Blink: An Optimal Proof of Proof-of-Work

Lukas Aumayr[1,4], Zeta Avarikioti[2,4], Matteo Maffei[2,5], Giulia Scaffino[2,4,5], and Dionysis Zindros[3,4]

[1] University of Edinburgh
[2] TU Wien
[3] Stanford University
[4] Common Prefix
[5] CDL-BOT

Abstract. Designing light clients to securely and efficiently read Proof-of-Work blockchains has been a foundational problem since the inception of blockchains. Nakamoto themselves, in the original Bitcoin paper, presented the first client protocol, i.e., the Simplified Payment Verification, which consumes an amount of bandwidth, computational, and storage resources that grows linearly in the system's lifetime $\mathcal{C}$.
Today, the blockchain ecosystem is more mature and presents a variety of applications and protocols deployed on-chain and, often, cross-chain. In this landscape, light clients have become the cornerstone of decentralized bridges, playing a pivotal role in the security and efficiency of cross-chain operations. These new use cases, combined with the growth of blockchains over time, raise the need for more minimalist clients, which further reduce the resource requirements and, when applicable, on-chain costs. Over the years, the light client resource consumption has been reduced from $\mathcal{O}(\mathcal{C})$ to $\mathcal{O}(\text{polylog}(\mathcal{C}))$, and then down to $\mathcal{O}(1)$ with zero-knowledge techniques at the cost of often assuming a trusted setup.
In this paper, we present Blink, the first *interactive provably secure* $\mathcal{O}(1)$ *PoW light client without trusted setup*. Blink can be used for a variety of applications ranging from payment verification and bootstrapping, to bridges. We prove Blink secure in the Bitcoin Backbone model, and we evaluate its proof size demonstrating that, at the moment of writing, Blink obtains a commitment to the current state of Bitcoin by downloading only 1.6KB, instead of 67.3MB and 197KB for SPV and zk-based clients, respectively.

1 Introduction

Blockchain systems run consensus protocols among a large and varying set of mutually distrusting participants, favoring decentralized trust over efficiency. To engage with blockchains, users need to read the latest state of the chain to find out, e.g., how many coins they own, if they got paid by another user, or, in more modern chains, which is the current state of a contract. To do so, they can use different techniques with different trade-offs. One approach is to run a full node, which downloads, re-executes, and stores the entire history of the ledger. This

is the most secure way but, obviously, also the most inefficient. It is like reading a whole long book from the beginning just to know if the protagonist defeats the villain at the end. An alternative way for a user to read the blockchain is to run a light node, which only downloads and validates block headers, trading off some security to gain efficiency. In our analogy, it is like skimming through the book, finding the page of interest, and only reading the sentence "Voldemort was dead, killed by his own rebounding curse" [1]. In the real world, users often have few resources and cannot read an entire (however good) book when checking a payment or trading assets. Users cannot be asked to run full nodes on their resource-constrained phones to use their wallet.

In the seminal Bitcoin white paper [2], Satoshi Nakamoto already predicted the 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, consuming an amount of resources that grows linearly with the system's lifetime. Several subsequent works optimized this concept, introducing *superlight clients* whose resource requirements are only polylogarithmic in the lifetime of the system [3,4,5,6]. Nevertheless, these protocols are not out-of-the-box compatible with Bitcoin and require a consensus fork. The increased performances of zero-knowledge techniques have also lead to *ultralight clients* [7,8,9] which, however, often rely on a trusted setup, trading off trust to gain efficiency.

As blockchains grow, and with them also the number of on-chain and cross-chain applications, the need for more efficient clients has become more pressing. Indeed, in today's more mature ecosystem, light clients are not only used by wallets, but they have become a pivotal component of many bridge protocols, whose operating costs are often dominated by the (inefficient) reads and verifications of blockchain data. Designing a client that only requires constant communication, computation, and storage resources has unfortunately remained an elusive goal over the past dozen years. This paper fills this gap, enabling critical resource optimizations for blockchain clients as well as reduced on-chain costs for cross-chain applications.

Contributions. In this work, we present Blink, a novel *interactive PoW light client with constant communication, computational, and storage complexity.* In a nutshell, the Blink client connects to multiple full nodes, so that at least one of them can be assumed honest. The client locally samples a random value η, includes it in a transaction Tx_η, and sends it to the full nodes. For instance, η could be a new, fresh address sampled with high entropy and Tx_η can be a payment to the vendor that owns this fresh address. Then, Blink waits for Tx_η to be included on-chain in a block and confirmed. The full nodes respond to the client with a proof π consisting of only $2k+1$ consecutive block headers, with the header of the block including Tx_η sitting in the middle (see Figure 1); k is the *common prefix* security parameter [10], e.g., the conventional 6 confirmation blocks in Bitcoin. The constant-sized proof π ensures that *the first block in the proof is stable* and,

therefore, it can be considered as a checkpoint or as a new genesis. Contrary to the $\mathcal{O}(1)$ zero-knowledge based clients, *Blink does not require any trusted setup.*

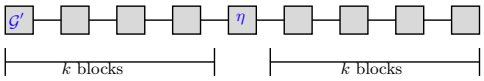

Fig. 1. Structure of the Blink's proof π. The proof π consists of $2k + 1$ consecutive block headers, with the one of the block including the entropy η in the middle, and k block headers before and after it. The block $\mathcal{G}'$ identified by the first header of the proof is stable in the chain and can act as a new genesis block.

Blink is proven secure in the *static population*, i.e., static difficulty, Bitcoin Backbone model [10] against an *adaptive minority adversary*. In this paper, we refine the problem of Proofs of Proofs-of-Work [3], i.e., techniques often used by light clients to prove on-chain inclusion by succinctly verifying the amount of Proof-of-Work done. We prove that Blink has a constant-sized proof verifiable in constant time, and constitutes therefore the first provably secure *Optimal Proof of Proof-of-Work (OPoPoW) without trusted setup.*

Furthermore, we showcase how Blink can be leveraged to develop a plethora of applications with enhanced efficiency compared to state-of-the-art protocols. For instance, it allows, for the first time, to securely and efficiently bootstrap light miners and full nodes, by providing a commitment to the state of the ledger with a short, constant-sized proof, bringing down the synchronization time from several days to a few hours. Furthermore, with Blink users can trustlessly verify their payments in a resource-constrained environment such as their phone with only a $\mathcal{O}(1)$ overhead, instead of $\mathcal{O}(\mathcal{C})$. Finally, Blink can replace SPV clients as a key component for bridges, further reducing the verification and storage costs incurred by contracts, while retaining the same security.

We provide a Proof-of-Concept *implementation* of Blink for Bitcoin, and evaluate its communication cost for the conventional $k = 6$ blocks. We underscore that Blink improves on all previous light client solutions in terms of proof size (and computational resources to verify it); at the time of writing, an SPV client has a proof size of 67MB, superlight clients [3,6] of 5-10KB, zk clients [9] of 197KB, and Blink of only 1.6KB.

