mathsf{B}}$ being a convergence event, imply $\hat{\mathsf{B}} = \mathsf{B}'$. Thus, $\mathsf{B}'$ descends from $\hat{\mathsf{B}}$, reaching a contradiction. ◀

▶ **Lemma 53.** *Let $r$ be the round in which a block* $\mathsf{B}$ *was produced. For any block* $\mathsf{B}$ *and any round $r'$, for which* $\mathsf{B}$ *is a convergence event at $r'$ and* $\mathsf{B} \in \mathcal{C}^{\bigcap}_{r'}[: -k]$*, blocks acceptable at any round after $r$ always extend* $\mathsf{B}$.

**Proof.** From Observation 46 and Lemma 52, honest parties will extend $\mathsf{B}$ in the rounds between $r$ and $r'$ (included). $\mathsf{B}$ is stable for all honest parties at round $r'$. Therefore, after $r'$ all honest parties only extend $\mathsf{B}$, otherwise common prefix is violated. ◀

We denote with $|X(S)|$, $|Y(S)|$, and $|Z(S)|$ the number of successful queries $X$, $Y$, and $Z$ in a set of consecutive rounds $S$.

▶ **Theorem 54** (General Eventual Stability)**.** *Consider a convergence event* $\tilde{\mathsf{B}}$ *at $r^*$, which was produced in round $\tilde{r}$ and it has height $\tilde{l}$. If there exists a block with height $l \geq \tilde{l} + k$ in $\mathcal{C}^{\bigcup}_{r^*}$ then, in a typical execution,* $\tilde{\mathsf{B}}$ *becomes stable for at least one honest party at most at round $\tilde{r} + u$, i.e.,* $\tilde{\mathsf{B}} \in \mathcal{C}^{\bigcup}_{\tilde{r}+u}[: -k]$.

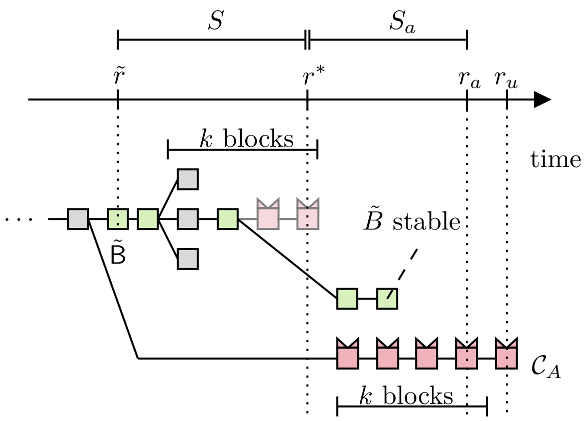


**Figure 11** This figure illustrates the proof of Theorem 54.

**Proof.** Consider Figure 11. Let $\tilde{l}$ be the height of $\tilde{\mathsf{B}}$ and $\tilde{r}$ the round at which $\tilde{\mathsf{B}}$ was produced. Since $\tilde{\mathsf{B}}$ is a convergence event, we know that it was honestly produced. Thus, at any round $r > \tilde{r}$, honest parties have adopted a chain of length at least $\tilde{l}$.

By round $r^*$, since $\tilde{\mathsf{B}}$ is a convergence event and due to causality, the acceptable blocks with height larger than $\tilde{l}$ have been mined at or after $\tilde{r}$ . Since the blocktree at round $r^*$ contains a block with a height $l \geq \tilde{l} + k$, at least $k$ consecutive blocks were mined in the set of consecutive rounds $S' := \{\tilde{r}, \dots, r^*\}$. Let $S := \{\tilde{r} - 1, \dots, r^* + 1\}$. We can thus apply the patience lemma (Lemma 34) to this set of rounds, which means that $|S| > \lambda$ and typicality bounds apply. In particular, $|X(S)| > |Z(S')|$, which implies $|X(S)| > \frac{k}{2}$ . From chain growth (Lemma 36), we know that in every round $r$ in which there is at least one honest block found, i.e. $X_r = 1$, honest parties increase the length of their chains by (at least) 1. It follows that in any round $r > r^*$, honest parties have adopted a chain longer than $\tilde{l} + \frac{k}{2}$.

Towards contradiction, suppose that $\tilde{\mathsf{B}} \notin \mathcal{C}^{\bigcup}_{\tilde{r}+u}[: -k]$. This means that there exists a round $r_u$ in which all honest parties have adopted a stable chain $\mathcal{C}_A$ of length $l_A \geq \tilde{l} + k$ which excludes $\tilde{\mathsf{B}}$. We note that *all* honest parties must have adopted $\mathcal{C}_A$, otherwise common prefix would be violated. It follows that: (i) $r_u < r + u$ because otherwise, by chain quality and chain growth, at round $r + u$, $\tilde{\mathsf{B}}$ would be stable; (ii) $r^* < r_u$ because, by definition of convergence event at $r^*$, $\tilde{\mathsf{B}}$ does not have any parallel acceptable adversarial block at round

$r^*$. The blocks $\mathcal{C}_A[-(k+1):]$ have a height of at least $\tilde{l}$ and are produced after $r^*$. We now proceed with a counting argument for the set of rounds $S_a := \{r^*, \ldots, r_a\}$, where $r_a \leq r_u$ is the first round in which $\mathcal{C}_A$ contains at least $k$ blocks with a height higher or equal to $\tilde{l}$. Again, since (at least) $k$ consecutive blocks were mined in $S_a$ and we can apply Lemma 34 to this set of rounds, which means that $|S_a| > \lambda$ and typicality bounds apply. In particular, $|X(S_a)| > |Z(S'_a)|$, which implies $|X(S_a)| > \frac{k}{2}$.

