IMPROVING THE PRIVACY, SCALABILITY, AND ECOLOGICAL IMPACT OF BLOCKCHAINS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Benedikt Bünz
June 2023

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Dan Boneh) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mary Wooters)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(David Mazierez)

Approved for the Stanford University Committee on Graduate Studies

Abstract

Despite the excitement around them, Blockchains today still suffer from major limitations. Popular blockchains have high transaction fees, limited privacy, and substantial ecological impact. These issues are largely due to the fact that in decentralized blockchain systems, each action performed by one party gets seen and verified by all parties in the network. In this dissertation, we develop solutions to these limitations. They rely on a common proof paradigm, in which a prover can generate a certificate that the actions it performed were in accordance with a set of rules. Importantly, the certificates can be highly efficient and privacy-preserving, even if the actions are not. We develop four such proof systems, each tailored to resolve one of the limitations of blockchains. Bulletproofs, a zero-knowledge proof system that is designed for private blockchain transactions. HyperPlonk, an efficient SNARK that enables outsourcing the processing of entire blocks of transactions. Verifiable Delay Functions, a key component for ecologically friendly blockchains, and ProtoStar, an incrementally verifiable proof system designed for building efficient Verifiable Delay Functions and succinct blockchains. These tools are not merely theoretical but have already found practical applications and are securing systems worth billions of dollars.

Acknowledgments

When I was asked what I wanted to be, as a child, I would respond: "Not a doctor" (my form of rebellion). Well, now I am one, just not the 'real' kind. This journey would not have been possible, and certainly not nearly as enjoyable, without the many amazing people who supported me along the way.

First and foremost, this is my advisor Dan Boneh. I walked into Dan's Cryptography class as a Master's student, knowing something about cryptocurrencies but very little about cryptography. Dan's excitement for this topic was, and to this day is, contagious, and I immediately fell in love with the subject. I could not have wished for a better advisor. Any time after meeting with him, I would feel energized, motivated, and usually filled with a new insight or answer from the 'oracle'. I, especially appreciated his support and his ability to open doors for his students. After my first year, he was asked to give a keynote at a big blockchain conference on the 'state of cryptography'. He suggested that I, who was far from an expert at that point, gave the talk. This single event was a major confidence builder, and I hope that I can emulate him as an advisor myself.

The second person that made this Ph.D. possible is Ben Fisch. My first memory of Ben was discussing (to the outside, it would have looked like a full-blown fight) the general merits of decentralized blockchains. I thought I did a terrible job for the defense and was just not getting through. This was the unorthodox start of an incredibly successful collaboration and more importantly, a great friendship. Ben has made me a significantly better researcher by forcing me to be less hand-wavy through endless discussions (not fights anymore) and by fixing most (unfortunately not always all) of the bugs I carefully hid in our proofs. We did end up working on quite a few blockchain-related topics, so maybe I did get through after all.

I also want to thank Binyi Chen, who I have tremendously enjoyed collaborating with in these last couple of years and who has become a close friend. I am looking forward to many more years of working together! Joe Bonneau taught me that good research can and should be combined with good humor and life balance. I cannot thank him enough for his support from before I started my Ph.D. till recently on the academic job market. I am very much looking forward to being colleagues with him! I would not have made it to Stanford without Sven Seuken and Ben Lubin. They first sparked my interest in research, and by treating me as an equal, gave me the confidence to think

that I could be a successful researcher.

I also want to thank all my other co-authors that I have had the pleasure of collaborating with: Mary Maller, Pratyush Mishra, Nirvan Tyagi, Jeremy Clark, Psi Vesely, Alessandro Chiesa, William Lin, Nicholas Spooner, Zhenfei Zhang, Alan Szepieniec, Shashank Agrawal, Mahdi Zamani, Lucianna Kiffer, Loi Luu, Jonathan Bootle, Andrew Poelstra, Pieter Wuille, Greg Maxwell, Fernando Krell, Alex Xiong, Philippe Camacho, Elaine Shi, and Yuncong Hu. It is a great joy to work in such a collaborative field, and I hope to add many more people to this list in years to come.

I started my Ph.D. in the Applied Cryptography Group together with Dima Kogan, Florian Tramer, Saba Eskandarian, and significantly overlapped with Ben, Henry Corrigan-Gibbs, Riad Whaby, David Wu, Sam Kim, and Alex Ozdemir. I am incredibly proud of this group and all that we have accomplished, and the culture we created. We had a great mix of support, friendship, and competitiveness that never once got toxic. I am very confident that this new generation in the Applied Cryptography Group will continue and build on this legacy. I also want to thank Ruth Harris and the rest of the Stanford staff, who graciously put up with me for all these years.