In Section 7, we discuss practicality, limitations, and extensions of Blink to the variable difficulty setting.

Related Work. The description of Nakamoto's SPV client appears already in the Bitcoin whitepaper [2]. A series of optimizations followed. The first succinct construction was the interactive *Proofs of Proof-of-Work protocol* [5] with polylogarithmic costs. Later work removed this interactivity and achieved security against 1/2 adversaries but succinctness only in the optimistic setting (against no adversaries) [3]. This construction was subsequently optimized [4], made practical [11], and redesigned with backwards compatibility in mind [12]. The optimistic setting limitation was alleviated in a follow-up work, achieving succinctness against all adversaries up to a 1/3 threshold [13]. An alternative

construction was also proposed, enabling security and succinctness against a 1/2 adversary, and adding support for variable difficulty [6]. All these solutions require polylogarithmic costs, whereas Blink requires only constant.

Recently, generic (recursive) zero-knowledge (ZK) techniques were utilized to build $\mathcal{O}(1)$ light clients [14,8,7]. However, these approaches incur prohibitively high computational costs (or necessitate specialized blockchain deployments [8,7] utilizing ZK-friendly cryptographic primitives [15]) and additionally require a trusted setup to generate and prove verification keys (which can only be removed if polylogarithmic communication is acceptable). Blink removes the trusted setup assumption.

Towards a $\mathcal{O}(1)$ light client without a trusted setup, the idea of using only a small segment of the chain near the tip was proposed [16]. However, the proposed construction was shown to be susceptible to pre-mining attacks and thus insecure [17]. Recently, Glimpse [17] has combined the idea of [16] with the injection of a *high-entropy* transaction Tx_η to prove the provided segment of the chain is "fresh" and not pre-mined. Nevertheless, we show that Glimpse suffers from safety and liveness attacks. In Blink we leverage and extend these ideas, proposing the first provably secure light client that consumes only a constant amount of resources and 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 [18]. For a review of the long-standing light client problem, see [19]. Light clients are also a cornerstone for building trustless bridges between chains, a question that has been explored in a multitude of works [20,21,22,23]. In this work, we demonstrate how Blink can serve as a building block for efficient optimistic bridges.

Comparison. In Table 1, we compare Blink with existing light client protocols. We denote by $\mathcal{C}$ 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 *constant* in the lifetime of the system, albeit with the trade-off of logarithmically increasing the probability of failure.

We first observe that Glimpse [17] consumes $\mathcal{O}(k)$ resources, but it is not secure (see Section 3); its exact resilience, if any, remains unknown. ZK clients, unlike Blink, rely on the trusted setup assumption. Blink consumes $\mathcal{O}(k)$ resources, it is finally provable secure, but requires one round of interaction.

Finally, contrary to other clients, Blink requires to publish on-chain the transaction Tx_η and wait for it to be k confirmed; despite this, in most real-world applications, all clients have the same latency as any payment or cross-chain request is considered final only after having k confirmation blocks. In Section 7, we propose practical solutions to reduce the costs of publishing Tx_η.

2 Preliminaries and Model

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

	SPV[2]	KLS[5], NIPoPoW [3,13] FlyClient[6]	Plumo[7], Mina[8], Coda[14]	ZeroSync[9]	Glimpse [17]	**Blink**
Resources	$\mathcal{O}(\mathcal{C})$	$O(k \text{ polylog}(\mathcal{C}))$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$
No Trusted Setup	✓	✓	✗	✓	✓	✓
Adv. Resilience	1/2	1/2	1/2	1/2	✗(?)	1/2
Rounds of Interactivity	0	0	0	0	1	1
On-Chain Transactions	None	None	None	None	1	1
Proof size	67 MB	5-10 MB	22 KB	197 KB	1.2 KB	1.6KB

Table 1. Comparison of light client solutions.

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 $\mathcal{C}_r^{\bigcap}[:-k]$ 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 r, pruned of the last k blocks.

Ledger Model. We assume a synchronous network, i.e., all honest parties are guaranteed to receive messages sent by honest parties 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 exposing to all parties the following methods:*

- execute*: Executes 1 round of the protocol, during which the machine can communicate with other parties.*
- write(Tx)*: Takes transaction* Tx *as input.*
- read()*: Outputs a ledger.*

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

Definition 3 (Safety). *A distributed ledger protocol is* safe *if it fulfills the following properties:*

Self-consistency *For any correct 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 correct 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* live *with liveness parameter u if all transactions written by any correct party at round r, appear in the ledgers of all correct parties by round $r + u$.*

The ledger uniquely defines the current state of the system. Consider an empty ledger $\mathcal{L}_0$ with genesis state S_0. To ascertain the i-th state S_i of a ledger $\mathcal{L}_i$, with $i > 0$, transactions $[\mathsf{Tx}_1, \ldots, \mathsf{Tx}_i]$ are applied as follows:

$$S_i := \delta(\ldots\delta(\delta(S_0, \mathsf{Tx}_1), \mathsf{Tx}_2)\ldots, \mathsf{Tx}_i).$$

As shorthand notation, we use $S_i := \delta^*(S_0, \mathcal{L}_i)$ to denote successive application of all transactions $\mathsf{Tx} \in \mathcal{L}_i$ to S_0.

We consider PoW ledgers whose block headers include state commitment, i.e., a succinct, constant size representation of the state of the ledger. We stress that state commitments are necessary for Blink only to extract the ledger's state and not to create a secure proof.

Prover-Verifier Model. A client protocol is an interactive protocol between the client, acting as verifier V, and a set $\mathcal{P}$ of full nodes, acting as provers. We focus on a client V that, when it bootstraps on the network for the first time, it is only aware of the genesis state. We assume that the client is honest, and in the set $\mathcal{P}$ there is at least one honest prover (existential honesty assumption). The client does not know which prover is honest. While honest parties adhere to the correct protocol execution, the adversary can execute any probabilistic polynomial-time algorithm.

Ledger Client Security. We now define what it means for a client of a ledger $\mathcal{L}$ to be *secure*. Among all the honest parties' ledgers, let $\mathcal{L}_r^{\bigcup}$ be the longest and $\mathcal{L}_r^{\bigcap}$ be the shortest at the end of round r.

Definition 5 (Ledger Client Security [24]). *An interactive Prover-Verifier protocol $\Pi(\mathcal{P}, V)$ is* secure *with safety parameter v if, when the protocol terminates ar r, V outputs a commitment to a state of the ledger $\mathcal{L}$ that, $\forall r' \geq r + v$, satisfies the following properties:*

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

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

When a protocol $\Pi(\mathcal{P}, V)$ correctly executes by downloading and verifying asymptotically less data in $\mathcal{L}$, it is a *light client* protocol. We measure the performance of a client protocol by defining the *communication cost* for the verifier; the computational and storage costs of a client are linear in the amount of data downloaded during the protocol.[6]

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 V.*

A client protocol has *optimal communication cost* if $\mathsf{cost}(\mathcal{E}, V) = \mathcal{O}(1)$, i.e., if V only receives a constant amount of data per protocol execution.

