Scalable Distributed Payment Systems with Minimal Trust Assumptions

by

Assimakis Agamemnon Kattis

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 2022

Assistant Professor Joseph Bonneau

Dedication

To my parents.

Acknowledgements

I would like to thank my advisor Joe, without whom I would have never had the opportunity to work on such exciting, difficult and rewarding problems.

I am extremely grateful to all my friends. Michali, Ari, Dionysi, Alex, Achillea, Giorgo and Fabian, thank you for your support and for continuing to put up with me.

Thank you to Memo, for keeping me in check, and to Loukia, for always being there.

Thank you Ana, without you this would have never happened.

Abstract

Over the last decade, the security and resilience of Bitcoin as a stable payment network has motivated substantial study of the viability of distributed payment protocols as alternatives to centralized payment processing. We investigate the design of scalable distributed payment systems in the permissionless setting, where no actors in the protocol can be trusted or identified with out-of-band information. Scalability is identified with two desirable properties: high transaction processing rate (or throughput) and low confirmation latency (or settlement times). We analyze the trade-offs inherent to distributed protocols that prevent naive optimization of the above parameters and study techniques from verifiable computation as potential tools for overcoming these bottlenecks.

One technique to increase throughput in distributed payment systems involves the use of Succinct Non-interactive ARguments of Knowledge (SNARKs, or SNARK proofs) to verify the integrity of transactions. Transaction rollups are one such solution, using SNARK computations to achieve scalability. Many instantiations of rollups leveraging SNARKs show encouraging evidence that this technique could achieve commercial-capacity throughput rates if implemented on top of current distributed payment systems, even in the smart-contract setting. Although promising, all rollup approaches require the resolution of an additional yet crucial question. For protocols operating in the permissionless setting, we need to ensure that a system relying on proof generation to scale also incentivizes actors to compute proofs cheaply and quickly. This is a governance problem, as the protocol needs to decide on how participants will be chosen to

perform these (expensive) computations. We pose the question of *who* will compute the proofs, identify it as a consensus problem and provide a technical proposal towards its resolution.

Our main contributions are twofold: in Part I, we design a permissionless consensus protocol that solves the problem of state verification for resource-limited clients in an incentive-compatible way. We show formal proofs of security and achieve minimal resource requirements for full ledger verification. This protocol showcases our key contribution: the design of a proof-of-work (PoW) process that computes SNARK proofs as valid outputs. Suitably choosing the statement whose proof is generated through PoW provides an incentive-compatible way to enforce the computation required by proof-based scaling techniques. In Part II, we look at one of the key components of SNARK-based throughput optimization: the non-interactive proof itself. We design a novel proof system which provides security guarantees in the trustless setting, while still being small and efficiently computable. This proof system (a transparent SNARK, or STARK) can be used directly for scaling throughput in distributed payments through transaction rollups. In conjunction with an incentivized PoW process, it also demonstrates a way for participants in consensus to quickly generate the rollup proofs in a permissionless way.

Contents

List of Figures

List of Tables

1 Introduction

We are interested in investigating how trust assumptions influence the design of payment systems. When building effective payment protocols, a primary concern is the safe and timely settlement of transactions between two end-users who don't necessarily trust each other. Such protocols usually also account for third parties: for example banks or clearing houses that settle the transaction on the end-users' behalf. In practice, payment systems enable commercial interactions between untrusted parties through trusted intermediaries: agents that both parties trust to perform the requisite verification and update the state of the participants' accounts. In this work, we focus on payment systems that require minimal trust assumptions by participants: this means that in order to guarantee secure and efficient settlement, the networking protocol should only require the honesty of the end-user and the majority of network participants, without relying on assurances (or prior-knowledge) about any other participant's behaviour.

We focus on designs with access to a universal ledger: payment networks in which every end-user has access to a distributed ledger of all account/balance pairs. The first question to resolve is how payments can be implemented without any requirements for trusted intermediation. Apart from preventing invalid transactions (or'double-spending'), the protocol needs to ideally satisfy two additional security properties: liveness, or that all valid transactions will eventually process, and (eventual) finality, or that a valid transaction cannot be reverted (with high probability) after it has been processed. The first protocol to achieve security in this setting is Bitcoin [Nakamoto 2008], which utilizes advances in cryptography and distributed systems to build a

single-asset settlement network. All parties have access to a universal ledger and utilize a peer-to-peer messaging layer to transact. This architecture has been developed further in Ethereum [Buterin 2014], where parties have the ability to encode arbitrary computation in transactions to decide how final balances are settled. We denote such systems as *decentralized* if they provide consensus guarantees over the state of the underlying ledger in the *pseudonymous* setting, where any party can leave or join the network at will in the presence of (a minority of) adversarial actors. As we will see, the notion of a pseudonymous network, in which each participant can label peers only locally without any (assumed) knowledge about them, is equivalent to a setting of minimal trust. This is because not having access to the 'actual' identities of peers means that no *a priori* knowledge assumptions can be made about their behaviour. We interchangeably refer to such protocols as having minimal trust assumptions, or as being secure in the permissionless threat model.

1.1 Throughput & Settlement Time

One crucial property that payment systems need to possess is a sustained capacity for high settlement rates. If the transaction processing rate (or throughput) is too low, then applicability of such systems to practically-relevant settings remains limited. Low transaction confirmation (settlement) times are also critical [Bech et al. 2017], and require strong liveness and finality guarantees. The first substantial improvements over daily settlement times for central bank transactions were achieved through Real-Time Gross Settlement (RTGS) [Allsopp et al. 2009] with the development of FedWire in 1970, permitting real-time settlement of transactions between US Federal Reserve banks. By the 1980s, similar systems were adopted in other countries, such as the Clearing House Automated Payments System (CHAPS) in the UK or SAGITTAIRE in France. Although secure, high-throughput and with instant settlement, RTGS has been heterogeneously implemented by central banks of different countries, usually in accordance with domestic ap-

proaches and regulations. In practice, it is used for large inter-bank transactions and only indirectly benefits end-users, who are still beholden to the legacy infrastructure operated by their (trusted) banking partners. RTGS is fundamentally different from traditional 'net settlement' systems such as Automated Clearing House (ACH) in the US or BACS in the UK, in which banks aggregate intra-day transactions and settle them all at end-of-day (and are thus necessarily restricted to daily settlement times). Efforts to expand the usage of RTGS systems and standardize their interactions are many and ongoing, due to their improved security, efficiency and simplicity. RTGS does, however, remain a means of conducting large transactions over central banks, and so its effects on smaller retail payments remain indirect. Upgrading the infrastructure of retail payment providers is an ongoing but orthogonal effort, substantially more challenging due to the many different actors involved and the heterogeneity of their banking infrastructure and assets.

Although efficient, these systems all rely on trusted intermediation for transaction processing. We investigate how to achieve comparable throughput and security guarantees in decentralized payment solutions that do not impose trust assumptions on participants. Following prior work, we focus on the application of succinct non-interactive proofs of (computational) validity to distributed payments, with the aim of leveraging their security and efficiency properties to improve the throughput of both payment-only networks such as Bitcoin and arbitrary-computation ('smart-contract') platforms such as Ethereum.

The access to a universal ledger and unified transaction semantics (i.e. all transactions have the same structure) means that such protocols have uniform average liveness and finality guarantees, albeit with subtle trade-offs. For example, transactions on Bitcoin and Ethereum usually process in minutes/hours, while using a clearing-house to settle a stock purchase usually takes 1 to 3 days. Liveness also follows, as any valid transaction will eventually be processed since it does not depend on (potentially erroneous) rejection by trusted intermediaries. Unfortunately, throughput is the main factor currently influencing such systems' adoption [BIS 2018]. Although Bitcoin and Ethereum currently reach $\sim$ 10 transactions per second (tps), this is at least an order-

of-magnitude behind commercial payment solutions ($\sim$ 500 tps), and many orders-of-magnitude behind the maximal supported throughput ($\sim$ 3,500 tps) of existing settlement networks. We investigate the inherent trade-offs in security and throughput/settlement times of decentralized networks and propose consensus-layer modifications that can achieve better throughput while retaining the same security and decentralization guarantees.

1.2 Protocol Governance

A second crucial question in the design of distributed payment systems is the type of privilege that different actors should have in modifying the common ledger. This question can be resolved in multiple ways, depending on the type of network that is required. If actors have *a priori* knowledge of other network participants, this defines a *permissioned* system and there exists a large body of work studying the optimal ways to achieve consensus in such settings (cf. Section 2.1). Although distinct from node honesty/compliance, requiring knowledge about specific nodes to be available to all translates, in practice, to the existence of trusted intermediaries performing specific functions that behave honestly. In the context of monetary policy, a network which confers such a power to a central monetary authority would necessitate restrictions on the types of actors that can modify the ledger: this is the starting point for Central Bank Digital Currency (CBDC) initiatives [Auer and Böhme 2020; Auer et al. 2021]. We should note that any efforts in optimizing the transaction settlement times and protocol throughput are still applicable in this context, i.e. the above is a claim about governance and can still leverage the full toolkit of efficiency improvements that we develop. Indeed, this more restricted threat model can be leveraged for throughput gains not only in the transaction layer, but also the consensus process. Since we are not bound to the pseudonymous setting (in which prior knowledge about participants is not provided), we can assign identities to participants and use these to perform consensus with improved throughput and settlement times. Such a system would be distributed and efficient, but its

ultimate security would rest on nodes being able to correctly identify honest participants, which is not a realistic assumption for large-scale trustless networks .

If we want to retain the threat model of minimizing trust in the system, we are thus forced to work in a fully *permissionless* system: a pseudonymous setting in which every participant has the opportunity to update the ledger with valid transactions. This means that we cannot expect to be able to assign identities to any participants in the system, i.e. like expecting honest behaviour from specific parties. In this world, governance rights iterate over all actors involved in transaction processing and ensure (eventual) finality if some proportion of participants acts honestly. Note that the generality of the threat model necessitates a severe restriction in the type of consensus algorithm that can be used: Nakamoto Consensus (NC) [Nakamoto 2008] and iterations over its basic premise offer the only suitable approaches through which security can be attained in this setting.

This observation leads us to a fundamental question: how can we retain these minimal trust assumptions in a protocol that is capable of processing transactions at commercially viable levels? Such a design would have to be permissionless, but with the necessary transaction layer semantics and economic incentives to ensure high throughput and fast settlement times between untrusted parties. Currently, the way in which leader-election is performed in permissionless distributed payment systems adds substantial inefficiencies to protocol throughput: since we operate trustlessly, every honest node needs to verify the integrity of all transactions it processes, leading to a large amount of replicated computation and bandwidth consumption. This added complexity means that permissionless networks impose an increased resource burden on both the underlying network and its honest participants, bounding attainable improvements in throughput and settlement times for a given security level. In our approach, this issue is tackled at the consensus layer using verifiable computation.

An independently relevant property of permissionless consensus is that it is inherently wasteful; pseudonymous participants are assigned ledger modification rights through expending large

amounts of iterated and unstructured computation, which is not useful in any other context. This property is critical for Sybil-resistance guarantees, or the ability of the network to withstand large volumes of malicious traffic. Since participants are not trusted, such a measure is necessary for system liveness and eventual finality guarantees. We thus look at this feature as an integral requirement for permissionless consensus, opting to leverage it for the performance of incentivized computation instead of removing it from the protocol and changing the threat model.

1.3 Overview of Contributions

In this work, we study the proposition that scaling throughput in decentralized payment systems can happen through the use of Succinct Non-interactive ARguments of Knowledge (SNARKs, c.f. Chapter 4). As described in Section 3.2, there exist approaches that enable substantially higher transaction settlement rates in decentralized payment systems by leveraging the security and efficiency properties of SNARKs at the transaction layer. We conduct an overview of the scaling solutions currently being considered and implemented, specify their trade-offs, and conclude that ZK-Rollups (c.f. Section 3.2.2) are the most secure approach to increase transaction throughput, mainly due to concomitant data-availability and security guarantees. Although this position is not new, we pose an unaddressed general problem facing all current solutions, such as ZK-Rollups, that utilize SNARKs as part of their protocol. Namely, which protocol actors are responsible for performing all the expensive computations for the generation of these proofs, and how do they interact (if at all) with the underlying system and its efficiency? Indeed, although many high-throughput (more than 500 tps) approaches have focused almost exclusively on providing transaction layer semantics for efficiently computable proofs that also achieve settlement times that make sense (within a minute), the total amount of proof computation required is so large that participants cannot be presumed to always be incentivized to provide it at consistent levels. Moreover, in all current ZK-Rollup approaches this issue is not addressed, meaning that

the computational costs of producing proofs are not factored in (i.e. proofs are presumed to be 'free') when reasoning about system performance. Unless we opt for trusted proving, this is fundamentally a governance problem about incentives and therefore needs to be resolved at the consensus level.

Thus, even though high throughput rates have already been attained experimentally for proof-based transaction layers, the governance question still remains: who will compute the proofs? In a completely specified distributed payment system, any proof-based scaling solution comes with the question of which actors will be expected, according to honest behaviour, to produce the requisite proofs. If actors are not correctly or adequately incentivized, the protocol cannot provide efficiency guarantees. Moreover, in the trust-minimized pseudonymous network model, this process becomes even more restrictive. We review this in Chapter 2, and identify related works (Section 2.2.2) that extend the NC protocol to fully optimize its use of the network layer to maximize throughput with respect to latency and bandwidth.

In Section 2.3, we discuss the key underlying primitive that forms the basis of our work: moderately hard or proof-of-work (PoW) puzzles. Our key contribution is a protocol secure in the trust-minimized setting that provides large amounts of valid proofs as a useful byproduct of consensus. This is done by designing the underlying computation performed for leader election (which is a PoW puzzle) to generate valid proofs for some relevant statement. This process requires specific security properties to be satisfied, but is otherwise modular with respect to any consensus protocol utilizing moderately hard puzzles as an underlying primitive. Since Sybil-resistance is necessary, we view PoW as an integral part of any payment system within such a threat model and focus on leveraging it to incentivize computation that can increase throughput.

In Part I, we construct a consensus protocol in which each valid PoW leadership ticket is also a proof of state validity. This means that instead of providing arbitrary computation for leader election, participants in the network are iteratively generating a SNARK testifying to the validity of their set of transactions to be processed (and on which previous state to do so) if they win.

This design has two main advantages:

1. Providing proofs of state validity is both useful and expensive, so doing this 'for free' through the wasteful PoW consensus process resolves the question of who will compute the state validity proof while retaining consensus-critical incentive compatibility.

2. We ensure that participants are also incentivized to produce faster and more power-efficient (hardware) solutions for proof computation. As these are modular processes for which specialized hardware can be constructed, such an effort would have positive vertical effects on all proof-based protocols since efficient proof-generation solutions, designed by actors seeking leadership tickets, could become available to the ecosystem as a whole.

The specification in Chapter 6 is designed with an orthogonal problem in mind: succinct state verification for entities who do not have access to the full ledger. In practice, this is important for light-clients (c.f. Section 3.3) and our approach provides a fully-specified system resolving this issue with optimal light-client computational requirements. The difference with other approaches is its incentive-compatibility: we know exactly who computes the proofs and the incentive to do so (quickly) always exists.

This protocol, however, is just a specific construction showcasing the feasibility and power of embedding proof computations into the leadership election process through PoW. Indeed, the techniques we develop (c.f. Chapters 7, 8) are broadly applicable and general, usable for the instantiation of other proof statements as PoW puzzles. An immediate corollary of this work is that it is possible to incentivize the creation of ZK-Rollup proofs of batched transaction validity (rollup proofs) by enforcing their generation at the consensus layer as the underlying PoW puzzle. In such a model, miners would be able to mine for the next block as they do in Bitcoin, but with the difference that a valid PoW solution is also a valid ZK-Rollup proof for the transactions to be processed. We can then design the transaction semantics of the PoW puzzle to commit a minimal witness on-chain (to maximize throughput, c.f. Section 3.2.1) and thus

achieve incentive-compatible scaling. Note that NC consensus improvements from the literature can be switched-in modularly to better leverage the underlying network without affecting the incentives of proof production. In this setting, the system is operating as a 'PoW-Rollup': settling transactions using (1) the ZK-Rollup architecture to achieve high throughput guarantees, and (2) an always-incentivized set of actors (the miners) that quickly compute the rollup proofs through the PoW process. This could provide a potential avenue for scaling that transcends existing decentralization-efficiency-security trade-offs.

In Part II, we look more deeply at what properties the underlying proof systems need to satisfy in order to achieve throughput increases (by computing rollup proofs) without introducing additional trust assumptions. We provide a new SNARK construction that concurrently satisfies a variety of properties (c.f. Sections 4.2, 4.3) necessary for trustless computation. In this process, we identify primitives relevant to efficient proof generation and provide an analysis of their security. Key to our approach is the observation (also noted in concurrent work [Chiesa et al. 2019b]) that SNARKs can be compiled using two modular objects: Interactive Oracle Proofs and Polynomial Commitments (c.f. Sections 11.3,4.5). We show secure compilation of SNARKs with a more general notion of Polynomial Commitments (which we call List Polynomial Commitments, see Chapter 12) that allows us to retain small proof sizes, which are the main bottleneck for higher throughput, and fast proving times, which correspondingly decrease the time to settlement. The two main design challenges here involve the security model and functionality of the proofs: we require no trusted intermediaries in the process of generating proofs (or their public inputs), while also requiring that all truth-statements can be trustlessly verified using the same set of initial public parameters.

We can use the contributions from Parts I and II to design trustless protocols for the verification of system state. This is done by modularly replacing the trusted proof system used in the design of the consensus protocol in Part I with a trustless proof system like the one designed in Part II. This is important since such a proof system would not require additional trust assump-

tions while remaining efficient. Even though a different design and security analysis of the PoW process is required due to such a change, the provided formalization framework and related security definitions offer a starting point for its investigation. The proof system designed in Part II is also extremely useful for throughput optimizations at the transaction layer alone. More specifically, the potential for fast recursion (as exemplified in subsequent work iterating on our model) is especially well-suited for efficient transaction rollups. This is very promising, and could imply that a fully trustless transaction rollup may be practical enough to scale the throughput capacity of decentralized systems. Bootstrapping such a distributed payment system with a PoW process computing the required rollup proof would then incentivize the perpetual functioning of the ZK-Rollup as a critical part of consensus. Note the two distinct aspects of this proposal: (1) at the transaction layer, operate a rollup using a trustless proving system, and (2) at the consensus layer, make the PoW process require the generation of rollup proofs. The second property is critical in enforcing the first, as it provides the incentives for participants to compute the proofs. In essence, this constructs a market for proof computation between miners and end-users.

2 | Fault Tolerance & Sybil Resistance

2.1 Byzantine Agreement

We work in a setting where participating actors cannot be assumed to be trusted. The most general problem in fault-tolerant distributed systems that captures this property is that of Byzantine agreement [Pease et al. 1980; Shostak et al. 1982], where n participating nodes have to arrive at consensus over some common value in the presence of t arbitrary (or Byzantine) adversarial nodes. It has been shown that if $n \leq 3t$ or there are less than $t + 1$ rounds of interaction, no such protocol exists [Pease et al. 1980; Fischer and Lynch 1981]. These properties are respectively known as resilience and early stopping. Prior works [Garay and Moses 1998; Dolev and Strong 1982] have successfully designed protocols with both optimal resilience and early stopping with polynomial message and computational overheads, while more recent work [Abraham and Dolev 2015] has focused on minimizing message complexity.

In our context, the above model will be too restrictive to be of use. This is because we are interested in designing a large-scale network where participants are not expected to know each other, and can enter or leave the network as they please. This means that nodes can't be expected to know their index in a total enumeration of all active nodes, which is a critical property on which the above results rest. This is a strictly weaker model of computation, which does not assume authenticated channels between participants. Since it is easy for adversarial nodes to control multiple pseudonymous identities in this context, we are also interested in Sybil-resistance, or

that an adversary will not be able to overwhelm the network even if they indirectly control a sufficiently large number of nodes.

2.2 Nakamoto Consensus

In order to achieve Byzantine agreement with Sybil-resistance guarantees, the Bitcoin protocol uses moderately hard puzzles as a key underlying primitive. This allows participating nodes to contribute computing power to the network (by solving the puzzles) in order to claim their identity, prohibiting a computationally bounded adversary from obtaining too many identities and thus corrupting the system. This insight requires the usage of a puzzle in the form of a non-interactive proof, which we equivalently denote as a proof-of-work (PoW) puzzle. The key security property it satisfies is formalized in the following section, but informally requires that (1) the difficulty of the puzzle can be set so that a solution arrives on average every λ timesteps, (2) verifying that a solution is correct is substantially faster than finding the solution to begin with, and (3) no marginal computational gains can be obtained by batching multiple puzzles in one computation. Equipped with such a primitive, Nakamoto Consensus (NC) can be defined as a protocol between n total and t Byzantine nodes with each having equivalent computational power, where consensus is obtained when $n \geq 2t + 1$. It has been shown [Miller and LaViola Jr 2014] that this resilience is optimal for protocols based on random oracles, regardless of the choice of PoW puzzle.

Each honest node in NC serially checks solutions to the underlying PoW puzzle until a valid solution is obtained, at which point it broadcasts the solution to the network where it is verified by other nodes. Using this puzzle solution, the node votes for its preferred consensus value (which it commits to before computing the PoW). The security guarantee is that, if the majority of the computation for PoW puzzle solutions is done by honest nodes ($2t+1 \leq n$), the protocol has a negligible chance of not achieving consensus over the underlying value: instead of requiring per-

fect consensus, disagreement on which input value is chosen happens with negligible probability over a consensus security parameter.

2.2.1 The Bitcoin Protocol

In Bitcoin, participating nodes are known as miners, as they serially compute solutions to the underlying PoW puzzle in order to cast votes on the correct state of the system. The state is given by a sequence of blocks, each containing a set of transactions in a specified order. Every time a miner begins to compute a new PoW solution, they have to pick the block they want to vote for (or 'build on'), choosing which ledger state they believe to be the correct one. They also pick the set of transactions to process into the state defined by this block. A miner issues a valid block if they can find a PoW solution, broadcast it to the network, and have a majority of miners accept it as the next update to system state. This happens because, when a new valid block is found, all participants reset their PoW puzzles to commit to it with a (new) set of valid transactions and begin mining again.

Honest nodes accept new blocks if (1) the PoW puzzle solution is valid, (2) the set of transactions to be processed is valid with respect to the provided block, and (3) the block builds on top of the honest node's canon chain. Honest nodes will be able to pick which chain they consider canon by looking at the quality score of the chain's latest block. Each block's quality score is obtained as the sum of the total number of valid PoW solutions in the block's ancestral chain, and is a measure of computational power used to generate this sequence of blocks. By picking the chain with the highest quality score, honest miners will always pick to extend the block for which the maximal amount of computation has been performed. The consensus security parameter q here is the number of valid blocks in the chain (or its quality), with splits between the consensus states of honest nodes happening with probability negligible in q. This holds only when more than half of the computational power is honest, as each subsequent block has probability $p < 1/2$ of being added by a Byzantine attacker. As q increases, the probability that an attacker will provide more

PoW solutions than the valid chain decays exponentially to zero. We refer to this as the 'longest' (or 'heaviest') chain quality-update rule.

Finally, the difficulty of the PoW puzzle is also modified every fixed number of blocks in order to ensure a fixed block frequency over time. This is done by time-stamping blocks and altering the difficulty level proportionally to ensure fixed block times. More specifically, every 2016 blocks, the average time between blocks is calculated and, based on this value, the difficulty d of finding a PoW solution is scaled proportionately to ensure that the system maintains a fixed 10-minute block time. This process works because the rate at which PoW solutions are found scales with 2^d, and so if blocks are becoming more frequent (which means more miners are joining the network), the difficulty parameter will increase until block frequency returns to its previous value.

2.2.2 Related Work

Formalizations of NC have been studied in a variety of contexts, identifying and optimizing trade-offs between the system's parameters and security properties. Prior work [Miller and LaViola Jr 2014] has provided a computational model in which the fundamental properties of NC are characterized, demonstrating that it achieves optimal resilience in the Random Oracle (RO) model for any moderately hard puzzle. This also formalizes the idea that a scalable consensus protocol has time and memory complexity requirements that are independent of the network's size.

Studies of NC's robustness against different adversarial mining strategies have also been performed. [Nakamoto 2008] mentions that miners are incentivized to behave honestly, through incentive compatibility with the design of the protocol. [Kroll et al. 2013] show that the Bitcoin reference implementation is a Nash equilibrium with respect to honest miner incentives in the setting of perfect information, although different equilibria are also shown to exist. For adversarial miners controlling more than a third of computational resources, [Bahack 2013; Eyal and Sirer 2014] present block withholding attacks, where the adversary withholds blocks they find and attempts to force a change of the canon chain by building faster on their private fork.

Extensions and improvements to NC have also been explored, with the works of [Sompolinsky et al. 2016; Sompolinsky and Zohar 2018] providing better throughput-security trade-offs by changing the way honest nodes represent previous transactions and choose which canon chain to follow. The work of [Eyal et al. 2016] proposes a higher-throughput 'batched' version of NC, wherein PoW solutions are required only every k blocks. Although it comes with specific attacks and a decrease in consensus security, this model was built upon by later works [Kogias et al. 2016; Pass and Shi 2016] aiming to decouple consensus from transaction processing. Recent works [Bagaria et al. 2019; Wang et al. 2020] have demonstrated that optimal consensus protocols satisfying the properties of NC can be constructed with respect to network capacity C and propagation delay Δ. In [Bagaria et al. 2019], the authors use the existence of a PoW puzzle to design a consensus algorithm which has optimal resilience $2t + 1$, optimal throughput up to capacity C, latency that scales with delay Δ, and which enforces an eventual ordering of all processed transactions with high probability.

2.3 Moderately Hard Puzzles

The PoW process in Bitcoin and most modern cryptocurrencies is based on HashCash [Back 2002] and involves solving a hard puzzle for which the difficulty can be adaptively set according to the number of participants. Hardness here is taken to mean that no adversary can compute solutions to the puzzle faster than randomly guessing. An important property of such systems is that they are *memoryless*, or that the probability of winning does not depend on time spent computing a solution. It is important to ensure that the PoW process is fair, meaning that a miner's hashrate (or puzzle-solving rate) is directly proportional to their computational power and hence that large miners do not enjoy algorithmic efficiency gains with growth. This is necessary to ensure that the network remains decentralized; without this property there would be an algorithmic incentive for miners to consolidate. Of course, there may be economic and logistic incentives for miner

consolidation (e.g. reduced administrative overhead) but we consider these out-of-scope.

Formally, a moderately hard (or, equivalently, PoW) puzzle is defined as a tuple PoW satisfying a set of efficiency and security properties. The definition below is adapted from [Ball et al. 2017a]:

Definition 2.1. The tuple $\mathsf{PoW} := (\mathsf{Gen}, \mathsf{Solve}, \mathsf{Verify})$ is a moderately hard puzzle if it satisfies all the following properties:

1. **Computational Efficiency:**
   - $\mathsf{Gen}(1^\lambda)$ runs in time $O(\lambda)$,
   - For any $c \leftarrow \mathsf{Gen}(1^\lambda)$, $\mathsf{Solve}(c)$ runs in time $t(\lambda) = \omega(\lambda) \in \mathsf{poly}(\lambda)$,
   - For any $c \leftarrow \mathsf{Gen}(1^\lambda)$ and π, $\mathsf{Verify}(c, \pi)$ runs in time $O(\lambda)$.
2. **Completeness:** For any $c \leftarrow \mathsf{Gen}(1^\lambda)$ and $\pi \leftarrow \mathsf{Solve}(c, \pi)$, $\Pr[\mathsf{Verify}(c, \pi) = 1] = 1$.
3. **Amortization Resistance:** For any polynomial $\ell(\cdot)$, constant $\epsilon > 0$ and adversary $\mathcal{A}$ running in time $\ell(\lambda) \cdot t(\lambda)^{1-\epsilon}$, the following is negligible in λ:

$$\Pr\left[\forall i \in [\ell(\lambda)], \mathsf{Verify}(c_i, \pi_i) = 1 \;\middle|\; \begin{array}{c} \{c_i\}_{i=1}^{\ell(\lambda)} \leftarrow \mathsf{Gen}(1^\lambda) \\ \{\pi_i\}_{i=1}^{\ell(\lambda)} \leftarrow \mathcal{A}(1^\lambda, \{c_i\}_{i=1}^{\ell(\lambda)}) \end{array}\right].$$

In the Bitcoin protocol, the PoW process is defined with respect to a random oracle O instantiated by the SHA-256 hash function. A valid solution to some puzzle c is a proof π for which the iterated output $\mathsf{out} := O(O(c, \pi))$ has d leading zeroes. Since we are working in the RO model, this process can only find solutions with repeated queries of (c, π^*) to O, thus ensuring hardness. Moreover, the random variable $B \in \{0, 1\}$ representing whether an unchecked pair (c, π) is a valid PoW solution has success probability $p := 2^{-d}$, and therefore the probability that we get X valid solutions out of $n := \ell(\lambda)$ different pairs (c_i, π_i) is Binomially distributed as $X \sim \mathsf{B}(n, p)$. As the number of miners n solving the PoW puzzle increases, the difficulty update mechanism increases

d (thus decreasing p) so that np converges to some constant μ. This is equivalent to requiring fixed block times on average, as a valid solution here corresponds to a new valid block. In the limit of $n \to \infty$, the update rule implies that $p \to 0, np \to \mu$, and by the Poisson limit theorem, $X \sim \text{Po}(\mu)$. We can thus approximate the Bitcoin PoW process by a Poisson random variable of rate μ, given that the number of miners and difficulty are sufficiently high.

2.3.1 Useful Proof-of-Work

It has long been an open challenge to design a PoW puzzle that is both suitable for NC and also *useful* for some independent purpose [Bonneau et al. 2015]. In addition to being memoryless, the puzzle must satisfy several other properties, such as derivation from public parameters and support for fine-tuned difficulty adjustments. Early candidates for useful PoW puzzles were highly structured problems, such as finding long Cunningham chains of related prime numbers [King 2013] or tables of relations for solving discrete log computations [Hastings et al. 2018]. Other PoW schemes have also focused on altering the task performed or rewards received [Król et al. 2019], each demonstrating different ways to utilize the underlying computational task for something useful: such as proofs-of-space or file retrievability [Miller et al. 2014; Sengupta et al. 2016; Zhang et al. 2017] or proofs-of-replication [Fisch 2019]. Trusted Execution Environments (TEE) such as Intel's Proof-of-Elapsed-Time (PoET) [Chen et al. 2017] have been developed to simulate the PoW procedure for efficiency gains, while others [Daian et al. 2017] propose a design that allows miners to reuse computation by controlling how much work is delegated to workers.

The work of [Bitansky et al. 2016] shows how PoW puzzles can be developed from cryptographic assumptions, namely worst-case hard non-amortizable problems with succinct randomized encodings. In [Ball et al. 2017a], the authors develop a framework to define PoW using worst-case fine-grained hardness assumptions. The work of [Ball et al. 2017b] builds on this by embedding specific NP–hard problems as PoW puzzles, showing non-amortizability results for certain languages.

3 | Distributed Payment Systems

3.1 Preliminaries

We model the processing of payments using a state machine. A state machine is defined by an initial state, a set of possible states, and a state transition function which governs the transition from one state to another given some information as input. Moreover, we work under the assumption that this is a *replicated state machine* (RSM), with local copies of the state machine in each node so as to achieve fault tolerance.

We define our payment system state machine as follows: we have a set of participants who share a broadcast communication channel, and who may join or leave the system at will. There are two types of nodes we concern ourselves with here: miners and light clients.

Miners: A mining (or full) node has access to the current state $\mathcal{S}_i \in \mathcal{S}$ at timestep i, performing any consensus-specific computation and verifying state transitions.

Light Clients: Light clients (or end-users) can issue transactions $t \in \mathcal{T}$ and verify their inclusion, but do not need to keep mutable state.

We investigate how the system transitions from $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$ while retaining consensus over state. Transitions between states happen through the processing of transactions by a model-specific

transition function NewState. We also require a transition validation function VerifyState that ensures the state update was done correctly. By defining the notion of *validity* between state transitions, we differentiate between legitimate and illegitimate transactions and only permit processing of the former. Moreover, we require that such tuples are also internally consistent, namely that all new states are correctly validated. For example, the Bitcoin and Ethereum protocols both define their own transition functions between blocks (states) and each one is based on its own notion of transaction validity.

Definition 3.1. A tuple of efficiently computable algorithms (VerifyState, NewState) is considered a *transition* tuple if the following conditions hold:

- $\mathsf{VerifyState} : 2^{\mathcal{T}} \times \mathcal{S} \times \mathcal{S} \times \{0,1\}^* \rightarrow$Yes/No
- $\mathsf{NewState} : 2^{\mathcal{T}} \times \mathcal{S} \times \{0,1\}^* \rightarrow \mathcal{S}$

and moreover we consider such a tuple *consistent* if $\forall \mathcal{S}_i, \mathcal{S}_{i+1} \in \mathcal{S}, \mathbf{t} \in 2^{\mathcal{T}}$:

$$\exists z_i \text{ s.t. } \mathsf{VerifyState}(\mathbf{t}, \mathcal{S}_i, \mathcal{S}_{i+1}, z_i) = \text{Yes} \Leftrightarrow \mathsf{NewState}(\mathbf{t}, \mathcal{S}_i) = \mathcal{S}_{i+1}.$$

$\Sigma^n = (\mathcal{S}_i, \mathbf{t}_i, z_i)_{i=1}^n$ is *valid* with respect to (VerifyState, NewState) if $\mathsf{VerifyState}(\mathbf{t}_i, \mathcal{S}_i, \mathcal{S}_{i+1}, z_{i+1}) =$ Yes, or equivalently $\mathsf{NewState}(\mathbf{t}_i, \mathcal{S}_i) = \mathcal{S}_{i+1}, \forall i \in [n]$.

The above notion can be used to define the minimal semantics for a model DPS, where $2^{\mathcal{T}}$ refers to the power set of $\mathcal{T}$. Note that the above does not imply deterministic state transitions. In addition, we associate the monetary value $c \in \mathbb{N}$ of each account with user address values $z \in \mathcal{Z}$, of which there can be multiple in a given state. This provides us with all the ingredients needed to define the fundamental system.

We require a theoretical model for a distributed payment system (DPS), defined as a tuple of algorithms necessary for minimal payment functionality. Many subsequent and concurrent

works have focused on developing various DPS architectures, each depending on a different set of trade-offs and desirable protocol properties. For us, the structure of the DPS is not the main goal, so we opt for working with a minimal design. We restrict ourselves to a simple construction based on a standard approach, which we fully specify in Appendix A.2. In terms of security, the system needs to provide both completeness and correctness guarantees. This requires that the protocol should guarantee that state transitions considered correct by VerifyState will not be rejected by compliant nodes. Similarly, satisfying correctness requires that transactions and state transitions that are invalid should not be accepted by compliant nodes. These definitions are constructed in the usual way in the auxiliary supportive materials.

Our model can easily be adapted to describe existing blockchain-based payment systems. We illustrate this informally for Bitcoin (in its original form) to provide intuition for what the essential components of a distributed payment system are.

Bitcoin: The Bitcoin protocol is a UTXO-based payment clearing system, for which a valid block update includes a set of valid ordered transactions and specific block header information. The components of the RSM are illustrated below:

- **State:** The list of all UTXOs.
- **Witness:** Not required; validation happens by inspection of the ledger.
- **NewState:** Generation of a new block.
- **VerifyState:** Validity of a block transition requires:
  - Verifying all UTXOs exist in state.
  - Verifying that the header is well formed.
  - Checking the nonce satisfies proof-of-work.
  - Ensuring all transactions are valid.

A similar treatment would allow us to characterize Ethereum using the same basic components. This paradigm also makes obvious that, in order to verify the state of the *whole* system without any external information, we would need to iteratively validate each state transition. We use the witness z_i to provide 'hints' to the validation function, which we will demonstrate later allows us to construct protocols tailored for much more efficient state verification.