Essentially, the only jour fixe in my last 7 years was Tuesdays, 6 pm running practice with the Peninsula Distance Club. PDC and the people in it were a vital constant in an otherwise, at times, tumultuous Ph.D. journey. Running in circles with Matt, Chris, Steven, Kevin, Trevor, Sarah, Maya, and Brendon, while getting told life stories and advice by our coach Dena is the single thing I most looked forward to in any given week. I have been blessed with many amazing friends here and on the other side of the world back home. They gave me the support and the distraction I needed, and I would have been miserable without them. My many Bay Area friends that have made this place a home for me: Chris Olley, Felix Crevier, Benoit Miquel, Lars Neustock, Anastasiya Vitko, Jean De Becdelievre, Emilie Leblanc, Maxime Bouton, Arnaud Dusser, Isabelle Durant, Behrad Afshar, Frances Silva-Roig, Annie Marsden, Jake Kohn, Timon Ruban, and Dan Zylberglejd. Special thanks go to my close friends overseas, Aidan Biggar, Marius Vollberg, Max Volmar, and Herny Clausen. I am so happy that we've been able to maintain our friendships over this distance and all these years, and it makes me even more confident that it will continue for many years. And finally, I want to thank my family, to whom this dissertation is dedicated: My parents Katharina, and Stephan, my siblings Johannes, Solveig, Franziska, and Jacob. Mit viel Liebe an Euch alle!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since their inception in 2008[Nak08], blockchains have received an enormous amount of hype and speculation. But beyond all of the noise, real transformational utility of blockchains is building. For example, blockchains are being used to avoid currency controls by oppressive governments[1], and a whole new decentralized finance space removes barriers to accessing financial tools such as loans or currency exchanges. In 2022, over 800 billion usd of trades happened on such decentralized exchanges. Further, stablecoin (digital copies of the dollar and euro that live on a blockchain) transfers exceeded 7.4 trillion dollars, more than Mastercard's and American Express' volume[2]. A blockchain is a public shared ledger that is accessible to everyone in the blockchain network. This gives blockchains a high level of transparency, including the ability for all users in the network to check the correctness of the state. Additionally, blockchains are controlled by a decentralized consensus instead of a centralized network operator. Together these features enable transparent, powerful networks without the need for a monopolistic centralized coordinator, that can become a single point of failure. Despite their hype, adoption, and promise, blockchains today still face several significant issues that hinder their utility:

High fees Blockchains such as Bitcoin and Ethereum have fees that can reach tens of dollars[3], especially in times of congestion. A key driver for digitalization is the reduction of costs. However, with at least the largest blockchains today, we see that despite their digital nature, they have significant costs per transaction.

Bad privacy Bitcoin was originally hailed for its privacy features. Satoshi proposed a privacy model where the public keys or addresses are decoupled from real-world identities. However, since then, several papers [Mei+13; And+13] as well as multiple companies[4], have shown the significant limitations of this model. In currencies like Bitcoin or Ethereum every transaction amount, as well as the senders' and receivers' addresses, are visible to everyone. This allows tracing the flow of funds

through the system. As soon as some endpoints are de-anonymized, it is often possible to backtrace the identities of many parties transacting in the system. This concern is a particular hindrance to companies that want to use blockchains for important economic activities. For instance, using Ethereum to pay all their employees would yield in the salaries of these employees being publicly leaked. Running a supply chain using such a blockchain could leak vital information to competitors.

Ecological Impact Perhaps the most prominent criticism of blockchains, and in particular Bitcoin is its environmental impact. Bitcoin relies on so-called Proof-of-work consensus. This consensus requires burning energy in order to gain voting power in the protocol. This has led to significant and wasteful energy consumption. Bitcoin is estimated to currently have energy consumption on the level of nation-states such as the Netherlands or the Philipines. In times of climate change and energy crisis, such wasteful consumption is unacceptable. It is, therefore, paramount to develop alternatives that preserve the attractive properties of proof-of-work consensuses, such as its decentralized, permissionless nature, while not relying on wasteful energy consumption.

Data broadcasting as the core issue There is a common theme to the issues of blockchains today. It is related to the data that is posted on the blockchain. In blockchains, generally, one user creates some data, e.g. a transaction or a transaction block, and then broadcasts it to every user in the network. Every user then verifies the integrity of the data, e.g. checking the correctness of the transaction. This stands in contrast to traditional centralized client-server architectures. In a centralized architecture, only one server (or one cluster) needs to process data. This redundant processing enables the high level of transparency in blockchains, as each user can independently verify the integrity of the blockchains state. However, it also is a key cause for the issues blockchains face today. If many transactions are issued and processed, then this can lead to *congestion*. In order to ensure that every node has sufficient capacity to receive and process all transactions, we need to limit the total throughput a blockchain network can process. This is especially true when we want a diverse, heterogeneous set of nodes in a wide variety of locations. However, this congestion naturally increases competition between transactions and drives up fees. This is not just theoretical but the relationship between congestion and high fees has often been observed in practice.