[6] For full nodes, assume transaction execution has a constant upper bound on the computation.

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

Chain Client Security. To model the Proof-of-Work setting we closely follow [10]. Importantly, we operate in the static model, where the number of consensus nodes remains fixed throughout the protocol execution. Furthermore, each of them is assumed to have an equal computational power (flat model). The static model implies *static difficulty*. For a more complete description of the PoW blockchain model, see Appendix B.1. Throughout this work, we will use the term block to mean a block header. Towards defining the security of a client for blockchain protocols, we first define the notion of *admissible block*.

Definition 8 ((u,k)-Admissible Block at r). *Consider $u, k \in \mathbb{N}$. Any block B that, at round r, fulfils the following properties is an* admissible block at r*:*
Safety *:* $\mathsf{B} \in \mathcal{C}^{\bigcup}_{r+u}[:-k]$
Liveness *:* $\mathsf{B} \notin \mathcal{C}^{\bigcap}_{r}[:-k]$

In other words, for $r \leq r^*$, a block B is admissible at round r^* if B is seen as stable by at least one honest party at round $r+u$ (safety), and B is not yet seen by all parties at round r (liveness). In the above definition, u and k are free parameters. From the proofs in Appendix D, it turns out that admissibility holds with u being the "wait time" parameter of liveness, and k the "depth" parameter of safety [10].

Definition 9 (Chain Client Security). *An interactive Prover-Verifier protocol $\Pi(\mathcal{P}, V)$ is* secure *if, when the protocol terminates at r^*, V outputs a block B that is* admissible *at $r \leq r^*$.*

We consider PoW blockchains whose blocks include state commitments. 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 UTXO Tree [2,24]. Equipped with state commitments, client protocols satisfying Definition 9 also satisfy Definition 5. We stress, however, that state commitments are necessary in Blink only for efficiently reading elements of the ledger's state, and not for creating a secure proof.

3 Blink

The ultimate goal of an OPoPoW client is to identify a recent, correct block of the ledger, by only receiving a constant-sized proof. We recall that in PoW blockchains, blocks are considered final if they have at least k confirmations, where k is the security parameter - in Bitcoin folklore $k = 6$.

A Naive Construction. We start considering a simple, naive construction. The provers give to the client the last $k+1$ consecutive blocks in their longest

chain. The client verifies the validity of these sequences of blocks and, among the valid ones, takes the sequence whose blocks have the greatest height, and considers safe and live the first block in the sequence. This construction trivially breaks safety: an adversarial prover may have pre-mined $k+1$ fake blocks and can therefore trick the client into accepting a block that is not part of the chain.

Preventing Safety Attacks. To prevent this pre-mining attack, the client V needs to randomize the tip of the chain and to only accept blocks in the randomized suffix. In this way, the adversary is not able to predict the random value and therefore it is required to produce fresh blocks. Therefore, V locally samples a random string $\eta \xleftarrow{\$} \{0,1\}^\lambda$, defines a timeout T after which it stops accepting incoming proofs, and sends (η, T) to the provers [17]. The provers embed η into an *entropy transaction* Tx_η, broadcast Tx_η to the blockchain network, and wait for Tx_η to be included in a block B_η and confirmed. As soon as a prover P sees B_η with k confirmation blocks, if T has not expired, it sends to V a proof π consisting of B_η with its k confirmation blocks. Finally, V considers B_η safe and live, and it extracts from it the commitment to the state of the ledger.

Randomizing the proof rules out pre-mining attacks but, unfortunately, it does not result in a secure client protocol. Consider an adversary that controls $t < n/2$ of the n total participants in the PoW game. The adversary has a probability $t/n < 1/2$ of being elected as block proposer, which results in a non-negligible probability of censoring Tx_η in the next $k-1$ blocks. If by T the chain is extended by fewer than $2k$ consecutive blocks, the adversary can violate the liveness of the client protocol with probability t/n, as honest parties do not have time to produce $k+1$ blocks by T. Figure 2 shows this attack.

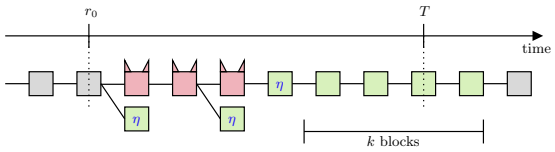

Fig. 2. Consider $k = 4$. The light client broadcasts η at round r_0. With non-negligible probability, a minority adversary can censor Tx_η in the next $k-1$ blocks. Honest parties do not have time to generate a proof of $k+1$ blocks before T expires.

Preventing Liveness Attacks. To protect the client from liveness attacks, one could remove the timeout T and ask V to accept the first incoming proof that consists of B_η and at least k confirmation blocks. Alternatively, one could ask V to accept the proof that, by T, has the most confirmation blocks on top of B_η. While both these attempts safeguard the liveness of the client, the client's safety is now broken again, as V might accept a block that is not part of any honest party's chain. We describe the safety attack for the case where T is removed, but a similar logic applies to the other case as well.

After Tx_η is broadcast, honest parties include it in the next block they create. The adversary, instead, extends the chain with blocks that censors Tx_η and it keeps these blocks private. Being k the security parameter, with non-negligible

probability the adversary can privately mine $k - l$ blocks, with $0 < l < k - 1$. Meanwhile, honest parties can mine B_η with at most $k - l - 2$ confirmations. Then, the adversary broadcasts the private chain, causing all honest parties to switch to the adversarial chain, as per the longest chain rule. Honest parties include Tx_η in the new longest chain and keep mining on top of it. At the same time, the adversary starts privately mining on top of the chain abandoned by the honest parties, i.e., the one that included Tx_η early on. Now, to create a valid proof, the adversary only needs to mine $l + 2 < k + 1$ blocks, whereas honest parties need to mine $k+1$ blocks. As a result, with non-negligible probability the adversary can find the first k confirmation blocks on top of (an abandoned) B_η, and trick the client into considering safe and live a block that will never be part of the longest chain, thereby breaking security. Figure 3 illustrates this attack. This attack breaks the security of Glimpse [17].