3.2 Maximizing Throughput

Given the set of constraints surrounding the design of a DPS, one of the most pressing concerns preventing large-scale commercial applications is the need for consistently high transaction throughput rates. More specifically, the number of transactions processed per unit of time is orders-of-magnitude below what is currently attained by retail payment solutions. This issue is critical for adoption, since a robust settlement layer would need to withstand seasonal adjustments to transaction processing demand while keeping transaction fees low enough to be competitive. However, an increase in transaction throughput comes with design trade-offs that prevent this from being achievable through protocol reparametrization.

A natural attempt at optimizing transaction throughput in this way is to increase either the number of transactions per block or the block issuance frequency. Both values are linearly correlated to transaction throughput, so any substantial increases would resolve the above problem. However, increasing the number of transactions per block implies a proportional increase in block size, which would need to be disseminated through the gossip network at the same rate. This adds a bandwidth burden to the network, and comes with an increased probability of chain forks, and thus a corresponding decrease in the security of the system's consensus.

To specify the exact trade-off between these two variables, it suffices to note that the underlying gossip network has an inherent latency that cannot be removed. This means that when a new block is issued there exists some minimal time parameter Δ which lower bounds the time

until all nodes receive the new block. As Δ gets larger, the probability that two different block sequences building on the same parent (a fork) grows. This is because the increased bandwidth costs due to the larger block size correspond to a larger probability a new block will be found before the previous one was disseminated to the whole network. It is easy to see that this issue still exists if we instead opt for increasing block frequency.

This constraint was formally analyzed in [Sompolinsky and Zohar 2015], where the authors use the empirical analysis of [Decker and Wattenhofer 2013] to notice that $\Delta(b) = \Delta_0 + b \cdot \Delta_b$, where b the size (in bits) of the block being propagated and Δ_0, Δ_b the propagation delay (in seconds) and bandwidth capacity (in seconds per bit) of the underlying gossip protocol respectively. If there exists some honest sub-network in the topology of all participating nodes that solves PoW puzzles at a rate of $\alpha \cdot \lambda$ with delay $\Delta(b)$, then for any $P \in (0, K/\Delta_b)$, the protocol can achieve a throughput of P transactions per second at a security threshold of at least $\alpha' := \alpha \cdot \left(1 - P \cdot \frac{D_b}{K}\right)$ for appropriate choices of λ, b. This means that, in the best case, an adversary would require an $\alpha' < \alpha$ portion of total computing power to grow the chain faster than the honest nodes and break consensus. The trade-off between throughput P and security α' here is exactly due to the aforementioned effects of increasing block frequency/size in the presence of network latency.

3.2.1 Segregated Witness

The constraint identified above means that in order to increase throughput without a tradeoff with security, we need to increase the amount of transactions per block without increasing block size. Therefore, succinct representations of the transaction set $\mathbf{t}$ that still contain the minimal amount of information needed to verify the state transition $\mathcal{S}_i \rightarrow \mathcal{S}_{i+1}$ would provide corresponding increases in throughput. This optimization was proposed and implemented in Bitcoin through BIP 141, and on a technical level decoupled the transaction set $\mathbf{t} = (t_1, ..., t_n)$ into statement witness pairs $t_i = (\phi_i, w_i)$, where ϕ_i the list of inputs to the transaction and w_i the witness elements required to prove its validity. This change, along with modifications to how transaction

weight was counted towards block capacity (which in practice led to a slight increase in block size), provided increases in transaction throughput.

However, this approach reaches a practical limit at a minimal representation of $\mathbf{t}$ for which the aggregation of all ϕ_i, w_i cannot be done in less space without affecting transaction validation times. Indeed, a very succinct representation (ϕ, w) for some transaction would require the full node to perform substantial computation in order to ensure block validity, burdening the network with an increase in the latency Δ due to the heavier computational burden of verifying incoming transactions. One promising approach to minimize this problem involves verifiable computation, in which the computationally intensive procedure of providing a proof of validity π can be performed once by a 'prover' and the subsequently validated quickly by everyone else. This means that each block could provide a minimally sized witness element for the whole block alongside a proof of state validity for the given state transition.

3.2.2 Transaction Rollups

If we extend the model to include transactions that encode arbitrary compute (or 'smart contracts' enforcing a specific computation), then new options for scaling become feasible. Returning to the above setting, we can encode each transaction as a minimally-sized statement witness pair $t_i = (\phi_i, w_i)$ in order to reach the theoretically maximal transaction throughput possible at the given security level (fixed by block time). If we can cheaply verify that each transaction is correct using a small-enough proof π_i, then a smart contract verifying π_i with access to (ϕ_i, w_i) is enough to ensure soundness. Moreover, if we can batch the validity verification of multiple transactions into one equally-sized proof π, then verifying a block on-chain would only require access to π and each of the included $t_i = (\phi_i, w_i)$. This is because we can perform the verification of π using a smart-contract on-chain if it has access the above data. This process is known as a roll-up architecture, since the proofs of validity of multiple transactions are 'rolled up' into one proof. Note that, modulo the security assumptions of the proof system used to implement

the construction of π, this approach retains the same soundness guarantees as the DPS threat model. Critical to this is the availability of all witness information, so that all parties are able to reconstruct the full transcript of state transitions from the same trust assumptions. Although this fact lower-bounds the total number of transactions per unit of memory we can achieve with rollups that have full data availability, it does not preclude order-of-magnitude improvements to throughput in comparison to current architectures. Of great importance is the choice of model through which the proofs π are generated.

Instead of attempting to change the properties of the consensus protocol and chain parameters, a more general approach is to build a second protocol operating on top of the base layer, but which has different properties that permit larger transaction throughput. These are denoted by Layer 2 (L2) solutions. We identify two approaches to resolving the above problem on L2: Optimistic Rollups (OR) [Adler and Quintyne-Collins 2019] rely on fraud proofs, which are generated by nodes when they spot an invalid transaction being processed in the rollup. This is because all transactions are presumed valid unless such a proof can be provided, relying on miner incentives to generate the fraud proofs. Although capable of achieving extremely high throughput values at low cost, this model differs fundamentally from our model, mainly due to incentive alignment being necessary for transaction validity.

A second approach [Buterin V.] involves leveraging verifiable computation, and more specifically succinct non-interactive proofs, in constructing π. These are known as 'zero-knowledge rollups' or ZK-Rollups and, although more complex on a design level, rely only on the security of the proof system used to create π. Note that 'zero-knowledge' here is a misnomer, since the proofs only require non-interactivity and succinctness to be suitable. Multiple approaches to realizing this on Ethereum are currently underway, with implementations differing around the choices of proof system, transaction semantics, and verifiable computation to be performed. Some efforts have focused on building Virtual Machines (VMs) for more 'proof friendly' state transition functions, while others have attempted to realize the full Ethereum VM (EVM) in their proof imple-

mentations. Although all approaches focus on resolving the underlying throughput optimization problem we identify above, there are many design choices when adapting such a system to an already existing DPS implementation that can affect its performance and fault-tolerance guarantees. Note that although the rollup approaches cited above are L2 solutions, the optimization itself can also be applied directly on Ethereum's semantics if we can provide proofs for EVM computations. This, however, would require a hard-fork.

3.2.3 Other Scaling Approaches

The Lightning Network (LN) [Poon and Dryja 2015] is the largest deployed L2 on Bitcoin to-date. LN relies on the creation of channels between participants, which allow them to perform large amounts of computation off-chain. The main differences rely on the need for some amount of interactivity between users, along with the need to 'lock up' coins in order to use the service. Another idea to leverage off-chain compute in the smart-contract setting is *state channels* [Dziembowski et al. 2018], in which Bob puts some amount of coin in a smart contract, and proceeds to pay Alice off-chain from this deposit by signing spend messages with his public key. When Alice wants to get paid, she will sign and publish the latest such message, which will be verified by the smart contract that disburses the money. This works if Bob is able to initiate a withdrawal after a specified time period, in the case that Alice is malicious or suffers a fault during computation. Although state channels are powerful, there are limits to their feasibility. For example, they cannot be used where one represented party is a smart contract, nor can they be deployed in large scale applications without a substantial amount of locked-up capital. A similar idea was proposed in [Poon and Buterin 2017] for the Ethereum protocol, wherein a 'blockchain tree' would be kept that differentiated 'child' from 'parent' chains such that child-chain transactions are substantially faster. We should note that all these approaches require additional trust assumptions, and that the ZK-Rollup approach is the only one which does not while having the promise of practicality.

3.3 Light Client Verification

The idea of providing portions of the blockchain to light clients for verification began with Bitcoin, where Simple Payment Verification (SPV) clients download only block headers and Merkle inclusion proofs for specific transactions to be convinced of their validity. While this approach forgoes downloading the whole blockchain, clients must trust that the downloaded blocks contain only valid transactions due to incentives provided to miners which discourage mining invalid blocks. This approach also still requires a linear amount of memory with respect to chain length. For Bitcoin downloading only block headers requires about 10 kB per day in bandwidth, which is reasonable for up-to-date clients but non-trivial for new clients which must download the entire chain of headers (currently about 40 MB and growing). For Ethereum this is much worse, with daily bandwidth requirements reaching 5 to 8 GB to download the block headers alone.

Sublinear memory complexity in SPV clients through skip-lists was first formally analyzed in [Kiayias et al. 2016, 2017; Karantias et al. 2019], where the authors propose keeping pointers to multiple previous blocks at every step to allow for fast verification. This allows the protocol to check for high-difficulty previous blocks (or 'superblocks'), of which verifying a logarithmic number suffices to ensure security for the whole chain. This approach, however, is only feasible in the regime of fixed difficulty and thus cannot be implemented as is.

Flyclient [Bünz et al. 2019c] also guarantees logarithmic complexity for transaction and proof-of-work verification, and is secure with high probability under variable difficulty even if fractions of the network are adversarially controlled. This is achieved by using Merkle Mountain Range Commitments to achieve memory improvements, and a random block sampling protocol to ensure security. However, Flyclient still requires resources linear with respect to each new transaction, and storage requirements still grow with blockchain size. Neither the work of Kiayias et al. or Flyclient enable efficient verification of transaction validity, both relying on the same argument from Bitcoin that economic incentives discourage mining a long chain of blocks containing

incorrect transactions.

By contrast, while Mimblewimble [Poelstra 2016] does not provide sublinear verification guarantees with respect to block header size h, it contributes an innovative framework for transaction verification. By compressing state in a 'UTXO set' of $u \leq t$ transactions that adaptively updates with each block, it provides asymptotically better transaction verification. In Mimblewimble, history verification is linear only in the number of currently unspent coins, not the total number of transactions. While we could theoretically adapt these techniques in our work to decrease proving costs, we choose not to given practical observations that the number of unspent coins is not much smaller than the total number of transactions. The more complex predicate to verify state in MimbleWimble would likely negate any gains in proving time from compressing some transaction history.

The above constructions can all be characterized as 'Ultralight' clients, for which verification costs are sublinear with respect to the underlying statement. In this work, we will use verifiable computation (c.f. Chapter 4) to construct an asymptotically optimal Ultralight client. This approach was subsequently generalized [Chen et al. 2020] for arbitrary transition functions. Further work [Gabizon et al. 2020] has also provided a performant Ultralight client for the Celo [Kamvar et al. 2019] blockchain that utilizes the same insights. As will be discussed in Section 5.2, this approach has also been adopted by the Mina protocol [Bonneau et al. 2020], although with changes to the consensus protocol. Finally, Vault [Leung et al. 2018] uses an alternative consensus mechanism and decouples recent transaction storage from account state in order to achieve similar light-client verification guarantees without the use of verifiable computation.

4 Verifiable Computation

4.1 Zero Knowledge Proofs

Zero-knowledge proofs [Goldwasser et al. 1989] have recently received increased amounts of attention for providing efficient verification while maintaining small proof sizes, even in the case of complex predicates. Initially limited to theoretical considerations, such proof systems have lately come to encompass the underlying technology in a wide variety of practical and industrial applications with delicate trade-offs between privacy and system security [Boneh et al. 2018; Chaidos et al. 2016; Setty et al. 2018]. In this work, we are interested in applications for which there is limited space availability in the underlying system, and thus for which minimal proof size is an important property. Moreover, we ideally want to focus on applications for which there exist no trusted parties at any point of the computation, and thus hope to achieve proof size minimization *without* compromising the trust model of the system.

The above motivation is most closely associated with applications of zero-knowledge proofs to cryptocurrency systems, such as Ethereum [Wood et al. 2014] or ZCash [Miers et al. 2013], in which participants have to verify state (or transaction) validity to ensure system soundness but for which there is limited space available in which to do so. Bridging the gap between these two requirements will allow for not only efficient but also trustless verification of state transition in such systems. This has the potential for scaling improvements, such as increased transaction throughput or better privacy guarantees.

The most widely used proof systems for such an application are preprocessing Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) [Groth 2016; Micali 2000; Gentry and Wichs 2011], for which proof size and verification time are polylogarithmic in the size of the circuit being verified. 'Preprocessing' here denotes that such systems rely on a one-time (often expensive) setup procedure to produce a proving/verification key-pair (pk, vk) (known as a Structured Reference String or SRS) that is used in all subsequent computation. The most efficient such construction is due to [Groth 2016] and achieves constant proof size consisting of 3 group elements, with state-of-the-art proving time.

A SNARK for relation generator $\mathcal{R}_\lambda$ (from which relation R is sampled) is comprised of the following four algorithms:

1. **Setup:** $\mathcal{G}(\mathcal{R}) \rightarrow (\tau, \mathsf{SRS} := (pk, vk))$. Key generation takes as input a relation $\mathcal{R}$ and a security parameter λ, outputting proving and verification keys as the SRS and trapdoor τ.

2. **Prover:** $\mathcal{P}(pk, \mathcal{R}, (x, w)) \rightarrow \pi$. The prover takes as input the proving key pk, relation $\mathcal{R}$ and a statement-witness pair (x, w), outputting proof π that $\mathcal{R}(x, w) = 1$.

3. **Verifier:** $\mathcal{V}(vk, \mathcal{R}, x, \pi) \rightarrow$ Yes/No. When given as input the verification key vk, relation $\mathcal{R}$, a proof π and statement x the verifier outputs Yes if π is a valid proof that x is valid, outputting No otherwise with high probability.

4. **Simulator:** $\mathcal{S}(\mathcal{R}, \tau, x) \rightarrow \pi$ The simulator takes as input the relation $\mathcal{R}$, trapdoor τ and statement x, outputting a proof π that x is valid.

The main security properties that SNARKs satisfy are completeness, knowledge soundness and zero-knowledge and are defined below. For some relation $\mathcal{R}$, the set of statements with a witness such that $\mathcal{R}(x, w) = 1$ is known as the language $\mathcal{L} := \mathcal{L}(\mathcal{R})$ defined by $\mathcal{R}$.

1. **Perfect Completeness:** An honest prover always convinces an honest verifier. For all

(x, w) such that $\mathcal{R}(x, y) = 1$ for $\mathcal{R} \leftarrow \mathcal{R}_\lambda$ and all $\lambda \in \mathbb{N}_+$:

$$\Pr\left(\mathcal{V}(vk, \mathcal{R}, x, \pi) = 1; \pi \leftarrow \mathcal{P}(pk, \mathcal{R}, (x, w)), (pk, vk) \leftarrow \mathcal{G}(\mathcal{R})\right) = 1.$$

2. **Perfect Zero-Knowledge:** For all sampled $(\mathcal{R}, z) \leftarrow \mathcal{R}_\lambda$, all $\lambda \in \mathbb{N}_+$ and valid statement-witness pairs (x, w), for all adversaries $\mathcal{A}$:

$$\Pr\left(\mathcal{A}(\mathsf{SRS}, \tau, \mathcal{R}, z, \pi) = 1; \pi \leftarrow \mathcal{P}(pk, \mathcal{R}, (x, w)), (\mathsf{SRS}, \tau) \leftarrow \mathcal{G}(\mathcal{R})\right) =$$

$$\Pr\left(\mathcal{A}(\mathsf{SRS}, \tau, \mathcal{R}, z, \pi) = 1; \pi \leftarrow \mathcal{S}(\mathcal{R}, \tau, x), (\mathsf{SRS}, \tau) \leftarrow \mathcal{G}(\mathcal{R})\right).$$

3. **Computational Knowledge Soundness:** For all non-uniform polynomial time (PT) adversaries $\mathcal{A}$, there exists a PT extractor $\mathcal{E}$ with access to the random coins and state of $\mathcal{A}$ for which the following is approximately zero:

$$\Pr\left(\mathcal{R}(x, w) \neq 1 \text{ and } \mathcal{V}(vk, \mathcal{R}, x, \pi) = 1; (x, w, \pi) \leftarrow \mathcal{E}(\mathsf{SRS}, \mathcal{R}, z), \mathsf{SRS} \leftarrow \mathcal{G}(\mathcal{R}), (\mathcal{R}, z) \leftarrow \mathcal{R}_\lambda\right).$$

In the following sections, the above security properties will be used and extended as required. We note that the notion of a predicate, relation and circuit all refer to the same concept: that of encoding a statement-witness pair as an arithmetic circuit in order to check whether it satisfies a given statement. We will be using these notions interchangeably.

4.2 Transparency

Most designs in the literature [Gennaro et al. 2013; Setty et al. 2013; Ben-Sasson et al. 2014] rely on a *trusted setup*, or a trusted $\mathcal{G}$ that generates a trapdoor (known as 'toxic waste') that should be destroyed in order for the system to retain its security guarantees. Such a security lapse would

be grave for all aforementioned applications. For example, in a cryptocurrency system such as ZCash an adversary possessing such waste would be able to spend non-existent tokens without being found. An adopted approach to mitigating this issue involves Multi-Party Computation, in which a single participant needs to destroy their parameters for security to hold [Bowe et al. 2017]. However, scaling such an approach to many participants comes with its own challenges, and can never reach be completely trustless.

The trust issue inherent in the above approach stems from the requirements for the generation of the SRS of the proof at the preprocessing stage. This is done once at the beginning of the protocol, encoding information that is used in the subsequent proof generation of any input arguments. More specifically, in most SNARKs (such as [Maller et al. 2019]) the trusted part of SRS generation stems from the usage of a polynomial commitment scheme that needs to sample (secret) randomness in order to provide commitments to some low-degree polynomial that in turn encodes the circuit in question. That information is then used by the prover to efficiently convince the verifier that a given value is indeed the evaluation of this polynomial, thus proving knowledge of the statement. Such systems use the polynomial commitment scheme of [Kate et al. 2010], from which the above trust model is derived.

In attempting to retain a trustless (or 'transparent') threat model, the main design challenge lies in the efficiency of the underlying protocol. Various threads of work in this domain have achieved different efficiency trade-offs. The work of [Goldwasser et al. 2015] produces proofs with size scaling as $O(d \log T)$, while the proofs in [Wahby et al. 2018] scale with $O(d \log G)$ where T, d and G the size, depth and width of the circuit respectively. Succinct Transparent ARguments of Knowledge (zk-STARKs) [Ben-Sasson et al. 2019a] achieve $O(\log^2 T)$ proof sizes for uniform (layered) circuits. However, in the context of *universal* SNARKs (arbitrary circuits), existing proof systems suffer from performance overheads with respect to preprocessing SNARKs such as [Groth 2016]. Some also require non-trivial circuit designs, similar to what is described in [Evgenya 2017]. Nevertheless, we should note that for the class of problems that can be efficiently

expressed as layered circuits, these proof systems may be more optimal than universal ones. Since we are also interested in verifier succinctness, transparent approaches such as [Bünz et al. 2018] do not suffice here due to the linear dependence between verification time and predicate size.

Below we informally describe the properties that an 'ideal' proof system should possess for satisfiability of a given circuit C, where $|C|$ denotes its size. The first three properties define what is known as a 'fully succinct' zk-SNARK:

- *Succinctness:* Verifier time and proof size are polylogarithmic in $|C|$.
- *Prover Efficiency:* Proving time is quasi-linear in $|C|$.
- *Transparent:* No trust assumptions are required.
- *Plausibly Quantum Resistant:* Not based on quantum-falsifiable assumptions.

4.3 Universality

A new approach to the above problem relies on creating a 'universal' SRS at the preprocessing phase, which can then be used in tandem with *any* possible predicate (or circuit). This has been the focus of many recent contributions (see [Maller et al. 2019; Xie et al. 2019; Setty 2020]) and most recently [Gabizon et al. 2019] that are also fully succinct zk-SNARKs in the above sense. The approach in such schemes relies on two main ingredients: (1) encoding the circuit satisfaction problem of the predicate in question as a property of some (low-degree) polynomial f, and then (2) committing to f using a polynomial commitment scheme. In all the above approaches, the polynomial commitment scheme in [Kate et al. 2010] is used due to its constant size complexity and efficient implementation. However, this is the only part in the protocol that introduces the trusted setup, as the setup phase in the scheme requires a trusted actor to create (and then destroy) a secret value that is only used in generating commitments.

4.4 Incrementally Verifiable Computation

We now briefly introduce Incrementally Verifiable Computation (IVC) [Valiant 2008], an efficient primitive instantiated using (preprocessing) SNARKs. Consider a set of system states $\mathcal{S}$ with initial state $\mathcal{S}_0 \in \mathcal{S}$. We denote the system's state transition function by UpdateState and construct a predicate Π_S that evaluates to 1 on input state $\mathcal{S}_{i+1}$ (or a commitment to it) if and only if there exists a valid transition from some $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$. A prover repeatedly applies state transitions on the initial state to acquire $\mathcal{S}_n$.

An IVC system allows a verifier that only sees (a commitment to) the last state $\mathcal{S}_n$ and a short proof π_n to be convinced that $\mathcal{S}_n$ is a valid system state, i.e. a state that can be derived from $\mathcal{S}_0$ by applying valid state transitions for all i in the chain. An IVC system is comprised of the following three algorithms:

1. **Setup:** $\mathcal{G}(\Pi_S, 1^\lambda) \rightarrow (pk, vk)$. Key generation takes as input a predicate Π_S, outputting proving and verification keys.

2. **Prover:** $\mathcal{P}(pk, \mathcal{S}_{i+1}, \mathcal{T}, \mathcal{S}_i, \pi_i, w) \rightarrow \pi_{i+1}$. The prover takes as input the proving key pk, state $\mathcal{S}_i$, a proof π_i that $\mathcal{S}_i$ is a valid state and a set of transactions $\mathbf{t} \in 2^{\mathcal{T}}$, outputting a proof π_{i+1} that $\mathcal{S}_i \rightarrow \mathcal{S}_{i+1}$ is a valid state transition.

3. **Verifier:** $\mathcal{V}(vk, B_i, \pi_i) \rightarrow$ Yes/No. When given as input the verification key vk, a proof π_i and a commitment B_i to state $\mathcal{S}_i$, the verifier outputs Yes if π_i is a valid proof that state $\mathcal{S}_i$ is valid, outputting No otherwise with high probability.

4.5 Polynomial Commitments

Polynomial commitment schemes allow for the efficient verification of the evaluations of some polynomial $f \in \mathbb{F}_p[X]$ at an arbitrary point in its domain. We start with the basic requirement for

a commitment scheme to commit to elements of a given polynomial. This will be provided as an oracle and we assume it's binding i.e. that the adversary will be able to forge commitments with negligible probability over the schemes security. A commitment scheme $\Sigma = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ is defined as follows:

- $\mathsf{Gen}(1^\lambda) \rightarrow \mathsf{pp}$ generates public parameters,
- $\mathsf{Com} : \mathbb{F}_{<d}[X] \rightarrow C$ generates commitment c to some f,
- $\mathsf{Open} : C \times \mathbb{F}_{<d}[X] \rightarrow \{0, 1\}$ checks validity of some commitment c with access to f.

We say that the tuple $(\mathsf{Gen}, \mathsf{Com})$ is ϵ-binding if there exists an Open function for which:

$$\Pr\left[\mathsf{Open}(\mathsf{Com}(f)) = 1 | \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)\right] = 1,$$

and for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\begin{array}{c|c} f \neq g, \mathsf{Open}(c, f) = 1 & (f, g, c) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{Open}(c, g) = 1 & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \end{array}\right] \leq \epsilon(\lambda).$$

Since the introduction of polynomial commitment schemes in [Kate et al. 2010], the first transparent such scheme was introduced in [Wahby et al. 2018] for multivariate polynomials, with $O(\sqrt{d})$ commitment size and verification complexity. Subsequent work in [Bünz et al. 2019b] introduces a scheme with $O(\mu \log d)$ size and verification complexity, where μ the number of variables of the polynomial in question and d the polynomial's degree. Even though the asymptotics of the approach in [Bünz et al. 2019b] suffice for the above motivation, the practical implementation of their system relies on cryptographic operations that are substantially more resource-heavy than previous approaches. This stems from the reliance of their system's security on class groups of unknown order. Although the proof sizes achieved are sufficiently succinct, this dependence could make practical deployment difficult at reasonable security levels when proof generation

time needs to also be substantially efficient. Moreover, the assumptions on which their construction rests are not quantum-resistant.

Part I

Proof of Necessary Work

5 | Contributions

In this work, our goal is to design a payment system supporting efficient verification of the system's entire history by any participant without trusting any third parties. Participants can join the system at any time and need only to obtain some fixed public parameters from a trusted source (e.g. the genesis block and the system's rules). Current systems such as Bitcoin require participants to process the entire system history to verify that the current state (the most recent block in the chain) is correct. This requirement makes joining the system prohibitively expensive for most clients, as downloading and verifying over 400 GB of system history (as of December 2020) takes days on an ordinary laptop. In practice, most clients instead rely on a trusted third party to assert the current state of the system.

We address this problem using succinct proofs of state validity. These enable clients to verify any snapshot of the system using minimal bandwidth and time, even if they have no other information except the genesis state and transaction validity rules. For any block in the system, these proofs demonstrate both that there exists a sequence of valid transactions from the genesis state $\mathcal{S}_0$ to the state committed in the current block, and that the block's *branch* (the sequence of predecessor blocks) is of quality q according to the consensus protocol. In this work we focus on aggregate PoW difficulty as the measure of branch quality, as used in Bitcoin consensus. Currently, systems such as Bitcoin or Ethereum require $O(t + h)$ work to completely verify a branch containing t transactions and h blocks. We are able to achieve optimal asymptotic performance of $O(1)$ verification costs for a client joining the system at an arbitrary point in its history.

Our techniques cannot help a client that is separated (or *eclipsed*) from the genuine system by a network partition. We assume a client can reach at least one node which will provide the most recent block and a proof. The client may also communicate with arbitrarily many attacker-controlled nodes; efficient verification means the client can quickly tell which block is canonical in the system.

Another major issue facing Bitcoin and related cryptocurrencies is energy usage. These systems employ PoW, which provides system security by publicly verifying energy consumption. This energy consumption, while necessary for the consensus protocol, is not used for anything else and hence is often described as wasted. We design a PoW puzzle which produces correctness proofs for each block as a useful byproduct, thus recycling some of the energy expended in achieving consensus. This requires carefully designing the proof-of-work to replicate the properties of Bitcoin's non-useful puzzle. Our main technical contribution is a method to deeply embed a nonce into the proof computation process, making it suitable as a (progress-free) PoW puzzle. We formalize this intuition by introducing the notion of ϵ-amortization resistance, and propose a protocol design based on this.

1. We design and prove the correctness and security of a protocol enabling *succinct state verification*. This ensures negligible computational requirements for any observer to verify the current system state.

2. We propose a variant of Nakamoto Consensus, which we call Proof of Necessary Work. This enforces computation of proofs of block/transaction validity as part of the consensus process, creating some useful work from the energy usage. We show security and in the process provide:

   (a) an upper bound for the honest prover based on its language's complexity, and

   (b) matching lower-bounds for a natural restriction to 'randomizable' languages.

Our results are based on the average-case hardness of multiexponentiation in the Generic Group Model (GGM) [Shoup 1997].

3. We implement the proof system in (1) with the consensus protocol variant in (2) at an 80-bit security level, benchmark its performance and establish feasibility. Our system:

   (a) produces block headers of size < 500 bytes for any number of txs/block,

   (b) allows stateless clients to verify a block in < 20ms, and

   (c) achieves throughput of 50 tx/block.

In terms of throughput and block header size, our prototype is about an order of magnitude worse than Bitcoin. Bitcoin block headers are 80 bytes and throughput is about 1,000 transactions per block. However, our system allows a stateless client to rapidly verify a block (and thus its complete history) in milliseconds with 500 bytes of data downloaded. In Bitcoin, a comparable full verification of a block requires many hours of computation time and downloading hundreds of gigabytes of data. The efficient block verification provided by our system does assist miners in quickly validating new blocks broadcast on the network, which may reduce the risk of block collisions and enable faster block frequency.

5.1 Incentivized State Compression

We identify a new approach to useful PoW by proposing that the work aid in the verification of the system itself. We denote this as *Proof of Necessary Work* and show how it can be used within a succinct blockchain architecture as a suitable PoW puzzle. A synergistic benefit of this approach is providing a direct incentive for hardware acceleration of zero-knowledge proofs, which could encourage the development of FPGA and ASIC designs for proof generation. This is relevant for many distributed payment systems whose underlying architecture depends on the generation of zero-knowledge proofs for the processing of transactions, such as [Bowe et al. 2018a; Bünz et al.

2019a; Fisch et al. 2018; Kamvar et al. 2019]. In all these systems, proof generation time is a critical bottleneck limiting transaction throughput and/or latency.

Improved proof generation times could yield order-of-magnitude improvements in not only latency and throughput but also energy cost per unit proof. Indeed, recent industry developments [Aleo 2022] based on our work have yielded interest in dedicating resources toward an industry-wide effort to maximize the performance of zero-knowledge proof systems. We believe this to be beneficial not only for distributed payments, but also for any application where high-throughput, low-latency and low-energy zero-knowledge proof generation is required.

5.2 Optimal Light Clients

Since we are interested in working with RSMs that facilitate state verification, we will also define the notion of "succinct verifiability". This restricts RSMs to be succinctly verifiable if the computational and memory resources they require to perform verification of the RSM's current state are small. Since in practice the size of the state of some RSM is extremely large (and grows with the number of processed transactions and blocks), any sufficiently efficient verification algorithm will need to take as input a "succinct representation" of the current state transition, while still being able to verify it. Otherwise, verification would require parsing $\mathcal{S}_i, \mathcal{S}_{i+1}$, which is prohibitively expensive. This is modelled as a function $\psi : 2^{\mathcal{T}} \times \mathcal{S} \times \mathcal{S} \to C$, which provides a commitment $c \in C$ that suffices for verification.

Definition 5.1. An RSM Σ^n with n state transitions is a tuple $\Sigma^n = (\mathcal{S}_i, \mathbf{t}_i, z_i)_{i=1}^n$ of states $\mathcal{S}_i \in \mathcal{S}$, sets of transactions $\mathbf{t}_i \in 2^{\mathcal{T}}$ (where $\mathcal{T}$ is the set of possible transactions), and witnesses $z_i \in \{0,1\}^*$. We denote $\mathcal{S}_n$ as the *current* state of Σ^n and $\mathcal{S}_0$ as its *genesis* state. Moreover, a valid RSM $\Sigma^n = (\mathcal{S}_i, \mathbf{t}_i, z_i)_{i=1}^n$ with respect to a consistent transition tuple (VerifyState, NewState) is considered *succinctly verifiable* if there exist $\psi : 2^{\mathcal{T}} \times \mathcal{S} \times \mathcal{S} \to C$ and SuccinctVerify : $C \times \{0,1\}^* \to$Yes/No

Technique	**Transaction Verif.**	**PoW Verif.**	**Memory Req.**
Bitcoin/Ethereum	$\Theta(t)$	$\Theta(h)$	$\Theta(h+t)$
Mimblewimble	$\Theta(u) = O(t)$	$\Theta(h)$	$O(\log^4(h))$
NIPoPoW	$\Theta(t)$	polylog (h)	$\log h \cdot (\log t + \log\log h)$
FlyClient	$\Theta(t)$	$O(\log^2 h)$	$O(\log^2 h)$
This work	$O(1)$	$O(1)$	$O(1)$

Table 5.1: Client verification times and memory requirements for t transactions in h blocks.

such that SuccinctVerify has $O(1)$ time and size complexity over $n, |\mathcal{S}_i|, |z_i|$ and:

$$\Pr\left(\mathsf{SuccinctVerify}(\psi(\mathbf{t}, \mathcal{S}_i, \mathcal{S}_{i+1}), z_i) \neq \mathsf{VerifyState}(\mathbf{t}, \mathcal{S}_i, \mathcal{S}_{i+1}, z_i)\right) \approx 0,$$

over the random coins of SuccinctVerify and ψ.

In Table 1, we provide a comparison of the asymptotic time and memory requirements of existing SPV protocols implementing transaction and/or proof-of-work verification. Given that transaction volume and chain length both grow linearly over time, we can ideally provide verification that is constant with respect to both. The only other work we are aware of with this goal is the Mina Protocol framework [Bonneau et al. 2020]. Mina takes a similar high-level approach as our work, encoding state transitions in a recursive proof system to asymptotically optimal verification time.

The two approaches are independent and vary in a number of technical details around predicate structure, with Mina choosing a different design for transaction proof aggregation. Most importantly, the main conceptual differences lie in our choice of consensus protocol. Mina implements a proof-of-stake [King and Nadal 2012] system, which must be carefully adapted for the succinct proof setting. By contrast, we implement a PoW system, which requires tackling an orthogonal set of design challenges to adapt to the succinct proof setting.

6 Succinct Verification

Here we demonstrate a specific instantiation of a DPS for which we define a transition function tailored to fast state verification by stateless clients. To achieve this, we leverage the capabilities of IVC systems and construct a succinct proof of state validity to represent each state transition. Since we will be basing our implementation of the proofs on SNARKs, we design the transition function so as to minimize SNARK proof sizes. This is critical for efficiency and feasibility.

Following the longest chain quality update rule defined in Section 2.2.1, our system updates the quality q of solving a PoW puzzle according to the depth of the chain. We are thus required to include (and commit to) q_i and n_i with every proof, where q_i is the quality of state $\mathcal{S}_i$ and n_i the associated nonce. This is because these quantities are needed by miners in order to follow the longest chain and achieve consensus.

6.1 Preliminaries

Each participant in our system has a public and secret key that they generate when they first join the network. The participants use these keys to digitally sign transactions and verify other participants' signatures. The state $\mathcal{S}_i$ contains the distribution of money between the participants (stored as a tree), state quality and a nonce corresponding to the most recent proof-of-work. We also distinguish between the i-th block, which in our case will be represented by a proof π_i that the i-th state transition is valid along with the set of transactions $\mathbf{t}_i$ corresponding to the transi-

tion, and *commitments* to state, which we denote by B_i and use for client verification. We require an account-based system (like Ethereum but not Bitcoin) and keep track of state with an 'Account Tree' of all account-value pairs. These building blocks are:

Account Tree: We use a Merkle tree construction with a compressible Collision Resistant (CRT) hash function $\mathcal{H} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$. We assume a fixed size tree T with height h throughout.

State: We denote S_i the state after the i-th update:

- Account tree T^i with leaves the lexicographically-ordered (by address) accounts in state.
- The block number i, quality q_i, and nonce n_i.

State Commitment: Set B_i as the commitment to S_i:

- The root rt_i of the Account tree T^i in $\mathcal{S}_i$.
- The block number i, quality q_i, and nonce n_i.

Protocol Initialization: Initially all accounts in the Account tree are set to null. In every transition, the tree allows the following modifications:

- *Account Initialization:* Set the public key to a non-null value and initialize the balance and the nonce. An account with a non-null public key is considered initialized. An account can be initialized only once. Uninitialized accounts have null public key.
- *Balance Update:* Modify account balance *bal*, ensuring money conservation.
- *Nonce Update:* Modify account nonce n to that of the current block.

We denote the initial state of the system (or "genesis state") by $\mathcal{S}_0$; this is agreed to by an out-of-band process. For example, a system might start with all addresses having a balance of zero or it might pre-populate some accounts with non-zero balance (colloquially known as "pre-mining").

Note that in the initial state, the Account tree is a full tree and contains one leaf/account for every address that can exist in the state. The genesis state can contain initialized and uninitialized accounts. All preliminary data structures have been included in the auxiliary supportive material.

6.2 State Transition Semantics

Below we define our semantics used for transaction and state transition validity.