An additional issue with the broadcasting of transactions is related to privacy. If transactions contain sensitive information, then an attacker listening to the network can trivially gather this information. In traditional blockchains such as Bitcoin and Ethereum, information such as the sender, the receiver, the amount transferred, and smart contract programs and states are publicly visible. On the contrary, the privacy model of traditional centralized financial systems relies on trusted parties. No information is revealed publicly, and as long as users trust their bank, they have sufficient financial privacy for their bank transactions.

Finally, the broadcasting of data requires consensus or agreement between all nodes on what data has been sent. Bitcoin pioneered using a so-called proof-of-work based consensus[Nak08].

In that system, the voting power within the consensus is roughly proportional to the amount of computation a node has performed. This leads to a competition between nodes who can perform the most computation, which indirectly leads to a competition of who can burn the most energy. The computation itself, finding partial per-images of hashes, has no inherent utility. This is incredibly wasteful and leads to massive energy consumption on the level of nation-states.

1.1 The proof paradigm

In this dissertation, we apply a powerful *proof paradigm* to tackle these issues. The core idea of the proof paradigm is to use certificates or proofs that attest that data has certain properties. In the blockchain setting, instead of broadcasting the data directly, a user broadcasts a proof of data validity, and nodes process the proofs instead of the data directly. The proofs are cryptographic in nature and give strong guarantees[5] about the integrity of the underlying data. Importantly, while the data, itself, may be large and reveal private information, we aim to design proofs that are succinct, efficient to verify, and do not reveal information beyond the integrity of the underlying data. The issues of blockchains are caused by an asymmetry, where one user's action requires verification by all other users. Proof systems can have an inverse asymmetry where one *prover* produces a proof that is easy to check for everyone else.

In this dissertation, we develop four different types of proof systems specifically tailored to address the issues that blockchains face today: In Chapter 2, we introduce Bulletproofs, a *zero-knowledge* proof system, designed for private transactions that hide the contents of the transaction, while still ensuring their validity. Bulletproofs has been deployed in private cryptocurrencies such as Monero or ZCash[Hop+22], which provide strong privacy guarantees. In Chapter 3, we design HyperPlonk, a highly efficient proof system that is tailored for large statements, such as proving that an entire Ethereum block was correctly processed. This is a vital component of so-called *rollups*. The idea of a rollup is to bundle transactions and post only a small summary of the transactions on the blockchain. It provides a proof that all the transactions were valid and the state was correctly updated. The proof is much shorter and more efficient to verify than the original transactions. HyperPlonk has precisely these properties and in addition, is designed to be very efficient to compute, even for large and complex statements. It is currently being implemented by the Ethereum Foundation[KMT22] as part of their efforts to facilitate rollups.

In Chapter 4, we define and introduce Verifiable Delay Functions (VDFs). VDFs are a proof-based cryptographic primitive that enables many mutually distrusting parties to generate shared, unbiased and unpredictable randomness. Such randomness is a vital component of modern proof-of-space and proof-of-stake consensus systems. These consensus systems serve as ecologically friendly replacements for the wasteful proof-of-work. VDFs are part of the roadmap for Ethereum's proof of stake consensus, ensuring an unbiased, efficient, and fair leader election[6]. In Chapter 5, we introduce

ProtoStar. ProtoStar enables incrementally verifiable computation. It's a proof system for showing that t steps of computation were executed correctly, such that verifying the proof is independent of t. This directly yields a VDF construction, if the underlying t-step computation is sequential, e.g. a chain of hashes. Furthermore, ProtoStar can be used to prove that all transaction blocks, in a blockchain, are valid, such that checking the proof is efficient, and that it can be efficiently updated, once a block is added to the chain.

We now give a more detailed introduction to each of the proof systems developed in this dissertation:

1.2 Bulletproofs: Privacy through zero-knowledge.

Bulletproofs is a zero-knowledge proof system. A zero-knowledge proof enables a *prover* to convince a *verifier* that some statement is valid without revealing any information other than the fact that the statement is true. Bulletproofs can be applied to two different types of statements: firstly, it applies to general computations, expressed as arithmetic circuits, i.e. a computation consisting of additions and multiplications. Secondly, we design a specifically efficient variant of Bulletproofs for so-called *range proofs*. A range proof shows that some value inside a commitment is within some small range of numbers. Bulletproofs is characterized by two important features: The proofs do not rely on a trusted setup and are built on the well-studied discrete logarithm assumption, as well as the Fiat-Shamir transform, to achieve non-interactive proofs. This stands in contrast to the highly efficient pairing-based SNARK systems[Gro16; KZG10]. Unlike Bulletproofs, they rely on trusted setups, where a private-coin setup algorithm produces proving and verification keys. The soundness of the proof system is reliant on these private coins being deleted post-setup. While it is possible to distribute the trust of these setups using multi-party computation[BGG19; Gro+18], they have led to significant, exploitable vulnerabilities in cryptocurrencies such as ZCash[Ben+14a]. Other alternatives to pairing-based SNARKs are hash-based proof systems such as STARKs[Ben+19a]. However, these proof systems have proofs of about 100s of kilobytes for even medium-sized statements (e.g. 10k constraints).