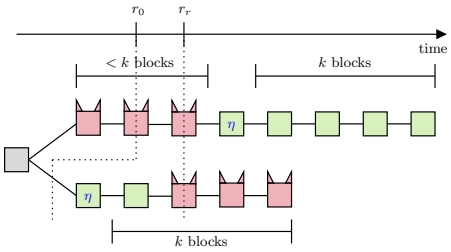

Fig. 3. Consider $k = 4$ and $l = 1$. The client broadcasts η at r_0. The adversary privately mines 3 blocks censoring Tx_η, while honest parties mine 2 blocks, the first of which includes Tx_η. The adversary releases the private chain at r_r and honest parties switch to it. Honest parties need to mine 5 blocks to find a valid proof, while the adversary only needs to mine 3 blocks. The adversary finds the proof first.

Blink. We recall that our goal is to let the client securely identify a block that is *safe*, i.e., already k deep in at least one honest party's chain [10], and *live*, i.e., sufficiently close to the tip of the chain. In the safety attack we just described, the adversary delays the inclusion of Tx_η in the main chain, but this censorship only succeeds for a limited time, specifically for less than k consecutive blocks: a private chain longer than k and longer than the honest parties' chain would break safety of the ledger [10]. This means that any honest majority will create k blocks faster than any minority adversary.

Knowing that Tx_η can only be censored for $k - 1$ blocks and that it takes k additional blocks for it to become safe (Figure 3), in our final construction we modify the proof such that it consists of $2k + 1$ blocks, with B_η in the middle, as depicted in Figure 1. To avoid the safety attack, the client now considers safe and live the first block of the first valid proof it receives, and not B_η as before; this is because a proof of $2k+1$ blocks must necessarily contain a safe block, i.e., a block that is at least k deep in the chain of all honest parties. To be precise, the first proof that the client gets contains at least one block that was safe even before Tx_η was broadcast: the honest abandoned subchain has length at most $k - 2$ and at least 1, therefore the first block in the proof was already part of

the honest parties' stable chain (Figure 4). Importantly, this is true even if the proof comes from the adversary. For any proof coming from an honest party, any block before B_η is already safe. We also note that the first block in the proof is live, as the block with η comes shortly after, and attached to genesis, otherwise honest parties would not have extended it.

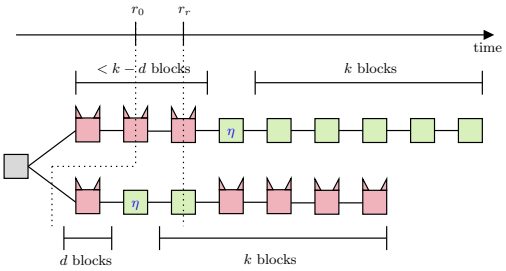

Fig. 4. Consider $k = 5$. The adversary censors Tx_η by $k - d$ blocks on the top branch and by d blocks on the lower branch, with the overall number of adversarial blocks before Tx_η being smaller than k ($d \leq k - 1$). This shows why taking fewer than k blocks before Tx_η is not sufficient.

To conclude, the first block of the first valid proof seen by the client is always safe, i.e., it has at least k confirmations in the view of an honest party, and live, i.e., it is at most k blocks far back from B_η.

We observe that the client receives a proof that consists of $2k+1$ blocks, with the number of blocks remaining constant in the system's lifetime (*optimal proof size*). After broadcasting η, the clients needs to wait for k blocks to be mined, but this waiting time is standard to all clients that want to have finality guarantees for a transaction. Finally, Blink can extract the state commitment from the first block of the proof. In Figure 5 we present the pseudocode of the Blink protocol. In Appendix A we show the algorithms run by the client and provers of Blink.

Proof Construction

1. V samples $\eta \xleftarrow{\$} \{0,1\}^\lambda$.
2. V broadcasts η to $\mathcal{P}$.
3. $\mathcal{P}$ creates a transaction Tx_η that includes η and broadcast it to the network.
4. When a party $P \in \mathcal{P}$ sees a block B_η including Tx_η and having k confirmations, P sends to V a proof consisting of B_η and the k blocks before and after it.

State Commitment Extraction

5. V accepts the first incoming proof π that consists of $2k + 1$ consecutive well-formed blocks, with the block in the middle containing η, i.e., $\pi[k] = \mathsf{B}_\eta$.
6. V extracts the state commitment from $\mathsf{B} := \pi[0]$ and terminates.

Fig. 5. The Blink protocol.

4 Applications

In this section, we describe how to use Blink for different applications.

Payment Verification. Consider a vendor that wants to check if its customer (the buyer) has paid for the purchase of a good, and that will ship the good only after the client's payment has been verified and finalized. The vendor gives the buyer a new, fresh address η sampled with high entropy, and uses Blink to efficiently read the chain and verify the payment. Blink only guarantees safety for the first block of the first proof it receives, and not for the block including Tx_η. Fortunately, the safety guarantees of Blink are strong and they help our vendor nonetheless: the first block in the Blink proof behaves as a secure checkpoint or as a new genesis: this means that it will never be reverted and the consensus rules applied to genesis are consistent with the consensus rules applied to the new genesis (we prove this in Appendix D.1).

Therefore, upon accepting a proof and identifying the new genesis $\mathcal{G}'$, the Blink client can broadcast $\mathcal{G}'$ to the provers, and provers start sending to the client the blocks descending from $\mathcal{G}'$. The client now maintains the longest chain descending from $\mathcal{G}'$, temporarily running an SPV client on top of $\mathcal{G}'$, bootstrapped in $\mathcal{O}(1)$. When Tx_η is in a block k-deep in the longest chain, the client considers the payment final and terminates. This protocol retains the proof optimality of Blink (Tx_η is final at most $3k$ consecutive blocks after $\mathcal{G}'$), albeit with one more round of communication. Blink has the same latency of an SPV: k blocks during normal operation, and $2k$ in case of adversarial attack.

This construction can be used out-of-the-box in the Bitcoin Backbone protocol in the static difficulty setting, without assuming block headers include a commitment to the state of the ledger. We refer to Section 7 for the variable difficulty setting and practical deployments.

Bootstrapping via Blink. In blockchains, there is interplay between different types of parties: *consensus nodes*, *full nodes*, and *light nodes*. *Consensus nodes*, also called miners, receive transactions from the network (environment) and execute a distributed ledger protocol that outputs a ledger. *Full nodes* do not participate in the distributed ledger protocol but receive the ledger from consensus nodes. They re-execute transactions to verify validity, keep audit proofs, and serve reads. Finally, *light nodes* connect to full nodes to verify inclusion of specific transactions or of specific state elements, e.g., an account balance. Bootstrapping these nodes usually requires a lot of time, e.g., several days for consensus and full nodes. It also requires a lot of bandwidth, compute, and disk space: starting from genesis, consensus and full nodes need to download and execute the entire ledger, while SPV light nodes need to download and verify the entire header chain.

We have already seen that Blink can be used for bootstrapping nodes: by identifying a recent block that behaves as a new genesis, SPV nodes can pinpoint the tip of the chain in $\mathcal{O}(1)$ and then run their protocol on top of it. Consensus nodes and full nodes, on the other hand, can use Blink to identify the new genesis and from it extract the commitment to the whole state of the ledger. Then, upon receiving from other nodes the state of the ledger, e.g., the set of

UTXOs, they can verify its validity, without downloading and re-executing the entire history of transaction.