Verifying Transactions: VerifyTx(t, T^i) $\rightarrow$Yes/No takes as input a transaction t and an Account tree T^i, outputting Yes/No (1 or 0). A transaction is considered valid if:

1. Sender and receiver are legitimate accounts in T^i.
2. Amount transferred is not more than sender's balance.
3. Signature authenticates over the sender's public key.
4. Sender and receiver accounts in the Account tree are updated correctly.
5. Recipient and Account public keys match, or the address is uninitialized.

Updating System State: UpdateState$(\mathcal{S}_i, \mathbf{t}, n) \rightarrow \mathcal{S}_{i+1}$ is a procedure that takes as input a state $\mathcal{S}_i$, am ordered set of transactions $\mathbf{t}$ with $|\mathbf{t}| = N$ and a nonce n. It outputs the next state $\mathcal{S}_{i+1}$ and a witness w of objects proving the update was done correctly. A transition is valid if:

1. All transactions in $\mathcal{T}$ are valid.
2. The previous state has performed proof-of-work.
3. Only last transaction t_N is of coinbase type.
4. Each transaction builds on top of the previous one; the first builds on the previous root.

6.3 State Transition as an NP statement

In order to instantiate a DPS that is capable of verifying a given state transition function, we encode the transition function ValidState as a compliance predicate Π_S. With every state transition, we include a proof that the transition was Π_S compliant. This is done by verifying the transition from the previous state and producing an attesting witness w in the process. In this context, we are interested in verifying the transition between two states of the Account tree by processing transactions between them into the system. This is achieved by tracking changes to the root rt_i of the Account tree after the input of each transaction.

We capture all requirements for transaction, proof-of-work and state validity in an NP language that only accepts commitments of the form $B_i = (rt^i, i, q_i, n_i)$ that build 'correctly' on top of a previous state. At a high level, the elements of this language are state commitments that, given some previous state's root, have only processed valid transactions.

6.3.0.1 Compliance Predicate

Given input $B_{i+1} = (rt_{i+1}, i+1, q_{i+1}, n_{i+1})$, the compliance predicate $\Pi_{\mathcal{S}}$ evaluates to 1 if and only if all of the following are satisfied:

1. Exists state $\mathcal{S}_i$ satisfying proof-of-work with nonce n_i and quality q_i.
2. Exists a tuple of ordered transactions $\mathbf{t}$ with $|\mathbf{t}| = N$. These transactions need to be sequentially valid with respect to $\mathcal{S}_i$.
3. $\textsc{UpdateState}(\mathcal{S}_i, \mathcal{T}, n_i) = \mathcal{S}_{i+1}$.

We use the compliance predicate $\Pi_{\mathcal{S}}$ to design an IVC system consisting of algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, where each message z_i is commitment B_i.

6.3.1 DPS Specification

Here we define how the system transitions from $\mathcal{S}_i \rightarrow \mathcal{S}_{i+1}$. Algorithm 1 generates a new state and associated proof of compliance, along with a nonce certifying that the system performed proof-of-work. When validating, we check that the new state $\mathcal{S}_{i+1}$ is a valid next state for the system by being (a) $\Pi_{\mathcal{S}}$ compliant and (b) providing proof-of-work. Note that the validation only requires the *root* of the Account tree corresponding to $\mathcal{S}_i$, thus making it efficient enough for light clients. A detailed specification alongside security definitions and proofs can be found in the attached auxiliary supportive material.

Algorithm 1 NewState

Require: $pp, \mathcal{T}, \mathcal{S}_i, \pi_i$
Ensure: $\mathcal{S}_{i+1}, \pi_{i+1}$
1: **procedure** NewState($pp, \mathcal{T}, \mathcal{S}_i, \pi_i$)
2: **if** $\mathcal{V}(vk, \mathcal{S}_i, \pi_i) = 0$ **then return** 0
3: **end if**
4: **while** $\mathcal{H}(\pi_{i+1}) > d$ **do**
5: Pick n_{i+1} uniformly at random
6: $(\mathcal{S}_{i+1}, w) \leftarrow$ UpdateState($\mathcal{S}_i, \mathcal{T}, n_{i+1}$)
7: $\pi_{i+1} \leftarrow \mathcal{P}(pk, \mathcal{S}_{i+1}, \mathcal{T}, \mathcal{S}_i, \pi_i, w)$
8: **end while**
9: **return** $(\mathcal{S}_{i+1}, \pi_{i+1})$
10: **end procedure**

When updating the state of the system, each participating miner receives π_{i+1} and $\mathbf{t}_{i+1}$. This allows them to update their own state to $\mathcal{S}_{i+1}$ and begin mining again. In Table 1, we provide a comparison of the asymptotic time and memory requirements of existing SPV protocols implementing transaction and/or proof-of-work verification. Given that transaction volume and chain length both grow linearly over time, we can ideally provide verification that is constant with respect to both.

7 | Proof of Necessary Work

To allow proof generation to serve as a proof-of-work puzzle, we require (a) a proof π_i whose generation algorithm $\mathcal{P}$ is moderately difficult to compute and (b) a proof-of-work puzzle $\mathsf{P}_V^{\mathcal{H},d}$ that requires the miner to fully recompute $\mathcal{P}$ to test a potential solution. The second property is necessary for the puzzle to be progress-free for fairness to miners of differing size. Indeed, if generating unique proofs π_i based on randomly sampled nonces n_i is sufficiently 'hard', then using $\mathsf{P}_V^{\mathcal{H},d}$ instead of a generic puzzle (such as computing the double SHA256 digest in Bitcoin) would allow us to not only perform proof-of-work with the same theoretical guarantees, but also compute a valid proof π_i in the process.

We do not formally analyze any consensus properties, since our goal is not to design a new consensus protocol but to retain that used by Bitcoin (and similar systems) and inherit its properties. However, we would like the work done to be useful by producing proofs of each block's validity. We introduce the notion of performing proof-of-work by proving the validity system state, denoted by *Proof of Necessary Work* (PoNW).

7.1 Definitions

We formalize this definition below, and provide the relevant security model.

Definition 7.1 (Proof of Necessary Work)**.** Given a pseudorandom function $\mathcal{H}$ and a proof $\pi_i \in \mathcal{Z}$ in some RSM with transition tuple $(\mathsf{NewState}, \mathsf{VerifyState})$, we define the *verification puzzle*

$\mathsf{P}_V^{\mathcal{H},d} : \mathcal{S} \times \mathcal{S} \times \mathcal{Z} \to \{0,1\}$ with difficulty d as the solution to the following function:

$$\mathsf{P}_V^{\mathcal{H}}(\mathcal{S}_i, \mathcal{S}_{i+1}, \pi_{i+1}) = \mathbf{1}\left[\begin{array}{c} \mathsf{VerifyState}(\mathcal{S}_i, \mathcal{S}_{i+1}, \pi_{i+1}) = 1 \\ \mathcal{H}(\pi_{i+1}) < d \end{array}\right],$$

where $\mathbf{1}[\cdot]$ is the indicator function.

By having access to a proof generating algorithm $\mathcal{P}(\mathbf{t}, \mathcal{S}_i, \mathcal{S}_{i+1}, n_i) \to \pi_{i+1}$ that generates unique (yet valid) π_{i+1} for each n_i, we can generate π_{i+1} for $\mathcal{S}_{i+1} = \mathsf{NewState}(\mathbf{t}, \mathcal{S}_i, \pi_i)$ using a uniformly randomly sampled n_i until the puzzle condition is satisfied:

$$\mathsf{P}_V^{\mathcal{H}}(\mathcal{S}_i, \mathcal{S}_{i+1}, \mathcal{P}(\mathbf{t}, \mathcal{S}_i, \mathcal{S}_{i+1}, n_i)) = 1.$$

Then π_{i+1} suffices for public verification that proof-of-work has been performed. This is because our prover will always fail with constant probability (when $\mathcal{H}(\pi_{i+1}) \geq d$), so iteratively sampling new proofs (by sampling new n_i) until a valid one is found can be shown, under the assumption that $\mathcal{P}$ is the most efficient way to find such an n_i, to be a memoryless exponential process and hence *fair*. Note that, by construction, we also guarantee that π_{i+1} is a valid witness for the RSM. The number of transactions verified is always *fixed* (with empty transactions still 'added') as otherwise miners would be incentivized to mine puzzles with the smallest blocks.

7.1.1 An Initial Approach

A natural thought would be to require the generation of proofs until $\mathcal{H}(\pi) < d$, as is proposed in the previous section. In the case that the proof is unique to the state and witness input, we can ensure that by adding a nonce in the input we will always get a different hash for π. However, this can lead to unfair outcomes. When computing π, the adversary can retain the parts of π that don't change between nonces and therefore substantially decrease proof generation time with

respect to other provers. This means the process is not memoryless, and so the fairness of the system is compromised.

7.1.2 Amortization Resistance

Like Nakamoto consensus, our puzzle needs the property that solutions are equally hard to test even after testing an arbitrary number of previous solutions. In other words, a miner should not be able to *amortize* costs while testing multiple potential solutions. This property is defined more formally below based on the μ-Incompressibility of [Miller et al. 2015], although we work in the bounded-size precomputation model. We model PoNW as a function $f^{\mathcal{O}}$ with limited access to some oracle $\mathcal{O}$ that performs a hard computation in an encoding of some group $\mathbb{G}$.

Definition 7.2 (ϵ-Amortization Resistance)**.** For inputs of length λ and ouputs $q \in \text{poly}(\lambda)$, function $f^{\mathcal{O}} = \{f^{\mathcal{O}}(n)\}_{n\in\mathcal{N}}$ is ϵ-amortization resistant on average with respect to a sampler S if for all adversaries $\mathcal{A} = (\mathcal{A}_1^{\mathcal{O}}, \mathcal{A}_2^{\mathcal{O}})$ with $\mathcal{A}$ performing less than $(1-\epsilon)qN$ queries to the oracle $\mathcal{O}$ on average, where N number of queries required for one evaluation of $f^{\mathcal{O}}(n)$ on average, the following is negligible in λ:

$$\Pr\left[\forall i \in [q], \pi_i = f^{\mathcal{O}}(n_i) \;\middle|\; \begin{array}{c} \{n_i\}_{i=1}^q \leftarrow n, (n, \text{aux}) \leftarrow \text{S}(1^\lambda) \\ \text{precomp} \leftarrow \mathcal{A}_1^{\mathcal{O}}(1^\lambda, \text{aux}) \\ \{\pi_i\}_{i=1}^q \leftarrow \mathcal{A}_2^{\mathcal{O}}(1^\lambda, n, \text{precomp}) \end{array}\right].$$

This definition captures the fact that computing multiple proofs does not come with marginal gains: indeed, provers cannot use larger computational resources to batch process proofs and achieve disproportionate performance improvements. By preventing large miners from achieving algorithmic returns-to-scale, this property is crucial in ensuring fairness. With the above objectives in mind, we now look at how to adapt our implementation to realize such a system.

7.1.3 Prover Computational Costs

Before we look at designing an amortization resistant PoNW system, we summarize the computationally expensive components of proof generation in the Quadratic Arithmetic Program (QAP) Non-Interactive Proof (NIPs) of [Parno et al. 2013] compiled with [Kate et al. 2010]. For an ℓ-size statement with m internal variables and n constraints, the prover $\mathcal{P}$ needs to (1) update inputs and witnesses, and (2) perform $9m + n$ exponentiations in $\mathbb{G}$ using elements from the proving key as bases. Since updating variable assignments is orders-of-magnitude faster, amortization resistance requires $\mathcal{P}$ to recompute (almost) all exponentiations for each new nonce. We provide the formal definition of QAP instances below for completeness.

Definition 7.3. A QAP Q over field $\mathbb{F}$ contains three sets of $m+1$ polynomials $\mathcal{V} = \{v_k(X)\}, \mathcal{W} = \{w_k(X)\}, \mathcal{Y} = \{y_k(X)\}$, for $k \in \{0, ..., m\}$ and a target polynomial $t(X)$ of degree n. Suppose F is a function that takes as input ℓ_1 elements and outputs ℓ_2 elements for a total of $\ell = \ell_1 + \ell_2$ elements. We say that Q computes F if: $(a_1, ..., a_\ell) \in \mathbb{F}^\ell$ is a valid assignment of F's inputs and outputs iff there exist $(a_{\ell+1}, ..., a_m)$ for which $t(X)$ divides $p(X)$ where

$$p(X) := \left(v_0(X) + \sum_{i=1}^{m} v_i(X)\right) \cdot \left(w_0(X) + \sum_{i=1}^{m} w_i(X)\right) - \left(y_0(X) + \sum_{i=1}^{m} y_i(X)\right).$$

7.1.4 Amortization of Multiexponentiation

Multiexponentiation is inherently amortizable [Gordon 1998; Henry 2010] given enough memory, although space requirements scale exponentially with the number of computed elements. This is because we can precompute the exponents of specific basis elements and perform look-ups that can be used by multiple evaluations at once. We make precise the relationship between size and amortization gain to demonstrate that non-negligible amortization gains require an infeasibly large amount of space. Since we are interested in average-case guarantees, all input elements

to the multiexponentiation algorithm (i.e. the enumerated exponents, or puzzle instances) are sampled uniformly randomly from some S.

We consider amortization in Shoup's Generic Group Model (GGM) [Shoup 1997],[1] in which the adversary can only compute products based on existing group elements (with non-negligible probability), or directly query the exponentiation of some index. The adversary has access to a multiplication oracle $O : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$, which returns the multiplication of the input elements over some random encoding $\sigma : \mathbb{Z}_p \to \mathbb{G}$. This oracle computes $O(\sigma(i), \sigma(j)) = \sigma(i + j)$. The adversary may also use a polynomially-sized precomputation string. Since they don't have access to the exponents of the bases that are being multiplied together (so as to perform a direct look-up), computing some $\sigma(k)$ requires the generation of an addition chain ending with $\sigma(k)$.

However, this is the only assumption underlying the lower-bound results which prove the optimality of (the generalized) Pippenger's algorithm [Henry 2010], as they obtain lower-bounds on the length of the minimal addition chain needed to compute some element. In short, our main formal contribution relies on adapting the packing lower-bound ideas of [Erdös 1960; Pippenger 1980] to formalize the relationship between amortization of multiexponentiation of random indices and the amount of space available to the adversary. We do this by making explicit the average-case lower bounds for multiexponentiation, which were only stated (but not proven) in [Erdös 1960; Pippenger 1980] to be a constant term away from the worst-case lower bounds.

Note that the notion of average-case hardness requires an underlying probability distribution over which the input indices are sampled. Obviously, the distribution of the sampled puzzle instances can affect the average-case bounds if, for example, the sampler provides structured output with high probability. Therefore, all results have to be taken with respect to the underlying distribution of the inputs, which is in turn specified by the choice of sampling algorithm S. Where this S is taken to be uniform (as in this work), the notion of average-case hardness defaults to the traditional average-case lower bound results referenced in the literature.

[1] Maurer proposed a slightly different GGM definition [Maurer 2005], for a comparison see [Zhandry 2022].

In order to make formal statements about the amortization resistance of computing multiple NIPs, we need to show that there exists some sampling algorithm S_{NIP} outputting instance-witness pairs (ϕ, w) so that, on average over its public coins, these output puzzle instances require a minimum number of oracle calls each for computation of their corresponding proof π. The first step towards this is to construct the equivalent multiexponentiation problem that the above will reduce to. In the following, we restrict ourselves to the NIP of [Parno et al. 2013], in which the valid output proof consists of 9 group elements of the form $\sum_{k=1}^{\kappa} w_k G_k^i$ for $i \in [9], w_k \in [N]$ and an additional element $\sum_{m=1}^{\mu} g(w_1, ..., w_\kappa)_m H_m$, where g an m-dimensional n-variable polynomial encoding the instance's witness and $G_i, H_m \in \mathbb{G}$.

Since the hardness of the above computation depends on the structure of w and g, it becomes apparent that we need to restrict the types of *predicates* that we are looking at. In subsequent sections, we make precise the following construction: a circuit with an efficient sampler S such that (1) accepting witness elements $w_1, ..., w_\kappa \in [N]$ are randomly distributed, (2) for each valid instance ϕ there exists only one valid w, and (3) for each valid w, there exists a unique valid g. Note that (1) and (2) are properties of the predicate, while (3) requires a stronger result on the NIP's knowledge guarantees. We will show that predicates satisfying (1) and (2) are enough to reduce the computation of a NIP from [Parno et al. 2013] (which satisfies (3)) to a multiexponentiation problem (Definition 7.4) whose amortization we can bound.

Definition 7.4. The (κ, μ)-length MultiExp function $f : [N]^\kappa \to \mathbb{G}^\nu$ of dimension ν for bases $\{G_i^{(1)}, ..., G_i^{(\nu-1)}\}_{i=1}^{\kappa}$, $\{G_i^{(\nu)}\}_{i=1}^{\mu}$, and function $g : [N]^\kappa \to K \subseteq [N]^\mu$ is

$$f(x_1, ..., x_\kappa) := \left(\sum_{i=1}^{\kappa} x_i G_i^{(1)}, ..., \sum_{i=1}^{\kappa} x_i G_i^{(\nu-1)}, \sum_{i=1}^{\mu} g(x)_i G_i^{(\nu)} \right),$$

where the x_i are given by sampler S, based on its random coins.

In order to provide a reduction that exactly captures the average-case hardness of the above problem, the structure of g becomes important. This requires a more technical treatment, so

here we work in the case where g is a weakly collision-resistant map from the witness elements $x = (x_1, ..., x_\kappa)$ to the values $(g(x)_1, ..., g(x)_\mu) \in K \subseteq [N]^\mu$. This defines a computationally unique correspondence between witness elements and representations of μ-degree polynomials with coefficients in $[N]$. We specifically require the mapping $g : [N]^\kappa \to K \subseteq [N]^\mu$ to be collision-resistant in each of its output coordinates, or that the following probability is negligible for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\exists i \text{ s.t. } g(\mathcal{A}(z))_i = z_i; z \leftarrow g(x), x \leftarrow_R [N]^\kappa\right] \approx 0,$$

where z_i denotes the i-th coordinate of z. This is enough to provide multiexponentiation amortization bounds, which are given below for the case when $\kappa = \mu$. Note that the general case for $\mu > \kappa$ can also be calculated in the exact same way, but has been omitted for simplicity.

Theorem 7.5. *The (κ, κ)-length MultiExp function (c.f. Definition 7.4) of dimension ν over index size $\lambda := \log(N)$, group $\mathbb{G}$ with $|\mathbb{G}| = 2^\lambda$, and storage size q is ϵ-amortization resistant with respect to the uniform sampler for all collision-resistant g, and for large enough κ, λ, ν, q satisfies:*

$$\epsilon \leq \frac{\log(q) + o(1)}{\log(q) + \log(\kappa) + \log(\nu) + \log(\lambda)}.$$

We prove Theorem 7.5 in Appendix A.1. This amortization gain is unavoidable for NIPs that reduce to multiexponentiation; such as by compilation with [Kate et al. 2010].

7.2 Amortization Resistance & Efficiency

We modify the DPS predicate Π to ensure that most of the proof variables change unpredictably with modifications of the nonce or state. This gives amortization resistance in exchange for increasing the number of variables and constraints in the predicate. The performance overhead originates from the need to commit to state and 'mask' the computation, which can be expensive

for large predicates.

The naive approach would be to isolate each of the different circuits in the system and show that they can be modified to change unpredictably based on some seed. The design challenge here is how to make this happen while conserving the proof's correctness guarantees. For this, we ideally want to leverage a property specific to our predicate in order to 'mask' the computations and treat the proving system as a black box. We leverage the Pedersen hash function to transform our predicate Π to an amortization resistant version in Section 8.3.

7.2.1 Committing to State

Given some nonce n, the prover might only change a part of the input in order to (re)check difficulty. This is an issue if the same nonce can be used with many inputs (in our case, transactions), as an adversarial prover would compute a proof and then only switch out a single transaction (or bit!), rechecking difficulty with no expensive recomputation. Define $\rho := \mathsf{PRF}_n(\text{state})$ that commits to state where PRF a pseudorandom function family. We need to commit to all block transactions, ensuring that changing one transaction changes ρ. This can be expensive if we exploit no information about the underlying predicate, since PRF would have to commit to every single original variable.

Fortunately, for our predicate the input to PRF is small: we use $\rho = \mathsf{PRF}_n(rt)$ where rt the root of the new state and n the given nonce. Since this input will anyways be computed as part of the protocol, we don't actually suffer any overhead apart from having to verify the above computation. Note that this is actually *constant* in predicate size. In the GGM, we can replace the PRF by a collision resistant hash function CRT instead, since the randomness of the group encoding is sufficient for the witness elements to look random to an adversary.

7.2.2 Masking the Computation

We can force unique changes to the Merkle path updating the account if we require n to be part of the leaf: since a change in the block (or nonce) would lead to a new n, all update paths need to be recomputed if any transaction is changed. However, we also need to enforce change to the *old* Merkle path checking account existence. This technique is thus not ideal, since these paths do not depend on the current nonce (or state) at all, meaning that around half our variables will remain the same, giving $\epsilon \approx 1/2$.

To get around this, we opt for a different approach. We 'mask' the input variables to $\mathcal{H}$ by interaction with ρ (which also commits to n) and transform the constraints of the hash function subcircuit $C_{\mathcal{H}}$ into a new circuit that retains the original Proof of Knowledge (PoK) guarantees by verifying the same underlying computation. By the unpredictability of ρ and randomness of n, we hope to achieve upper bounds for amortization resistance based on the security of the CRT. In this case, the sampler would need to provide valid witnesses for $C_{\mathcal{H}}$ of the form $w = (w_1, ..., w_m)$ whose encodings are indistinguishable from random, given n sampled uniformly randomly and access to a multiplication oracle O for a randomized encoding of some $\mathbb{G}$.

7.3 Consensus Security

Our proposal introduces two novel effects to the consensus protocol due to the fact that checking even one proof-of-work solution (on the order of tens of seconds to minutes) can take a non-negligible fraction of the average block frequency (ten minutes in the case of Bitcoin). We can measure these effects assuming a single puzzle solution takes time τ to check (with the mean block arrival time normalized to 1).

7.3.1 Quantization Effects

When τ becomes a significant fraction of the average block generation time ($\tau \sim 1$), miners face a loss of efficiency as they will often be forced to discard a partially-checked puzzle solution when a block is broadcast while checking previous solutions. We prove the scale of this efficiency loss in a short theorem:

Theorem 7.6. *A miner in a proof-of-work protocol with puzzle checking time τ will discard a fraction $1 - \frac{\tau}{e^{\tau}-1}$ of their work due to newly broadcast solutions.*

Note that as $\tau \longrightarrow 0$ (fast puzzle checking time relative to block interval), the fraction of wasted work drops to 0. This is why this effect has never been considered in prior work. In the reverse direction, as $\tau \longrightarrow \infty$ the fraction of wasted work approaches 1. For $\tau = 1$ (solutions take as long to check as the mean block interval), the fraction of wasted work is $\frac{e-2}{e-1} \approx 0.42$, suggesting that we should aim to keep the time (even for slow miners) to get a solution significantly shorter than the mean block time.

7.3.2 Stubborn Mining and Collisions

Slow puzzle checking time also introduces a concern that miners might refuse to stop working on a partially-checked solution (and hence discard partial work) even if a valid solution is found and broadcast. These *stubborn* miners might cause collisions in the blockchain (two blocks being found at the same height in the chain). We can analyse a worst-case scenario in which all miners are synchronized with identical proving time, in effect making all miners stubborn and maximizing the probability of simultaneous solutions. If miners aren't synchronized, they may opt to finish their current effort after a block is found, but even if all miners do so this reduces to the above case where all miners finish checking a solution simultaneously. We call each synchronized period in which all miners check a solution a *round*.

Theorem 7.7. *The expected number of solutions in a synchronized mining round is defined by a Poisson distribution with $\lambda = \tau$. The proportion of rounds with multiple solutions (of rounds with any solution) is upper bounded by $\tau/2$.*

By Theorem 7.7, an unoptimized 100 second proving time (and 10 minute block time) has collisions for fewer than $\frac{1}{12}$ of blocks in the worst case.

8 | Design & Instantiation

8.1 Proof System & Predicate

Since we'll be broadcasting each proof π_i to the network, we would like them to be quite small (ideally $<$ 1kB). We also require that the size of π_i does not increase with i, ideally staying the same size after every state transition. With these design choices in mind, we prototype our system using libsnark[SCIPRLab 2017], a C++ library implementing the IVC system in [Ben-Sasson et al. 2017] using the construction from [Parno et al. 2013]. This is done using Succinct Non-Interactive Arguments of Knowledge (SNARKs) [Ben-Sasson et al. 2014], non-interactive proofs of knowledge with the additional property of *succinctness*: producing constant-sized proofs that can be instantly verified. We can equivalently consider Π_S as an arithmetic circuit C_Π, evaluating to 1 on some input B_i if and only if B_i is a valid commitment to the output of UpdateState given some transaction set $\mathbf{t}$ and $\mathcal{S}_{i-1}$. In our implementation, C_Π is expressed as a QAP.

The circuit is encoded over elliptic curve elements through vectors in $\mathbb{F}_p$, where the number of gates increases with the size of π_i and the time required to generate it. By manually designing a circuit C_Π, we minimize the number of gates used and provide a deployable implementation. Note that the system need also allow for *recursive proof composition*, or the capability of new proofs to check the validity of previous proofs efficiently. Since this construction depends on SNARKs over pairs of elliptic curves that form *IVC-friendly cycles*, we use the same pair of non-supersingular curves of prime order as [Ben-Sasson et al. 2017] with 80 bits of security and field size $\log p \approx 298$.

8.2 Circuit Requirements

A tree depth of 32 for our implementation allows for 4.2 billion accounts. We compare this to 32 million unique used wallets on the Bitcoin blockchain after 10 years of operation. This requires $32 \cdot 4 = 128$ hash checks for each transaction. We use the circuits in libsnark to verify such proofs of inclusion and modification.

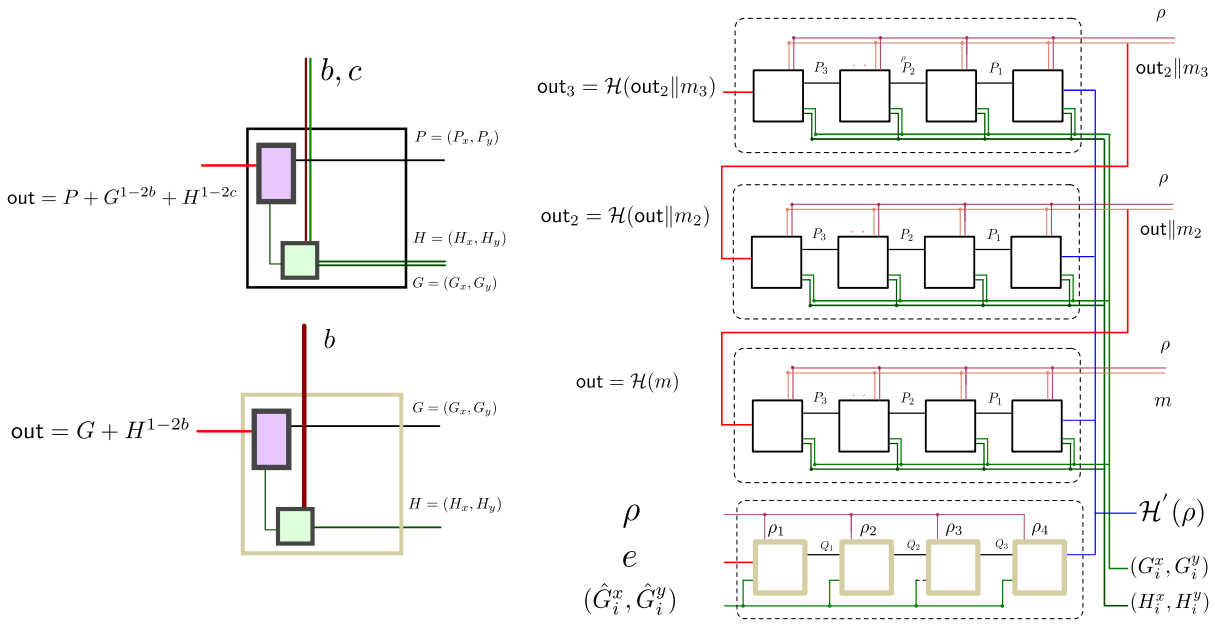
8.2.1 Pedersen Hashes

Since it is desirable for $\mathcal{H}$ to be efficiently represented with a low gate count, we opt for using Pedersen hashes [Damgård et al. 1993]. We modify the Pedersen hash to compute $\prod_{i=1}^{D} G_i^{1-2x_i}$ where $\{x_i\}_{i=1}^{D}$ is the bit representation of the input x and $\{G_i\}_{i=1}^{D}$ is a set of primitive roots for an elliptic curve group $E(\mathbb{F}_p)$. We encode each root as two field elements and, based on the sign of each input x_i, perform multiplication of an intermediate field variable c by each G_i to arrive at the digest if the corresponding $x_i = 1$. We use the same underlying elliptic curve for the SNARK with $|p| = 2^{298}$, which reduces in security to the elliptic curve discrete-logarithm problem (ECDLP) at a security of 80 bits.

8.2.2 Signature Scheme

We use Schnorr signatures [Schnorr 1989] over an elliptic curve (EC), based on the hardness of DLP. This choice is motivated by our desire to minimize the size of the verifying circuit, since this has to be built inside C_{Π}. The Schnorr verification circuit only requires two exponentiations, a hash computation, and a comparison between scalars. The same curve from the IVC construction is also used here, offering a security of 80 bits. Schnorr signatures use elliptic curve elements as public keys, resulting in key sizes of 596 bits, or $298 + 1 = 299$ bits with point compression. Secret keys are sampled as random 298-bit strings.

Figure 8.1: *Left*: The TwoBitGroupAddition and SymmetricGroupAddition circuits from top to bottom respectively. *Right*: Layout of a single Merkle authentication path circuit, with $M = 3$ evaluations of $\mathcal{H}$ on an $n = 4$-bit Pedersen hash. $\hat{G}_i = (\hat{G}_i^x, \hat{G}_i^y) = G_i + G_i + H_i$ and $\mathcal{H}'(\rho) = \prod_{i=1}^{4} \hat{G}_i^{1-2\rho_i}$ and e the identity.



8.3 Randomizing the Pedersen Hash

In addition to some input x of length n bits, our evaluation requires a pseudorandom seed $\rho \in \{0,1\}^n$. Consider the following modification, which can be thought of as masking the underlying evaluation by using two sets of input variables: $\mathcal{H}_G(\rho)^2 \cdot \mathcal{H}_H(\rho)$ and x_i for $i \in [n]$, where $\mathcal{H}_G(\cdot)$ the evaluation of the Pedersen function $\mathcal{H}_G(x) = \prod_{i=1}^{n} G_i^{1-2x_i}$.

The variable $h_0 = \mathcal{H}_G(\rho)^2 \cdot \mathcal{H}_H(\rho)$ forms the 'starting point' of the evaluation. In the beginning, the prover will have access to generator constants $\{H_i, H_i^{-1}, G_i^{-2}H_i^{-1}, G_i^2 H_i\}$ for the specific instance of the problem. It would then perform a 2-bit lookup based on x_i and ρ_i, multiplying the intermediate variable c_i by one of the above. By carefully choosing these q_i, we can design the circuit in such a way that unpredictability based on the seed is retained by all intermediate variables except the output y, which we ensure equals $\mathcal{H}_G(x)$.

Correctness follows from the following observation: at step 0, the variable $c_0 = \mathcal{H}_H(\rho) \cdot$

Algorithm 2 MaskedPedersen

Require: $x, \rho \in \{0,1\}^n, G, H \in \mathbb{G}^n$
Ensure: $y \in \mathbb{G}$

procedure CacheGenerators(ρ, G, H)
  Parse $\{\rho_i\}_{i=1}^n \leftarrow \rho$
  Compute $h \leftarrow \mathcal{H}(\rho, G)$, $h_2 \leftarrow \mathcal{H}(\rho, H)$, $h_0 = h^2 \cdot h_2$
  return h_0, h
end procedure
procedure MaskedHash(x, ρ, h_0, h)
  Parse $\{x_i\}_{i=1}^n \leftarrow x$, $\{\rho_i\}_{i=1}^n \leftarrow \rho$
  Define $q = \{q_i\}_{i=1}^n$, $c = \{c_i\}_{i=0}^n$ and set $c_0 = h_0$
  for $i \leq n$ **do**
    if $\rho_i = 0, x_i = 0$ **then** $q_i = H_i^{-1}$
    else if $\rho_i = 0, x_i = 1$ **then** $q_i = G_i^{-2} \cdot H_i^{-1}$
    else if $\rho_i = 1, x_i = 0$ **then** $q_i = G_i^2 \cdot H_i$
    else if $\rho_i = 1, x_i = 1$ **then** $q_i = H_i$
    end if
    $c_i = c_{i-1} \cdot q_i$
  end for
  $y = c_n \cdot h^{-1}$
  return y
end procedure

$\mathcal{H}_G(\rho)^2 = \mathcal{H}_G(\rho) \cdot \prod_{i=1}^n G_i^{1-2\rho_i} \cdot H_i^{1-2\rho_i}$ is initialized as the hash of the seed. For all intermediate steps $j < n$, we have that $c_j = \mathcal{H}_G(\rho) \cdot \left(\prod_{i=1}^j G_i^{1-2x_i}\right) \cdot \left(\prod_{i=j+1}^n G_i^{1-2\rho_i} H_i^{1-2\rho_i}\right)$. Finally, after the n-th bit has been processed the final intermediate variable c_n is equal to the Pedersen hash of the original input x multiplied by (the unpredictable) $\mathcal{H}_G(\rho)$. By multiplying with $\mathcal{H}_G(\rho)^{-1}$, we get $\mathcal{H}_G(x)$. This follows easily from the fact that at every step we are performing the following operation: $c_i = c_{i-1} \cdot (H_i \cdot \mathbf{1}[\rho_i, x_i = 1] + H_i^{-1} \cdot \mathbf{1}[\rho_i, x_i = 0] + G_i^{-2} H_i^{-1} \cdot \mathbf{1}[\rho_i = 0, x_i = 1] + G_i^2 H_i \cdot \mathbf{1}[\rho_i = 1, x_i = 0])$. It can be quickly checked that this computation ensures the previous recursive property when initialized with $c_0 = \mathcal{H}_H(\rho) \cdot \mathcal{H}_G(\rho)^2$. By induction, this implies that after the n-th bit, only $\mathcal{H}_G(\rho)$ and the exponentiations due to the bits of x remain in the output variable i.e. $c_n = \mathcal{H}_G(\rho) \cdot \prod_{i=1}^n G_i^{1-2x_i}$.

We observe that in all cases where we know that the variable a_i has small support (when,

for example, it is boolean $a_i \in \{0, 1\}$), the prover can always precompute once and use the same answers without performing exponentiations. However, this is not a problem since all miners would know what the precomputed answers are from the very beginning and can incorporate them with a small memory cost.

The problem with creating variables that become more and more 'deterministic' is that at some point their support becomes so small that an adversary will be able to precompute some oracle queries. However, since the end value of the sequence of variables $\{c_i\}_{i=1}^{n}$ is $h \cdot \mathcal{H}_G(x)$ which is also unpredictable due to h, it is not feasible to predict any index $i \in [n]$ without violating the security of the operation $\mathcal{H}_G(\rho) = h$ even if $\mathcal{H}_G(x)$ is previously known. Note that h can be 'offset' by a random element I as $h_i^{'} = h + I_i$ for each path $i \in [N]$. This provides independence between authentication paths using the same nonce.

8.4 Security

8.4.1 Unique Witness Extraction

We must restrict the proof systems used because certain constructions are inherently insecure: Groth16 [Groth 2016] can easily be rerandomized, for example, with a couple additional group multiplications. We thus need a notion of non-malleability, or that we cannot construct proofs given access to previous valid proofs. To this end, we show that Pinocchio [Parno et al. 2013] satisfies *unique witness extractability*. This property requires the proof system to output proofs with unique encodings for each distinct statement-witness pair.