The other key feature of Bulletproofs is that they are highly succinct. The statements are logarithmic in the size of the statement with small concrete constants. A Bulletproof for a circuit of a million gates is less than 1.7KB, and 32 range proofs of 64-bit length in less than 1kb. Bulletproofs[1] remain, to this day, the shortest proofs without trusted setup.

Bulletproofs are built on an inner-product argument (IPA) which was pioneered by Bootle et al.[Boo+16]. This is a special proof system for showing that given two committed vectors $\mathbf{a}$ and $\mathbf{b}$ and a scalar c, $c = \langle \mathbf{a}, \mathbf{b} \rangle$. We improve the inner-product argument and reduce the proof size by a factor of 3 to just $2\log_2(n)$ group elements for vectors of size n. At the core of this IPA there

[1] Up to additive constants which were improved in Bulletproofs+[Chu+20] and Bulletproofs++[Eag22]

is a randomized reduction from a statement about vectors of length n to vectors of length $n/2$. Repeating this reduction $\log_2(n)$ time yields the final protocol. We then use this improved IPA to build an argument for general arithmetic circuits. The constraints imposed by the arithmetic circuits are decomposed into multiplication constraints (corresponding to multiplication gates) and linear combination constraints (corresponding to additions and multiplications by constants). These constraints are then compiled down to claims about inner products, which can be proven using the IPA. We also design a specialized range-proof protocol. The range proof shows that a value inside a homomorphic Pedersen commitment[Ped92] is within some small positive range. While, in theory, this statement could be proven using our arithmetic circuit protocol and implementing the commitment function as such a circuit, our protocol is far more direct and efficient. It takes advantage of the homomorphic nature of the commitment and of Bulletproofs proof system. The value inside the commitment can be directly fed into the proof statement. Concretely, to show that a value v is in a range $B = [0, 2^k)$, we show that v can be decomposed into k bits $\mathbf{b} = b_1, \ldots, b_k$ and that $\langle \mathbf{b}, (1, 2, 4, \ldots, 2^{k-1}) \rangle = v$.

1.2.1 Application to confidential transactions

Bulletproofs is designed to improve the privacy of blockchain payments. In traditional blockchain transactions, such as Bitcoin transactions, the transferred amount is publicly visible. Any Bitcoin transaction destroys one or more specific records (called a UTXOs) and creates one or more new records such that the total value of the created UTXOs equals the total value of the destroyed UTXOs. Transaction validation checks the authenticity of a transaction by verifying a signature and possibly some additional transaction rules. It also checks the balance of the transaction by comparing the value of the created and destroyed UTXOs. However, this directly leaks the transacted amount. *Confidential transactions* [Max16] resolve this privacy issue by leveraging cryptographic commitments (See Definition 3.4). A cryptographic commitment is the digital analog of an envelope. The commitment of some value v is *hiding*, in that it reveals no information about v. It is also *binding* in that it can only be opened to v and no other value v'. Confidential transactions commit to the transferred amounts. However, this seems to hinder the ability that the transaction remains balanced. Furthermore, we need to guarantee that all created records have positive value, as otherwise, an adversary could use a transaction to print money. All of these statements can be proven using a range proof. This proof is then attached to the transaction. The specific system design leads to a few important design constraints for the range proofs:

Small proofs Because in blockchain systems, we need to broadcast transactions to the entire network, it is vital that the proof itself is small. Additionally, we need that the proof reveals no information about the newly created records other than that the transaction is balanced. Finally, in confidential transactions, there is a particular need for strong security. While, strong security,

is important in all cryptography applications, the potential damage of an attack on confidential transactions is particularly high.

Strong security without trusted setup If an attacker could break the so-called *soundness* property of a confidential transaction's range proof, then they can create a transaction that creates records with more value than the input records. In a blockchain-based cryptocurrency, this would be equivalent to printing money out of thin air. Even more detrimental, because of the private nature of confidential transactions, such an attack would be undetectable. This would enable an attacker to cause maximal damage to the blockchain system. Because of this, confidential transactions need range proofs that are very secure, i.e. rely on minimal cryptographic assumptions and no or minimal trusted setups. Bulletproofs satisfies both of these properties and additionally has more features tailored to the confidential-transactions application.

Aggregate range proofs Bulletproofs enables proving that multiple values are within the same range using a single proof. The proof scales logarithmically. Thus proving that k values are within a range only has an additive additional cost of $\lceil \log_2(k) \rceil 64$ bytes. This is particularly useful if a transaction has many outputs.