State Verification. As stated in Section 2, Blink extracts a commitment to the global state of the ledger on the premise that each block embeds state commitments. Commitments come in different flavors (Merkle trees, accumulators, vector commitments), and they are used to efficiently download and validate the state of the ledger without having to receive and re-execute a complete copy of the ledger. In a chain with state commitments, *Blink allows to verify account balances and read the current state of on-chain contracts in* $\mathcal{O}(1)$. For a discussion on chains that have state commitments and how to introduce them to systems like Bitcoin, see Section 7.

Historical Transaction Verification. However uncommon, some applications might need to verify the on-chain inclusion of a transaction that is, e.g., 1 day old. Consider the case where a user wants to verify a day old transaction inclusion. With Blink, they can identify the new genesis, and then hop back, block by block, until hitting the block with the desired transaction. This naive approach for checking past transactions comes with a linear overhead in the age of the transaction: the older the transaction, the more blocks the client has to process. More efficient techniques exist under the name of *proofs of ancestry*: these add within block headers a commitment to a data structure, e.g., a Merkle Mountain Range or a vector commitment, that allows to navigate the chain backward with a logarithmic or constant overhead in the age of the transaction, respectively. Combining Blink with chains that have proofs of ancestry allows for more efficient historical transaction verification.

Bridging with Blink. In recent years, light clients have become a pivotal component for bridge protocols, allowing to efficiently and securely read blockchain data within new resource-constrained environment: the blockchain itself. Successful bridges move a high volume of transactions: ideally, at least one transaction per block. Therefore, every block including a cross-chain transaction must be relayed from the source to the destination chain by default, in an SPV-like fashion. However, the on-chain costs of the bridge contract can be minimized by skipping the block verification: blocks can be optimistically accepted and only verified on-demand in case a dispute is raised. This is what an *optimistic bridge* does. We now show how to *use Blink for creating succinct fraud proofs* for an optimistic bridge.

Consider a PoW source blockchain $\mathcal{C}_S$ equipped with ancestry proofs and a destination chain $\mathcal{C}_D$. *Relayers* of the bridge optimistically relay a block B from $\mathcal{C}_S$ to the bridge contract in $\mathcal{C}_D$, along with a random string η_R freshly sampled for the block. Should a *challenger* notice that relayers submitted an invalid block, they have a time window to trigger a challenge, pinpointing the contested block B, and revealing a newly sampled random string η_C. The challenger also publishes a transaction Tx_η on $\mathcal{C}_S$, which includes $\eta := \eta_R \oplus \eta_C$ ($\oplus$ is bit-wise xor). The bridge contract will accept the first valid Blink proof received and, via ancestry proof, verify whether the contested block B is an ancestor of the first block in π. If this is not the case, B is removed from the bridge contract. We note

that both parties need to contribute with a random string to prevent each of them from cheating, i.e., pre-mining a fake proof. Honest behavior can be incentivized through collateral that is slashed or redistributed in case of misbehavior.

5 Analysis

In this section, we present the main theorems and an overview on the proof strategy. For model and definitions from the Bitcoin Backbone, see Appendix B. For our general results, see Appendix C. For the complete analysis of Blink security, see Appendix D. In this paper, we prove the following main theorem:

Theorem 1 (Ledger Client Security for Blink). *Blink is* ledger client secure *as per Definition 5, with the safety parameter v being the wait time parameter u of liveness.*

We start by identifying a special type of block: a *convergence event at round r* (Definition 19). A convergence event is an honestly produced block that, by r, has no acceptable parallel block (Definition 18); a block is *acceptable* if it is valid and there is at least one honest party who may extend it. Convergence events have interesting properties: (i) acceptable blocks at r descend from all convergence events at r (Lemma 7); (ii) let $\tilde{\mathsf{B}}$ be a convergence event at r_c, and $\hat{\mathsf{B}}$ be an existing valid block whose height is larger than the one of $\tilde{\mathsf{B}}$ by at least k. Then, $\tilde{\mathsf{B}}$ is destined to become stable for all honest parties, even if $\hat{\mathsf{B}}$ is only known to the adversary (Lemma 8); (iii) for any block, the prior convergence event is always less than k blocks back (Theorem 5). These properties are formally presented in Appendix C: they are general and not specific to our construction, therefore they might be of independent interest.

We prove the admissibility of $\pi[0]$, arguing about its liveness and safety (Theorem 8). Towards proving safety of $\pi[0]$, we show that $\pi[k:]$ always extends $\tilde{\mathsf{B}}$ which, at the time that π is found, is the convergence event closest to $\pi[k:]$ (Theorem 7). Since $\pi[k]$ is fewer than k blocks in the future from its closest convergence event, $\tilde{\mathsf{B}} \in \pi$. Also, $\tilde{\mathsf{B}}$ will become stable (Lemma 8), and thus $\pi[0]$ is safe. Towards proving liveness for $\pi[0]$, we know that $\pi[k]$ is fresh because it contains the newly sampled η; $\pi[0]$ is k blocks distant, and thus fresh. As a result of honest majority, a proof extending $\tilde{\mathsf{B}}$ is found first. To conclude, assuming that block include state commitments, we prove by reduction that any client construction that fulfils the chain client security definition, also fulfills the ledger client security definition (Corollary 3).

Theorem 2. *Blink has optimal* communication cost*, i.e., $\mathcal{O}(k)$.*

The *communication cost* (Definition 6) measures the bits sent/received by V during an execution $\mathcal{E}$ of a protocol $\Pi(\mathcal{P}, V)$. V sends η whose size is $\mathcal{O}_\lambda(1)$ to all provers, and it receives (at most) one proof consisting of $2k+1$ block headers for each $P \in \mathcal{P}$, plus an inclusion proof for Tx_η which is of size $\log N$, with N being the average number of transactions in a block; $\log N$ can be considered a constant. This makes for a total size of $\mathcal{O}(k)$.

Note that in any light client construction the communication cost increases linearly in the number of provers. Nevertheless, the *communication cost* of Blink remains constant in the system's lifetime. From Theorems 1 and 2, Blink is an Optimal Proof of Proof-of-Work (Definition 7).

6 Evaluation

We evaluate Blink by measuring its proof size and waiting time for Bitcoin. A Proof-of-Concept implementation is available at [25], and entropy transactions broadcast during the evaluation can be inspected here [26]. The client uses the python-bitcoin-utils library [27] to create entropy transactions and communicates via RPC APIs using the python request HTTP library [28].