Definition 8.1. Let (Setup, Prove, Verify, Simulate) denote a NIP for relation $\mathcal{R}$. Define the PPT

algorithm $\mathcal{A}$ with extractor $\chi_{\mathcal{A}}$, $\mathbf{Adv}^{uwe}_{\mathcal{BG},R,\mathcal{A},\chi_{\mathcal{A}}}(\lambda) = \Pr[\mathcal{G}^{uwe}_{\mathcal{BG},R,\mathcal{A},\chi_{\mathcal{A}}}(\lambda)]$, and $\mathcal{G}^{uwe}_{\mathcal{BG},R,\mathcal{A},\chi_{\mathcal{A}}}(\lambda)$ as:

Main $\mathcal{G}^{uwe}_{\mathcal{BG},R,\mathcal{A},\chi_{\mathcal{A}}}(\lambda)$

$(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g) \leftarrow \mathcal{BG}(1^\lambda)$

$(\mathbf{crs}, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$

$(\phi, \pi_1, \pi_2) \leftarrow \mathcal{A}^{O}(\mathbf{crs})$

$(w_1, w_2) \leftarrow \chi_{\mathcal{A}}(\mathsf{tr}_{\mathcal{A}})$

$b_1 \leftarrow (w_1 = w_2) \cup (\mathcal{R}(\phi, w_1) \neq 1) \cup (\mathcal{R}(\phi, w_2) \neq 1)$

$b_2 \leftarrow \mathsf{Verify}(\mathbf{crs}, \phi, \pi_1) \cap \mathsf{Verify}(\mathbf{crs}, \phi, \pi_2) \cap ((\phi, \pi_1) \notin Q) \cap ((\phi, \pi_1) \notin Q) \cap (\pi_1 \neq \pi_2)$

Return $b_1 \cap b_2$

$O(\phi)$

$\pi \leftarrow \mathsf{Simulate}(\mathbf{crs}, \tau, \phi)$

$Q = (\phi, \pi) \cup Q$

Return π

The NIP is unique witness extractable if for all PPT $\mathcal{A}$ $\exists \chi_{\mathcal{A}}$ such that $\mathbf{Adv}^{uwe}_{\mathcal{BG},R,\mathcal{A},\chi_{\mathcal{A}}}(\lambda) \in negl(\lambda)$.

Theorem 8.2. *Assume the q-PDH, 2q-SDH and d-PKE assumptions hold for* $q \geq \max(2d-1, d+2)$. *The Pinocchio NIP [Parno et al. 2013] satisfies unique witness extractability.*

8.4.2 Single Witness Hardness

The ability to resample witnesses for a provided statement-witness pair can also be advantageous to an adversary, since an 'easy' witness could be found by repeated sampling. We follow the definition of 2-hard instances in [Dahari and Lindell 2020] and define *single witness hard* languages, for which it is hard to find a new witness given an existing one.

Definition 8.3. Let R_L be a relation, and $\mathcal{L} = \{\phi | \exists w \text{ s.t. } R_L(\phi, w) = 1\}$ an NP language. $\mathcal{L}$ is a hard single-witness language if:

1. **Efficient Sampling:** There exists a PPT sampler $\mathsf{S}(1^\lambda)$ outputting a statement-witness pair $\langle \mathsf{S}^x, \mathsf{S}^w \rangle$ with $\mathsf{S}^x \in \{0,1\}^\lambda$ and $(\mathsf{S}^x, \mathsf{S}^w) \in R_L$.

2. **Witness Intractability:** For every PPT $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that:

$$\Pr\left[\left(\mathsf{S}^x(1^\lambda), \mathcal{A}(\mathsf{S}(1^\lambda), 1^\lambda)\right) \in R_{\mathcal{L}}, \mathcal{A}(\mathsf{S}(1^\lambda), 1^\lambda) \neq \mathsf{S}^w(1^\lambda)\right] \leq \mu(\lambda).$$

A relation whose statements are outputs of a CRT hash function $\mathcal{H}$ defines a hard single-witness language. We show this for $\mathcal{L}(\mathcal{H}_P) = \{\phi : \exists w \text{ s.t. } \mathcal{H}^G_{P,|w|}(w) = \phi\}$ where $\mathcal{H}^G_{P,n}$ a weakly collision-resistant hash function.

We show that computing a [Parno et al. 2013] proof for the evaluation of MaskedHash (and our DPS predicate) will take on average a similar number of queries as a suitably parametrized MultiExp instance. We restrict to the case of outputs from a sampler S which samples a ρ randomly and generates valid witnesses. Since we are working in the GGM, the witness variables of the MaskedHash instance have an encoding that is indistinguishable from random. Therefore, the amortization bounds of Theorem 7.5 apply.

Theorem 8.4. *There exists a sampler* S *and QAP R evaluating N parallel instances of k-bit inputs of* MaskedHash *for which the [Parno et al. 2013] prover and the* $(4N(k+1), 8N(k+1)+2k)$*-length MultiExp problem of dimension* 10 *are equivalent up to constant terms with respect to multiplicative hardness.*

The vast majority of the constraints and variables in the predicate of the designed system are hash evaluations, so Theorem 8.4 can be used to show that there exists a proof system verifying state transitions for the DPS with bounded amortization-resistance guarantees. This is because the DPS predicate spends the vast majority of its time computing a proof whose hardness can be bounded by Theorem 8.4, since it is a sequence of iterated Pedersen hashes over a unique simulation extractable NIP.

Corollary 8.5. *The DPS in Section 6.3.1 with block size T, state tree depth d, and index size λ admits a Proof of Necessary Work that is ϵ-amortization resistant w.r.t. a multiplication oracle and for which:*

$$\epsilon \lessapprox \frac{\log(q)}{\log(q) + \log(dT\lambda) + \log(\lambda)},$$

where q is memory size measured in proof elements.

8.5 Performance

We construct the DPS based on the above specifications and investigate its running time and memory consumption. Results are displayed in Table 2. Our benchmark machine was an Amazon Web Services (AWS) c5.24xlarge instance, with 96 vCPUs and 192GiB of RAM. The security properties of the DPS are based on the guarantee of Π-compliance provided by IVC. It is apparent that setup and proving times dominate both the running time and memory consumption in the protocol. Setup takes place once by a trusted third-party and hence is less critical for day-to-day system performance.

The prover is run by the miners, or full nodes. These generate proof-of-work solutions repeatedly and would compute proof instances for many input nonces. Thus, larger storage requirements ($\sim$ 5.42GB key sizes) could be easily met by these nodes, as could the need for more parallelism and better computing power to bring down the proving rate.

We normalize the block time to achieve $\tau = 1/3$ in the sense of Theorem 7.6 for a proof including 30 transactions. This gives us that a miner will discard in expectation 15.59% of their work for an efficiency of $\sim$ 84% if all miners operated based on the above benchmarks. Theorem 7.7 then gives an upper bound on the block orphan rate (or likelihood of block collisions) of 16.65%. Since we are keeping block times constant at 10 minutes, we note that any improvements in SNARK proof generation times will correspondingly decrease the amount of wasted work

Txs	**Constraints**	**Generator**	**Prover**	**Verifier**	**Size**		
#	#	Avg (s)	Avg (s)	Avg (ms)	pk (GB)	vk (kB)	π (B)
3	3658281	53.99	24.57	16.0	0.74	0.76	373
10	10071527	161.24	88.14		1.96		
20	19233307	268.93	185.10		3.74		
30	28395087	354.83	198.61		5.61		
40	37556867	485.52	286.50		7.15		
50	46718647	570.09	358.95		9.01		

Table 8.1: Prototype Times and Key Sizes for Predicates verifying different numbers of transactions: Average running times for setup $\mathcal{G}$, prover $\mathcal{P}$ and verifier $\mathcal{V}$ over 10 iterations are shown alongside proving/verification key and proof sizes.

and orphan rate. Moreover, this does not depend on the way that the proofs are generated: distributed techniques among many participants (such as [Wu et al. 2018]) would also benefit efficiency through the corresponding decrease of average proof time.

9 | Open Questions

9.1 Waste in Nakamoto Consensus

It should be noted that the wastefulness of the puzzle is proportional to N the number of miners: we take this as inherent to Nakamoto consensus while maximizing useful computation. However, a portion of the wasted work can be recovered by relaxing the protocol to accept chains of k proofs if the last proof satisfies difficulty and all proofs have committed to the same nonce (we present the case of $k = 1$ in this paper), preventing miners from discarding proofs that don't satisfy difficulty. This would provide an $O(k)$ efficiency improvement, where k would in practice depend on memory considerations. If k is large enough, we can improve the above trade-off. We do not analyze this here, as the associated efficiency analysis for the choice of k is quite complex.

9.2 Trusted Setup & Quantum Resistance

Using SNARKs as a building block in our system introduces the issue of the one-time trusted setup. Like in other cryptocurrency systems built using SNARKs [Sasson et al. 2014], an adversary with knowledge of the secret parameters would be allowed to forge proofs. One approach is a ceremony with many participants through a multiparty protocol [Bowe et al. 2017, 2018b], in which only one is honest. We also use SNARKs based on elliptic-curve hardness assumptions which are not quantum-resistant. Recent work on practical instantiations of SNARK constructions based on

lattice assumptions [Gennaro et al. 2018] or point-based PCPS and IOPs [Ben-Sasson et al. 2018b], may offer an option for quantum-resistant SNARKs. Recent work has sought to construct SNARK systems which require limited or no trusted setup. SONIC [Maller et al. 2019] uses an adaptively changing structured reference string (the proving/verification keys). More recent advances such as Marlin [Chiesa et al. 2019a] and Fractal [Chiesa et al. 2019b] provide structured reference strings for *all* predicates trustlessly.

9.3 Privacy & Complex Transactions:

We did not consider transaction privacy, focusing instead on a simple distributed payment ledger closely matching the properties of Bitcoin. However, while we use SNARKs for their succinctness properties, the constructions here readily extend to provide zero-knowledge succinct arguments as well (zk-SNARKs). It should be straightforward to adapt to a zk-SNARK-based privacy-preserving transaction format (such as Zerocash [Sasson et al. 2014]) and provide no additional overhead for chain verification. The main cost is that users compute SNARK proofs to post transactions, imposing a heavier burden for the system. It would also require careful thought to achieve amortization resistance when users are computing proofs of transaction validity.

Bitcoin supports more complex payment scripts, enabling applications like atomic cross-chain swaps and off-chain payment channels that our system does not. Ethereum supports fully programmable smart contracts to control payment. In principle, a programmable state machine like Ethereum's can be supported on our architecture using "universal" SNARK techniques such as TinyRAM [Ben-Sasson et al. 2013]. Recent work on achieving privacy in smart contract platforms using SNARKs [Bünz et al. 2019a; Bowe et al. 2018a] can potentially be adapted to our setting to enable a more powerful programming model with efficient verification.

9.4 Other Consensus Protocols

We adapted Bitcoin's relatively simple linear longest-chain rule. More complex DAG-based proposals exist which improve on Bitcoin's consensus protocol. They involve different formulas for computing the quality of a specific block in the chain. Our approach does not preclude the use of more complex predicates for consensus. By setting the quality accordingly and correctness of π_i, *any* consensus protocol can be used in this way.

9.5 Hardware Acceleration & Parallelism

The design of Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs) would lead to order-of-magnitude improvements in proving time and thus substantially minimize quantization effects (c.f. Section 7.3). Such hardware would also provide a barrier to entry for some miners due to the upfront costs for its design. This can impact the fairness of the system. Our construction uses naive parallelism. Recent advances [Wu et al. 2018] construct larger proofs using parallel workers, a model that adapts readily to cryptocurrencies which typically feature large mining pools. Exploring this is an important avenue for future work, especially given the order-of-magnitude improvements in the size of computable predicates.

Part II

RedShift

10 | Contributions

The works of [Ben-Sasson et al. 2018a] and [Ben-Sasson et al. 2019b] introduced the Fast Reed Solomon IOP of Proximity (FRI IOPP) - a novel protocol for efficient proximity testing, or checking if a given function is close to any low degree polynomial. Such a proximity tester may be naively turned into a transparent PCS, which provides commitments of size $O(\log^2 d)$ for polynomials of degree d. However, the soundness error of such a commitment scheme is rather large, and the protocol should be iterated many times to reach a sufficient security level. This results in large proof sizes and computational burden. The reason for the large soundness error hides in the low sensitivity of FRI: when the Hamming distance between two different polynomials is smaller than some predefined constant, it is impossible for FRI to efficiently distinguish them.

In this work, we generalize the PCS in the sense that we construct a commitment to a list of proximate polynomials. We introduce a new cryptographic primitive for fast verification of polynomial evaluations we call a *list polynomial commitment* (LPC). At a high level, this scheme retains the necessary security guarantees that are required for polynomial-based proof systems such as [Gabizon et al. 2019] and [Maller et al. 2019] to compile into zk-SNARKs. In the language of IOP formalization, this primitive can be thought of as an alternative compiler for public-coin IOP protocols.

10.1 Compilation of IOPs with LPCs

The above contribution implicitly provides a general framework that demonstrates how the LPC can be used to compile any polynomial IOP into a preprocessing zk-SNARK. As previously mentioned, this follows the approach in [Chiesa et al. 2019a] and [Bünz et al. 2019b] with the main difference being that we do not require a PCS in their (more restrictive) sense.

10.2 RedShift

We demonstrate the security and practicality of this approach by compiling PLONK [Gabizon et al. 2019] using the framework above. By fitting an implementation of the list commitment scheme on PLONK with suitable adaptations and optimizations, we remove all trusted computation while retaining efficiency in both proof size and generation time. We call this new proof system RedShift, and provide:

1. formal proofs of correctness and security,

2. a proof-of-concept implementation, along with benchmarks establishing feasibility.

At an 80-bit security level and for circuits of size 2^{20}, RedShift provides proofs of size ~ 515 KB with a proof generation time of about half a minute. Overall, RedShift is an efficient instantiation of a plausibly post-quantum transparent preprocessing zk-SNARK suitable for practical deployment at high security.

Prior work on transparent zk-SNARKs has explored a variety of trade-offs and different design choices in order to achieve efficiency gains. We briefly discuss concurrent and previous efforts in building efficient and transparent proof systems and compare with our approach. The works of [Ben-Sasson et al. 2018c] and [Chiesa et al. 2019b] design IOPs for the Rank-1 Constraint System (R1CS) arithmetization, providing a compilation framework equivalent to our methods. They

require a holographic lincheck argument/IOP of Proximity respectively in order to construct their IOP for R1CS, which our approach avoids by using a suitable transformation of the proof system in [Gabizon et al. 2019] instead. The works of [Zhang et al. 2020] (and [Xie et al. 2019], although not transparent) use a similar approach but with the IOP from [Goldwasser et al. 1989]. This makes our approach easier to formalize.

The authors of [Ben-Sasson et al. 2019a] and [Ben-Sasson et al. 2018a] use FRI implicitly as a PCS and design the ALI-IOP for compilation. Our results are directly applicable and generalize their approach. In [Boneh et al. 2020], the authors define a restricted version of PCS that are 'additive' (with homomorphic properties), benchmarking with Bulletproofs [Bünz et al. 2018] for fast recursion. While an additive PCS is too restrictive to include FRI, batching efficiency gains are possible if this definition is relaxed. The batch evaluation problem in Section 4 of [Boneh et al. 2020] is equivalent to a multivariate commitment, and the LCS is a relaxation of that notion that we selectively apply to proof witnesses. By relaxing the binding property of PCS, our LCS replaces [Kate et al. 2010] for multivariate commitments to witness polynomials more generally.

Subsequent work [Zero 2021] has built on these ideas, looking at how two modular modifications to the proof system affect performance by using a smaller field for faster modular operations and implementing a leaner PLONK-derived IOP called TURBOPLONK [Gabizon and Williamson 2020] instead. The authors explore an additional avenue for optimization of proof efficiency by counterbalancing any soundness loss due to the smaller field by applying the tight parallel repetition theorem to the IOP in order to boost soundness and performing FRI in a suitable extension field. This is then shown to lend itself to efficient recursive proving times, providing a promising avenue for transparent recursive proof computations at scale.

11 | Overview

PLONK [Gabizon et al. 2019] is based on polynomial commitments: the prover's (secret) witness is encoded as a set of univariate polynomials, while the verifier wishes to ensure this encoding satisfies some polynomial relations. The prover commits to her witness polynomials and later the verifier queries their values at a set of randomly selected points, checking if all relations are indeed satisfied. As the points were randomly sampled, it is highly likely that the given polynomial relations hold identically.

The state-of-the-art PCS used in the construction of zk-SNARKs is the Kate commitment [Kate et al. 2010], which is based on pairings of points of elliptic curves. The security of this scheme reduces to the Discrete Logarithm assumption, while in the case of a perfectly hiding commitment the t-Strong Diffie Hellman assumption is required. The main drawback of Kate commitments is that some secret value is sampled during the parameter generation process. For security to hold, this value should never be revealed to the prover and verifier. Such a requirement is very strong, as it means that every proof system using Kate will require a 'trusted' setup process. We denote proof systems without this requirement as *transparent*. In fact, the only reason PLONK requires a trusted setup is due to Kate. In this work we aim to investigate a suitable replacement for Kate, turning PLONK (and similar systems) into zero-knowledge Succinct Transparent ARguments of Knowledge, i.e. zk-STARKs.

We utilize the FRI protocol, which is a key component of STARKs such as [Ben-Sasson et al. 2019a] and [Ben-Sasson et al. 2018c]. FRI is focused on solving the following *proximity* problem:

the verifier is given oracle access to an evaluation of some function f on a fixed domain $D \subset \mathbb{F}$. The prover wants to convince the verifier that this function f is close (in some metric) to a polynomial of (predefined) degree d. If the verifier queries f on all of D and then computes the interpolation polynomial herself, she can verify the degree bound d. However, this requires $O(d)$ complexity. FRI solves this problem by requiring only a polylogarithmic number of queries in d.

We now describe a naive way to design a PCS using FRI:

1. The prover commits to the evaluations of f on some predefined domain D.

2. The prover and verifier engage in FRI for f with respect to some degree d. If the prover passes the check, the verifier is convinced with high probability that f is close to a polynomial of degree less than d.

3. The verifier wants to retrieve the value of f at point $i \notin D$. The prover sends the corresponding opening $z = f(i)$ and both parties conduct an instance of FRI with respect to a quotient function $q(X) = (f(X) - z)/(X - i)$ and degree $d - 1$. Note that the verifier has oracle access to q via oracle access to f and also knows i and z. If the prover passes the last instance of FRI then $q(X)$ is in fact a polynomial function of degree less than $d - 1$.

4. This implies that $f(i) = z$ which follows from Bézout's theorem stating that $h(X)$ has value y at point t iff $h(X) - y$ is divisible by $X - t$ in the ring $\mathbb{F}[X]$.

In reality, this simplified protocol doesn't suffice. There are several reasons for that, among which are the following:

1. FRI has a sensitivity bound: it is incapable of distinguishing between precise polynomials and functions sufficiently close to them in some predefined metric (which in our case is the relative Hamming distance).

2. For implementation coherency, we want the same domain for both FRI instances. However, FRI has an interdependence between the degree d and the size of the domain $|D|$ measured

in terms of its rate $\rho = d/|D|$. The structure of FRI requires the rate to be "2-adic", i.e. of the form 2^{-R} for some $R \in \mathbb{N}$. However, this property cannot simultaneously hold for two adjacent degrees d and $d-1$ without protocol modification.

The first problem means that the scheme needs to correctly process the case when the function is not a polynomial, but close to one; a property not naturally supported by existing commitment schemes. Even more so, allowing the oracle f to not be strictly polynomial and to take as the prover's commitment the polynomial f' lying in a small δ-ball around f (where δ is taken according to the sensitivity of FRI) then we cannot guarantee *a priori* that this polynomial f' is unique in the chosen neighborhood of f. The set of polynomials $\{f'_1, f'_2, \ldots, f'_n\}$ lying in the δ-neighborhood of f is the *δ-list* of f, which we denote by $L_\delta := L_\delta(f)$. For small values of δ, the list L_δ contains only one polynomial: δ lies in the *unique-decoding radius*. Unfortunately such values of δ require larger proof sizes for the same soundness guarantees. Thus, increasing δ to reduce proof sizes would lead to the size L_δ being greater than 1.

To solve this, we consider a *relaxed* treatment of commitment schemes, where the commitment opens to a polynomial in the δ-list L_δ. When the prover is asked for an evaluation at point i, they respond with some value $f'(i)$, where $f' \in L_\delta$. In subsequent sections we show that this scheme is sufficient for the compilation of holographic IOPs. During the execution of PLONK, the prover and verifier need to evaluate a set of initial 'constraint' (or setup) polynomials $c(X)$ encoding the constraint system itself. In order to achieve succinctness, the verifier never calculates the value $c(i)$ at point i by herself. PLONK instead relies on Kate: the prover and verifier run Kate with the commitment to c and value i as inputs. By its binding property, the verifier is convinced that the prover actually sends the evaluation $c(i)$ of the polynomial $c(X)$ in question. Since our relaxation commits to a whole neighborhood $L_\delta(c)$ of $c(X)$ instead of only $c(X)$ itself, we lose uniqueness. This means we can't use such a 'relaxed' scheme as is. However, we show that with minor changes our LPC can be turned into a PCS. We call this construction a polynomial evaluation scheme and it constitutes the second key sub-protocol of the paper.

With the list polynomial commitments and polynomial evaluation schemes, we can modify PLONK to achieve full transparency. We call the modified version RedShift and prove its correctness in the IOP model. A large portion of our approach remains the same as in [Gabizon et al. 2019]: our modification doesn't modify the completeness property of the system. However, the FRI-based protocol doesn't possess the hiding capabilities of Kate. This means that we need to take additional measures to achieve zero-knowledge for our system. We also need to change the security model as the original PLONK protocol was proven secure in the Algebraic Group Model (AGM) [Fuchsbauer et al. 2018]. The dependence of our scheme on FRI means that we need to conduct our security analysis in the IOP model, affecting the soundness proof as well as the proof of knowledge approaches.

11.1 Definitions

In this section, we lay out the building blocks that are necessary to describe our constructions. We use the following notation throughout:

- $\mathbb{F}_q$ is a prime field with modulus q
- $D \subset \mathbb{F}$ evaluation domain for Reed Solomon codes
- $f|_D$ is a restriction of function f to domain D
- For function pair f, g, the relative Hamming distance with respect to some domain D is given by:

$$\Delta(f, g) = \frac{|\{x \in D : f(x) \neq g(x)\}|}{|D|}.$$

11.2 Reed-Solomon codes

Most of the information covered in this section can be found in most standard on algebraic codes (e.g. [Huffman and Pless 2003]).

Definition 11.1 (Reed-Solomon Codes)**.** For a subset of some field $D \subseteq \mathbb{F}$ and a rate parameter $\rho \in (0, 1]$, we denote by $\mathsf{RS}[\mathbb{F}, D, \rho]$ the set of all functions $f : D \to \mathbb{F}$ for which there exists $\hat{f} \in \mathbb{F}_{<\rho|D|}[X]$ agreeing with f on D. A *prime field RS code family* is a code family $\mathsf{RS}[\mathbb{F}, D, \rho]$ for which $\mathbb{F} = \mathbb{F}_q$ for q prime. In this case, D is a multiplicative subgroup of $\mathbb{F}_q^*$.

Definition 11.2 (List Decoding)**.** Let $V = \mathsf{RS}[\mathbb{F}, D, \rho] \subset \mathbb{F}^D$ be an RS code family. Set a distance parameter $\delta \in [0, 1]$. For $u \in \mathbb{F}^D$, we define $L(u, V, \delta)$ to be the set of elements in V that are at most δ-far from u in relative Hamming distance. The code V is said to be (δ, N)-list-decodable if $|L(u, V, \delta)| \leq N$ for all $u \in \mathbb{F}_q^D$. Let $L_\delta^{\max} = L(\mathbb{F}, D, d, \delta)$ be the maximum size of $L(u, V, \delta)$ taken over all $u \in \mathbb{F}^D$ for $V = \mathsf{RS}[\mathbb{F}, D, \rho = d/|D|]$.

Theorem 11.3 (Johnson Bound)**.** *For every $\rho > 0$, there exists a constant C_ρ such that the code family $\mathsf{RS}[\mathbb{F}, D, \rho]$ is list-decodable from a $1 - \sqrt{\rho} - \epsilon$ fraction of errors with the following list size:*

$$L(\mathbb{F}, D, \rho|D|, 1 - \sqrt{\rho} - \epsilon) \leq \frac{C_\rho}{\epsilon\sqrt{\rho}} := J_{\rho,\epsilon},$$

for every $\epsilon \in (0, 1 - \sqrt{\rho})$.

We also provide a (strong) conjecture that substantially improves on the above bound, which appears in [Ben-Sasson et al. 2019b].

Conjecture 11.4 (List Decodability up to Capacity)**.** *For every $\rho > 0$, there exists a constant C_ρ such that the code family $\mathsf{RS}[\mathbb{F}, D, \rho]$ is list-decodable from a $1 - \rho - \epsilon$ fraction of errors with the following list size:*

$$L(\mathbb{F}, D, \rho|D|, 1 - \rho - \epsilon) \leq \left(\frac{|D|}{\epsilon}\right)^{C_\rho},$$

for every $\epsilon \in (0, 1-\rho)$.

We now look at which distance parameters δ provide unique decodability. To this end, we provide some standard results on the unique decodability of RS codes.

Definition 11.5. We call δ_0 the *unique decoding radius* (UDR) for code family C if it is the maximum δ_0 for which $L_{\delta_0}^{\max} \leq 1$. We denote all $\delta < \delta_0$ as being within the unique decoding radius.

Theorem 11.6. *The UDR for* $\mathsf{RS}[\mathbb{F}, D, \rho]$ *is* $\delta_0 = (1-\rho)/2$.

The decoding problem for the Reed-Solomon code $V = \mathsf{RS}[\mathbb{F}, D, \rho]$ is the problem of finding a codeword $u \in V$ that is within Hamming distance δ of a given word $v \in \mathbb{F}^D$. There exists a standard polynomial-time solution known as the *Guruswami-Sudan* [Guruswami and Sudan 1999] algorithm. Its output includes *all* codewords lying in the δ-ball of v.

Theorem 11.7 (Guruswami-Sudan)**.** *For all* $\delta \leq 1 - \sqrt{\rho}$*, the code* $V = \mathsf{RS}[\mathbb{F}, D, \rho]$ *can be list-decoded in time* $O(|D|^{15})$*. If* $\delta < 1 - \sqrt{\rho}$*, this reduces to* $O(|D|^3)$.

11.3 Interactive Oracle Proofs

Given some relation $\mathcal{R} \subseteq S \times T$, we denote by $\mathcal{L}(\mathcal{R}) \subseteq S$ the set of $s \in S$ such that there exists $t \in T$ with $(s, t) \in \mathcal{R}$ (also known as the *language* defined by $\mathcal{R}$). We also denote by $\mathcal{R}|_s \subseteq T$ the set $\{t \in T : (s, t) \in \mathcal{R}\}$. For pairs $(x, w) \in \mathcal{R}$, we call x the instance and w the witness.

The security analysis in this section will be conducted in the *Interactive Oracle Proof* (IOP) model [Ben-Sasson et al. 2016] which is a generalization of Interactive Proofs and Probabilistically Checkable Proofs. More specifically, we will be looking at *holographic* IOPs, or IOPs in which (preprocessed) indices are provided to the participating parties through oracles. The model consists of a prover/verifier tuple (P, V) of two probabilistic interactive algorithms. The number of interactive rounds, denoted $k = r(x)$, is called the *round complexity* of the system. During a

single round, the prover P sends a message a_i (which may depend on prior interaction) to which the verifier V provides some response m_i. The final output of V after interacting with P is either *accept* or *reject*. We denote the result of this interaction by $\langle P(x, w) \leftrightarrow V(x) \rangle$, where the input to V is $x \in S$ and the input to P is $(x, w) \in S \times T$. The proof length is the sum of lengths of all messages sent by the prover, herein denoted $l(x) = \sum_{i=1}^{k} a_i$. The query complexity of the protocol, denoted $q(x)$, is the total number of entries read by V.

Definition 11.8. A pair of interactive PPT algorithms (P, V) is an interactive oracle proof system for some language $\mathcal{R} \subseteq S \times T$ with $\epsilon : \{0,1\}^* \to [0,1]$ soundness and $k : \{0,1\}^* \to \mathbb{N}$ rounds of interaction if it satisfies the following properties:

1. **Completeness:**

$$Pr\left[\langle P(x, w) \leftrightarrow V(x) \rangle = \text{Accept} \mid (x, w) \in \mathcal{R}\right] = 1$$

2. **Soundness:** For all computationally unbounded malicious provers P^*:

$$Pr\left[\langle P^*(x, w) \leftrightarrow V(x) \rangle = \text{Accept} \mid (x, w) \notin \mathcal{R}\right] \leq \epsilon(x).$$

   Probabilities are over the randomness of P and V, which engage in at most $k(x)$ rounds of (adaptive) interaction.

Definition 11.9. Let A, B be PPT algorithms, $x, y \in \{0,1\}^*$ and $\mathsf{View}(B(x, y), A(x))$ the *view* (or *transcript*) of $A(x)$ in an IOP with $B(x, y)$. This is the random variable $(x, r, \{a_i, m_i\}_{i=1}^{n})$ where x, r are A's input and randomness and a_i is B's (i-th) answer to A's query m_i.

State-Restoration Knowledge Soundness Strengthening the notion of soundness, we say the IOP has knowledge soundness $e : \{0,1\}^* \to [0,1]$ if every prover P^* who is capable of convincing the verifier that $x \in \mathcal{L}(\mathcal{R})$ actually knows some witness $w \in \mathcal{R}|_x$. Put differently, the IOP

is knowledge sound if for all adversaries $\mathcal{A}$ there exists a (non-uniform) PPT extractor $\mathcal{E}^{\mathcal{A}}(x)$ which gets full access to the adversary's transcript at any stage. However, this does not include $\mathcal{A}$'s random coins, auxiliary inputs and internal code. We say that (P, V) has proof of knowledge ϵ if there exists $\mathcal{E}$ s.t. for every $x \in S$ and PPT $\mathcal{A}$:

$$\Pr\left[(x, \mathcal{E}^{\mathcal{A}}(x)) \in \mathcal{R}\right] \geq \Pr\left[(\mathcal{A}, V) = 1\right] - e(x).$$

Since we are ultimately interested in compiling the IOP into a non-interactive proof, the stronger notion of state-restoration IOP soundness error $\epsilon_{\mathsf{sr}}(r)$, where r the maximal number of rounds, is needed. This is because the protocol should be robust against state-restoration attacks, in which the prover has the ability to move to a previous state of the protocol up to r times. It was shown in [Ben-Sasson et al. 2016] that this notion suffices in compiling proofs using Fiat-Shamir in the random oracle model, while [Canetti et al. 2018] show this for correlation-intractable hash functions as well. In order to prove state-restoration bounds, the idea of round-by-round soundness error ϵ_{rbr} is leveraged. It can be shown that $\epsilon_{\mathsf{sr}} \leq r \cdot \epsilon_{\mathsf{rbr}}$, which is sufficient for security if r is a polynomial number of rewinds/rounds. This holds since we can apply the round-by-round extractor to every partial transcript that comprises $\mathsf{tr}_{\mathsf{sr}}$ and output the first valid witness. Since the empty transcript is rejecting and $\mathsf{tr}_{\mathsf{sr}}$ accepts, then some partial transcript moves from rejecting to accepting. The round-by-round extractor fails with probability ϵ_{rbr}, so the result follows by a union bound. We follow the approach in [Canetti et al. 2018] and provide the required definitions.

Definition 11.10 (Round-by-Round Soundness)**.** An IOP (P, V) for language $\mathcal{L}(\mathcal{R})$ has round-by-round knowledge soundness ϵ_{rbr} if there exists a function State from the set of transcripts to $\{0, 1\}$ and a polynomial-time extractor $\mathcal{E}$ such that for all (x, tr) for which $\mathsf{State}(x, \mathsf{tr}) = 0$, and all messages $\mathbf{a}$ received from the prover, if $\Pr_m[\mathsf{State}(x, \mathsf{tr}|\mathbf{a}|m) = 1] > \epsilon_{\mathsf{rbr}}$ then $(x, \mathcal{E}(x, \mathsf{tr}|\mathbf{a})) \in \mathcal{R}$.

Definition 11.11 (State-Restoration Soundness)**.** An IOP (P, V) for language $\mathcal{L}(\mathcal{R})$ has state-restoration knowledge soundness ϵ_{sr} if there exists a polynomial time extractor $\mathcal{E}$ such that for

all x and every state-restoration prover P^*:

$$\Pr\left(\begin{array}{c|c} \mathsf{tr}_{\mathsf{sr}} \text{ accepts} & \mathsf{tr}_{\mathsf{sr}} \leftarrow \mathsf{View}(P^*(x), V(x)) \\ (x,w) \notin \mathcal{R} & w \leftarrow \mathcal{E}(x, \mathsf{tr}_{\mathsf{sr}}) \end{array}\right) \leq \epsilon_{\mathsf{sr}}.$$

Definition 11.12 (Zero Knowledge)**.** For a given relation $\mathcal{R}$ and some $z : \{0,1\}^* \to [0,1]$, $\langle P, V\rangle$ has z-statistical honest-verifier zero knowledge if there exists a PPT algorithm S (the *simulator*) s.t. $\forall(\phi, w) \in \mathcal{R}$, $S(x, \phi)$ and $\mathsf{View}(P(\phi, w), V(\phi))$ are $z(x)$-close.

An important subclass of IOP protocols is given below.

IOPP. An *Interactive Oracle Proof of Proximity (IOPP)* is an r-round interactive IOP system for the following problem. Given a field $\mathbb{F}$, degree $d \in \mathbb{N}$, proximity parameter $\delta > 0$ and domain $D \subset \mathbb{F}$, the prover is provided with the representation of some function f and the verifier is given oracle access to its evaluation on domain D (i.e. an oracle $\hat{f}(x)$ to $f(x)|_D$). The prover then needs to convince the verifier that $f|_D$ is the evaluation of some degree d polynomial on this domain. Namely, that $f \in \mathsf{RS}[\mathbb{F}, D, \rho = d/|D|]$. We follow the formalization in [Ben-Sasson et al. 2018a]:

Definition 11.13 (IOPP)**.** An r-round IOP of Proximity (P, V) is an $r+1$-round IOP. (P, V) is an IOPP for the error-correcting code $C = \{f : S \to \Sigma\}$ and soundness $\epsilon : [0,1] \to [0,1]$ with respect to some metric Δ if the following hold:

1. **First message format**: the first prover message is a purported code-word, i.e. $f^0 \in C$,
2. **Completeness**:

$$\Pr[\langle P(f^0, C) \leftrightarrow V(C)\rangle = \text{Accept} \mid \Delta(f^0, C) = 0] = 1,$$

3. **Soundness**: For any P^*,

$$\Pr[\left\langle P^*(f^0, C) \leftrightarrow V(C)\right\rangle = \text{Reject} \mid \Delta(f, C) = \delta] \geq \epsilon(\delta).$$

We note that the notions of proof and query complexity of IOPs translate naturally to the context of IOPPs. In the rest of the sections, we use $\mathsf{IOPP}(f^0, C) \longrightarrow \{0, 1\}$ to denote an IOPP protocol IOPP over error-correcting code family C with purported code-word $f^0 \in C$.