Batch verification A node within a blockchain network needs to verify entire blocks of transactions. If these are confidential transaction then each will have a range proof associated with them. We propose a batch-verification algorithm for Bulletproofs that enables verifying k Bulletproofs faster then k times the cost of verifying a single Bulletproofs. The marginal cost of verifying a 64-bit range proof is only about $5x$, the cost of verifying an ECDSA signature. This is a relevant comparison as non-confidential transaction validation requires digital signature verification (which cannot be batched).

Collaborative proof generation Due to the aggregated range proofs feature, Bulletproofs are particularly efficient for transactions with many outputs. We design a protocol that enables multiple users to generate a single proof for all of their transactions. This protocol does not require the users to share any secret values and has minimal overhead over a single-user proof generation.

Generic Circuit proofs for more complex statements Bulletproofs do not just provide the ability proof that a committed value is within a range. It can also be used to prove that some circuit was evaluated correctly. We combine the technique and enable proving a circuit where some of the inputs are committed. This makes Bulletproofs more composable with more complex protocols. For example, one could prove that a confidential transaction satisfies the balance property and additionally that some other transaction rules are satisfied.

Impact

Bulletproofs has been deployed in several blockchain systems. Within a year of publishing the first pre-print Monero [Mon], the largest (both by market capitalization and by number of transactions) blockchain with private transactions, adopted Bulletproofs. Replacing the previously used range proofs, which were significantly larger and slower to verify, led to a significant reduction in transaction size. This, in turn, led to a 97% decrease in transaction fees[7]. Bulletproofs was further adopted by other blockchains and systems, such as Grin, Beam, Signal's MobileCoin and Chase Bank's Quorum[8]. In addition ZCash recently transitioned to a poof system that is based on Bulletproofs' improved inner-product-argument[9]. This massive deployment has been made possible by a large variety of high-quality industry implementations of Bulletproofs. A recent paper[Dao+23] lists 15 such implementations. Bulletproofs has also received attention from academia. The protocol has been extended to bilinear groups[Bün+21b; Lee21], groups of unknown-order[BFS20; BCS21], and lattices[Boo+20; ACK21; BF22]. Even though Bulletproofs itself does not have succinct verification, later work showed how to build IVC from using its inner product argument[BGH19; Bün+20].

Several papers have performed security analyses of several aspects of Bulletproofs. Showing that it is non-malleable[Gan+21; DG23], secure in the random-oracle[Wik21; AFK22], and given a tighter security analysis[JT20; GT21]. Other works have given alternative security proofs of Bulletproofs through elegant reductions to Σ-protocols[AC20] and sumcheck arguments[BCS21].

1.3 HyperPlonk: A proof system for the zkEVM

Zero-knowledge proofs, such as Bulletproofs, enable a prover to convince a verifier that some statement is true without revealing why it's true. Some zero-knowledge proofs (but not Bulletproofs) have an equally powerful but orthogonal property called *verifiable computation*. These proofs enable showing that a statement is true, and checking the proof is (exponentially) faster than checking the statement itself. This enables outsourcing computation. A powerful server can run some computation and attach a proof that this computation was performed correctly. The verifier only checks the proof and is convinced that the computation has been done correctly. Proof systems that satisfy this verifiable-computation property are also referred to as SNARKs[2]. Modern SNARKs are usually built from two separate components. A polynomial interactive-oracle proof (PIOP)[BFS20; Chi+20] and a polynomial-commitment[KZG10]. The PIOP is an information-theoretic interactive proof system where the prover's messages are oracles to (possibly multi-variate) polynomials. The verifier is public-coin, i.e. all its messages are random challenges. The verifier the checks the proof by evaluating the polynomial oracles. The polynomials itself may be large in size so to make this efficient we replace the oracles with so-called polynomial commitments. This is a succinct commitment

[2]Technically, the S in SNARKs refers to succinct or short proofs. This is a necessary but not sufficient requirement for verifiable computing. Bulletproofs has succinct proofs, but the verifier runs in time linear in the statement.

to a polynomial. There exists a specialized SNARK for these commitments that enables the prover to convince a verifier that the polynomial was correctly evaluated. The Bulletproofs IPA can be used to build a polynomial commitment. Given any such PIOP for some statement and any secure polynomial commitment, as well as a secure cryptographic hash function, we can build a SNARK for that statement[BFS20; Chi+20]. The security and efficiency properties of that statement are the sum of the properties of the PIOP and the polynomial commitment.