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 a complete copy of the ledger and they allow us to broadcast transactions to the Bitcoin network as well as retrieve blocks, transactions, and inclusion proofs. We ran our custom implementation client on commodity hardware. The client begins by sampling uniformly at random a 160-bit string η and creating Tx_η by placing η in an `OP_RETURN` output. The size of Tx_η is 222 bytes. Then, the client connects to the two Bitcoin full nodes, broadcasts Tx_η, and waits for it to be k-confirmed. We set $k = 6$ according to Bitcoin folklore. When one of the two full nodes reports Tx_η k-deep, the client downloads and verifies the Blink proof ($2k + 1$ block headers) by checking their parent-child relation and the PoW inequality. The client additionally downloads and verifies the inclusion proof of Tx_η in the middle block.

Proof Size. We measure all the data received by the client from the full node that first reports Tx_η with k confirmations. This data amounts to 7728 bytes (7360 for the Blink proof, 368 for verifying transaction inclusion).

The 7728 bytes are due to full nodes using the inefficient JSON format and to the available standard RPC endpoints of the bitcoin daemon 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 (1040 bytes for the 13 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 Tx_η). In Table 2, for height 841368, we compare the amount of data downloaded by Blink (1.6KB) to the one of a full node (684GB), to an SPV client (67.3MB), to NIPoPoW and FlyClient clients (10.0KB and ∼5KB, respectively), and, finally, to a PoW ZK-STARK client (ZeroSync[9], 197KB). Importantly, *the differences between these clients increase as the blockchain grows.*

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 Tx_η to be included in the next 1 or 2 blocks. The average waiting time for the client to accept a proof is 59 minutes, with a standard deviation

Full node	SPV[2]	KLS[5], NIPoPoW [3], Mining LogSpace[13]	FlyClient[6]	ZK ZeroSync Client [9]	**Blink**
684GB	67.3MB	10KB	∼5KB	197KB	1.6KB

Table 2. Amount of data light clients download for Bitcoin at height 841368, $k = 6$.

of 17 minutes, in accordance with the Bitcoin folklore of 6 blocks per hour. Any node that waits for 6 confirmations to report transactions final incurs the same waiting time.

7 Practicality, Limitations, and Extensions of Blink

State Commitments. Some of the applications presented in Section 4 operate 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. NIPoPoWs, i.e., the polylogarithmic clients described in [3,6], have the potential to be added retroactively via a velvet fork [12,29]. The idea of introducing state commitments for Blink via velvet fork is appealing, however, its practical deployment is still undetermined.

Entropy Transaction. We observe that to make light clients more efficient, the computational and storage complexity is often shifted from the light client to the full nodes or to the consensus nodes: for instance, NIPoPoW clients require consensus nodes to augment the chain with an interliking structure, FlyClient requires full nodes to store and update an MMR where each leaf is a block of the chain (plus additional metadata), and zk clients require full nodes to compute heavy proofs. In this respect, Blink has *optimal resource consumption for all the parties involved*: the light client, the full nodes, and the consensus nodes. Blink only requires to post on-chain a transaction with high entropy.

The Blink's entropy transaction does not need to have a Blink-specific format, but it can be, for instance, a transaction that sends money to a fresh address sampled with high entropy. In this case, the fee required to publish the entropy transaction is absorbed by the fee of performing a payment. If the entropy transaction is posted with the sole purpose of injecting entropy on-chain (e.g., by using the `OP_RETURN` opcode), fees can be paid in the form, for instance, of an anyone-can-spend output. The way the light client pays the on-chain fees can be optimized on the application level: For instance, dedicated contracts or untrusted services can mitigate the clients' costs by batching different requests and compressing multiple entropy values into a single transaction. 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 client instance needs a proof of inclusion of its randomness in the tree.

Interactivity. Blink demands one round of interactivity between the client and the full nodes, unlike its predecessors that operate non-interactively [6,3,5]. This is the trade-off we incur for achieving a constant-sized proof instead of a polylogarithmic one. 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 $\mathcal{O}(1)$ non-interactive client is possible without extra assumptions.

Variable Difficulty. Blink is analyzed in the static setting [10], 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 the difficulty beforehand by looking it up on a trusted service, or making assumptions on the computational power of the adversary. Ideally, designing a secure client in the variable difficulty setting [30] should be possible by using *difficulty balloons* to succinctly measure the current difficulty [31]. This approach utilizes entropy proofs to estimate (within some error) the current PoW difficulty of the network, by which point Blink can be applied as is. However, we anticipate that such an approach would only be secure under a weaker adversary, i.e., one 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 abstains from mining while the clients measures the difficulty, thus creating a false sense of the total computational power and of the number of blocks they can create in a set of rounds. Then, the adversary takes advantage of this false estimation to mine privately the required proof, violating the safety of Blink. We estimate that this adversarial advantage may be mitigated if honest nodes can produce double as many blocks as the adversary.

Alternatively, one could change the selection rule for the proof: the client can choose the proof with the most work after taking the intersection of all proofs within a timeout. We conjecture such an approach may alleviate the possible attack vectors of a minority adversary, and we plan to explore it in future work.

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, and by Input Output (http://iohk.io) through their funding of the Edinburgh Blockchain Technology Lab.

References

1. J.K. Rowling. Harry Potter and the Deathly Hallows, Chapter 36: The Flaw in the Plan, 2009.
2. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. http://bitcoin.org/bitcoin.pdf.
3. Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-Interactive Proofs of Proof-of-Work. In *Financial Cryptography and Data Security*. Springer International Publishing, 2020.
4. Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Compact Storage of Superblocks for NIPoPoW Applications. In *Mathematical Research for Blockchain Economy*. Springer International Publishing, 2020.
5. Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of Proofs of Work with Sublinear Complexity. In *Financial Cryptography and Data Security, 2016*. Springer Berlin Heidelberg.
6. Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. FlyClient: Super-Light Clients for Cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy*, 2020.
7. Psi Vesely, Kobi Gurkan, Michael Straka, Ariel Gabizon, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, and Eran Tromer. Plumo: An Ultralight Blockchain Client. In *Financial Cryptography and Data Security, 2022*, Berlin, Heidelberg, 2022. Springer-Verlag.
8. Mina docs, 2023. https://docs.minaprotocol.com/about-mina.
9. Linus Robin, George Lukas, Milson Andrew, and Steffens Tino. ZeroSync - STARK proofs for Bitcoin, 2024. https://github.com/ZeroSync/header_chain.
10. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. New York, NY, USA, 2024. Association for Computing Machinery.
11. 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*, 2020.
12. 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.
13. Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. Mining in Logarithmic Space. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, New York, NY, USA, 2021. Association for Computing Machinery.
14. Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized Cryptocurrency at Scale. *IACR Cryptol. ePrint Arch.*, 2020.
15. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *30th USENIX Security Symposium 2021*. USENIX Association, 2021.
16. How to validate Bitcoin payments in Ethereum (for only 700k gas!), 2018. https://medium.com/summa-technology/cross-chain-auction-technical-f16710bfe69f.
17. Giulia Scaffino, Lukas Aumayr, Zeta Avarikioti, and Matteo Maffei. Glimpse: On-demand PoW light client with constant-size storage for DeFi. In *32nd USENIX Security Symposium 2023*. USENIX Association, 2023.