11.4 FRI: Fast Reed-Solomon IOP of Proximity

In our construction, we opt for using FRI [Ben-Sasson et al. 2018a,d]. We provide an overview of its relevant properties below. For a given RS code family $\mathsf{RS}[\mathbb{F}, D, \rho]$ for which $|D| = n = 2^k$ and rate $\rho = 2^{-R}$ for $k, R \in \mathbb{N}$. This implies that the degree bound d is 2^{k-R}. Fix $r \in [1, \log d = k - R]$ to be the number of rounds in the protocol. For every $\eta \in (0, 1]$, let $J_\eta : [0, 1] \longrightarrow [0, 1]$ be the Johnson function $J_\eta(x) = 1 - \sqrt{1 - x(1 - \eta)}$. Given this parameter choice, FRI has the following properties (asymptotics are in terms of field operations over $\mathbb{F}$):

1. **Prover Complexity:** $O(n)$
2. **Verifier Complexity:** $O(\log n)$
3. **Completeness:** If $f \in \mathsf{RS}[\mathbb{F}, D, \rho]$, for an honest prover the verifier always accepts.
4. **Soundness:** If $\Delta(f, \mathsf{RS}) = \delta$ and $\delta \in (0, J_\eta^{[3/2]}(1 - \rho))$, then $\forall \eta \in (0, 1]$ the soundness error $\epsilon(\delta)$ is bounded above by:

$$\frac{2 \log |D|}{\eta^3 |\mathbb{F}|} + \left(1 - \min\left\{\delta_0, \delta\right\} + \eta \log |D|\right)^l,$$

where l the number of queries the verifier performs.

12 | List Polynomial Commitment

Following Section 4.5, we introduce the main ingredient underlying the transparency of our proving system, which we call a List Polynomial Commitment (LPC) scheme. This cryptographic primitive most resembles a polynomial commitment scheme, with the main difference arising from the need to show that $g(z) = y$ where g is a polynomial in a δ neighborhood around f (in a predefined metric Δ), rather than requiring the evaluation of f itself. As before, we denote $L_\delta(f)$ as the δ-list of f or the set of all $g \in \mathsf{RS}[\mathbb{F}, D, \rho]$ such that $\Delta(f, g) < \delta$.

12.1 Specification

Here we define the generic primitive that formalizes the notion of a list commitment scheme. We will build on this construction later to show that it admits (1) an efficient implementation, and (2) modifications that provide stronger proof of knowledge guarantees.

Definition 12.1. An (ϵ, k)-list polynomial commitment scheme for some metric $\Delta : \mathbb{F}[X] \times \mathbb{F}[X] \to [0, 1]$ and all $\delta > 0$ consists of the following:

- $\mathsf{Gen}(1^\lambda) \to \mathsf{pp}$ generates public parameters,
- $\mathsf{Com} : \mathbb{F}_{<d}[X] \to C$ generates commitment c to some f,

- An IOP system (P, V) with $\epsilon(\delta)$ soundness and $k(\delta)$ rounds of interaction for the relation

$$\mathcal{R}^{\delta}(\mathsf{pp}) := \left(\langle (d, N, \{z_i, y_i\}_{i=1}^{N}, c); f \rangle \;\middle|\; \begin{array}{c} \exists\, g \in \mathbb{F}_{<d}[X],\ \Delta(f, g) < \delta, \\ \forall i \in [N], g(z_i) = y_i, \mathsf{Com}(g) = c \end{array} \right)$$

  for which (P, V) are both provided with degree bound d, and a set of point-evaluation pairs $\{(z_i, y_i)\}_{i=1}^{N}$ and commitment $c \in C$, while P is also provided with a representation of $f \in \mathbb{F}[X]$. Both P and V have access to an oracle for $\mathsf{Com}(\cdot)$.

12.2 Instantiation

We assume existence of an IOPP protocol FRI in the sense of Definition 11.13, and specify the LPC routine below. More specifically, since we are concerned with polynomial commitment schemes, we present the scheme based on the existence of FRI (Theorem 2 in [Ben-Sasson et al. 2018a]) for the prime field RS code family $C = \mathsf{RS}[\mathbb{F}, D, \rho]$, where $\rho = 2^{-\mathcal{R}}$ for some $\mathcal{R} \in \mathbb{N}$, $\mathcal{R} > 2$ and $\rho \cdot |D| > 16$. Note that we model the Com function as an oracle, so we do not deal with its security here. In this case, we set the public parameters to be $\mathsf{pp} = (\mathbb{F}, D)$:

Algorithm 3 LPC Routine

1: **procedure** $\mathsf{LPC}(\mathsf{pp}, d, N, \{z_i, y_i\}_{i=1}^{N}, c; f)$
2: P and V define the interpolation polynomial $U(X)$ s.t.

$$\forall i \in [N], U(z_i) = y_i.$$

3: P and V define the quotient polynomial

$$q(X) = \frac{f(X) - U(X)}{\Pi_{i=1}^{N}(X - z_i)}.$$

4: P and V return $\mathsf{FRI}(q, \mathsf{RS}(\mathbb{F}, D, (d - N)/|D|))$.
5: **end procedure**

The oracle provided here is to $f|_D$, which allows both parties to simulate FRI over the coset domain D by calculating the values of $q|_D$. This is since both parties explicitly construct the

interpolation polynomials and have access to $f|_D$. Hence, the verifier is above to check that $c = \mathsf{Com}(q)$ using oracle calls to $f|_D$ in order to simulate $q|_D$. That the above satisfies the definition of an LPC scheme is immediate from the security properties of FRI.

Theorem 12.2. *The prime field* $\mathsf{RS}[\mathbb{F}, D, \rho]$ *code family with rate* $\rho = 2^{-\mathcal{R}}, \mathcal{R} \geq 2, \mathcal{R} \in \mathbb{N}$ *and* $|D| = n$ *has an* (ϵ, k)*-LPC over the Hamming distance* Δ *with* $\epsilon = \epsilon_{\mathsf{FRI}}(\delta)$ *soundness and* $k = k_{\mathsf{FRI}}(\delta)$ *rounds of interaction for all* $v \in (0, 1 - \sqrt{\rho}), \delta \in (0, J_v^{[3/2]}(1-\rho)), N, d \in \mathbb{N}$ *for which* $\log(n) - \mathcal{R} = \log(d - N)$ *and* $d - N > 16$*. The LPC IOP admits inputs of size n for which:*

- *Prover Complexity:* $p_{\mathsf{FRI}}(n) + O(N \log^3 N)$,
- *Verifier Complexity:* $v_{\mathsf{FRI}}(n) + O(N \log^3 N)$,

where $p_{\mathsf{FRI}}(n), v_{\mathsf{FRI}}(n)$ *the FRI prover and verifier complexities on input size n.*

Remark 1: We note that the above scheme retains verifier succinctness when $N = O(\log d)$.

12.3 Polynomial Commitments from LPCs

The scheme introduced above works when dealing with “witness" polynomials $w(X)$ within our proof system, since we only require the existence (and not uniqueness) of such a polynomial. However, extra care should be taken outside of this regime, when working with “setup" polynomials $c(X)$ encoding the constraint system itself. In this case, we want to ensure that the openings provided by the prover are indeed the evaluations of the polynomial $c(X)$ itself and not of some polynomial $g \in L_\delta(c)$. The verifier can evaluate setup polynomial values themselves. However, this doesn't retain succinctness as evaluations require $O(d)$ computations.

We leverage the fact that for a given setup polynomial $c(X)$ the list $L_\delta(c)$ is computable by both the prover and verifier. They can hence find a *distinguishing* point i at which $c(i)$ differs from the evaluations of *all* other polynomials $g \in L_\delta(c)$. This is naively achieved by using a

list-decoding algorithm at the beginning to find all $g \in L_\delta(c)$ and then pick $i \in \mathbb{F}$ at random until $c(i) \neq g(i)\ \forall\ g \in L_\delta(c)$. This, however, has overhead polynomial in $|D|$.

The key to our approach is that the procedure of enumerating all such elements and picking a suitable candidate is (1) fully transparent, and (2) executed and verified only once for every circuit. We thus add an *offline phase* that is performed only once during Gen. The task of the offline phase is to search for such a distinguishing point i. This allows us to strengthen the proof of knowledge guarantee for the LPC to imply that all evaluations come from the *specific* polynomial $c(X)$. Note that this is equivalent to the general proof of knowledge guarantee provided by polynomial commitment schemes. We equivalently call this the *preprocessing phase*, which is analogous to the work of the *indexer* in [Chiesa et al. 2019a].

Definition 12.3. A PPT algorithm $\mathcal{D} : \mathbb{F}[X] \to (\mathbb{F} \times \mathbb{F})^\mu$ is called a μ-dimensional ϵ-list distinguisher for some metric $\Delta : \mathbb{F}[X] \times \mathbb{F}[X] \to [0, 1]$ if $\forall f \in \mathbb{F}[X], \delta > 0$ the following hold with probability $1 - \epsilon(\delta)$ over the randomness of $\mathcal{D}$:

$$\exists i \in [\mu], \forall g \in L_\delta(f) \backslash \{f\}, f(\mathcal{D}(f)_{i,1}) \neq g(\mathcal{D}(f)_{i,1}),$$

$$\forall i \in [\mu], f(\mathcal{D}(f)_{i,1}) = \mathcal{D}(f)_{i,2},$$

where $\mathcal{D}(f)_{i,j}$ the (i, j)-th output element, $i \in [\mu], j \in \{1, 2\}$.

Definition 12.4. An (ϵ, k, η)-polynomial evaluation scheme for some metric Δ is a tuple $\Pi = (\mathcal{D}, \Sigma)$ where $\Sigma = (P, V)$ an (ϵ, k)-LPC scheme and $\mathcal{D}$ an η-list distinguisher.

Theorem 12.5. *For every $\delta > 0$, an (ϵ, k, η)-polynomial evaluation scheme admits an IOP with soundness $\epsilon(\delta) + \eta(\delta)$ and $k(\delta)$ rounds of interaction for the relation*

$$\mathcal{R} := \{\langle (d, N, \{z_i, y_i\}_{i=1}^N), c; f\rangle | \forall i \in [N], f(z_i) = y_i, \mathsf{Com}(f) = c\}.$$

The above theorem relies on a simple observation: if we have access to some distinguishing

point-evaluation pair $(x, f(x))$ such that $f(x) \neq g(x)$ for all $g \in L_\delta \backslash \{f\}$, then adding $(x, f(x))$ to the openings performed by the LPC means that only f is a valid witness. This process is done once during the setup of the LPC scheme and each instantiation of an LPC that requires binding security can then retrieve these points from the proving key.

12.3.0.1 Instantiation

We now provide two instantiations of the PES based on different list distinguisher choices, and discuss the trade-offs between the two.

List Decodability: The most obvious way to construct a distinguisher $\mathcal{D}$ is by using a list-decoding algorithm for the given code to enumerate all $g \in L_\delta(f)$. By sampling random values $r \in \mathbb{F}$ and checking the required relation, this algorithm can be used to construct $\mathcal{D}$ with no soundness error but high time complexity. Indeed, for $\delta < 1 - (d - N - \mu)/|D|$ the algorithm takes $O(|D|^3)$ time, while in the case of equality between the two this can go up to $O(|D|^{15})$!

Random Sampling: Due to the above inefficiency considerations, we opt to add some soundness error to the distinguisher in exchange for a large increase in the efficiency of the protocol. We construct the distinguisher by sampling μ random points and simply returning them along with their evaluations. Due to the Schwartz-Zippel lemma, there is a high chance that the random points will indeed separate f from its corresponding δ-list. This takes time $O(\mu \cdot d)$ which, although linear with respect to d, is substantially faster than list-decoding when $d \sim |D|$.

Claim 12.6. *The random sampling algorithm defines an η-list distinguisher $\mathcal{D}$ taking time $O(\mu \cdot d)$ with soundness error:*

$$\eta(\delta) = \left(\frac{d \cdot (|L_\delta| - 1)}{|\mathbb{F}|} \right)^\mu .$$

We can use the LPC scheme constructed in the previous section along with the distinguisher defined above to put everything together in the following theorem.

Theorem 12.7. *The prime field* $\mathsf{RS}[\mathbb{F}, D, \rho]$ *code family with rate* $\rho = 2^{-\mathcal{R}}, \mathcal{R} \geq 2, \mathcal{R} \in \mathbb{N}$ *and* $|D| = n$ *has an* (ϵ, k, η)*-PES* $\Pi = (\mathcal{D}, \Sigma)$ *over the Hamming distance* Δ.

Σ *is an* (ϵ, k)*-LPC scheme with* $\epsilon = \epsilon_{\mathsf{FRI}}(\delta)$ *soundness and* $k = k_{\mathsf{FRI}}(\delta)$ *rounds of interaction for* $\nu \in (0, 1 - \sqrt{\rho}), \delta \in (0, J_\nu^{[3/2]}(1-\rho)), N, d, \mu \in \mathbb{N}$ *for which* $\log(n) - \mathcal{R} = \log(d - N - \mu)$ *and* $d - N - \mu > 16$. Σ *also admits inputs of size n for which:*

- *Prover Complexity:* $p_{\mathsf{FRI}}(n) + O((N+\mu)\log^3(N+\mu))$,
- *Verifier Complexity:* $v_{\mathsf{FRI}}(n) + O((N+\mu)\log^3(N+\mu))$,

where $p_{\mathsf{FRI}}(n), v_{\mathsf{FRI}}(n)$ *the FRI prover and verifier complexities on input size n.*

$\mathcal{D}$ *is a* μ*-dimensional* η*-list distinguiser for the* $\mathsf{RS}[\mathbb{F}, D, \rho]$ *code family taking* $O(\mu \cdot d)$ *and soundness error:*

$$\eta(\delta) = \left(\frac{d}{|\mathbb{F}|} \cdot (J_{\rho,\nu} - 1)\right)^{\mu}.$$

This provides the proof of knowledge guarantees needed for setup polynomials through direct application of Theorem 12.5.

13 | RedShift

13.1 Constraint System

We follow the language and notation of PLONK [Gabizon et al. 2019].

Definition 13.1 (PLONK Constraint System)**.** $\mathcal{L} = (\mathcal{V}, \mathcal{Q})$ is a constraint system with n gates and m wires for which $n \leq m \leq 2n$ and where:

- $\mathcal{V}$ is of the form $\mathcal{V} = (\mathbf{a}, \mathbf{b}, \mathbf{c})$, where $\mathbf{a}, \mathbf{b}, \mathbf{c} \in [m]^n$,
- $\mathcal{Q}$ is of the form

$$\mathcal{Q} = (\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C}) \in (\mathbb{F}^n)^5,$$

  where $\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C}$ are the “selector" vectors.

Moreover, $\mathbf{x} \in \mathbb{F}^m$ is said to satisfy $\mathcal{L}$ if $\forall i \in [n]$:

$$(\mathbf{q_L})_i \cdot \mathbf{x}_{\mathbf{a}i} + (\mathbf{q_R})_i \cdot \mathbf{x}_{\mathbf{b}i} + (\mathbf{q_O})_i \cdot \mathbf{x}_{\mathbf{c}i} +$$
$$+(\mathbf{q_M})_i \cdot (\mathbf{x}_{\mathbf{a}i} \cdot \mathbf{x}_{\mathbf{b}i}) + (\mathbf{q_C})_i = 0.$$

To define a relation based on $\mathcal{L}$, we extend it to include a positive integer $l \leq m$, and subset $I = [l] \subseteq [m]$ of “public inputs”. We can naturally set $\mathcal{R}_{\mathcal{L}}$ as the set of pairs (x, ω) with $x \in$

$\mathbb{F}^l, \omega \in \mathbb{F}^{m-l}$ such that $\mathbf{x} := (x, \omega)$ satisfies $\mathcal{L}$. We say $\mathcal{L}$ is 'prepared' for l public inputs if $\forall i \in [l]$:

$$\mathbf{a_i} = i,\ (\mathbf{q_L})_i = 1,\ (\mathbf{q_M})_i = (\mathbf{q_R})_i = (\mathbf{q_O})_i = (\mathbf{q_C})_i = 0.$$

From here on, we will assume that the constraint system is given in prepared form.

In order to reformulate this constraint system in polynomial terms, we require some additional ingredients. Let $g \in \mathbb{F}^*$ be an element of order $n+1$, $D = \langle g \rangle \subseteq \mathbb{F}^*$ the cyclic subgroup generated by g, and $D^* := D/\{e\}$ where $e = g^0$ the identity. For $i \in [n+1]$, denote by $L_i(X)$ the element of $\mathbb{F}_{\leq n}[X]$ with $L_i(g^i) = 1$ and $L_i(a) = 0$ for all $a \in D$ different to g^i. By construction, $\{L_i(X)\}_{i=1}^{n+1}$ form a Lagrange basis for D. Finally, we set $Z(X) := \prod_{a \in D^*}(X - a) \in \mathbb{F}_{\leq n}[X]$ to be a domain polynomial for D^*, i.e. zero only on D^*.

Definition 13.2 (Domain Permutations)**.** For sets of k polynomials $\{f_i\}_{i=1}^k, \{h_i\}_{i=1}^k$ for which $h_i, f_i \in \mathbb{F}[X]$ and permutation $\sigma : [kn] \to [kn]$, we say that the set $(h_1, ..., h_k) = \sigma(f_1, ..., f_k)$ if, for all $l \in [kn]$, the sequences $\left(f_{(1)}, \ldots, f_{(kn)}\right), \left(h_{(1)}, \ldots, h_{(kn)}\right) \in \mathbb{F}^{kn}$, defined as:

$$f_{((j-1)\cdot n+i)} := f_j(g^i),\ h_{((j-1)\cdot n+i)} := h_j(g^i),$$

for each $j \in [k], i \in [n]$, satisfy $h_{(l)} = f_{(\sigma(l))}$.

Definition 13.3. Let $\mathcal{T} = \{T_i\}_{i=1}^s$ be a partition of $[kn]$ into s disjoint blocks, where $k, n, s \in \mathbb{N}$. We say that $\{f_i\}_{i=1}^k \in \mathbb{F}[X]$ *copy-satisfies* $\mathcal{T}$ if, when defining $(f_{(1)}, \ldots, f_{(kn)}) \in \mathbb{F}^{kn}$ as above, we have $f_{(l)} = f_{(l')}$ whenever $\exists i$ s.t. $l, l' \in T_i$.

We define a permutation $\sigma(\mathcal{T})$ on $[kn]$ such that for each block T_i of $\mathcal{T}$, $\sigma(\mathcal{T})$ contains a cycle only going over all the elements of T_i. Many possibilities exist: for example, we can rearrange elements in the cycles corresponding to T_i, or $\sigma(\mathcal{T})$ can be chosen arbitrarily from the set of all valid permutations. It is simple to check that $(f_1, \ldots, f_k)$ copy-satisfies $\mathcal{T}$ if and only if $(f_1, \ldots, f_k) = \sigma(f_1, \ldots, f_k)$. We can thus equivalently say that $(f_1, \ldots, f_k)$ copy-satisfy σ.

Definition 13.4. Fix domain D^* and size parameter $n \in \mathbb{N}$. The constraint system $\mathcal{L}'$ is defined as follows:

$$\mathcal{L}' := (\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C}, \sigma, n),$$

where:

- $\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C} \in \mathbb{F}[X]$ the selector polynomials,
- $\sigma : [3n] \to [3n]$ a permutation over $3n$ elements.

The relation $\mathcal{R}_{\mathcal{L}'}$ for $\mathcal{L}'$ is defined as the set

$$(x, \omega) := (\mathbf{PI}(X), \langle \mathbf{f_L}(X), \mathbf{f_R}(X), \mathbf{f_O}(X) \rangle) \in \mathbb{F}[X] \times (\mathbb{F}[X])^3,$$

with the following properties:

1. $\mathbf{f_L}(X), \mathbf{f_R}(X), \mathbf{f_O}(X)$ copy-satisfy σ.
2. $\forall a \in D^*$ it holds that:

$$\mathbf{q_L} \cdot \mathbf{f_L} + \mathbf{q_R} \cdot \mathbf{f_R} + \mathbf{q_O} \cdot \mathbf{f_O} + \mathbf{q_M} \cdot \mathbf{f_L} \cdot \mathbf{f_R} + (\mathbf{q_C} + \mathbf{PI}) = 0.$$

For completeness we also provide a formal statement of the equivalence between the two constraint systems, whose proof can be found in the appendix. $\mathbf{PI}(X)$ is the *public input* polynomial and encodes public data that is used to define the predicate of choice, while $\mathbf{f_L}(X), \mathbf{f_R}(X), \mathbf{f_O}(X)$ are the *left*, *right* and *output* wire polynomials respectively and encode prover-only private data.

Lemma 13.5. *The constraint systems proposed in Definition 13.1 and Definition 13.4 are equivalent.*

Remark: Note that the degrees of $\mathbf{f_L}, \mathbf{f_R}, \mathbf{f_C}$ are $n - 1$, where n is the size of $\mathcal{L}$. However in RedShift this is relaxed to some degree $k > n$ to attain zero-knowledge.

13.2 IOP Protocol

Let $\mathcal{L}' = (\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C}, \sigma, n)$ be the constraint system in question. Define $k_1, k_2, k_3 \in \mathbb{F}^*$ to be representations of different cosets in $\mathbb{F}^* \backslash D$, with $k_1 = e$ set as the identity. Let τ be the bijection between the sets $P_1 = [3n]$ and $P_2 = D^* \cup k_2 D^* \cup k_3 D^*$ defined by:

$$\tau[n \cdot (j-1) + i] = k_j g^i, \ i \in [n], j \in [3].$$

Since σ is a permutation on P_1, $\sigma' = \tau \circ \sigma \circ \tau^{-1}$ is a permutation on P_2. Define $\{S_{id_i}(X)\}_{i=1}^3, \{S_{\sigma_j}(X)\}_{j=1}^3$ each of degree at most n as the set of *permutation* polynomials as follows:

1. $S_{id_j}(X) = k_j X$ for $j \in [3]$,
2. $S_{\sigma_j}(g^i) = \sigma'(k_j g^i)$ for $i \in [n], j \in [3]$.

These will be used as part of the setup polynomials to define the problem instance.

Setup & Witness Polynomials: We can now draw the link between the "setup" and "witness" polynomials of the previous sections with their concrete definitions in the instantiation of RedShift. The selector polynomials $\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C}$, permutation polynomials $\{S_{id_i}\}_{i\in[3]}, \{S_{\sigma_j}\}_{j\in[3]}$ and Lagrange-basis polynomials $\{L_i\}_{i\in[n+1]}$ play the role of "setup" polynomials. Moreover, the wire polynomials $\mathbf{f_L}, \mathbf{f_R}, \mathbf{f_O}$ form the set of "witness" polynomials. This choice makes intuitive sense: the former fully specify the relation in question and hence need to be unique in order to prevent malleability in the proof of knowledge guarantees. The latter (which tend to be much larger in most practical deployments) however, do not require uniqueness guarantees.

We are now ready to specify the RedShift protocol. At the interactive level, it is most similar to the DEEP-ALI protocol, but for the PLONK constraint system. We instantiate the distinguisher oracle $O^{\mathcal{D}}$ queries as evaluations by an indexer algorithm $\mathcal{I}$ that receives a low-degree polyno-

mial input f and outputs μ separation points along with their evaluations $\{x_i, f(x_i)\}_{i=1}^{\mu}$. We can implement the random sampling distinguisher in the non-interactive setting using $n_c \cdot \mu \cdot n$ queries (where n_c the number of constraint polynomials), and provide the set of points as input to the IOP for the PLONK proof system (c.f. Section 7 in [Gabizon et al. 2019]).

In the interests of modularity and ease of exposition, we make use of an (ϵ, k)-list polynomial commitment scheme LPC while simulating an (ϵ, k, η)-polynomial evaluation scheme $PES = (\mathcal{D}, LPC)$ with access to a μ-dimensional η-distinguisher $\mathcal{D}$. The idea is to replace all instances of commitment to some low-degree polynomial, which is required for the knowledge soundness guarantees of the proof system, to commitment using the LPC and PES schemes above. These will be implemented using the FRI protocol, but at the maximal levels of δ in order to achieve improvements in proof size. We identify two types of polynomials with differing knowledge guarantee requirements. These are the witness and setup polynomials mentioned above. In the first, we only require the existence of *some* low degree polynomial to exist, while in the latter the prover needs to know the specific polynomial they are providing commitments to.

In the construction below, we assume that $\{S_{id_i}\}_{i\in[3]}$, $\{S_{\sigma_j}\}_{j\in[3]}$, $\{L_i\}_{i\in[n+1]}$, $\mathbf{q_L}$, $\mathbf{q_R}$, $\mathbf{q_O}$, $\mathbf{q_M}$, $\mathbf{q_C} \in$ pp have been precomputed over D^* and can be accessed through commitments provided by an oracle, along with a representation of the public input polynomial $\mathbf{PI}(X)$ given to the verifier. To highlight the above difference, we explicitly instantiate the commitments that need to be sent at each step and show how these are used in the PES and LPC schemes. In the IOP formalism, this is equivalent to sending the respective polynomials to a trusted intermediary $\mathcal{I}$.

Completeness holds because for honest provers the F_i are identically zero on domain D^*, which means that all the $F_i(X)$ are divisible by $Z(X)$ in the ring $\mathbb{F}[X]$, hence so is their linear combination $F(X) = \sum_{i=1}^{6} a_i F_i(X)$. Note that the $\{F_i\}_{i\in[5]}$ are responsible for checking the copy-satisfiability of the witness polynomials. This is directly equivalent to the completeness argument in PLONK and a full proof can be found in Appendix A.6.

Algorithm 4 RedShift Routine

procedure RedShift(pp, $\mathbf{PI}, n, k, N; \mathbf{f_L}, \mathbf{f_R}, \mathbf{f_O}$)

P chooses masking polynomials $h_1, h_2, h_3 \in_R \mathbb{F}_{<k}[X]$.

P defines masking witness polynomials and sends commitments $\mathsf{cm}_i, i \in [3]$ for them to V:

$$f_1(X) := \mathbf{f_L}(X) + h_1(X)Z(X), f_2(X) := \mathbf{f_R}(X) + h_2(X)Z(X), f_3(X) := \mathbf{f_O}(X) + h_3(X)Z(X).$$

V sends random $\beta, \gamma \in \mathbb{F}$ to P. For $j \in [3]$, P computes

$$p_j := f_j + \beta \cdot S_{id_j} + \gamma, q_j = f_j + \beta \cdot S_{\sigma_j} + \gamma, p'(X) := \prod_{j \in [3]} p_j(X), q'(X) := \prod_{j \in [3]} q_j(X).$$

P computes $P, Q \in \mathbb{F}_{<n+1}[X]$ such that $P(g) = Q(g) = 1$ and for $i \in [n+1] \backslash \{1\}$:

$$P(g^i) = \prod_{1 \le j < i} p'(g^j), Q(g^i) = \prod_{1 \le j < i} q'(g^j).$$

V sends random $a_1, \ldots, a_6 \in \mathbb{F}$ to P. P defines the following polynomials:

- $F_1(X) = L_1(X)(P(X) - 1)$ and $F_2(X) = L_1(X)(Q(X) - 1)$
- $F_3(X) = P(X)p'(X) - P(X \cdot g)$ and $F_4(X) = Q(X)q'(X) - Q(X \cdot g)$
- $F_5(X) = L_n(X)(P(X \cdot g) - Q(X \cdot g))$
- $F_6(X) = \mathbf{q_L}(X)f_L(X) + \mathbf{q_R}(X)f_R(X) + \mathbf{q_O}(X)f_O(X) + \mathbf{q_M}(X)f_L(X)f_R(X) + (\mathbf{q_C}(X) + \mathbf{PI}(X))$

P defines $F = \sum_{i=1}^{6} a_i F_i$ and computes $T(X) = F(X)/Z(X)$, sending V a commitment cm_6 to T.

V sends P a random evaluation point $y_m \in \mathbb{F} \backslash D, m \in [N]$.

P responds with N sets of points, where $m \in [N], \mathbf{j} \in \{\mathbf{L}, \mathbf{R}, \mathbf{O}, \mathbf{M}, \mathbf{C}\} := \mathbf{J}$:

$$T(y_m), P(y_m), Q(y_m), \{f_i(y_m)\}_{i=1}^3, \{S_{id_i}(y_m)\}_{i=1}^3, \{S_{\sigma_i}(y_m)\}_{i=1}^3, \{\mathbf{q_j}(y_m)\}_{\mathbf{j} \in \mathbf{J}}.$$

P and V engage in sub-protocols, outputting 0 if any fail:

$$\mathsf{PES}(\mathsf{pp}, n+1, N, \{(y_m, \mathbf{q_j}(y_m))\}_{m \in [N]}, \mathsf{cm}_{\mathbf{q_j}}; \mathbf{q_j}), \mathbf{j} \in \mathbf{J},$$

$$\mathsf{PES}(\mathsf{pp}, n+1, N, \{(y_m, S_{id_j}(y_m))\}_{m \in [N]}, \mathsf{cm}_{\mathsf{id_j}}; S_{id_j}), j \in [3],$$

$$\mathsf{PES}(\mathsf{pp}, n+1, N, \{(y_m, S_{\sigma_j}(y_m))\}_{l \in [N]}, \mathsf{cm}_{\sigma_j}; S_{\sigma_j}), j \in [3],$$

$$\mathsf{LPC}(\mathsf{pp}, n+1, N, \{(y_m, f_i(y_m))\}_{m \in [N]}, \mathsf{cm_i}; f_i), i \in [3],$$

$$\mathsf{LPC}(\mathsf{pp}, n+1, 2N, \{(y_m, P(y_m)), (y_m \cdot g, P(y_m \cdot g))\}_{m \in [N]}, \mathsf{cm}_4; P),$$

$$\mathsf{LPC}(\mathsf{pp}, n+1, 2N, \{(y_m, Q(y_m)), (y_m \cdot g, Q(y_m \cdot g)\}_{m \in [N]}, \mathsf{cm}_5; Q),$$

$$\mathsf{LPC}(\mathsf{pp}, 3n+1, N, \{(y_m, T(y_m))\}_{m \in [N]}, \mathsf{cm}_6; T).$$

$\forall m \in [N]$ V computes $\{F_i(y_m)\}_{i \in [6]}$, outputting 1 if the following holds:

$$\sum_{i=1}^{6} a_i F_i(y_m) = Z(y_m)T(y_m).$$

end procedure

We also briefly explain the intuition behind the S_{id_j} and S_{σ_j} polynomials: S_{id_j} is only required to map D to the disjoint sets P_1, P_2, P_3. S_{σ_j} should then map to the same set $P = P_1 \cup P_2 \cup P_3$ but in a "permuted" fashion. We construct a map τ for permutation σ from domain $[n]$ to P. The simplest way to define S_{id_k} is to map $[n]$ to $[1, ..., n]$, $[n+1, ..., 2n]$, $[2n+1, ..., 3n]$ respectively, in this case there is no need to apply the map τ as then there is no need for domain translation ($P = [n]$). The problem is that all of the S_{id_j} polynomials will be of degree n in general. We construct S_{id_j} so as to be of minimal possible degree 1, so it is easy for the verifier to calculate evaluations of those polynomials by themselves without requiring the usage of the (more expensive) evaluation procedures. This optimization is taken from [Gabizon et al. 2019].

We opt for sampling y outside of the domain D in order to achieve perfect-zero knowledge guarantees instead of statistical. N here denotes the number of random challenge points sampled and is set to $N = 1$ for our implementation. Finally, due to the restrictions on the degree it may be necessary to split T into separate polynomials $\{T_0, T_1, T_2\}$ and commit to them independently, since its degree is $3n$ we can ensure each of the T_i has degree at most n. This and other optimizations are further discussed in Appendices A.8 and A.9, while the formal analysis of the security and zero-knowledge properties of the protocol can be found in Appendix A.6. We end with a theorem capturing the security properties of RedShift.

Theorem 13.6. *The prime field* $\mathsf{RS}[\mathbb{F}, D, \rho]$ *code family with rate* $\rho = 2^{-\mathcal{R}}, \mathcal{R} \geq 2, \mathcal{R} \in \mathbb{N}$ *and* $\rho|D| = n + 1$ *admits an IOP for the constraint system* $\mathcal{S}$ *described in Definition 13.1, where* n *the size of the instance in* $\mathcal{S}$ *and* N *a repetition parameter. This IOP achieves perfect completeness, perfect zero-knowledge, and* $\forall \nu \in (0, 1 - \sqrt{\rho}), \delta \in (0, J_\nu^{[3/2]}(1 - \rho))$ *has round-by-round knowledge soundness:*

$$\epsilon_\pi(\delta) \leq \max\left(\epsilon_{\mathsf{FRI}}(\delta), \epsilon_{\mathsf{IOP}}^N, \frac{1}{|\mathbb{F}|}\right), \text{ where } \epsilon_{\mathsf{IOP}} := (J_{\rho,\nu})^6 \cdot \frac{4n}{|\mathbb{F} \backslash D|}.$$

The above IOP is instantiated as a Non-Interactive Random Oracle Proof (NIROP) using the "CS-proof" technique [Micali 2000] to compile the oracles to the constraint functions. More

specifically, this assumes the existence of a Random Oracle (RO) for the hash function $\mathcal{H} : \mathbb{F}\times\mathbb{F} \to \mathbb{F}$ [Ben-Sasson et al. 2016] and constructs $\mathrm{Com}(f)$ as the root c of a Merkle tree where the leaves form the evaluations of f on D. The RO is not required, as correlation-intractable hash functions have been shown to suffice (see [Canetti et al. 2018]). In the non-interactive setting, the prover will provide the verifier with a Merkle authentication path to some $f(i)$ upon a query for $i \in D$ to the oracle for f. Note that this adds $\log|D|$ overhead to the protocol for each query.

13.3 Soundness Parameters

The various components of RedShift can now be put together in order to arrive at security guarantees of a functional implementation. By modularity of the above approach, it is possible to analyze the security guarantees of the system based on different choices for the rate, distinguisher, and LPC scheme. The starting point for this is the soundness bound of Theorem 13.6. Since this depends on the list decodability of the RS codes, the analysis can be split into two axes: based on Theorem 11.3 and on Conjecture 11.4, each of which gives different bounds on protocol soundness. We analyze the above based on a changing rate parameter ρ at a given security level, and look the total number of queries that need to be executed by the FRI protocol to achieve this. In the analysis below, we work with a field of size $\log|\mathbb{F}| = 256$ bits and aim for 80 bit security. First we focus on the contribution from the list-decoding bound: $J^8_{\rho,\nu}\frac{4n}{|\mathbb{F}/D|}$: note the exponent of 8 instead of 6 due to splitting T into 3 smaller polynomials. In the syntax of Theorem 11.3, a choice of $\nu = |\mathbb{F}|^{-1/20}$ provides a list size of $\frac{1}{2}|\mathbb{F}|^{1/20}\rho^{-1/2}$. For our choice of field size, this yields an error contribution on the order of 2^{-128}. Now we focus on how such a choice of $\nu \neq 0$ will affect the soundness of FRI. By picking $\rho = 1/16$ we get the following contribution to the FRI soundness error for $\delta < J_\nu^{[3/2]}(1-\rho)$:

$$p(\nu,\rho) = \left(1 - \min\left\{\delta_0, J_\nu^{[3/2]}(1-\rho)\right\} + \nu\log|D|\right).$$

A smaller such value allows for fewer queries for a given level of soundness, shrinking proof size in the non-interactive setting. In the case of $\nu = 0$, we have $p(0, 1/16) = 1/2$. For domain size $|D| = 2^{32}$ (which equates to a degree bound $d = \rho|D| = n+1 = 2^{28}$) we have that $p(|\mathbb{F}|^{-1/20}, 1/16) \sim 0.504$. For comparison purposes, if we instead use $\rho = 1/32$ we get $p(0, 1/32) \sim 0.421$ and $p(|\mathbb{F}|^{-1/20}, 1/32) \sim 0.425$.

We are now ready to look at the effect of changing δ on the total number of queries required to achieve a constant security level for the overall protocol. We fix a rate of $\rho = 1/64$ with $\nu = 0$ for simplicity and provide the total query number required to achieve an 80 bit security level for three regimes for δ: (1) unique decoding radius $\delta = \delta_0$, (2) δ within the 'one-and-a-half' Johnson bound (used in FRI [Ben-Sasson et al. 2018d]), and (3) δ within the Johnson bound (used in DEEP-FRI [Ben-Sasson et al. 2019b]). Note that (1) denotes the worst-case error and is provided as a reference for the relative efficiency of the two LPC instantiations. We note a 51% reduction in the query number at this security level in using FRI, while DEEP-FRI achieves a 67% reduction at rate $\rho = 1/64$. As rate increases, this improvement is less pronounced; however, we still get a 32% and 55% improvement in query complexity for the two respective instantiations even at $\rho = 1/16$. This demonstrates concrete efficiency improvements to the underlying proof even with small rate deviations.