From Lemma 36, we know that there are at least $|X(S_a)|$ consecutive blocks extending $\tilde{\mathsf{B}}$. From Lemma 34, we know that $|S_a| \geq \lambda$, which means that typicality bounds apply, i.e., $|X(S_a)| > |Z(S_a)|$, hence $|X(S_a)| > \frac{k}{2}$ and $Z(S_a) < \frac{k}{2}$. The chain $\mathcal{C}_A$ which extends $\mathsf{B}'$ but not $\tilde{\mathsf{B}}$, has a length of at most $\tilde{l} - 1 + \frac{k}{2}$, as honest parties do not extend shorter chains. Therefore, at round $r_a$, all honest parties cannot have adopted $\mathcal{C}_A$, because they have a chain of length at least $\tilde{l} + k$, which includes $\tilde{\mathsf{B}}$. This concludes the contradiction. ◀

▶ **Theorem 55** (General Vicinity). *Consider any acceptable block $\dot{\mathsf{B}}$ at round $r$, produced in $\dot{r}$ and having a height larger than any honestly produced block in any round before $\dot{r}$. Let $\tilde{\mathsf{B}}$ be a convergence event at $\dot{r}$, such that $\tilde{\mathsf{B}}$ is the closest convergence event to $\dot{\mathsf{B}}$ in terms of height, and such that the height $\tilde{l}$ of $\tilde{\mathsf{B}}$ is smaller or equal to the height $\dot{l}$ of $\dot{\mathsf{B}}$, i.e., $\tilde{l} \leq \dot{l}$. In a typical execution, $\dot{l} - \tilde{l} < k$.*

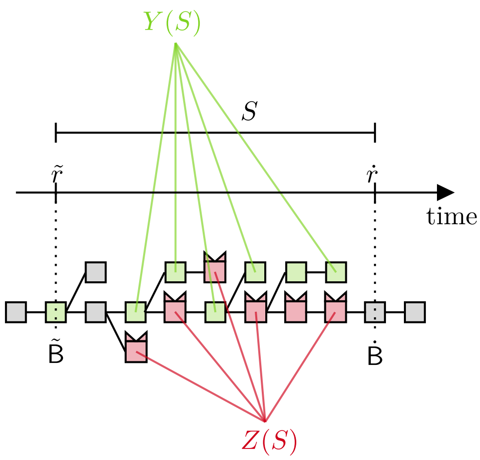


**Figure 12** This figure illustrates the proof of Theorem 55.

**Proof.** Consider Figure 12. Let $\tilde{r} \leq \dot{r}$ be the round in which $\tilde{\mathsf{B}}$ was produced. We now look at the blocks $\{\mathsf{B}\}_{Y(S)}$ that were honestly produced in the uniquely successful rounds in $S := \{\tilde{r}+1, \ldots, \dot{r}-1\}$, i.e., $Y(S)$. By definition, every block $\mathsf{B} \in \{\mathsf{B}\}_{Y(S)}$ has a height smaller than $\dot{\mathsf{B}}$. However, due to Lemma 52, every block $\mathsf{B} \in \{\mathsf{B}\}_{Y(S)}$ also extends $\tilde{\mathsf{B}}$ and thus has a height larger than $\tilde{\mathsf{B}}$.

Because $\tilde{\mathsf{B}}$ is the nearest convergence event at $\dot{r}$, any block $\mathsf{B} \in \{\mathsf{B}\}_{Y(S)}$ needs to have a parallel, acceptable at $\dot{r}$ (and thus mined at or before $\dot{r}$) block. Otherwise, $\mathsf{B}$ would be the nearest convergence event to $\dot{\mathsf{B}}$. Because these parallel blocks are acceptable, they need to extend $\tilde{\mathsf{B}}$ (Lemma 52) and thus by causality, need to have been produced at or after $\tilde{r}$ and at or before $\dot{r}$, which means they are produced in $S' := \{\tilde{r}, \ldots, \dot{r}\}$. Additionally, from Lemma 35, we know that these parallel blocks need to be adversarially produced. Thus, there needs to be at least one successful adversarial query within $S'$ for each uniquely successful round in $S$, i.e., $|Z(S')| \geq |Y(S)|$.

Due to causality, the blocks between $\tilde{\mathsf{B}}$ and $\dot{\mathsf{B}}$ need to have been produced in $S'' := \{\tilde{r}, \ldots, \dot{r} - 1\}$. Suppose towards a contradiction, the difference in height between $\dot{\mathsf{B}}$ and $\tilde{\mathsf{B}}$ is $k$ or more. From Lemma 34 we know that $|S''| > \lambda$, thus $|S| \geq \lambda$, and thus, typicality bounds apply to this set of rounds. Thus, by Lemma 33 it holds that $|Z(S')| < |Y(S)|$, which contradicts the above. ◀