18. Shresth Agrawal, Joachim Neu, Ertem Nusret Tas, and Dionysis Zindros. Proofs of Proof-Of-Stake with Sublinear Complexity. In *5th Conference on Advances in Financial Technologies (AFT 2023)*, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
19. Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. SoK: Blockchain Light Clients. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022*, Berlin, Heidelberg, 2022. Springer-Verlag.
20. 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.
21. 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.
22. Aggelos Kiayias and Dionysis Zindros. Proof-of-Work Sidechains. In *IACR Cryptology ePrint Archive*, 2019.
23. Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-Stake Sidechains. In *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.
24. Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light Clients for Lazy Blockchains. In *Financial Cryptography and Data Security FC 2024*. Springer-Verlag, 2024.
25. Blink Implementation, 2024. `https://github.com/scaffino/Blink`.
26. Bitcoin Address. `https://shorturl.at/9gQP2`.
27. Bitcoin Utils, 2024. `https://pypi.org/project/bitcoin-utils/`.
28. Python Request Library, 2024. `https://pypi.org/project/requests/`.
29. 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. In *Financial Cryptography and Data Security, FC 2019*, Berlin, Heidelberg, 2019. Springer Berlin Heidelberg.
30. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology – CRYPTO 2017*. Springer International Publishing, 2017.
31. Dionysis Zindros. *Decentralized Blockchain Interoperability*. PhD thesis, University of Athens, Apr 2020.

A Algorithms and Pseudocodes

Figure 6 shows the Blink protocol for payment verification.

Algorithm 1 showcases 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.

B Background from the Bitcoin Backbone [10]

We now introduce notation, definitions, theorems, and lemmas stated in [10] which will be necessary for our analysis.

Proof Construction

5. V accepts the first π it receives consisting of $2k+1$ consecutive well-formed blocks where the middle block contains η, i.e., $\pi[k] = \mathsf{B}_\eta$
6. Upon accepting π, V extracts the new genesis $\mathcal{G}' := \pi[0]$ and sends $\mathcal{G}'$ to all $P \in \mathcal{P}$
7. Each $P \in \mathcal{P}$ keeps sending all the blocks descending from $\mathcal{G}'$ in their chain

State Extraction

8. V maintains the longest chain $\mathcal{C}$ descending from $\mathcal{G}'$
9. V terminates when it sees Tx_η k-deep in $\mathcal{C}$

Fig. 6. Payment verification with Blink. Steps 1-4 are akin to Figure 5.

Algorithm 1 The algorithm ran by the verifier V, i.e., the Blink client. We split the proof π into (π_0, π_1), with π_0 allowing to identify a stable and recent block of the blockchain, i.e., the new genesis $\mathcal{G}'$, and π_1 being the Merkle proof that verifies inclusion of η into the middle block of π_0.

```
1: function VERIFIER_G()
2:     η ←-- {0,1}^λ
3:     for P ∈ P do
4:         η --→ P
5:         while True do
6:             π ←-- P                ▷ Only constant amount of data downloaded
7:             (π_0, π_1) = π
8:             if VALID_G(π, η) then
9:                 return π_0[0]
10:            end if
11:        end while
12:    end for
13: end function
```

B.1 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.

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 $\mathsf{B} \in \pi_0$ **do**
    if $\mathsf{B}.\mathrm{s} \neq h$ **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 Tx_η 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

To model the Proof-of-Work setting, the *q-bounded synchronous setting* defined in [10] 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 [10] 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 B. 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.

B.2 Formal Properties and Definitions

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 κ (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 10 (Common Prefix Property [10]). *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 $\text{view}^{t,n}_{\Pi,A,Z}$ respectively, it holds that $C_1^{\lvert k \rvert} \preceq C_2$.*

Definition 11 (Chain Quality Property [10]). *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 $\text{view}^{t,n}_{\Pi,A,Z}$, it holds that for any ℓ consecutive blocks of C, the ratio of honest blocks is at least μ.*

Definition 12 (Chain Growth Property [10]). *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[10], 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 13 (Typical Execution [10]). *An execution is (ϵ, λ)-typical (or just typical), for $\epsilon \in (0, 1)$ and integer $\lambda \geq 2/f$, if, for any set S of at least λ 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 λ, κ security parameters. Finally, we denote with ν the min-entropy of the value that the miner attempts to insert in the chain.

Theorem 3 (Theorem 4.5 in [10]). *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 1 (Lemma 4.6 in [10]). *The following hold for any set S of at least λ 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 2 (Lemma 4.8 in [10], (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 3 (Lemma 4.1 in [10], (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 4 (Lemma 4.2 in [10], (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 4 (Theorem 4.11 in [10], (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 1 (Corollary 4.12 in [10]). *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 λ 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 3 typical execution fails with negligible probability, resulting in our proofs holding with overwhelming probability.

C General Results

In this section, we introduce some definitions, observations, and lemmas that will be used as building blocks in the formal analysis of Blink security (Appendix D.1).

C.1 Notation

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 5 (Appendix B), 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.

C.2 Our General Results

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 14 (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 15 (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 16 (Valid Block). *A block is* valid *if it belongs to a valid chain.*

Lemma 5. *The following equality holds:*

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

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 17 (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 18 (Acceptable Block at r). *A block is* acceptable at r *if it belongs to an acceptable chain at* r*.*

Observation 1 *If a block is stable in an honest party's view, it is also acceptable.*

Observation 2 *All honestly produced blocks are acceptable in the round in which they are produced.*

Observation 3 *All honestly produced blocks only descend from blocks that are acceptable in the round in which the former are produced.*

Observation 4 *Any block* 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 19 (Convergence Event at r). *A block* 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 5 *A convergence event is always honestly produced.*

Lemma 6. *If a block* 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 B. Therefore, B fulfills the definition of convergence event at all rounds $\{r_B, \ldots, r\}$.

Lemma 7. *An acceptable block* B *at* r *must descend from all convergence events at* r *with a height smaller or equal to* B*'s height.*

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

Lemma 8. *Let* r *be the round in which a block* B *was produced. For any block* B *and any round* r'*, for which* 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* B*.*

Proof. From Observation 2 and Lemma 7, honest parties will extend B in the rounds between r and r' (included). B is stable for all honest parties at round r'. Therefore, after r' all honest parties only extend 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.