Subsequent work [Ben-Sasson et al. 2020] has demonstrated that for FRI instances over large fields we can do strictly better. If $q > |D|^2$, the maximal δ for which FRI is sound is actually equal to the Johnson bound $J_\nu(1-\rho)$ instead of $J_\nu^{[3/2]}(1-\rho)$. In practice, this gives us the same soundness error for FRI and DEEP-FRI since the number of queries performed dominates the error contribution. Finally, by assuming Conjecture 11.4 we can do even better: δ is pushed beyond the Johnson bound to $1-\rho$ and, due to the constant list size assumption, the soundness of FRI and DEEP-FRI persists in this larger range. Note, however, that going beyond the Johnson bound comes with a loss of knowledge security: we cannot use the Sudan list-decoding algorithm to extract a witness, so any knowledge claim would have to be non-extractable in this setting.

Table 13.1: LPC Instantiation Comparisons.

Method	δ Bound	p Bound ($\nu = 0$)	Query Number		
			1/64	1/32	1/16
Unique Decoding	$(1-\rho)/2$	$(1+\rho)/2$	82	84	88
FRI [Ben-Sasson et al. 2018d]	$J_\nu^{[3/2]}(1-\rho)$	$\sqrt[3]{\rho}$	40	48	60
FRI with $q > \|D\|^2$ [Ben-Sasson et al. 2020]	$J_\nu(1-\rho)$	$\sqrt{\rho}$	27	32	40
DEEP-FRI [Ben-Sasson et al. 2019b]	$J_\nu(1-\rho)$	$\sqrt{\rho}$	27	32	40
FRI with Conjecture 11.4	$1-\rho$	ρ	14	16	20

13.4 Benchmarks

We instantiate RedShift with $r = 576460752303423505$, $q = r \cdot 2^{192} + 1$ which is a Proth prime and use $\rho = 1/16$. Oracles were instantiated as Merkle trees using the Blake2s hashing algorithm. The PES was instantiated using the random sampling approach, where a random point was sampled using Fiat-Shamir: i.e. by placing all individual root hashes of the oracles to the setup polynomials into the transcript. Circuit sizes were chosen so as to set $n + 1 = 2^k$ which, in the case of RedShift, implies a degree bound $d = n + 1$ for FRI. All implementations use a certain degree of precomputation: we precompute the low degree extensions of setup polynomials and the Merkle trees of the setup polynomial oracles.

The soundness error due to FRI depends on the total number of queries performed. This *does not* change the proof generation time and only affects proof size and verification. We follow the approach in Section 13.3 in order to set these parameters, targeting an 80-bit security level. We note that the final soundness error is dominated by the FRI error, while the size contribution is dominated by the total number of queries performed. We used an Apple MacBook Pro 18.2 with an M1 Max 10-core processor and 64 GB RAM to record proof generation times, verification times and proof sizes for different predicate sizes presented in Fig. 13.1

The verification times and proof sizes provided rely on Conjecture 11.4. If we remove this assumption, we need to double the proof sizes (and verification times) but the proof generation

times would remain unchanged. This is because FRI operates over a large enough field that soundness holds for $\delta \leq J_v(1-\rho)$. Table 13.2 includes calculations for projected proof sizes at different security levels and rates. We present expected numbers for three different scenarios: (1) the current implementation, (2) the implementation after optimization (see Appendix A.8), and (3) the optimized proof system assuming Conjecture 11.4 also holds.

Our empirical results cannot be directly compared to other transparent proof systems such as [Chiesa et al. 2019a,b; Ben-Sasson et al. 2018c; Xie et al. 2019; Zhang et al. 2020], as the underlying constraint systems differ (all use R1CS, except [Zhang et al. 2020]) and thus predicate sizes (number of gates) don't exactly capture the same complexity across prototypes. Unlike all other approaches, the PLONK constraint system also allows for 'custom' gates, which means that it can be further modified to be more efficient at expressing specific types of circuits by using fewer circuit-specific gates. This makes a precise comparison between approaches difficult.

Even so, certain comparative observations can be made. Firstly, proof generation times are competitive with state-of-the-art systems, such as Aurora and Fractal that achieve a performance of $\sim$ 200 seconds on predicates of size 2^{20} at 128-bit security. Note that for us proof generation is only influenced by the rate parameter ρ, which we fix at $\rho = 1/16$ in Fig. 13.1 for *all* considered security levels to achieve 35 second proving time for size 2^{20} circuits. Our results are most comparable with Virgo [Zhang et al. 2020] at $\sim$ 10 seconds.

Verification times for 80-bit security stay at around $3-6$ ms for circuit sizes up to 2^{20}, clearly outperforming Aurora ($\sim$ 4 seconds for the same). At 128-bit security, a linear scaling of verification time (due to the linear increase in the number of queries) gives around $\sim$ 9ms. This matches the performance of Fractal ($<$ 10 ms at 128-bit security) and Virgo ($\sim$ 12 ms at 100-bit security). Note that verification time is proportional to proof size through the number of queries verified, so any corresponding improvements to proof sizes will affect verification proportionately.

The data clearly shows a logarithmic relationship between proof and predicate sizes, which is the biggest drawback of the current scheme. To this end, we propose optimizations (specified

in Appendix A.8) of how the Merkle tree represents data and query calls to decrease proof size (and verification times) by about two orders of magnitude ($\sim 4\times$). In this case, projected proof sizes are still larger ($\sim$ 300kB) than those of both Aurora and Fractal ($\sim$ 150kB) for a circuit of 2^{20} gates at 128-bit security but comparable to Virgo ($\sim$ 200kB) at 100-bit security. Note that this issue is specific to transparent systems: trusted proof systems usually [Groth 2016] have size $<$ 1kB proofs for all circuits.

Table 13.2: Projected Proof Sizes.

Security (Bits)	Rate (-log ρ)	Circuit Size (log $\|C\|$)	Proof Size (KB)		
			Unoptimized	Optimized	Conjectured
80	4	10	597	151	76
		15	1052	264	133
		20	1634	410	206
	5	20	1308	328	165
	6		1090	274	138
	7		934	235	118
	8		818	206	104
120	4	10	894	225	114
		15	1576	396	199
		20	2450	614	308

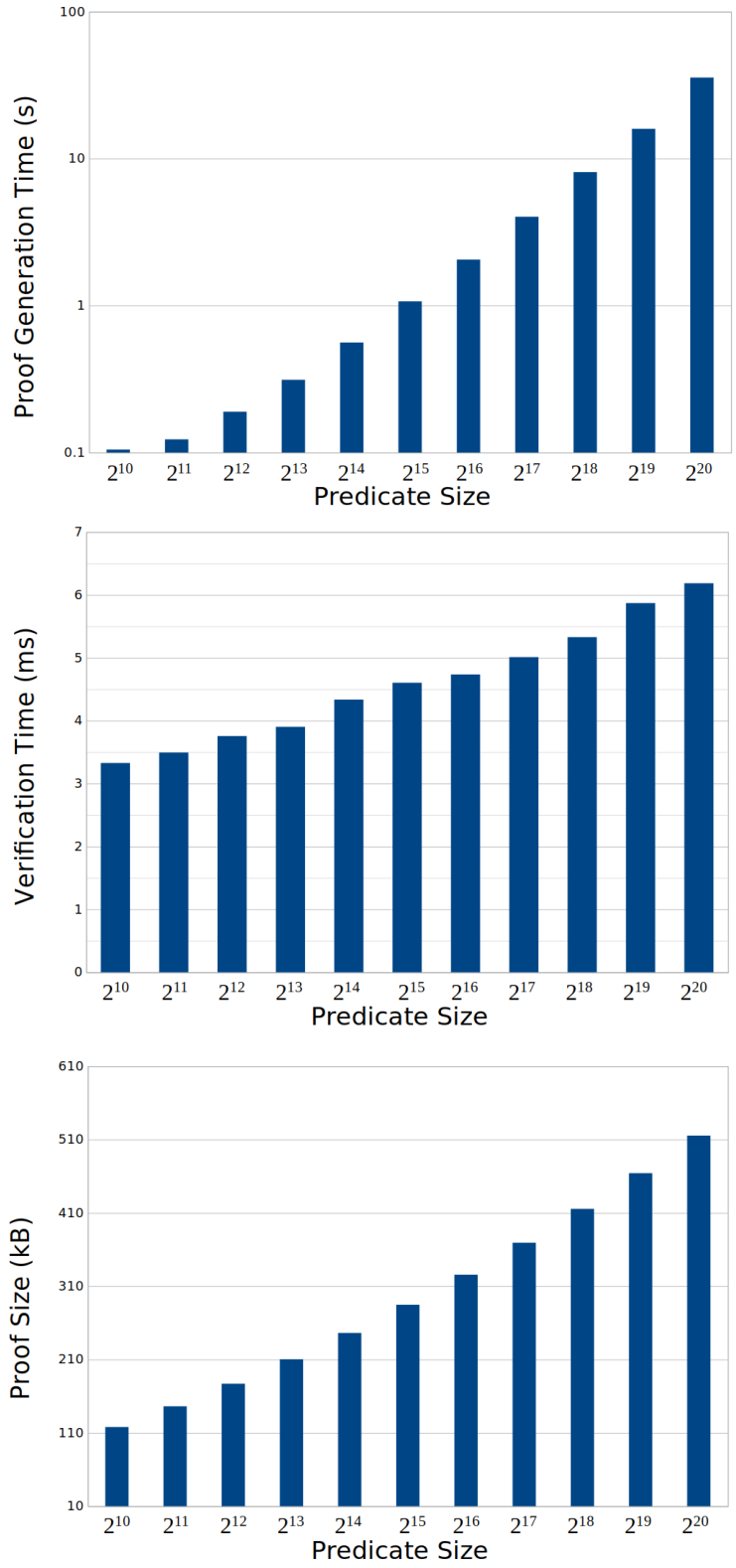


Figure 13.1: Benchmark for RedShift with $\rho = 1/16$. *Top:* Proof Generation Time (seconds). *Center:* Verification Time (ms). *Bottom:* Proof Size (kB).

14 | Open Questions

14.1 Extensions

14.1.1 Batching Multiple FRI Instances

Polynomial commitments and evaluations in RedShift reduce to the following check: whether particular functions $f_1, \ldots, f_k$ represented as oracles are close to the space of degree d polynomials. The batching approach is to replace all those separate and independent FRI queries by exactly one instance of FRI w.r.t a linear combination of functions f_i, where the coefficients of linear dependence are provided by the verifier. This can be done, with details in Appendix A.9. Such an optimization is important in the context of ZK-Rollups, as in practice the circuits that are verified are large (in the order of 2^{24} gates), and therefore being able to amortize the production of FRI proofs results in significant gains. This optimization has been independently proposed in a variety of proof systems using FRI and is critical to current implementations.

14.1.2 Binary Fields

Recall that PLONK is restricted to prime fields only. This is because [Kate et al. 2010] requires embedding a field $\mathbb{F}$ into a group of points on some pairing-friendly elliptic curve. [Ben-Sasson et al. 2018a] provides a version of the FRI protocol for binary fields which exploits additive and vector space structure of the underlying field. The rest of PLONK is field agnostic and only the

permutation argument would require modification. This means that RedShift can be instantiated for binary as well as prime fields and all constructions and proofs follow through by replacing the multiplicative domain $|D|$ by an affine subspace. The binary variant of PLONK is especially effective for computations that require a lot of bit manipulations.

14.1.3 Recursive Proofs

RedShift verification subroutines can be expressed as circuits, where the dominating operation will be the verification of Merkle paths or inclusion proofs in some other cryptographic accumulator. All remaining arithmetic operations are performed over the same field that the original circuit (for which the verifier is expressed) is defined, so there is no requirement for cycles over pairing-friendly elliptic curves as in previous work. A hybrid approach exists that performs the last step of recursion using a 'pairing-based' PLONK, e.g. the BLS12-381 curve has a main subgroup of order $|G|$ such that $2^{32} \mid (|G| - 1)$. This allows for instantiating RedShift.

Table 13.2 shows that in the case of recursive constructions one could use a higher rate for the "inner" and a lower one for the "outer" level of recursion that verify the nested proofs, since a smaller inner circuit is much cheaper to verify. This has been subsequently explored as an avenue to practical recursion in [Zero 2021].

14.1.4 Different Constraint Systems

The LPC and PES primitives can be applied to proof systems such as SONIC [Maller et al. 2019] and Marlin [Chiesa et al. 2019a] that use univariate polynomial commitments. Of interest here are the proximity testing parameters e.g. testing inclusion in $\mathrm{RS}[\mathbb{F}_q, D, (d-1)/|D|]$, where (in the case of Marlin) $d = 2^k$. Aurora contains a description of such a subroutine. Moreover, Virgo uses a different proof system with FRI in a functionally equivalent way, further demonstrating the modularity and applicability of our approach.

14.2 Discussion

We have constructed a zk-STARK capable of acting as the underlying primitive for a ZK-Rollup. We succeeded in minimizing proof generation and verification times, achieving encouraging results both with respect to decreasing latency and achieving high throughput. The main observation behind our methods was that the FRI protocol can be modularly composed with any IOP to generate a zk-SNARK, choosing a design which maximizes the efficiency of our proof system. One important focus for future work involves decreasing the size of the proofs. This will decrease the total cost of publishing transactions on-chain, thus contributing an increase in throughput.

We also believe that there exists substantial promise in the development of hardware acceleration methods for proof systems like RedShift. Indeed, an order-of-magnitude improvement in proof generation times would mean that scalable ZK-Rollups with extremely low latency and high throughput can be designed. If the entities computing the rollup proofs are doing so on dedicated hardware, the potential for cost-efficient mass transaction settlement is much greater. Indeed, using highly specialized but low-power hardware to quickly compute proofs would mean not only that proof generation times would be low, but also that the total *cost* per proof is also small. Driving down the energy cost of generating such proofs is thus an extremely important yet orthogonal research direction, as high-cost proof generation would inevitably drive up transaction fees in any implementation.

Finally, creating a PoW process out of a transparent SNARK is another intriguing open problem. Unfortunately, we can easily show that RedShift (or any FRI-based SNARK) is not suitable for such a purpose. This is because we can easily rerandomize witnesses using mask polynomials, meaning that the proof system is not unique witness extractable. Looking more closely at the knowledge soundness guarantees of transparent proof systems would be an important first step to understanding how we can design efficient PoW processes out of them.

A | Appendix

A.1 Multiexponentiation Bounds & Theorem Proofs

We borrow notation from [Pippenger 1980] and parametrize with q input indices, p outputs and maximum index size 2^λ. Where not specified, $H = pq\lambda$. Let $L(\mathbf{y})$ be the minimum number of multiplications to compute $\mathbf{y} = (y_1, ..., y_p)$ with $y_i \in [2^\lambda]^q$ and $[2^\lambda] = \{1, ..., 2^\lambda - 1\}$ from the inputs and unit vectors and $L(p, q, 2^\lambda)$ be the maximum over all of them.

Lemma A.1. *For any value of $c \leq L(p, q, N)$, there are at most:*

$$\left(\frac{H^2}{c}\right)^c 2^{q+1} e^c (q+1) 2^{O(1)},$$

addition chains of length at most c.

Lemma A.2. *Define $H := \kappa q \nu \lambda$, $\phi(q, \kappa, \nu, \lambda) :=$*

$$q\kappa\nu \log(q\kappa\nu) + \kappa \log(H) + q + \log(q+1) + 1,$$

and fix $\mu := \delta H$, corresponding to:

$$c_\delta := \frac{(1-\delta)H - \phi(q, \kappa, \nu, \lambda)}{\log(H) - \log(e) + \log(\mu) + \log(1/\delta)}.$$

For the (κ, κ)-length MultiExp function of dimension ν for collision resistant g:

$$\Pr_{\mathbf{x}\in_R[2^\lambda]^{\kappa\times q}, \mathrm{G}\in_R\mathbb{G}^{\kappa\times\nu}}\left[L(f(x_1),...,f(x_q)) \le c_\delta\right] \le \left(\frac{1}{2}\right)^{\mu}.$$

Proof. Write $G_k^{(j)} = r_{jk}G$. As the $x_i \in [2^\lambda]^\kappa$ and $r_{jk} \in [2^\lambda]$ are sampled randomly, the values $x_{ik}G_k^{(j)} = x_{ik}r_{jk}G$ for $i \in [q], j \in [\nu-1], k \in [\kappa]$ will be distinct w.h.p. The $\kappa \cdot q$ values $g(x_i)_k \cdot r_{\nu k}G$ will also be distinct w.h.p. as g is collision resistant in each of its κ output coordinates.

Let M be the $q \times (\kappa\nu)$ sized matrix with these values as entries. As each entry is an element in $[2^\lambda]$, the number of matrices M with $q\kappa\nu$ distinct elements is:

$$\binom{2^\lambda}{q\kappa\nu} \ge \frac{2^{\lambda q\kappa\nu}}{(q\kappa\nu)^{q\kappa\nu}},$$

and to each M there corresponds a unique matrix $F = (f(x_1),...,f(x_q))$ with dimension $q \times \nu$, where the κ products over random bases for each x_i have been computed. Note that $L(F) = L(M) + \kappa - 1$.

We can thus upper bound the minimal addition chain size $L(F)$ using $L(M)$ and the number of matrices M:

$$\Pr_{\mathbf{x}\in_R[2^\lambda]^{\kappa\times q}, \mathrm{G}\in_R\mathbb{G}^{\kappa\times\nu}}[L(F) \le c] \le \frac{|\{\mathbf{z} : L(\mathbf{z}) \le c\}|}{2^{H - q\kappa\nu\log(q\kappa\nu)}}.$$

The numerator is upper bounded by Lemma A.1 and the fact that a single chain corresponds to at most H^κ matrices, giving:

$$\Pr_{\mathbf{x}\in_R[2^\lambda]^{\kappa\times q}, \mathrm{G}\in_R\mathbb{G}^{\kappa\times\nu}}[L(F) \le c] \le \left(\frac{1}{2}\right)^{H-\psi(c)},$$

where $\psi(c) := c(2\log H + \log e) + \phi(q,\kappa,\nu,\lambda) - c\log(c)$.

Suffices to show that for $c \le c_\delta$, $\psi(c) \le (1-\delta)H$. Since $\psi(c)$ is increasing for $c \le L(\kappa, \nu q, 2^\lambda)$,

required to show that $\rho \geq c_\delta$ for $\psi(\rho) = (1-\delta)H$:

$$\rho(2\log H + \log e) + \phi(q,\kappa,\nu,\lambda) \geq (1-\delta)\cdot H,$$

$$\log \rho \geq \log\left((1-\delta)\cdot H - \phi(q,\kappa,\nu,\lambda)\right) - \log\left(2\log H - \log(e)\right),$$

$$\therefore \rho \geq \frac{(1-\delta)H - \phi(q,\kappa,\nu,\lambda)}{\log H - \log(e) + \log(\mu) + \log(1/\delta)},$$

since $\mu = \delta H$.

□

Corollary A.3. *Fix $\delta > 0$ and let $\psi(\rho_\delta) - (1-\delta)\cdot H = 0$.*

$$\mathbb{E}[L(f(x_1),...,f(x_q))] \geq \rho_\delta \cdot (1 - 2^{-\delta H}).$$

Proof. By Markov's inequality:

$$\Pr[L(f(x_1),...,f(x_q)) \geq \rho_\delta] \cdot \rho_\delta \leq \mathbb{E}[L(\mathbf{x})],$$

$$(1 - \Pr[L(f(x_1),...,f(x_q)) < \rho_\delta]) \cdot \rho_\delta \leq \mathbb{E}[L(f(x_1),...,f(x_q))].$$

□

Proof of Theorem 7.5. Required to compute q iterations of the MultiExp function. Each iteration includes ν multiproducts over random bases, with the indices also sampled from $[2^\lambda]$.

Using c oracle queries to do this corresponds to knowledge of an addition chain of length c containing all of $F = (f(x_1),...,f(x_q))$ with $x_i \in [2^\lambda]^\kappa$. Therefore, the probability that we compute F for $\mathbf{x} \in_R [2^\lambda]^{\kappa\times q}$ with less than c queries is upper bounded by the probability that $L(F) \leq c$.

Fix $\delta > 0$. Lemma A.2 states that $\exists c_\delta$ s.t. this probability is negligible in $\mu := \delta\kappa\nu q\lambda$ for $c \leq c_\delta$.

One function computation of dimension ν with κ inputs has an upper bound on the expected number of multiplications of:

$$\min(\kappa, \nu) \cdot \lambda + \frac{\kappa\nu\lambda}{\log(\kappa\nu\lambda)} \cdot (1 + o(1)).$$

Corollary A.3 implies that:

$$\epsilon \leq 1 - q^{-1} \cdot \left(\min(\kappa, \nu) \cdot \lambda + \frac{\kappa\nu\lambda}{\log(\kappa\nu\lambda)} \cdot (1 + o(1))\right)^{-1} \cdot (1 - 2^{-\delta\kappa\nu q\lambda}) \cdot c_\delta$$

$$\leq \frac{\log(q) + \delta\log(\kappa\nu\lambda) + o(1)}{\log(\kappa\nu q\lambda)} \leq \frac{\log(q) + o(1)}{\log(\kappa\nu q\lambda)},$$

where we have taken $\delta \leq 1/\log(\kappa\nu\lambda)$. □

Proof of Theorem 8.2. We know that the NIP has a PKE extractor from its security proof and so $\mathcal{A}$ can extract two witnesses almost surely using extractor $\chi_{\mathcal{A}}^{PKE}$. If the polynomials are distinct, so are their witnesses. This follows directly from the fact that, since $\pi_1 \neq \pi_2$, either (1) one of $u_i(X), v_i(X), w_i(X)$ differs in one of the proofs, or (2) the extracted witnesses differ. Since the predicate is the same, it follows that the witnesses must differ. □

Lemma A.4. *Let $\mathcal{H}_P = \{\mathcal{H}_{P,\lambda}^G\}_{\lambda\in\mathbb{N}_+}$ be a family of efficiently computable functions for which each $\mathcal{H}_{P,\lambda}^G : \{0,1\}^\lambda \to \mathbb{G}$ is weakly collision-resistant. $\mathcal{L}(\mathcal{H}_P)$ is hard single-witness.*

Proof of Lemma A.4. Define S in the natural way: fix $\lambda \in \mathbb{N}_+$ and define S to randomly sample an element $x \in \{0,1\}^\lambda$, outputting $(\mathcal{H}_{P,\lambda}(x), x)$. The sampler is efficient by the efficiency of $\mathcal{H}_{P,\lambda}(x)$, and $(\mathcal{H}_{P,\lambda}(x), x) \in R_{\mathcal{L}(\mathcal{H}_{P,\lambda})}\}$ by definition. Witness intractability (WI) follows from the collision resistance of $\mathcal{H}_{P,\lambda}$ on constant-size inputs. If some $\mathcal{A}$ exists that violates WI, then running S on 1^λ and then $\mathcal{A}$ on $\mathsf{S}(1^\lambda)$ and 1^λ, we non-negligibly find a collision in $\mathcal{H}_{P,\lambda}^G$. □

Proof of Theorem 8.4. The MaskedHash QAP has $4N(k+1)$ intermediate witness variables (and

$8N(k+1)+2k$ constraints) which admits witnesses from a sampler S where the seed ρ is uniformly random and so all witness variables (with full support) also look random by the randomness of the group encoding. This is as the intermediate values are distinct powers of a group element that is random due to ρ and the independence of the I_j index elements. By unique witness extractability and single witness hardness of CRT functions, all valid witnesses have a unique encoding and hence a unique witness polynomial h.

We start with ℓ instances of N k-bit hash evaluations from S, and require ℓ valid proofs. We reduce to the $4N(k+1)$-length MultiExp problem for ℓ instances and g equal to the function evaluating the representation of h given the witness elements. We provide ℓ of the $4N(k+1)$ intermediate witness variables and the corresponding 9 sets of bases to the MultiExp function. The representation of h will be unique w.r.t. the witness (since the instance is single witness hard) and thus look random due to the inputs. Note that $\mu = 8N(k+1)+2k$. We finally perform a linear in ℓ number of multiplications to add any witness variables that were not included (i.e. not randomly distributed). Since the MultiExp index distributions are also random, a proof verifies iff the MultiExp solution is valid.

Conversely, given ℓ $(4N(k+1), 8N(k+1)+2k)$-length MultiExp instances of dimension 10 with inputs and bases sampled from the QAP's sampler and proving key respectively, we reduce to computing ℓ proofs for N k-bit hash evaluations. This is because the unassigned witness variables can be discerned from the auxiliary input to g, which comes from the QAP sampler. By the uniqueness of the proof's encoding, the set of ℓ valid proofs will have to equal the MultiExp instances after a linear in ℓ number of operations to 'undo' products by any of the additional variables. □

A.2 DPS Transaction Semantics

In the definitions below, we denote the supplementary information string by $*$, but make no assumptions about the type of information provided. This is done to ensure that information required by an (arbitrary) transition function is encompassed by our definition.

Definition A.5. Given a consistent RSM Σ, a Distributed Payment System (DPS) is a tuple $\Delta(\Sigma)$ consisting of:

Setup : $1^{\lambda} \longrightarrow pp$

- Inputs: Security parameter λ
- Outputs: Public parameters pp

NewCoinbase : $S \times z_a \times c \times (pk, sk)_a \times * \longrightarrow t$

- Inputs: Subset of current state $S \subseteq \mathcal{S}_i$, z_a address of sender a, c value transferred, public-private key pair $(pk, sk)_a$
- Outputs: Transaction t

NewTransaction : $S \times z_{\{a,b\}} \times c_{\{a,b\}} \times (pk, sk)_a \times pk_R \times * \longrightarrow t$

- Inputs: Subset of current state $S \subseteq \mathcal{S}_i$, $z_{a,b}$ addresses of sender/receiver, $c_{\{a,b\}}$ value transferred, public-private key pair $(pk, sk)_a$, public key pk_R
- Outputs: Transaction t

VerifyTransaction : $t \times S \times * \longrightarrow$ Yes/No

- Inputs: Subset of current state $S \subseteq \mathcal{S}_i$, transaction t
- Outputs: Yes/No

NewState : $S \times \mathbf{t} \times * \rightarrow \mathcal{S}_{i+1}$

- Inputs: Subset of current state $S \subseteq \mathcal{S}_i$, list of ordered transactions $\mathbf{t} \in 2^{\mathcal{T}}$
- Outputs: State $\mathcal{S}_{i+1}$

VerifyState : $\mathbf{t} \times S_1 \times S_2 \times * \rightarrow$ Yes/No

- Inputs: Subsets of current and next state $S_1 \subseteq \mathcal{S}_i, S_2 \subseteq \mathcal{S}_{i+1}$, list of ordered transactions $\mathbf{t} \in 2^{\mathcal{T}}$
- Outputs: Yes/No

CreateAddress : $pp_S \rightarrow (pk, sk)$

- Inputs: Public parameters pp_S
- Outputs: New public/private keys $pk, sk \in \{0,1\}^*$

GetBalance : $S \times pk \times * \rightarrow c$

- Inputs: Subset of current state $S \subseteq \mathcal{S}_i$, pk a **CreateAddress** output
- Outputs: Balance c corresponding to pk

GetQuality : $S \rightarrow q$

- Inputs: Subset of current state $S \subseteq \mathcal{S}_i$
- Outputs: Quality q of state $\mathcal{S}_i$

A.2.1 Security Properties

A.2.1.1 Completeness:

Definition A.6. A DPS Π is *complete* if, for all poly(λ)-size algorithms $\mathcal{A}$ and large enough λ:

$$Pr[\text{INCOMP}(\mathcal{A}, \Pi, \lambda) = 1] \leq negl(\lambda)$$

INCOMP$(\Pi, \lambda, \mathcal{A})$:

1. C samples $pp \leftarrow \text{Setup}(1^\lambda)$, sending pp to $\mathcal{A}$
2. $\mathcal{A}$ sends C the following:
   (a) A state $\mathcal{S}_i$
   (b) Three addresses z_a, z_b, z_{CB}
   (c) Positive integer values c_s, c_f, c_m
   (d) A key pair (PK, SK) corresponding to address **a**
   (e) A public key PK_B
   (f) A key pair (PK_{CB},SK_{CB})
   (g) Two signatures σ, σ_{CB}
   (h) Information strings $\mathsf{info}_S, \mathsf{info}_\sigma, \mathsf{info}_{\sigma_2}$
3. C checks that the following hold, outputting 0 if any test fails:
   (a) Check that the key pairs are well formed
   (b) Check that all addresses are different
   (c) Check that
   $$\text{VS}(\text{PK}, z_a \| z_b \| c_s \| c_f \| \mathsf{info}_\sigma, \sigma) = 1$$
   (d) Check that
   $$\text{VS}(\text{PK}_{CB}, z_{CB} \| c_m \| \mathsf{info}_{\sigma_2}, \sigma_{CB}) = 1$$
   (e) $\text{VerifyState}(pp, \mathcal{S}_i, \mathsf{info}_S) = 1$
   (f) Account **a** with **a**.addr = z_a exists in $\mathcal{S}_i$ and is non-null with public key PK
   (g) If account **b** with **b**.addr = z_b is initialized, check that it has public key PK_B
   (h) $\text{GetBalance}(pp, \mathcal{S}_i, \text{PK}) \geq c_s + c_f$

(i) $c_f \leq c_m$

4. C constructs a send transaction t with the given parameters:

$$\text{NewTransaction}(pp, z_a, z_b, c_s, c_f, (\text{PK}, \text{SK}), \text{PK}_B)$$

5. C constructs a coinbase transaction t_{CB} with the given parameters:

$$\text{NewCoinbase}(pp, z_{CB}, c_m, (\text{PK}_{CB}, \text{SK}_{CB}))$$

6. C checks that the following hold, outputting 0 if any test fails:

   (a) $\text{VerifyTransaction}(pp, t, \mathcal{S}_i) = 1$

   (b) $\text{VerifyTransaction}(pp, t_{CB}, \mathcal{S}_i) = 1$

7. Compute the state transition:

$$(\mathcal{S}_{i+1}, \text{info}_{S_2}) = \text{NewState}(pp, \{t, t_{CB}\}, \mathcal{S}_i, \text{info}_S)$$

8. Output 1 if any of the following hold:

   (a) $t \neq (z_a, z_b, c_s, c_f, \text{PK}, \sigma, \text{PK}_R)$

   (b) $t_{CB} \neq (z_{CB}, c_m, \text{PK}_{CB}, \sigma_{CB})$

   (c) $\text{GetBalance}(\text{PK}_B, \mathcal{S}_{i+1}) \neq \text{GetBalance}(\text{PK}_B, \mathcal{S}_i) + c_s$

   (d) $\text{GetBalance}(\text{PK}_{CB}, \mathcal{S}_{i+1}) \neq \text{GetBalance}(\text{PK}_{CB}, \mathcal{S}_i) + c_m$

   (e) $\text{GetBalance}(\text{PK}, \mathcal{S}_i) \neq \text{GetBalance}(\text{PK}, \mathcal{S}_{i+1}) + c_s + c_f$

   (f) $\text{VerifyState}(pp, \mathcal{S}_{i+1}, \text{info}_{S_2}) = 0$

A.2.1.2 Correctness:

Define oracle O that initializes the DPS based on public parameters pp, keeps system state $\mathcal{S}_i$ and string info_S, keeps counters C, D, E initialized to zero and an (initially empty) set ADDR. The oracle allows the following queries:

CreateAddress:

1. Generate $(\text{PK}, \text{SK}) \leftarrow \text{CA}(pp_{sig})$
2. Add (PK, SK) to ADDR
3. Return PK

LookupAddress:(z_a)

1. Find the public key PK for account z_a
2. If PK is not in ADDR, return zero
3. Return GetBalance$(\text{PK}, \mathcal{S})$

RequestTransactions:

$$(z_a, z_b, c_s, c_f, \text{PK}, \text{PK}_B, z_{CB}, c_m, \text{PK}_{CB})$$

1. Check that $\text{PK}, \text{PK}_{CB} \in$ ADDR
2. Retrieve SK and SK_{CB} and obtain σ, σ_{CB}
3. $t \leftarrow (z_a, z_b, c_s, c_f, \text{PK}, \sigma, \text{PK}_B)$
4. $t_{CB} \leftarrow (z_{CB}, c_m, \text{PK}_{CB}, \sigma_{CB})$
5. Check that VerifyTransaction$(pp, t, \mathcal{S}) = 1$
6. Check that VerifyTransaction$(pp, t_{CB}, \mathcal{S}) = 1$
7. $E_0 =$ GetBalance$(\text{PK}_B, \mathcal{S})$

8. Update the state and information string:

$$(\mathcal{S}_2, \mathsf{info}_{S_2}) \leftarrow \text{NewState}(pp, \{t, t_{CB}\}, \mathcal{S}, \mathsf{info}_S)$$

9. If $\text{VerifyState}(pp, \mathcal{S}_2, \mathsf{info}_{S_2}) = 1$, set

$$\mathcal{S} = \mathcal{S}_2, \mathsf{info}_S = \mathsf{info}_{S_2}$$

10. Check that $c_s = \text{GetBalance}(\text{PK}_B, \mathcal{S}) - E_0$
11. If $\text{PK}_B \notin \text{ADDR}$, set $E = E + c_s$.

AddTransactions:(t, t_{CB})

1. $(z_a, z_b, c_s, c_f, \text{PK}, \sigma, \text{PK}_B) \leftarrow t$
2. $(z_{CB}, c_m, \text{PK}_{CB}, \sigma_{CB}) \leftarrow t_{CB}$
3. Check that $\text{VerifyTransaction}(pp, t, \mathcal{S}) = 1$
4. Check that $\text{VerifyTransaction}(pp, t_{CB}, \mathcal{S}) = 1$
5. $C_0 = \text{LookupAddress}(z_b)$
6. $D_0 = \text{GetBalance}(\text{PK}_{CB}, \mathcal{S})$
7. Update the state and information string:

$$(\mathcal{S}_2, \mathsf{info}_{S_2}) \leftarrow \text{NewState}(pp, \{t, t_{CB}\}, \mathcal{S}, \mathsf{info}_S)$$

8. Check that $c_m = \text{GetBalance}(\text{PK}_{CB}, \mathcal{S}) - D_0$
9. If $\text{PK}_B \in \text{ADDR}$, check that

$$c_s = \text{LookupAddress}(z_b) - C_0$$

10. If VerifyState$(pp, \mathcal{S}_2, \mathsf{info}_{S_2}) = 1$, set

$$\mathcal{S} = \mathcal{S}_2, \mathsf{info}_S = \mathsf{info}_{S_2}$$

11. Set $C = C + \text{LookupAddress}(z_b) - C_0$
12. Set $D = D + c_m$

Definition A.7. A DPS Π is *correct* if, for all poly(λ)-size adversaries $\mathcal{A}$ and large enough λ, the adversary wins INCOR with at most negligible probability:

$$Pr[\text{INCOR}(\mathcal{S}, \Pi, \lambda) = 1] \leq negl(\lambda)$$

INCOR$(\Pi, \lambda, \mathcal{A})$:

1. C samples $pp \leftarrow \text{Setup}(1^\lambda)$, sending pp to $\mathcal{A}$
2. C instantiates an oracle O based on Π
3. $\mathcal{A}$ issues queries to O
4. $\mathcal{A}$ sends a set of addresses $\{z_i\}_{i=1}^K$ to C
5. C adds together in a variable v all the balances of the addresses PK_i corresponding to z_i for which $\text{PK}_i \notin \text{ADDR}$
6. C outputs 1 if $v + C > D + E$

A.2.2 Basic Data Structures

Account: An account **a** is a tuple $(\text{addr}, \text{PK}, bal, n)$ where:

1. addr is the address of **a**.
2. PK is the public key of **a**.

3. *bal* the balance of the account (non-negative).

4. n the nonce of the block that contains the transaction that last modified **a**.

Note: Each PK does not have to have a unique address.