This recipe is attractive because it enables separately designing each component. Additionally, it has been used to build SNARKs that are highly efficient and only require a minimal, universal and updatable trusted setup[Mal+19; Chi+20; GWC19; Set20]. Most previous PIOPs were built using univariate polynomials and required the prover to multiply large polynomials. This can be done most efficiently using a Fast-Fourrier-Transform in time $O(d \log(d))$ for degree d polynomials. The degree of these polynomials is usually proportional to the size of the circuit n. Unfortunately, FFTs introduce several limitations when applied to large polynomials (E.g. $d > 1M$):

- They become a bottleneck of the computation. For smaller sizes, the computation is usually bottlenecked by the polynomial commitment. However since this operation scales linearly in n and FFTs scale super-linearly in n, they eventually become a bottleneck.
- They necessitate the use of FFT-friendly fields. These are fields such that there exists an n-th root of the identity element 1. This requires using particular elliptic curves and limits the composability with other protocols.
- The FFTs have complex memory accesses. They require loading the entire polynomial into memory which becomes a bottleneck if the FFT becomes too large. Other components of the proof system are data-parallel, i.e. there is no minimum memory size required.
- Plonk, the most widely used PIOP, has achieved part of its popularity due to the use of high-degree gates. These are gates that do not just perform addition or multiplication but higher degree operations, such as $(x^3 + y) \cdot z$ for three inputs x, y, z. In Plonk and similar systems, the prover needs to compute and send so-called quotient polynomials. These polynomials are of degree $D \cdot n$ where D is the highest degree of any gate and n is the number of gates in the circuit. These quotient polynomials limit the utility of high-degree gates if D gets too large (in practice, $D \leq 5$).

With these limitations in mind we designed HyperPlonk. HyperPlonk is a multi-variate PIOP that mirrors the features and arithmetization of Plonk[GWC19] but does not require the use of FFTs. Instead, it is built on the famous sumcheck protocol[Lun+92]. Instead of interpolating the witness as a univariate polynomial over roots of unity, HyperPlonk uses multi-linear polynomials and encodes the witness over the boolean hypercube. We carefully replace each sub-component of Plonk to get rid of the requirement of FFTs. This yields a PIOP where the prover is fully linear in the witness instead of quasi-linear. Accompanying this HyperPlonk has a few additional attractive features:

High degree gates Using multi-linear polynomials we now are able to use much higher degree custom gates. The reason is that within the sum-check protocol, the prover needs to compute and send polynomials of size D instead of size $n \cdot D$.

Efficient Lookup We design an efficient lookup protocol that enables proving that some value is within a table of values. This can be useful for range proofs if the table contains all values within a range. Unlike previous online lookups[3], the prover is fully linear in the size of the table.

Soundness in small fields HyperPlonk can be instantiated such that it is sound even in small, polynomial-size fields. Concretely, the soundness error is only $O(\frac{\text{polylog}n}{|\mathbb{F}|})$ instead of $O(\frac{n}{|\mathbb{F}|})$ for univariate PIOPs.

Batch evaluation of polynomial commitments Using the same sumcheck protocol, we build an efficient batch evaluation protocol, that enables reducing k multi-linear polynomial evaluations to a single polynomial evaluation. The protocol applies to any homomorphic polynomial commitment.

Orion+ We also propose Orion+, an improvement on Orion[XZS22b]. Orion+ has significantly smaller proofs (7kb vs 3MB) while retaining the same asymptotic, linear prover complexity.

1.3.1 Application to Rollups and ZK-EVMs

We designed HyperPlonk specifically for large complex statements. One set of applications where this is useful are so-called rollups and the ZK-EVM. In these rollups, a sequencer collects a list of transactions and then proves that these transactions lead to a specific state transition. Using verifiable computing, it is possible to create proofs that are significantly more efficient to verify than replaying all the rolled-up transactions. Current efforts aim to roll-up entire blocks with thousands of transactions. On Ethereum, these transactions can be more than just simple transfers of funds but complex programs, called smart contracts. Efforts to build rollups for the Ethereum VM (EVM) are referred to as ZK-EVMs[4]. Current implementation efforts of the ZK-EVM[10], heavily rely on high-degree gates and lookups. There are also multiple companies[11] trying to build hardware solutions to facilitate efficient rollups. HyperPlonk is designed with these features and to be particularly hardware friendly. For these reasons, the Ethereum foundation has started to build its own HyperPlonk implementation (based on the implementation reported in this paper) and include it in their ZK-EVM efforts[12].

[3] Recently, there has also been an effort in offline lookups where the table can be preprocessed.[Zam+21; EFG22]. These have different applications and can potentially be combined.

[4] This is mainly a misnomer as the zero-knowledge or ZK property is not important here. We only require that the proofs or SNARKs have the verifiable computing property.

1.4 Verifiable Delay Functions

Early blockchains, such as Bitcoin[Nak08] and Ethereum, were built using proof-of-work consensus. This means that participants' voting power is proportional to the amount of computation or work they perform. Concretely, in Bitcoin, nodes continuously attempt to solve a computational puzzle, and the person solving it first gets elected as leader, i.e proposes the next block of transactions. The process is random, such that a node controlling x percent of the network's computational power, measured in hashes per second, has an x percent chance of solving the puzzle first. Randomness is vital for the consensus, ensuring that an adversary will not be able to produce all transaction blocks, even if it controls a plurality of the computational power. Nodes get rewarded for producing such a block with newly minted Bitcoin and transaction fees.