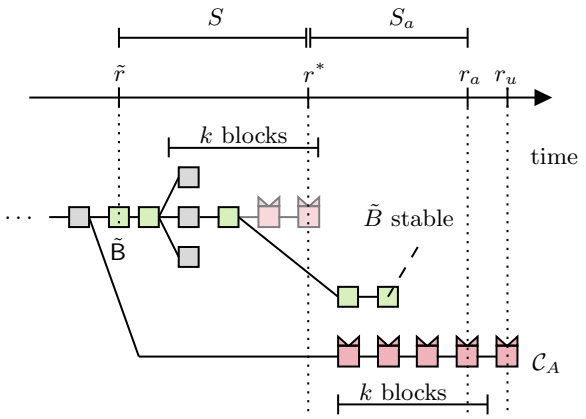

Fig. 7. This figure illustrates the proof of Theorem 5.

Theorem 5 (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* $\mathcal{C}^{\cup}_{r^*}$ *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.,* $\tilde{\mathsf{B}} \in \mathcal{C}^{\cup}_{\tilde{r}+u}[: -k]$.

Proof. Consider Figure 7. 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 $S' := \{\tilde{r}, \ldots, r^*\}$. Let $S := \{\tilde{r} - 1, \ldots, r^* + 1\}$. We can thus apply the patience lemma (Lemma 2) 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 4), 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 $\tilde{\mathsf{B}} \notin \mathcal{C}^{\cup}_{\tilde{r}+u}[: -k]$. 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 2 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 4, we know that there are at least $|X(S_a)|$ consecutive blocks extending $\tilde{\mathsf{B}}$. From Lemma 2, 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 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 6 (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, $\dot{l} - \tilde{l} < k$.*

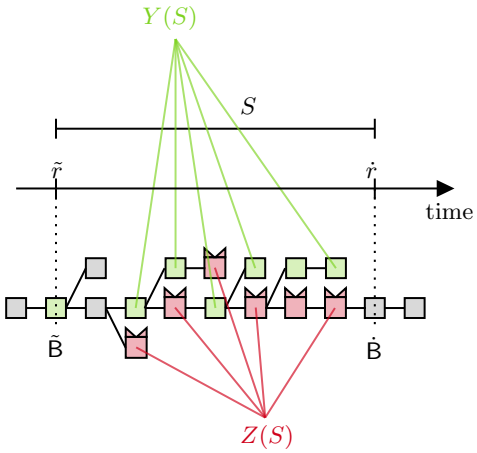

Fig. 8. This figure illustrates the proof of Theorem 6.

Proof. Consider Figure 8. 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 7, 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, 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 7) 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 3, 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 2 we know that $|S''| > \lambda$, thus $|S| \geq \lambda$, and thus, typicality bounds apply to this set of rounds. Thus, by Lemma 1 it holds that $|Z(S')| < |Y(S)|$, which contradicts the above.

D Analysis of Blink

D.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 η. η is received by the blockchain nodes at round r_0. We say the proof π 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 Tx_η to become stable. Finally, the client terminates when Tx_η 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 η 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 9. B' *exists.*

Proof. The genesis block satisfies the definition of B'.

We denote the round in which B' was produced as r', with $r' < r_0$. From Lemma 8, we know that all honest blocks produced after r' extend 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 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 5 and 6, 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 $\tilde{S} := \{\tilde{r}, \ldots, r^*\}$.

Lemma 10. $\tilde{\mathsf{B}}$ *exists.*

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

Lemma 11. *Acceptable blocks produced in* $\tilde{S}$ *descend from* $\tilde{\mathsf{B}}$.

Proof. This follows directly from Lemmas 6 and 7 (Appendix C).

As a consequence of Lemma 11 and Observation 2, all honest blocks produced in $\tilde{S}$ descend from $\tilde{\mathsf{B}}$.

Lemma 12. *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 [10] or Appendix B.

Theorem 7 (Anchor). *In a typical execution, the block with* η *and its* k *subsequent blocks of the proof* π *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 9 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^*.

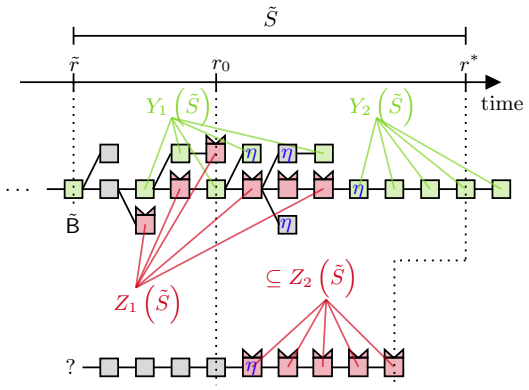

Fig. 9. This figure illustrates the proof of Theorem 7.

Claim. If $|Y_2| = k+1$, at round r^*+1 the client has received a proof π 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. 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 η in a block, if η was not included before. It follows that the block in Y_2 with the smallest height descends from a block including η. The k blocks produced by the remaining queries in Y_2 extend the block with η by one block each, as they are uniquely successful and there are no parallel, adversarial acceptable by r^* blocks.

Claim. 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 7, suppose that at round r^*+1 the client accepts a proof π 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 2 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 7.

Lemma 13. $\tilde{\mathsf{B}} \in \pi$.

Proof. Because the block containing η, $\pi[k]$ 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 6 that the nearest convergence event at $\dot{r}$ has a height difference smaller than k blocks.

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

Proof. (Safety) From Lemma 13 we know that π includes $\tilde{\mathsf{B}}$. From Theorem 5, 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 B'. Define $\mathsf{B}'' := \mathcal{C}^{\bigcap}_{r_0}[-k-1]$, and denote its height with height l''. Since B' is by definition either 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 B'' is stable for all honest parties at round r_0. Since $\pi[k]$ includes η it has to be mined after r_0, which is the round in which η 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 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 π, $\pi[0]$ 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 14 (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 2 (Chain Client Security for Blink). *Blink is* chain client secure *according to Definition 9.*

Given a client protocol Π which outputs a block B, one can build another protocol Π' that runs Π and reports the state commitment in B.[7]

Corollary 3. *For any client protocol Π that is chain client secure, the corresponding protocol Π' constructed in the above manner is* ledger client secure *(Definition 5).*

This follows from a simple reduction since Π' merely reports the state commitment of B. If the state commitment was such that Π' is not ledger client state secure, the corresponding B cannot have been admissible. This concludes the proof of the main Theorem 1.

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

D.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 B_η), 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 15 (Stability of Tx_η). *In a typical execution, a block B_η including Tx_η becomes stable for all honest parties at most at round $r_0 + u$, i.e., $\mathsf{B}_\eta \in \mathcal{C}^{\bigcap}_{r_0+u}[: -k]$.*

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

Lemma 16 (Vicinity of Tx_η). *In a typical execution, a block B_η including Tx_η 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 15, the entropy transaction Tx_η 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 B_η (Corollary 1), and Tx_η is at least k blocks deep in every honest party's chain. It follows that after at most $3k$ consecutive blocks are produced, B_η is stable for all honest parties.