Transactions: There are two types of transactions:

1. Coinbase:

$$t_c = (\text{addr}, v, \text{PK}, \sigma)$$

   (a) addr: the address of the recipient

   (b) v: the value it receives

   (c) σ: The digital signature of the transaction

2. Standard

$$t = (\text{addr}_s, \text{addr}_r, v, f, \text{PK}, \sigma, \text{PK}_R)$$

   (a) $\text{addr}_s, \text{addr}_r$: the addresses of the sender and the receiver respectively

   (b) v the value to be transferred from the sender to the receiver (it is a positive integer)

   (c) f: Total mining fee provided

   (d) σ: The signature of the transaction

   (e) PK: the public key that validates σ

   (f) PK_R: the public key of the recipient

A.2.3 Transaction Semantics

Setup: This algorithm is run once by a trusted third party to initialize the parameters of the IVC and signature schemes.

NewTransaction & NewCoinbase: See specification above.

Algorithm 5 NewTransaction & NewCoinbase

Require: $pp, \text{addr}_s, \text{addr}_r, v, f, \text{PK}, \text{SK}, \text{PK}_R$
  procedure NewTx($pp, \text{addr}_{\{s,r\}}, v, f, \text{PK}, \text{SK}, \text{PK}_R$)
    $\sigma \leftarrow \text{Sign}(\text{SK}, \text{addr}_s \| \text{addr}_r \| v \| f \| \text{addr}_s.n)$
    return $(\text{addr}_s, \text{addr}_r, v, f, \text{PK}, \sigma, \text{PK}_R)$
  end procedure
Require: $pp, \text{addr}_r, v, \text{PK}, \text{SK}$
  procedure CoinbaseTx($pp, \text{addr}_r, v, \text{PK}, \text{SK}$)
    $\sigma \leftarrow \text{Sign}(SK, \text{addr}_r \| v \| \text{addr}_r.n)$
    return $(\text{addr}_r, v, \text{PK}, \sigma)$
  end procedure

CreateAddress: CreateAddress$(pp_{sig}) \rightarrow (\text{PK}, \text{SK})$

GetQuality: GetQuality$(\mathcal{S}_i) \rightarrow q_i$

GetBalance: GetBalance$(\text{PK}, \mathcal{S}_i) \rightarrow \mathbf{a}.v$, $\mathbf{a}$ the leaf in $\mathcal{S}_i$ with $\mathbf{a}$.PK = PK.

A.2.4 Digital Signature Schemes

- SC-Setup$(1^\mu) \rightarrow pp_{sig}$
- CA$(pp_{sig}) \rightarrow (\text{PK}, \text{SK})$
- Sign$(SK, m) \rightarrow \sigma$
- VS$(\text{PK}, m, \sigma) \rightarrow$ Yes/No

Completeness: VS$(\text{PK}, m, \text{Sign}(\text{SK}, m)) = 1$.

Security: For all non-uniform PPT $\mathcal{A}$, the following is negligible in μ:

$$\Pr\left[\begin{array}{c|c} \sigma \notin Q & pp_{sig} \leftarrow \text{SC-Setup}(1^\mu) \\ \text{VS}(\text{PK}, m, \sigma) = 1 & (\text{PK}, \text{SK}) \leftarrow \text{CA}(pp_{sig}) \\ & (m, \sigma) \leftarrow \mathcal{A}^{\text{Sign}(\text{SK}, \cdot)}(\text{PK}, 1^\mu) \end{array}\right].$$

$\mathcal{A}$ has access to Sign$(\text{SK}, \cdot)$, Q set of queries by $\mathcal{A}$ toSign, which knows PK and μ. $\mathcal{A}$ cannot query m.

A.2.5 DPS Transition Functions

Algorithm 6 VerifyTx

Require: (t, T^i)

procedure VerifyTx(t, T^i)
  if $t = (\text{addr}_s, \text{addr}_r, v, f, \text{PK}, \sigma, \text{PK}_R)$ **then**
    if $v < 0$ **then return** 0
    end if
    Let $\mathbf{a}_s$ the account in T^i s.t. $\mathbf{a}_s.\text{addr} = \text{addr}_s$
    if $\mathbf{a}_s.\text{PK} \neq \text{PK}$ **then return** 0
    end if
    if $\mathbf{a}_r.\text{PK} \neq \text{nil}$ and $\mathbf{a}_r.\text{PK} \neq \text{PK}_R$ **then return** 0
    end if
    $m \leftarrow \text{addr}_s \| \text{addr}_r \| v \| f \| \mathbf{a}_r.n$
    if $\text{VS}(\text{PK}, m, \sigma) = 0$ **then return** 0
    end if
    if $\mathbf{a}_s.bal < v + f$ **then return** 0
    end if
  else if $tx = (\text{addr}, v, \text{PK}, \sigma)$ **then**
    if $\text{VS}(\text{PK}, \text{addr} \| v \| \text{addr}.n, \sigma) = 0$ **then return** 0
    end if
    Let $\mathbf{a}$ the account in T^i s.t. $\mathbf{a}.\text{addr} = \text{addr}$
    if $\mathbf{a}.\text{PK} \neq \text{nil}$ and $\mathbf{a}.\text{PK} \neq \text{PK}$ **then return** 0
    end if
  else return 0
  end if
  return 1
end procedure

A.3 Constraint System Equivalence

Proof. Suffices to show a polynomial time transition betwee the two constraint systems.

Suppose $\mathcal{V} = (\mathbf{a}, \mathbf{b}, \mathbf{c})$; think of $\mathcal{V}$ as a vector in $[m]^{3n}$. For $i \in [m]$, let $T_i \subset [3n]$ be the set of indices $j \in [3n]$ such that $\mathcal{V}_j = i$. Now define $T_{\mathcal{L}} := \{T_i\}_{i \in [m]}$ - partition of $[3n]$ into non-intersecting chunks. Define a permutation $\sigma(T_{\mathcal{L}})$ on $[3n]$ in the following way: for each block T_i of $T_{\mathcal{L}}$, $\sigma(T_{\mathcal{L}})$ contains a cycle going over all elements of T_i. For simplicity we write $\sigma = \sigma(T_{\mathcal{L}})$

Algorithm 7 UpdateState

Require: $(\mathcal{S}_i, \mathbf{t}, n)$
Ensure: $(\mathcal{S}_{i+1})$

procedure UpdateState($\mathcal{S}_i, \mathbf{t}, n$)
  Parse $\mathcal{S}_i \leftarrow (T^i, i, q_i, n_i)$, **return** 0 if this fails
  $N \leftarrow |\mathcal{T}|$, $T^i_0 \leftarrow T^i$, $v_{Tfee} \leftarrow 0$
  for $j \leftarrow 1, \ldots, N-1$ **do**
    $t_j \leftarrow (\text{addr}_s, \text{addr}_r, v, f, \text{PK}, \sigma, \text{PK}_R)$
    if VerifyTx$(t_j, T^i_{j-1}) = 0$ **then return** 0
    end if
    $T^i_j \leftarrow T^i_{j-1}$ // Initialize T^i_j
    Define $\mathbf{a}_s$ as leaf of T^i_{j-1} s.t. $\mathbf{a}_s$.addr = addr_s
    Update T^i_j:
      Set $\mathbf{a}_s.bal \leftarrow \mathbf{a}_s.bal - tx_j.v - tx_j.f$, $\mathbf{a}_s.n \leftarrow n$
    Define $\mathbf{a}_r$ as leaf of T^i_{j-1} s.t. $\mathbf{a}_r$.addr = addr_r
    Update T^i_j:
      Set $\mathbf{a}_r.bal \leftarrow \mathbf{a}_r.bal + tx_j.v$, $\mathbf{a}_r.n \leftarrow n$
      If $\mathbf{a}_r$.PK = nil, set $\mathbf{a}_r$.PK $\leftarrow \text{PK}_R$
    $v_{Tfee} = v_{Tfee} + tx_j.f$
  end for
  $t_N \leftarrow (\text{addr}, v, \text{PK}, \sigma)$
  if VerifyTx$(t_N, T^i_{N-1}) = 0$ **then return** 0
  end if
  if $v \neq v_{mint} + v_{Tfee}$ **then return** 0
  end if
  $T^i_N \leftarrow T^i_{N-1}$, $\mathbf{a}_m$ leaf of T^i_{N-1} s.t. $\mathbf{a}_m$.addr = addr_m
  Update T^i_N:
    Set $\mathbf{a}_m.bal \leftarrow \mathbf{a}_m.bal + v$, $\mathbf{a}_m.n \leftarrow n$
    If $\mathbf{a}_m$.PK = nil, set $\mathbf{a}_m$.PK $\leftarrow$ PK
  $q_{i+1} \leftarrow q_i + 1$, $\mathcal{S}_{i+1} = (T^i_N, i+1, q_{i+1}, n)$
  return $\mathcal{S}_{i+1}$
end procedure

Algorithm 8 VerifyState

Require: $pp, B_i, \pi_i, B_{i+1}, \pi_{i+1}$
Ensure: $\{0, 1\}$
1: **procedure** VERIFYSTATE($pp, B_i, \pi_i, B_{i+1}, \pi_{i+1}$)
2: **if** $\mathcal{V}(vk, B_{\{i,i+1\}}, \pi_{\{i,i+1\}}) = 0$ **then return** 0
3: **end if**
4: **if** $\mathcal{H}(\pi_{i+1}) > d$ **then return** 0
5: **end if**
6: **return** 1
7: **end procedure**

Overloading notation, set the selector polynomials $\mathbf{q_L}, \mathbf{q_R}, \mathbf{q_O}, \mathbf{q_M}, \mathbf{q_C} \in \mathbb{F}[X]$ defined for each $i \in [n]$ by

$$\mathbf{q_L}(g^i) := (\mathbf{q_L})_i,\ \mathbf{q_R}(g^i) := (\mathbf{q_R})_i, \mathbf{q_O}(g^i) := (\mathbf{q_O})_i,$$

$$\mathbf{q_M}(g^i) := (\mathbf{q_M})_i,\ \mathbf{q_C}(g^i) := (\mathbf{q_C})_i.$$

If (x, ω) is a relation $\mathcal{L}$ prepared for l public inputs, then $(x'\omega')$ is a relation for $\mathcal{L}'$ computed in the following way:

1. $\mathbf{PI(X)} := \sum_{i\in[l]} -x_i \cdot L_i(X)$

2. $\mathbf{f_L}, \mathbf{f_R}, \mathbf{f_O} \in \mathbb{F}[X]$ are defined by the following condition: $\forall i \in [n]$

$$\mathbf{f_L}(i) = \mathbf{x_{a}}_i,\ \mathbf{f_R}(i) = \mathbf{x_{b}}_i,\ \mathbf{f_O}(i) = \mathbf{x_{c}}_i.$$

It is easy to check that such a transition can be reversed, which yields the proof. □

Remark 1: Note that calculation of x' requires only the access to statement x and no access to secret witness ω.

Remark 2: Note that permutation σ was chosen in such a way that ω is a valid witness for $\mathcal{L}|_x$ iff $\mathbf{f_L}, \mathbf{f_R}, \mathbf{f_O}$ constructed as described before from a valid witness for $\mathcal{L}'|_{x'}$.

A.4 FRI Overview

Definition A.8. For a function $f : S \to \mathbb{F}$, let interpolant^f be the unique degree $< |S|$ polynomial that satisfies $\text{interpolant}^f(s) = f(s)$ for all $s \in S$. This polynomial can be constructed by Lagrange interpolation.

Setup phase. In the setup phase, the prover and verifier agree on the following parameters

- A prime field $\mathbb{F}$.
- A positive integer $R \in \mathbb{Z}_{>0}$ and the rate $\rho = 2^{-R}$.
- A multiplicative domain $D = D^{(0)} = \{\omega, \omega^2, \ldots, \omega^n\}$ generated by an element $\omega = \omega_0 \in \mathbb{F}^*$ of order $n = 2^k$ for some $k \in \mathbb{N}$. For chosen $\rho = 2^{-R}$ and $n = 2^k$ the protocol will check if f is of degree $< \rho n = 2^{k-R}$.
- The prover and verifier agree on a number of rounds $r < k - R \in \mathbb{N}$ and a sequence of subdomains $D^{(0)}, D^{(1)}, D^{(2)}, \ldots, D^{(r)}$, constructed inductively as follows: suppose $D^{(i)}$ was already defined and generated (as a cyclic group) by ω_i. Let $q(X) : \mathbb{F} \to \mathbb{F}$ be the map defined by the rule: $q(X) = X^2$. Then define $D^{(i+1)} = q(D^{(i)})$. Note that $D^{(i+1)}$ is cyclic subgroup of $\mathbb{F}^*$ generated by $\omega_{i+1} = \omega_i^2$. Note that $|D^{(i)}| = |D^{(0)}|/2^i$ and that $\forall i \in \{0, 1, \ldots, r-1\}$, $D^{(i)}$ can be split into cosets $\bigcup_j s_{ij} H^{(i)}$ where $H^{(i)}$ is the kernel of the homomorphism $q(X)|_{D^{(i)}} : D^{(i)} \to D^{(i+1)}$. Note that all cosets have equal size $|D^{(i)}|/|D^{(i+1)}| = 2$ and the number of cosets $j = |D^{(i)}|/2^{i+1}$.

When we say that prover commits to function f on domain D this means prover sends an oracle containing $f|_D$ i.e. all evaluations of function f on domain D.

Commit phase. In the commitment phase, the prover inductively constructs and commits to a sequence of functions $f^{(0)}, \ldots, f^{(r-1)}$ and a sequence of coefficients $a_0, \ldots, a_d$ with which the

verifier will construct the final function $f^{(r)}$.

- Input: a purported low degree polynomial $f^{(0)} := f \in \mathsf{RS}[\mathbb{F}, D^{(0)}, \rho]$. The prover commits to $f^{(0)}$ on $D^{(0)}$.

- For $0 \leq i < r$, given that $f^{(i)}$ was already defined (and committed to), the prover constructs $f^{(i+1)} : D^{(i+1)} \to \mathbb{F}$ in the following way:

  - The verifier sends a random $x^{(i)} \in \mathbb{F}$.
  - For $y \in D^{(i+1)}$, let $S_y = \{x \in D^{(i)} : q(x) = y\}$ be the coset of $D^{(i)}$ mapped to y.
  - Using interpolation, the prover constructs the polynomial

$$p_y^{(i)}(X) := \mathsf{interpolant}^{f^{(i)}|_{S_y}(X)},$$

    and defines

$$f^{(i+1)}(y) := p_y^{(i)}(x^{(i)}).$$

- If $i < r - 1$, the prover commits to the values of $f^{(i+1)}$ on $D^{(i+1)}$. If $i = r - 1$ then $f^{(r)}$ is a purported polynomial of degree $< \rho|D^{(r)}|$, in which case the prover commits to its coefficients $a_0, \ldots, a_d$.

Query phase. In the query phase, the verifier (probabilistically) validates the proof sent by the prover.

- Input: a sequence of oracles $f^{(0)}, \ldots, f^{(r-1)}$, and coefficients $a_0, \ldots, a_d$, with which the verifier constructs $f^{(r)}$, by

$$f^{(r)}(X) := \sum_{k=0}^{d} a_k X^k \in \mathsf{RS}[\mathbb{F}, D^{(r)}, \rho].$$

- Verifier generates a random $s^{(0)} \in D^{(0)}$ and for all $0 \leq i < r$ lets

1. $s^{(i+1)} := q(s^{(i)})$
2. $S^{(i)}$ be the coset of $H^{(i)}$ in $D^{(i)}$ containing $s^{(i)}$.

- For $0 \leq i < r-1$ the verifier checks that given $f^{(i)}$, the function $f^{(i+1)}$ was constructed according to the protocol:
  - She queries $f^{(i)}$ on all of $S^{(i)}$, and
  - computes $p^{(i)} = \mathsf{interpolant}^{f^{(i)}|_{S^{(i)}}}$, and
  - performs a "round consistency" check:

$$f^{(i+1)}(s^{(i+1)}) = p^{(i)}(x^{(i)}).$$

Note that in the last check, the function considered is $f^{(r)}$ which is in $\mathsf{RS}[\mathbb{F}, D^{(r)}, \rho]$ by construction. If all tests pass, the verifier accepts the proof. Otherwise, she rejects.

Remark: Instead of taking a family of nested sub-domains to be multiplicative subgroups it is also possible to take the cosets of them. To be more precise, consider any shift $g \in \mathbb{F}^* \backslash D$. There is a modification to the FRI protocol operating over the domains $D^{(0)'} = gD^{(0)}, D^{(1)'} = gD^{(1)}, \ldots, D^{(r)'} = gD^{(r)}$ The function mapping $D^{(i)'}$ to $D^{(i+1)'}$ is $q'(X) = q^{-1}X^2$. The modified version of FRI has the same security guarantees as the original one.

A.5 Supplementary Proofs

Proof of Theorem 12.2. It is immediate from the completeness of the FRI protocol that Algorithm 1 satisfies the completeness property for the given relation, which is verified directly by inspection. We also assume the existence of a (Gen, Com) binding commitment tuple and model it as an oracle to the IOP on domain D.

For the soundness bound, it suffices to show that the only source of soundness error comes from the FRI protocol. We concern ourselves with the situation when $q(X)$ passes the FRI check and the verifier is convinced that $q(X)$ is δ-close to some polynomial $h(X)$ with $\deg(h) < d - l$. This implies that, except at a δ-fraction of points on domain D, the following relation holds:

$$f(X) = U(X) + h(X) \prod_{i=1}^{l} (X - z_i).$$

Note that $t(X) = U(X) + h(X) \prod_{i=1}^{l} (X - z_i)$ is a polynomial of degree less than d. From the second equation we get that this polynomial is δ-close to $f(X)$ or that $\Delta(f, t) < \delta$. Moreover, we have that $\forall i \in [l]$, $t(z_i) = U(z_i) = y_i$ by the definition of $U(X)$. This means $t(X)$ satisfies all the requirements for the candidate polynomial g in the definition of $\mathcal{R}^{\delta}(\text{pp})$. Prover and verifier complexity results follow immediately by inspection of the IOP and the fact that the construction of an interpolation polynomial of degree k can be achieved with $O(k \log^3 k)$ field operations.

□

Proof of Theorem 12.5. We construct the given IOP by using the LPC scheme equipped with an additional oracle $O^{\mathcal{D}}$ providing access to $\mathcal{D}$ for both parties. Initially, the prover P queries $O^{\mathcal{D}}(f)$ and appends the output $\{x_i, w_i\}_{i=1}^{\mu}$ to its initial message to the verifier V. Subsequently, P and V simulate the LPC IOP for the input set $S = \{x_i, w_i\}_{i=1}^{\mu} \cup \{z_i, y_i\}_{i=1}^{l}$ of $\mu + l$ pairs. This is possible as both parties have access to S as the $\{z_i, y_i\}_{i=1}^{l}$ were provided as public input. By the security properties of the LPC scheme, except with probability $\epsilon(\delta)$ the prover can convince the verifier of the existence of some g for which $deg(g) < d$, $\Delta(f, g) < \delta$ and $\forall i \in [l], j \in [\mu]$ we have that $g(z_i) = y_i$ and $g(x_j) = w_j$.

Suffices to argue that $f = g$. We know by the properties of the distinguisher that with probability $1 - \eta(\delta)$, $\exists k \in [\mu]$ for which $\forall g \in L_{\delta}(f) \backslash \{f\}$, $g(x_k) \neq f(x_k)$. However, from the knowledge claim above we know that $\forall j \in [\mu], g(x_j) = w_j$ where $w_j = f(x_j)$ (since it was an oracle query response). This means that $g(x_k) = f(x_k)$ and, since $\Delta(f, g) < \delta$, we have that $f = g$. □

Proof of Claim 12.6. We begin with the case that $\mu = 1$. If at the setup step the choice of $x \in \mathbb{F}$ was random, by the Schwartz-Zippel lemma the probability that any degree d polynomial $g \in L_\delta(f)$ satisfies $g(x) = f(x)$ is:

$$\Pr_x[g(x) - f(x) = 0] \leq \frac{\deg\left(g(X) - f(X)\right)}{|\mathbb{F}|} \leq \frac{d}{|\mathbb{F}|}.$$

Enumerating over all $g_j \in L_\delta(f) \backslash \{f\}$ with a union bound:

$$\Pr_x\left[\bigcup_{j \in |L_\delta|-1} g_j(x) - f(x) = 0\right] \leq \frac{d}{|\mathbb{F}|} \cdot (|L_\delta| - 1).$$

For the multivariate case, the distinguisher needs to find at least one such point out of μ i.i.d random samples. This will only fail if all μ random values are not separation points. The result follows from the independence of the random samples, while the time complexity bound follows from the fact that one evaluation of f takes $O(d)$ time. □

Proof of Theorem 12.7. The random sampling distinguisher alongside the FRI-based LPC scheme define a PES that satisfies the above claim. The statement follows directly from Theorem 12.2 and Claim 12.6, while the upper bound on the soundness of η is obtained using Theorem 11.3. □

A.6 RedShift Security Analysis

Below we provide a proof of Theorem 13.6. For clarity, we consider the completeness and knowledge soundness cases separately. More specifically, the protocol satisfying the theorem statement is $\mathsf{RedShift}(\mathsf{pp}, \mathbf{PI}, n, \perp, N; \mathbf{f_L}, \mathbf{f_R}, \mathbf{f_O})$. For simplicity, we assume that any polynomial that uses the IOPP schemes can be decomposed into parts that each have at most degree n, thus allowing us to only require one RS code family for n. Note that here we ignore the mask polynomials by setting a null value for k.

A.6.1 Completeness

Assume P possesses a valid witness consisting of polynomials $\mathbf{f_L}, \mathbf{f_R}, \mathbf{f_O}$. which copy-satisfy σ. Note that the addition of masking polynomials doesn't change the values of $\mathbf{f_L}, \mathbf{f_R}, \mathbf{f_O}$ on D. It is straightforward to check that $F_6(X)$ is identically zero on D^* by the definition of witness polynomials, $F_1(X), F_2(X), F_3(X), F_4(X)$ will be zero on D^* by construction of $P(X)$ and $Q(X)$. To prove completeness of the protocol it is then enough to check that $F_5(X)$ is identically zero on D^*. Using the properties of the Lagrange basis, this is equivalent for $P(g^{n+1}) = Q(g^{n+1})$.

By definition of $P(X)$ and $Q(X)$, the above becomes:

$$\prod_{i=1}^{n}\prod_{j=1}^{3}\Big(f_j(g^i) + \beta \cdot k_j g^i + \gamma\Big) = \prod_{i=1}^{n}\prod_{j=1}^{3}\Big(f_j(g^i) + \beta \cdot \sigma'(k_j g^i) + \gamma\Big).$$

Since $\sigma' = \tau \circ \sigma \circ \tau^{-1}$, we rewrite this as follows:

$$\prod_{i=1}^{n}\prod_{j=1}^{3}\Big(f_{(j-1)n+i} + \beta \cdot \tau\big((j-1)n+i\big) + \gamma\Big) = \prod_{i=1}^{n}\prod_{j=1}^{3}\Big(f_{(j-1)n+i} + \beta \cdot \tau \circ \sigma\big((j-1)n+i\big) + \gamma\Big).$$

Now we use the fact that f_1, f_2, f_3 copy-satisfy $\sigma : f_{(j-1)n+i} = f_{\sigma((j-1)n+i)}$. Enumerating products on both sides proves equality, and hence completeness.

A.6.2 Knowledge Soundness

We require two auxiliary lemmas, proved in [Gabizon et al. 2019].

Lemma A.9. *Let $k \in \mathbb{N}$. Fix $F_1, \ldots, F_k \in \mathbb{F}[X]$ and $Z \in \mathbb{F}[X]$. Suppose that for some $i \in [k]$, $Z \nmid F_i$. Then, except with probability $\frac{1}{|\mathbb{F}|}$ over uniformly random $a_1, \ldots, a_k \in \mathbb{F}$, $Z \nmid F$, where $F := \sum_{i=1}^{k} a_i F_i$.*

Lemma A.10. *Let $n \in \mathbb{N}$. Fix a permutation σ of $[n]$, and $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{F}$. Suppose that*

for some $i \in [n]$ $b_i \neq a_{\sigma(i)}$. *Then except with probability* $\frac{n}{|\mathbb{F}|}$ *over random* $\beta, \gamma \in \mathbb{F}$*:*

$$\prod_{i=1}^{n} (a_i + \beta i + \gamma) = \prod_{i=1}^{n} (b_i + \beta\sigma(i) + \gamma) .$$

Let t_1 and t_2 denote the number of PES and LPC instances.

1. The PES is used on $S_{\sigma_1}, S_{\sigma_2}, S_{\sigma_3}, \mathbf{q_L}, \mathbf{q_R}, \mathbf{q_M}, \mathbf{q_O}, \mathbf{q_C}$ at point $y \in \mathbb{F}\backslash D$ sent by V, hence $t_1 = 8$. Although technically required, we do not evaluate the PES on $L_1, L_n, Z, S_{id_1}, S_{id_2}, S_{id_3}$ as these polynomials are in reduced form and can be evaluated by V without any help from P. More precisely, polynomials S_{id_j} for $j \in [3]$ are linear, $L_i(X)$ for $i \in [n+1]$ are of the form:

$$L_(X) = \frac{c_i(X^{n+1} - 1)}{X - g^i}$$

for some constant c_i and $Z(X)$ is of the form:

$$Z(X) = \prod_{a \in H^*} (X - a) = \frac{X^n - 1}{X - 1}.$$

2. LPC instances for witness polynomials $f_1(X) = \mathbf{f_L}(X), f_2(X) = \mathbf{f_R}(X), f_3(X) = \mathbf{f_O}(X), T_0(X)$, $T_1(X), T_2(X)$ (for $T = T_0 + X^n T_1 + X^{2n} T_2$) are evaluated at point y. For polynomials $P(X)$ and $Q(X)$ they are evaluated at points y and $y \cdot g$ (within one instance), giving $t_2 = 8$. Note that $t_2 = 6$ if we have T instead of the optimized T_0, T_1, T_2.

We begin by showing round-by-round soundness error. Suffices to construct a State function using the transcript of the proof, which is of the following form:

$$\mathsf{tr} := (f_1, f_2, f_3, \beta, \gamma, P, Q, \mathbf{a}, T, \mathbf{y}, \mathbf{g} \cdot \mathbf{y}, w, \mathsf{tr}_{LPC})$$

where $\mathbf{a} = (a_1, ..., a_6)$ and $\mathbf{y}, \mathbf{g} \cdot \mathbf{y} \in \mathbb{F}^N$ while for $j \in [3]$, $w :=$

$$T(\mathbf{y}), P(\mathbf{y}), P(\mathbf{g} \cdot \mathbf{y}), Q(\mathbf{y}), Q(\mathbf{g} \cdot \mathbf{y}), f_j(\mathbf{y}), S_{id_j}(\mathbf{y}), S_{\sigma_j}(\mathbf{y})$$

the openings of each evaluated polynomial at $\mathbf{y}, \mathbf{g} \cdot \mathbf{y}$ and $\text{tr}_{LPC} = (\text{tr}^1_{LPC}, ..., \text{tr}^{t_1+t_2}_{LPC})$ the transcript of the LPC evaluation routines, where tr^i_{LPC} the transcript of the i-th LPC routine. Note that the LPC routines have round-by-round soundness error ϵ_{FRI} and therefore admit a set of functions State^i_{LPC} for $i \in [t_1 + t_2]$.

$\text{State}(pp, \mathbf{PI}, n, \perp, N, \text{tr})$

1. If f_1, f_2, f_3, P, Q, T δ-close to codewords $\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{P}, \hat{Q}, \hat{T}$:
   (a) If $\hat{f}_1, \hat{f}_2, \hat{f}_3$ copy-satisfy σ and $q_{\mathbf{C}} + \mathbf{PI} + q_{\mathbf{L}}\hat{f}_1 + q_{\mathbf{R}}\hat{f}_2 + q_{\mathbf{O}}\hat{f}_3 + q_{\mathbf{M}}\hat{f}_1\hat{f}_2$‘ $= 0$, accept
   (b) If $Z(X)|\hat{F}_i(X)$ for all $i \in [6]$, accept
   (c) If any element of $\mathbf{a}$ is empty, reject
   (d) If $Z(X)|\sum_{i=1}^6 a_i\hat{F}_i(X)$, accept
   (e) If any element of $\mathbf{y}$ is empty, reject
   (f) If $\forall m \in [N], \sum_{i=1}^6 a_i\hat{F}_i(y_m) = \hat{T}(y_m) \cdot Z(y_m)$, accept
   (g) Reject
2. If $\text{tn}^i_{LPC} = \perp$ for some $i \in [t_1 + t_2]$, reject.
3. Return $\cap_{i=1}^{t_1+t_2}\text{State}^i_{LPC}(pp, \mathbf{PI}, n, N, \text{tr}^i_{LPC})$.

Note that $\text{tr} = \perp$, $\text{State}(pp, \mathbf{PI}, n, \perp, N, \perp) = 0$. We begin with Step 1, when the (partial) transcript provided contains a set of functions δ-close to codewords. Suppose that we have a non-satisfying assignment. By definition of the constraint system argument, this is a set of codewords $\hat{f}_1, \hat{f}_2, \hat{f}_3$ that either (1) don't copy-satisfy σ, or (2) don't satisfy the equality $q_{\mathbf{C}} + \mathbf{PI} + q_{\mathbf{L}}\hat{f}_1 + q_{\mathbf{R}}\hat{f}_2 +$

$q_{\mathrm{O}}\hat{f}_3 + q_{\mathrm{M}}\hat{f}_1\hat{f}_2$‘ $= 0$ on D^*. In this case, part (a) never holds. Part (b) outputs accept if $Z(X)$ divides all of the provided $\hat{F}_i$, which can be inferred from the transcript using $\hat{f}_1, \hat{f}_2, \hat{f}_3, \beta, \gamma, \hat{P}, \hat{Q}$. This however means that it divides $\hat{F}_6$, and therefore that the second condition in part (a) is satisfied. Thus the only way for part (b) to output success is if the $\hat{f}_1, \hat{f}_2, \hat{f}_3$ don't copy-satisfy σ and all $\hat{F}_i$ are divisible by Z. This happens with probability at most $1/|\mathbb{F}|$. This is because by the proof of completeness property:

$$\prod_{i=1}^{q}\prod_{j=1}^{3}\left(\hat{f}_{(j-1)q+i} + \beta\cdot\tau\big((j-1)q+i\big) + \gamma\right) =$$

$$\prod_{i=1}^{q}\prod_{j=1}^{3}\left(\hat{f}_{(j-1)q+i} + \beta\cdot\tau\circ\sigma\big((j-1)q+i\big) + \gamma\right). \qquad (\bullet)$$

However, P doesn't posses a valid witness (else part (a) would succeed) and, since $q = 1$ here, Lemma A.10 gives:

$$\Pr(\bullet \text{ holds } \mid \hat{f}_1, \hat{f}_2, \hat{f}_3 \text{ don't copy-satisfy } \sigma) \leq \frac{1}{|\mathbb{F}|}.$$

The probability of moving to accept in (d) is also upper bounded by $1/|\mathbb{F}|$, since $\mathbf{a}$ is randomly distributed (and some $\hat{F}_i$ is not divisible by Z, else (b) would accept); the result follows from Lemma 2. Part (f) moves to accept only in the case where no $\hat{T}$ exists for which the given equality holds identically, else part (d) would have accepted. Since the evaluations are at a random point $\mathbf{y} \in \mathbb{F}$, the probability that they coincide for two different polynomials is bounded above as $4n/|\mathbb{F}\backslash D|$, since the degrees of $\sum_{i=1}^{6} a_i\hat{F}_i, \hat{T}\cdot Z$ are at most $4n$ and thus the polynomials can only agree on up to $4n$ points without being identically equal. Using the Schwartz-Zippel lemma and taking a union bound over all possible tuples (note that the Johnson bound $J_{\rho,\nu}$ upper bounds the list size) yields an error term of $\epsilon_{\mathsf{IOP}} := J_{\rho,\nu}^{t_2}\cdot 4n/|\mathbb{F}\backslash D|$. To succeed for N independently random points, this yields an error term of $\epsilon_{\mathsf{IOP}}^N$. This gives a round-by-round error-contribution term of

$\epsilon_1 = \max\left(1/|\mathbb{F}|, \epsilon_{\mathsf{IOP}}^N\right)$ in the case when the input polynomials are δ-close to codewords.

If the transcript terminates within the first step, then we have that the error is at most ϵ_1 by the above analysis. In the case that the transcript ends after this, it must hold that there exists at least one of the prover provided oracles that is not δ-close to a codeword. We evaluate State_{LPC}^i $\forall i \in [t_1 + t_2]$ and an accepting output happens only if all transcripts tr_{LPC}^i for $i \leq t_1 + t_2$ lead to accepting states. However, at least one function is not δ-close to a codeword, which would add ϵ_{FRI} soundness error. Let the corresponding transcript of this function be tn_{LPC}^i, and notice that by the soundness of the LPC scheme, $\Pr(\text{State}_{LPC}^i(pp, \mathbf{PI}, n, N, \text{tr}_{LPC}^i) = 1) \leq \epsilon_{\mathsf{FRI}}$. By the Frechet inequality, the probability over a conjunction of events is upper bounded by the minimum of the probabilities of the individual events. We thus get that Step 3 has round-by-round soundness upper bounded by ϵ_{FRI}.

Remains to show that if State outputs 0, then so will the verifier. There are two situations in which State rejects a full transcript: if none of parts (a), (b), (d), (f) accept, or if some State_{LPC}^i rejects. In the former, the verifier will output 0 since the final step in verification will fail as otherwise step (f) would have passed. In the latter, rejection by State_{LPC}^i means that the verifier for the LPC will also fail. However, this means that the verifier for the whole IOP fails, as the second last step in verification is to check that all LPC/PES instances verify.

The extractor $\mathcal{E}(pp, \mathbf{PI}, n, \perp, N, \text{tr})$ with access to the first three functions as the transcript runs the Guruswami-Sudan list-decoding algorithm and returns the codewords $\hat{f_1}, \hat{f_2}, \hat{f_3}$ δ-close to f_1, f_2, f_3. If State moves to accept with probability greater than $\max(\epsilon_1, \epsilon_{\mathsf{FRI}})$, then $\hat{f_1}, \hat{f_2}, \hat{f_3}$ all copy-satisfy σ and identically satisfy $q_{\mathrm{C}} + \mathbf{PI} + q_{\mathrm{L}}\hat{f_1} + q_{\mathrm{R}}\hat{f_2} + q_{\mathrm{O}}\hat{f_3} + q_{\mathrm{M}}\hat{f_1}\hat{f_2}$‘ $= 0$, as part (a) has to accept in step 1 of State. This is by the round-by-round soundness analysis above, which bounds the probability of moving to an accepting state from an invalid starting witness to less than $\max(\epsilon_1, \epsilon_{\mathsf{FRI}})$.

A.6.3 Zero-Knowledge

Suffices to show that there exists a simulator S not possessing a valid witness for $\mathcal{L}$ such that S is able to generate a transcript $\langle S \rangle$ which is indistinguishable from the view of an honest prover-verifier interaction up to z. Since we need perfect zero knowledge, we will show that $z = 0$. For simplicity we denote the transcript in this context as the random variables $\langle a_1, a_2, \ldots, a_n \rangle$, where a_i represents either the verifier's messages/queries or the prover's responses to the corresponding queries. Note that the transcript doesn't capture any information about the oracles themselves: we treat all oracles as ideal and hence as exposing no data except for the elements sent in response to the verifier's queries (which are encoded inside a_i). This is a public-coin protocol in which all of the verifier's queries are randomly distributed field elements.

We are going to construct the simulator S and transcript $\langle S \rangle$ in the following way: we will set as many variables of the transcript as possible to be uniformly and randomly distributed. All remaining values will be uniquely fixed by the choice of the previous random variables. We will then show that such an approach finally results in the requirement for the witness polynomials f_i to have uniformly and randomly distributed values over some domains K_i (K_i are in general different for each witness polynomial). Then we will show that adding to each witness polynomial a masking polynomial $H_i(X)$ of degree at least $|K_i|$ is enough to achieve the required uniform distribution of values over K_i, which suffices for the proof. More specifically, to retain soundness we will add masking polynomials of the form $Z(X)H_i(X)$, as we don't want to change the values of $f_i(X)$ on the domain D^* defined by $Z(X)$.