Unfortunately, this process is incredibly wasteful. The block production and thus the rewards are proportional to the entire systems computational power. Thus, nodes are incentivized to try to solve the puzzle as fast as possible, acquire more computational power, and burn more energy. Unfortunately, the puzzle itself has no inherent use[13]. Bitcoin is estimated to use about 100 TWh per year. That is on the level of nation states such as the Netherlands or the Philippines[14]. From another perspective, the energy consumption of a single Bitcoin transaction is equivalent to the power 20 US Households consume in a day. This underscores the imperative for exploring alternative solutions. Fortunately, such solutions exist with proof-of-space and proof-of-stake consensus systems. Informally, in proof-of-space, the voting power is proportional to the storage a user allocates, and in proof-of-stake, it is proportional to the amount tokens a user owns and allocates. Both methods do not require heavy computation and are thus much more environmentally friendly than proof of work. They are now being widely deployed in modern blockchains. Ethereum recently upgraded to a proof-of-stake consensus[15], which is estimated to have reduced Ethereum's energy consumption by 99.95%. Proof-of-stake and proof-of-space are not without technical difficulties and limitations. One important one is related to leader election. Proof-of-work directly provides a mechanism for randomly electing a leader proportional to the work. For neither of the other protocols, this is the case. Focussing on proof-of-stake, we have a list of public keys and associated weights (the staked values) to these keys. A consensus protocol, in every round, needs to select a key proportional to the weights. The owner of that key then gets to publish the next block. It is important that this selection is unbiased, even in the face of an adversary that controls some f fraction of stake. Additionally, we need the selection to be unpredictable up to some point t as otherwise, an adversary could register keys that have higher chances of winning the leader selection.

These properties can be achieved though a so-called *random beacon*[Rab83]. A random beacon is a protocol for periodically emitting uniform, unbiased, and unpredictable values. It can be thought of as a multi-party protocol between n parties, such that if a beacon value appears at time t, then for any time $t' < t - k$, no adversary can predict the beacon value with more than negligible probability, and no adversary can bias the output of the beacon by more than a negligible amount. There are

several ways to produce a random beacon: One could rely on a trusted party, but this is undesirable in many applications. Alternatively, one could use a commit-and-reveal protocol where parties commit to some random value and then, after some time, reveal it. This is the kind of protocol that is currently used in Ethereum[Ran]. Unfortunately, it is possible to bias it as a node can simply choose not to reveal the value. Ethereum deals with this by economically incentivizing nodes to open, but these incentives do not provide strong guarantees. Thirdly, one can rely on threshold cryptography[Syt+17] to generate a shared secret. The protocols require that $f < 1/3$rd. They also have a minimum communication complexity of n^2 broadcasts, i.e. each party needs to send n values to every other party.

1.4.1 An unbiasable, ecological and communication-efficient beacon from VDFs

With these limitations in mind, we introduce Verifiable Delay Functions in Chapter 4. Verifiable Delay Functions are a novel cryptographic primitive that, among many other applications, enables building a secure random beacon, which is secure against $n-1$ corruptions and only has an $O(n)$ communication complexity. VDFs are characterized by the three properties that make up its name. A VDF is a *function* that maps each input from some domain to a unique output in its image. Evaluating the VDF incurs a *delay*. That is, it T sequential steps to evaluate, and no parallel polynomial-time adversary can evaluate it faster than ϵT. Finally, it is *verifiable*, in that it is possible to verify that the VDF was evaluated correctly in time polylog$T$. This verification may take in an optional proof. The beacon construction from VDFs is simple and elegant. We assume a common ledger or some other verifiable broadcast channel. Each party posts some random value r to the ledger in some time window $[t_0, t_0 + t]$. t has to be large enough such that all honest parties are able to post a message within that time-frame. Then all posted values, are hashed using a secure hash function. We then apply a VDF to the output with parameters such that $\epsilon T > t$. After time T some party (could be any honest party), posts the verifiable VDF output. This output is hashed and becomes the random beacon. To get an intuition for why this is a secure beacon, consider and adversary that controls $n-1$ parties and one honest party. Let's also assume that the hash function is modeled as a random oracle. The honest party sends its value at the beginning of the time window. This gives the adversary time t to decide what values to post. However, it cannot evaluate the VDF in time t, as $\epsilon T > t$. Thus, unless it has precomputed the input to the VDF, it cannot learn the output before $t_0 + \epsilon T$, and thus the beacon value. Precomputing the input is improbable for an exponential input domain, as the single honest party chooses its value randomly. Finally, since the VDF is a deterministic function, an adversary that can bias the output significantly must also be able to guess the output with non-negligible probability. This contradicts the delay property of the VDF.