We can rewrite the transcript of RedShift in the following form:

$$\langle \beta, \gamma, z, \mathcal{T}\langle f_1 \rangle, \mathcal{T}\langle f_2 \rangle, \mathcal{T}\langle f_3 \rangle, \mathcal{T}\langle T \rangle, \mathcal{T}\langle P \rangle, \mathcal{T}\langle Q \rangle \rangle,$$

where $\mathcal{T}\langle f \rangle$ denotes the part of the transcript corresponding to the LPC with respect to the witness oracle f. Note that we do not list the transcripts corresponding to the instances of the

elements that are precomputed as part of the public parameters.

Let β, γ to be uniformly randomly distributed over $\mathbb{F}$ and y to be uniformly and randomly distributed over $\mathbb{F}\backslash H$. As those values are also taken at random on exactly the same domains by a honest verifier during the actual interaction with the prover, this part of the transcript in $\langle S \rangle$ and $\langle P, V \rangle$ is equidistributed. For $\langle S \rangle$ we also take the openings of each witness function except for $T(X), P(X), Q(X)$ to be uniformly randomly distributed over $\mathbb{F}$. The evaluation of $T(X)$ at y is uniquely determined for any true transcript $\langle P, V \rangle$ and hence the same relation between variables should hold for the simulator's transcript $\langle S \rangle$ for them to be indistinguishable. Note, we have used the fact that $y \notin H$ here: in this case $Z(y) \neq 0$ and so we can obtain a unique value for the RHS of (1) that will satisfy (1) for any random choice of evaluations on LHS. Similarly for the values of $P(X)$ and $Q(X)$: by construction, these values are uniquely specified based on f_1, f_2, f_3. At any point when we sample a random element for the value of a witness polynomial $f_i(g^j)$ for $g^j \in H^*$ (which we have not sampled before), we make sure that the values of $T(g^j), P(g^j), Q(g^j)$ are updated so as to satisfy the above constraints. Note that this means we also need to sample $f_k(g^j), k \in [3]\backslash\{i\}$ as this is needed for P, Q, T. We keep track of all such oracle calls and return the provided value in the case of repeated queries, retaining consistency. In the analysis below, we implicitly do this check (and update) whenever it is stated that we randomly sample a new element as the evaluation of some function of a witness polynomial.

We now analyze the transcript $\mathcal{T}(f)$ for a given witness polynomial where the LPC is instantiated with FRI. In the actual interaction between the prover and verifier the transcript $\mathcal{T}\langle f \rangle$ is of the following form:

$$i_1, i_2, \ldots, i_k$$

$$z_1, z_2, \ldots, z_k$$

$$x^{(0)}, x^{(1)}, \ldots, x^{(r-1)}$$

$$a_0, s^{(0)}$$

$$q^{(0)}(s^{(0)}),\ q^{(0)}(t^{(0)}),\ \ldots,\ q^{(r-1)}(s^{(r-1)}),\ q^{(r-1)}(t^{(r-1)})$$

where:

1. $i_1, i_2, \ldots, i_k \in \mathbb{F}$ are the points at which the verifier asks to open oracle f. $k = 1$ for a single-point evaluation (conducted for witness polynomials f_1, f_2, f_3, T at y) and $k = 2$ for double evaluation (conducted for P and Q at the points y and $g \cdot y$).

2. $z_1, z_2, \ldots, z_k$ are the corresponding prover-sent openings.

3. $x^{(0)}, x^{(1)}, \ldots, x^{(r-1)}$ are random elements of $\mathbb{F}$ sent by the verifier during the FRI COMMIT phase, which is conducted with respect to the quotient function:

$$q(X) = q^{(0)}(X) = \frac{f(X) - U(X)}{\prod_{l=1}^{k}(X - i_l)}.$$

4. a_0 is the coefficient of $f^{(r)} \in \mathbb{F}$ sent by the prover at the end of the FRI COMMIT phase. Note that according to the remark at the end of the FRI section we assume all our instantiations of FRI are fully unrolled and hence that $f^{(r)}(x)$ is constant. The proof for the general case $\deg(f^{(r)}) > 0$ is only a little more involved and is handled in a similar fashion.

5. $s^{(0)} \in D$ is the value chosen by the verifier at the beginning of the FRI QUERY phase.

6. Every $s^{(i+1)} = q(s^{(i)})$ (for the definition of $q(x)$ refer to FRI section). $s^{(i)}, t^{(i)}$ is the coset of

$s^{(i+1)}$.

The simulated transcript $\langle S \rangle$ of FRI on f is constructed in the following way:

1. The point i (or two points (i_1, i_2)) are already fixed by the previous history of $\langle S \rangle$: i.e. $i = y$ or $(i_1, i_2) = (y, g \cdot y)$.

2. Similarly for the corresponding evaluations z_1 (or (z_1, z_2)): recall that they are either chosen at random (for witness polynomials f_1, f_2, f_3) or defined uniquely by all the previous values (for T, Q, P).

3. The values $x^{(i)}$ are distributed uniformly over $\mathbb{F}$ for an honest verifier V. We take the same approach in the simulator S: in $\langle S \rangle$ every $x^{(i)}$ is chosen uniformly at random from $\mathbb{F}$.

4. For $\langle S \rangle$ we take $s^{(0)}$ to be uniformly random over $D = D^{(0)}$.

5. In $\langle S \rangle$ the values of $q^{(0)}(s^0)$ and $q^{(0)}(t^{(0)})$ are also taken uniformly at random over $\mathbb{F}$.

6. Recall that in the FRI protocol we have:

$$q^{(i+1)}(s^{(i+1)}) = p^{(i)}_{s^{(i+1)}}(x^{(i)})$$

   where:

$$p^{(i)}_{s^{(i)}}(X) := \mathsf{interpolant}^{q^{(i)}|_{\{s^{(i)}, t^{(i)}\}}}(X),$$

   hence the value of every $q^{(i+1)}(s^{(i)})$ is uniquely determined by the values of $q^{(i)}(s^{(i)})$ and $q^{(i)}(t^{(i)})$ that were chosen at the previous iteration. In the simulator transcript, this relation between $q^{(i)}(s^{(i)}), q^{(i)}(t^{(i)}), q^{(i+1)}(s^{(i+1)})$ should remain unchanged. Since we have fixed the values of $q^{(0)}(s^{(0)}), q^{(0)}(t^{(0)})$ above, $q^{(1)}(s^{(1)})$ in $\langle S \rangle$ is then uniquely determined.

7. We proceed by induction: to fix the value $q^{(i)}(s^{(i)})$ we choose the value $q^{(i)}(t^{(i)})$ to be uniformly randomly distributed over $\mathbb{F}$ and compute $q^{(i+1)}(s^{(i+1)})$ for $i \in [r-2] \backslash \{1\}$.

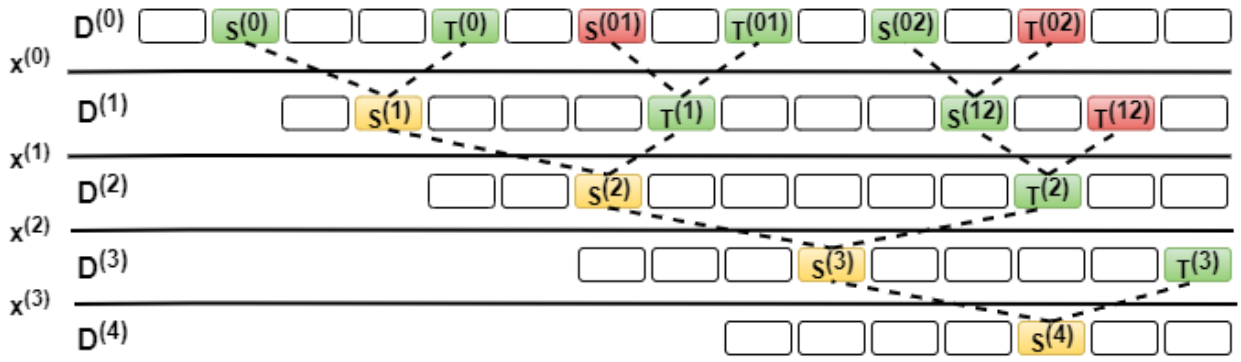


Figure A.1: FRI Transcript. Bold lines separate the adjacent levels of FRI, green blocks illustrate the values that are taken uniformly at random, yellow blocks represent the values that are uniquely determined by the coset of the previous layer, while red blocks have no impact on the construction of $\langle S \rangle$.

8. Compute a_0 based on $q^{(r-1)}(s^{(r-1)})$ and $q^{(r-1)}(t^{(r-1)})$.

Remains to show that this achieves the same distribution for transcripts in the honest prover-verifier interaction $\langle P, V \rangle$. First, for all witness polynomials (except for T, P, Q) we want their values at y to look like randomly distributed values over $\mathbb{F}$. Moreover, due to our construction of $\langle S \rangle$ we want the values

$$q^{(0)}(s^{(0)}), q^{(0)}(t^{(0)}), q^{(1)}(t^{(1)}), q^{(2)}(t^{(2)}), \ldots, q^{(r-1)}(t^{(r-1)})$$

to look uniformly random. In order to show this we need the following lemma:

Lemma A.11. *Let $f(X)$ denote the interpolation polynomial of $Z = \{z_1, z_2, \ldots, z_n\}$ over domain $I = \{i_1, \ldots, i_n\}$. Let $x \in \mathbb{F}$ be different from all $i_2, \ldots i_n$. If z_1 is uniformly random over $\mathbb{F}$ then $f(x)$ is also uniformly random over $\mathbb{F}$.*

Proof. Recall the Lagrange interpolation polynomial:

$$f(X) = \sum_{j=1}^{n} \prod_{k \neq j} \frac{X - i_k}{i_j - i_k} z_j.$$

Fix $x, i_1, \ldots, i_n$ and $z_1, \ldots, z_n$, $f(x)$ as a function of z_1 equals:

$$f(x) = az_1 + b.$$

where a, b - constants $\in \mathbb{F}$. Note that:

$$a = \prod_{k \neq 1} \frac{x - i_k}{i_1 - i_k} \neq 0,$$

provided x being different from all of $i_2, \ldots i_n$. For a linear function, the claim follows by the randomness of z_1. □

Consider $s^{(01)}$ and $t^{(01)}$ (the coset of $t^{(1)}$). Indeed, at least one of $s^{(01)}$ or $t^{(01)}$ is unequal to $x^{(0)}$. Without loss of generality assume $t^{(01)} \neq x^{(0)}$. We use Lemma A.11 for $I = \{s^{(01)}, t^{(01)}\}$, $Z = \{q^{(0)}(s^{(01)}), q^{(0)}(t^{(01)})\}$ and $x = x^{(0)}$ which implies that a uniform distribution of $q^{(0)}(t^{(01)})$ results in a uniform distribution for $q^{(1)}(t^{(1)})$, independent of the value of $q^{(0)}(s^{(01)})$.

We proceed by induction through repeated use of Lemma A.11. To achieve a uniformly random distribution for $q^{(2)}(t^{(2)})$ we need a uniformly random distribution for one of the values from the previous level: $q^{(1)}(s^{(12)})$ or $q^{(1)}(t^{(12)})$. Assume that $s^{(12)} \neq x^{(1)}$ (hence satisfying the conditions of Lemma A.11). The uniform distribution of $q^{(2)}(t^{(2)})$ then follows from the uniformly random distribution of one of $q^{(1)}(s^{(12)})$ which in turn follows from the uniform distribution of $q^{(0)}(s^{(02)})$. The same logic is then applied for all downstream layers of FRI. This is illustrated in Fig. A.1.

Finally, to achieve the same distribution of variables in transcripts $\langle P, V \rangle$ and $\langle S \rangle$ we need to add more “degrees of freedom" for each witness polynomial $f_i, i \in [3]$. More precisely, we want

the evaluation:

$$q^{(0)}(s^{(0)}),\ q^{(0)}(t^{(0)}),\ q^{(0)}(t^{(01)}),$$
$$q^{(0)}(s^{(02)}),\ \ldots\ (r_i + 1 \text{ values in total})$$

over the set $K_i^{'} = \{s^{(0)}, t^{(0)}, t^{(01)}, s^{(02)}, \ldots\}$ on the top level of FRI to be uniformly random for each $i \in [3]$. Now, recall that:

$$q^{(0)}(X) = q(X) = \frac{f(X) - U(X)}{\prod_{l=1}^{k}(X - i_l)}. \quad (**)$$

In this case the sets $\{i_1, \ldots, i_k\}$ and D are disjoint. This in turn means that a uniformly random distribution of values of $q^{(0)}(X)$ over $K_i^{'}$ is exactly the same as the uniformly random distribution of values of f_i over the same domain (as all other terms in $(**)$ are now fixed by previous considerations). Plugging in the the requirement for $f_i(y)$ to be also uniformly randomly distributed we arrive at the set $K_i = K_i^{'} \cup y$ with $|K_i| = r_i + 2$ at which the values of f_i should look like random elements in $\mathbb{F}$ for $i \in [3]$.

Since T, P, Q are each fully specified by f_1, f_2, f_3 on D, we need to add to K_i any potential oracle queries in the FRI instances for T, P, Q that sampled values for which the f_i were not already queried. Denote the number of these new calls by r_T, r_P, r_Q. Since for all of T, P, Q queried at a point $g \in D$ the values of $f_i(g)$ have to also be queried if they were not queried already, any potential extra queries due to these variables are added to the respective degrees of all three witness polynomials f_i. To achieve this property, it is enough to replace $f_i(X)$ by $f_i^{'}(X) = f(X) + H_i(X)Z(X)$ where $H_i(x)$ is a random polynomial of degree $r_i + r_T + r_P + r_Q + 1$.

A.7 FRI parameters

As described in the main text of the paper and in particular in Section 13.3, one has the freedom to pick FRI parameters that also affect contributions into the soundness error of RedShift due to the list size $|L|$. In general, *smaller* list sizes will lead to a *smaller* δ parameter (this is intuitively expected, as a larger list size requires less sensitivity) that in turn *reduces* FRI soundness for a chosen domain D, parameter ρ and number of queries. Alternatively, one can pick another limit in the FRI soundness formula

$$p(\rho, \nu) = \left(1 - min\left\{\delta_0, J_\nu(J_\nu(1-\rho))\right\} + \nu \log |D|\right)$$

and set $\delta_0 = (1-\rho)/2$ to be in the unique decoding radius. In this case list size $|L| = 1$, but FRI has *smaller* soundness for the same number of queries. This means that one has to pay particular attention to the final system soundness as described in Section 13.4: for in the case where $\epsilon_1, \epsilon_{FRI} \sim \epsilon_1$ one should also consider the case of the limit $\delta_0 = (1-\rho)/2$ in the FRI soundness term and can recalculate ϵ_1 and thus a final soundness in case of list size $|L| = 1$. Such checks are also important if one would want to reduce the field size for a corresponding reduction in proof size.

A.8 Proof Size Optimizations

There are various options that can reduce the proof size. Some of the are described in detail in [Chiesa et al. 2019b]. There are two essential parts to check satisfiability at the random point y:

1. Consistency between polynomial openings at y. The prover sends the purported evaluations to the verifier and these values are used for two subroutines:

   (a) Check the equations from Section 13 at y.

(b) Simulate oracles to the quotient function $q(y)$.

2. Proximity testing performed by the invocation of FRI.

Merging Oracles: As described in [Chiesa et al. 2019b], the prover can join the evaluations of the different polynomials over the domain D into a single oracle by placing the corresponding values into the same leaf of the Merkle tree. This reduces the total number of Merkle paths required for authentication, which is a bottleneck for proof size. We can perform such a joining operation for the following sets of polynomials:

1. Constraint polynomials: selectors q_L, q_R, q_O, q_M, q_C and permutation polynomials $S_{id_1}, S_{\sigma_1}, S_{\sigma_2}, S_{\sigma_3}$, as all those are independent and prepared at setup.

2. Witness polynomials: f_L, f_R, f_O.

3. Grand product polynomials: P, Q.

4. Polynomials T_0, T_1, T_2 for which

$$T(X) = X^{2n}T_2(X) + X^nT_1(X) + T_0.$$

While we initially have to provide 17 independent Merkle paths for the authentication of various oracle values, this optimization reduces their number to 4, directly reducing proof size and verification time. This is due to the smaller number of prover-provided Merkle paths which imply a smaller number of hash function invocations, which are the current verification time bottleneck. Such an argument universally applies to all the optimization described below that also reduce proof sizes.

Bitreversed Domain Element Enumeration as Merkle Tree Leaves: Another important optimizations for FRI is to use "bitreverse" enumeration when placing the claimed LDE values into

the Merkle tree. In this case, values that form the coset required for the FRI "folding" step are always adjacent and can thus be placed in the same leaf (combined with the optimization below), sharing a single Merkle path per FRI intermediate oracle query step. We do not use this optimization in the prototype implementation.

Concatenating Merkle Tree Leaves: We can place more values into the leaves of every Merkle tree used to instantiate the oracles. This optimization allows us to use a larger 'localization parameter' for FRI and thus reduce the number of intermediate oracles. In practice implementations follow an adaptive strategy where the localization parameter is large (usually 8) for the initial FRI stages when the Merkle path is "long" and decreases it when the tree becomes more shallow.

Other optimizations exist: e.g. performing proof-of-work on top of challenge values obtained from the transcript to reduce the number of required FRI queries (as used in [StarkWare 2022]) and other estimates for the number of required queries. To the best of our knowledge, there is no public analysis for such optimizations and we thus do not use them in our analysis.

For completeness we should also mention that for a substantial number of queries of Merkle path elements that are close to the root are often duplicate between queries and thus it may be beneficial to send them once, only later sending the values that are missing to complete the path to the leafs. One can also perform a smaller number of FRI "folding" steps and output not just a single coefficient of the claimed low degree polynomial, but settle on a larger degree based on the expected number of queries.

A.9 Batched FRI

Consider the following theorem found in [Ben-Sasson et al. 2018a]. We can define $J_v^{[k]}(\lambda) := J_v(J_v(\cdots(J_v(\lambda))))$, where there are k iterations of the function J_v. We also denote the *relative*

hamming distance of set $S \subseteq \mathbb{F}^n$ as

$$\Delta(S) = \min\{\Delta(w, w_0) | w, w_0 \in S, w \neq w_0\}.$$

Theorem A.12. *Let $V \subseteq \mathbb{F}^n$ be a linear space over a finite field $\mathbb{F}$ with $\Delta(V) = \lambda$. Let $u^* \in \mathbb{F}^n$ and $\epsilon > 0$ satisfy $\delta < J_\epsilon^{[l+1]}(\lambda)$. For $u_1, u_2, \ldots, u_l \in \mathbb{F}^n$ define*

$$A = \left\{ \alpha \in \mathbb{F}^* \middle| \Delta(u^* + \sum_{i=1}^{l} \alpha^i u_i, V) < \delta \right\}.$$

If $|A| > l \cdot (2/\epsilon)^{l+2}$, then $\forall j \in [l], \exists v^*, v_j \in V$ such that:

$$\left| \left\{ i \;\middle|\; (u_i^* = v_i^*) \wedge \left(\bigwedge_{j=1}^{l} (u_j)_i = (v^l)_i \right) \right\} \right| \geq (1 - \delta - \epsilon)n,$$

where $(u_j)_i$ denotes the i-th coordinate of u_j and $i \in [n]$.

In particular, $\Delta(u^*, v^*) \leq \delta + \epsilon$ and $\forall i \in [l]$:

$$\Delta(u_i, v_i) \leq \delta + \epsilon.$$

Specifying this theorem for $V = \mathsf{RS}[\mathbb{F}, D, \rho]$ (for which $\lambda = \Delta(V) = 1 - \rho$), the contrapositive yields the following corollary:

Corollary A.13. *Let $V = RS[\mathbb{F}, D, \rho]$ be the family of RS-codes. Let $\epsilon \in (0, 1), \delta > 0$ satisfy $\delta < J_\epsilon^{[l]}(1 - \rho)$. Let $l \geq 2 \in \mathbb{N}$ and $u_1, u_2, \ldots, u_l \in \mathbb{F}^n$, such that there exists $i \in [l]$ for which $\Delta(u_i, V) > \delta + \epsilon$. Then it holds that:*

$$|A| \leq (l - 1) \left(\frac{2}{\epsilon} \right)^{l+1}.$$

We sketch a batched FRI protocol, the correctness of which is a trivial consequence of the

previous corollary.

Batched FRI protocol:

1. P publishes oracles to $f_1, \ldots, f_k$.
2. V selects random $\alpha \in \mathbb{F}^*$ and sends it to P.
3. P and V perform FRI w.r.t $f = \sum_{i=1}^{k} \alpha^{i-1} f_i$.
4. V accepts if the previous step accepts.

Bibliography

Abraham, I. and Dolev, D. (2015). Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 605–614.

Adler, J. and Quintyne-Collins, M. (2019). Building scalable decentralized payment systems. *arXiv preprint arXiv:1904.06441*.

Aleo (2022). Announcing the ZPrize Competition. https://www.aleo.org/post/announcing-the-zprize-competition. Accessed: 2022-08-09.

Allsopp, P., Summers, B., and Veale, J. (2009). The evolution of real-time gross settlement. *The World Bank, Financial Infrastructure Series*.

Auer, R. and Böhme, R. (2020). The technology of retail central bank digital currency. *BIS Quarterly Review, March*.

Auer, R., Monnet, C., and Shin, H. S. (2021). Permissioned distributed ledgers and the governance of money. *Available at SSRN 3770075*.

Back, A. (2002). Hashcash-a denial of service counter-measure.

Bagaria, V., Kannan, S., Tse, D., Fanti, G., and Viswanath, P. (2019). Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602.

Bahack, L. (2013). Theoretical bitcoin attacks with less than half of the computational power (draft). *arXiv preprint arXiv:1312.7013*.

Ball, M., Rosen, A., Sabin, M., and Vasudevan, P. N. (2017a). Average-case fine-grained hardness. In *ACM SIGACT Symposium on Theory of Computing*.

Ball, M., Rosen, A., Sabin, M., and Vasudevan, P. N. (2017b). Proofs of useful work. *IACR Cryptology ePrint Archive*, 2017:203.

Bech, M. L., Shimizu, Y., and Wong, P. (2017). The quest for speed in payments. *BIS Quarterly Review March*.

Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. (2018a). Fast reed-solomon interactive oracle proofs of proximity. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. (2018b). Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046.

Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. (2019a). Scalable zero knowledge with no trusted setup. In *Annual International Cryptology Conference*, pages 701–732. Springer.

Ben-Sasson, E., Carmon, D., Ishai, Y., Kopparty, S., and Saraf, S. (2020). Proximity gaps for reed–solomon codes. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 900–909. IEEE.

Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., and Virza, M. (2013). Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*.

Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., and Ward, N. P. (2018c). Aurora: Transparent succinct arguments for r1cs. Cryptology ePrint Archive, Report 2018/828. https://eprint.iacr.org/2018/828.

Ben-Sasson, E., Chiesa, A., and Spooner, N. (2016). Interactive oracle proofs. In *Theory of Cryptography Conference*, pages 31–60. Springer.

Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. (2014). Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796.

Ben-Sasson, E., Chiesa, A., Tromer, E., and Virza, M. (2017). Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4).

Ben-Sasson, E., Goldberg, L., Kopparty, S., and Saraf, S. (2019b). Deep-fri: Sampling outside the box improves soundness. *arXiv preprint arXiv:1903.12243*.

Ben-Sasson, E., Kopparty, S., and Saraf, S. (2018d). Worst-case to average case reductions for the distance to a code. In *Proceedings of the 33rd Computational Complexity Conference*, CCC '18, pages 24:1–24:23, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

BIS, C. (2018). Looking beyond the hype. *Bank of International Settlement, Basel*.

Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., and Waters, B. (2016). Time-lock puzzles from randomized encodings. In *ACM ITCS*.

Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. (2018). Verifiable delay functions. In *Annual International Cryptology Conference*, pages 757–788. Springer.

Boneh, D., Drake, J., Fisch, B., and Gabizon, A. (2020). Halo infinite: Recursive zk-snarks from any additive polynomial commitment scheme. *Cryptology ePrint Archive*.

Bonneau, J., Meckler, I., Rao, V., and Shapiro, E. (2020). Mina: Decentralized cryptocurrency at scale. https://docs.minaprotocol.com/static/pdf/technicalWhitepaper.pdf. Accessed: 2022-08-09.

Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., and Felten, E. W. (2015). Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *IEEE Security and Privacy*.

Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., and Wu, H. (2018a). Zexe: Enabling decentralized private computation. Cryptology ePrint Archive, Report 2018/962.

Bowe, S., Gabizon, A., and Green, M. D. (2018b). A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. In *Financial Crypto*.

Bowe, S., Gabizon, A., and Miers, I. (2017). Scalable multi-party computation for zk-snark parameters in the random beacon model. *IACR Cryptology ePrint Archive*, 2017:1050.

Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. (2018). Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE.

Buterin, V. (2014). Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper. Accessed: 2022-08-09.

Buterin, V. (2018). On-chain scaling to potentially 500 tx/sec through mass tx validation. https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477. Accessed: 2022-08-09.

Bünz, B., Agrawal, S., Zamani, M., and Boneh, D. (2019a). Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Report 2019/191.

Bünz, B., Fisch, B., and Szepieniec, A. (2019b). Transparent snarks from dark compilers. Cryptology ePrint Archive, Report 2019/1229. https://eprint.iacr.org/2019/1229.

Bünz, B., Kiffer, L., Luu, L., and Zamani, M. (2019c). Flyclient: Super-light clients for cryptocurrencies. Cryptology ePrint Archive, Report 2019/226.

Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G. N., and Rothblum, R. D. (2018). Fiat-shamir from simpler assumptions. *Cryptology ePrint Archive*.

Chaidos, P., Cortier, V., Fuchsbauer, G., and Galindo, D. (2016). Beleniosrf: A non-interactive receipt-free electronic voting scheme. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1614–1625. ACM.

Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., and Shi, W. (2017). On security analysis of proof-of-elapsed-time (PoET). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*.

Chen, W., Chiesa, A., Dauterman, E., and Ward, N. P. (2020). Reducing participation costs via incremental verification for ledger systems. *Cryptology ePrint Archive*.

Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., and Ward, N. (2019a). Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Report 2019/1047. https://eprint.iacr.org/2019/1047.

Chiesa, A., Ojha, D., and Spooner, N. (2019b). Fractal: Post-quantum and transparent recursive proofs from holography. Cryptology ePrint Archive, Report 2019/1076. https://eprint.iacr.org/2019/1076.

Dahari, H. and Lindell, Y. (2020). Deterministic-prover zero-knowledge proofs. Cryptology ePrint Archive, Paper 2020/141.

Daian, P., Eyal, I., Juels, A., and Sirer, E. G. (2017). (Short paper) Piecework: Generalized outsourcing control for proofs of work. In *Financial Crypto*.

Damgård, I. B., Pedersen, T. P., and Pfitzmann, B. (1993). On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *Annual International Cryptology Conference*, pages 250–265. Springer.

Decker, C. and Wattenhofer, R. (2013). Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE.

Dolev, D. and Strong, H. R. (1982). Polynomial algorithms for multiple processor agreement. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 401–407.

Dziembowski, S., Faust, S., and Hostáková, K. (2018). General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966.

Erdös, P. (1960). Remarks on number theory III. On addition chains. *Acta Arithmetica*, 6.

Evgenya, P. (2017). Algebraic ram. Master's thesis, Technion.

Eyal, I., Gencer, A. E., Sirer, E. G., and Van Renesse, R. (2016). Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 45–59.

Eyal, I. and Sirer, E. G. (2014). Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer.

Fisch, B. (2019). Tight proofs of space and replication. In *Eurocrypt*.

Fisch, B., Bonneau, J., Greco, N., and Benet, J. (2018). Scaling proof-of-replication for Filecoin mining. Technical report, Stanford University.

Fischer, M. J. and Lynch, N. A. (1981). A lower bound for the time to assure interactive consistency. Technical report, Georgia Institute of Tech Atlanta School of Information and Computer Science.

Fuchsbauer, G., Kiltz, E., and Loss, J. (2018). The algebraic group model and its applications. In *Annual International Cryptology Conference*, pages 33–62. Springer.

Gabizon, A., Gurkan, K., Jovanovic, P., Konstantopoulos, G., Oines, A., Olszewski, M., Straka, M., Tromer, E., and Vesely, P. (2020). Plumo: towards scalable interoperable blockchains using ultra light validation systems.

Gabizon, A. and Williamson, Z. J. (2020). Proposal: The turbo-plonk program syntax for specifying snark programs. https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf. Accessed: 2022-08-09.

Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Technical report, Cryptology ePrint Archive, Report 2019/953.

Garay, J. A. and Moses, Y. (1998). Fully polynomial byzantine agreement for n> 3 t processors in t+ 1 rounds. *SIAM Journal on Computing*, 27(1):247–290.

Gennaro, R., Gentry, C., Parno, B., and Raykova, M. (2013). Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer.

Gennaro, R., Minelli, M., Nitulescu, A., and Orrù, M. (2018). Lattice-based zk-snarks from square span programs. In *ACM CCS*.

Gentry, C. and Wichs, D. (2011). Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108. ACM.

Goldwasser, S., Kalai, Y. T., and Rothblum, G. N. (2015). Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):27.

Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208.

Gordon, D. M. (1998). A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1).

Groth, J. (2016). On the size of pairing-based non-interactive arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 305–326. Springer.

Guruswami, V. and Sudan, M. (1999). Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767.

Hastings, M., Heninger, N., and Wustrow, E. (2018). The proof is in the pudding: Proofs of work for solving discrete logarithms. Cryptology ePrint Archive, Report 2018/939.

Henry, R. (2010). Pippenger's multiproduct and multiexponentiation algorithms. Technical report, University of Waterloo.

Huffman, W. C. and Pless, V. (2003). *Fundamentals of error-correcting codes*. Cambridge Univ. Press, Cambridge.

Kamvar, S., Olszewski, M., and Reinsberg, R. (2019). Celo: A multi-asset cryptographic protocol for decentralized social payments.

Karantias, K., Kiayias, A., Leonardos, N., and Zindros, D. (2019). Compact Storage of Superblocks for NIPoPoW Applications. Cryptology ePrint Archive, Report 2019/1444.

Kate, A., Zaverucha, G. M., and Goldberg, I. (2010). Constant-size commitments to polynomials and their applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 177–194. Springer.

Kiayias, A., Lamprou, N., and Stouka, A.-P. (2016). Proofs of proofs of work with sublinear complexity. In *Financial Crypto*.

Kiayias, A., Miller, A., and Zindros, D. (2017). Non-interactive proofs of proof-of-work. *IACR Cryptology ePrint Archive*, 2017:963.

King, S. (2013). Primecoin: Cryptocurrency with prime number proof-of-work. http://launch.primecoin.org/static/primecoin-paper.pdf. Accessed: 2022-08-09.

King, S. and Nadal, S. (2012). Peercoin: Peer-to-peer crypto-currency with proof-of-stake. https://peercoin.net/whitepaper. Accessed: 2022-08-09.

Kogias, E. K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., and Ford, B. (2016). Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th usenix security symposium (usenix security 16)*, pages 279–296.

Król, M., Sonnino, A., Al-Bassam, M., Tasiopoulos, A., and Psaras, I. (2019). Proof-of-prestige: A useful work reward system for unverifiable tasks. In *IEEE Conference on Blockchain and Cryptocurrency (ICBC)*.

Kroll, J. A., Davey, I. C., and Felten, E. W. (2013). The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013. Washington, DC.

Leung, D., Suhl, A., Gilad, Y., and Zeldovich, N. (2018). Vault: Fast bootstrapping for cryptocurrencies.

Maller, M., Bowe, S., Kohlweiss, M., and Meiklejohn, S. (2019). Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. *IACR Cryptology ePrint Archive*, 2019:99.

Maurer, U. (2005). Abstract models of computation in cryptography. In *IMA International Conference on Cryptography and Coding*.

Micali, S. (2000). Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298.

Miers, I., Garman, C., Green, M., and Rubin, A. D. (2013). Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE.

Miller, A., Juels, A., Shi, E., Parno, B., and Katz, J. (2014). Permacoin: Repurposing bitcoin work for data preservation. In *IEEE Symposium on Security and Privacy*.

Miller, A., Kosba, A., Katz, J., and Shi, E. (2015). Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *ACM CCS*.

Miller, A. and LaViola Jr, J. J. (2014). Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. *Available on line: http://nakamotoinstitute. org/research/anonymous-byzantine-consensus*.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf. Accessed: 2022-08-09.

Parno, B., Gentry, C., Howell, J., and Raykova, M. (2013). Pinocchio: Nearly Practical Verifiable Computation. Cryptology ePrint Archive, Report 2013/279.

Pass, R. and Shi, E. (2016). Hybrid consensus: Efficient consensus in the permissionless model. *Cryptology ePrint Archive*.

Pease, M., Shostak, R., and Lamport, L. (1980). Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234.

Pippenger, N. (1980). On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2).

Poelstra, A. (2016). Mimblewimble. https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf. Accessed: 2022-08-09.

Poon, J. and Buterin, V. (2017). Plasma: Scalable autonomous smart contracts. Accessed: 2022-08-09.

Poon, J. and Dryja, T. (2015). The bitcoin lightning network. *Scalable on-chain instant payments*.

Sasson, E. B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. (2014). Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*.

Schnorr, C.-P. (1989). Efficient identification and signatures for smart cards. In *Eurocrypt*.

SCIPRLab (2017). libsnark: a c++ library for zksnark proofs. https://github.com/scipr-lab/libsnark.

Sengupta, B., Bag, S., Ruj, S., and Sakurai, K. (2016). Retricoin: Bitcoin based on compact proofs of retrievability. In *ICDCN*.

Setty, S. (2020). Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer.

Setty, S., Angel, S., Gupta, T., and Lee, J. (2018). Proving the correct execution of concurrent services in zero-knowledge. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 339–356.

Setty, S., Braun, B., Vu, V., Blumberg, A. J., Parno, B., and Walfish, M. (2013). Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84. ACM.

Shostak, R., Pease, M., and Lamport, L. (1982). The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401.

Shoup, V. (1997). Lower bounds for discrete logarithms and related problems. In *Eurocrypt*.

Sompolinsky, Y., Lewenberg, Y., and Zohar, A. (2016). Spectre: A fast and scalable cryptocurrency protocol. *Cryptology ePrint Archive*.

Sompolinsky, Y. and Zohar, A. (2015). Secure high-rate transaction processing in bitcoin. In *International conference on financial cryptography and data security*, pages 507–527. Springer.

Sompolinsky, Y. and Zohar, A. (2018). Phantom. *IACR Cryptology ePrint Archive, Report 2018/104*.

StarkWare (2022). Starkdex: Bringing starks to ethereum. https://blog.0xproject.com/starkdex-bringing-starks-to-ethereum-6a03fffc0eb7. Accessed: 2022-08-09.

Valiant, P. (2008). Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography Conference*, pages 1–18. Springer.

Wahby, R. S., Tzialla, I., Shelat, A., Thaler, J., and Walfish, M. (2018). Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943. IEEE.

Wang, G., Wang, S., Bagaria, V., Tse, D., and Viswanath, P. (2020). Prism removes consensus bottleneck for smart contracts. In *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 68–77. IEEE.

Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. Accessed: 2022-08-09.

Wu, H., Zheng, W., Chiesa, A., Popa, R. A., and Stoica, I. (2018). DIZK: A Distributed Zero Knowledge Proof System. In *USENIX Security*.

Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., and Song, D. (2019). Libra: Succinct zero-knowledge proofs with optimal prover computation. *IACR Cryptology ePrint Archive*, 2019:317.

Zero, P. (2021). Plonky2: Fast recursive arguments with plonk and fri. https://github.com/mir-protocol/plonky2/blob/main/plonky2/plonky2.pdf. Accessed: 2022-08-09.

Zhandry, M. (2022). To label, or not to label (in generic groups). Cryptology ePrint Archive, Paper 2022/226.

Zhang, F., Eyal, I., Escriva, R., Juels, A., and Van Renesse, R. (2017). REM: Resource-efficient mining for blockchains. In *USENIX Security*.

Zhang, J., Xie, T., Zhang, Y., and Song, D. (2020). Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 859–876. IEEE.