In this dissertation, we formally define VDFs, including the security definitions that enable

this beacon construction. We also give several constructions of VDFs. From incremental verifiable computation as well as from plain verifiable computation, and give several optimizations and concrete underlying primitives that improve the practicality of these constructions.

1.4.2 Impact

VDFs had a significant theoretical and practical impact. Multiple blockchains such as Chia[16] , Filecoin[17] , and Solana[18] use variants of VDFs in their ecological consensus systems. Furthermore, Ethereum is planning to use a VDF based on incrementally verifiable computation and a delay function that is nearly identical to the construction in our paper[KMT22]. There has also been significant academic interest in VDFs. Pietrzak[Pie19a] and Wesolowski [Wes19] gave elegant direct VDF constructions in groups of unknown order. De Feo, Masson, Petit, and Sanso[De +19] gave a VDF construction that does not require a proof using isogenies. VDFs have also been used to show complexity theoretic connections between unique games (PPAD hardness) and repeated squaring[Cho+19; Eph+20b; LV20]. Further, they have been used to achieve dynamic availability for proof-of-stake and proof-of-space protocols[DKT21]. Dynamic availability is the ability for honest nodes to sync with the current state after being offline without additional trust assumptions.

1.5 ProtoStar: Proofs for VDFs and Succinct Blockchains

Verifiable Delay Functions can be built by taking a sequential computation, e.g. chaining of hashes, and using a SNARK to prove that the computation was executed correctly. This construction has several disadvantages. Firstly, the SNARK can only be constructed after the computation is done. Also computing the SNARK, tends to be many orders of magnitude more expensive than performing the underlying computation. This means that an adversary can learn the output of the VDF, long before an honest party is able to construct a convincing proof of the VDFs correctness. Secondly, this construction is incompatible with continuous VDFs[Eph+20a]. In continuous VDFs, the delay function can always be evaluated for one more step, and at every point, the output can be efficiently verified. Using a SNARK to achieve the verifiability property of a continuous VDF would imply that the entire SNARK needs to be recomputed at every step. Fortunately, we describe another VDF construction in Chapter 4 that does not suffer from these drawbacks. It is built on *Incrementally Verifiable Computation*(IVC)[Val08]. IVC enables proving that a t-step computation was performed correctly, with two important properties: Checking the proof takes at most polylog(t) time and given the state of the computation and a proof for the first t steps, any prover can continue the computation and efficiently update the proof. Beyond it's application to VDFs, IVC has many more applications to Blockchains and beyond:

Succinct Blockchains IVC can be used to build blockchain protocols such that new nodes, or nodes that were offline, only need to download and check a constant amount of data. The idea is

that along with every transaction block, block producers construct an IVC proof that the entire blockchain, including the consensus and all transactions, up to that point was valid. Nodes then download the state or a succinct commitment to the state along with the IVC proof and are convinced that the blockchain contains only valid transactions and that the consensus is correct. This is particularly useful in the sleepy model of consensus[PS17] that allows honest nodes to go offline for long periods of time. This vision of succinct blockchains has been realized in the Mina Blockchain system[Bon+20b], that utilizes a precursor of ProtoStar[BGH19; Bün+20].

Rollups IVC can be used to compress the verification of multiple blocks in a Blockchain. However, it can also be used to compress the verification of many transactions within a block. This can be used to build rollups as described in Section 1.3 or simply to speed up the verifications of blocks. One interesting property of IVC and its generalization, Proof Carrying Data[Bit+13a], is that they enable multiple distributed provers to construct a single proof together. This would allow multiple rollup servers to construct rollup proofs for a distinct set of transactions and then generate a single proof for the entire rollup block. Each prover's proving work is only proportional to the number of transactions in their set.

Proofs of VMs More generally, IVC can be used to prove the correct execution of a t-step virtual machine. This machine could be the Ethereum Virtual Machine (EVM) or a more classic architecture such as web assembly. Such proofs, again, have the ability to outsource verification. Such VMs consist of a control unit, that indicates what instruction to execute, an logical unit that executes the specific instructions, and a memory unit that stores the VMs state. ProtoStar is designed to efficiently support these units with minimal overhead. In systems such as ProtoStar, the verification is naively as expensive as a single step of the VM. However, ProtoStar could be combined with SNARKs, such as HyperPlonk, in order to compress even that final verification step.

1.5.1 Efficient and Flexible IVC from accumulation

IVC is clearly a very powerful primitive with a myriad of applications. However, until recently, constructing IVC was very expensive, as it relied on so-called recursive SNARKs. The idea is to take a SNARK and prove that a step of computation was executed correctly. Then construct a second SNARK that proves the next step of the computation *and* proofs that the previous SNARK was correct. This continues for perpetuity. Implementing this recursive step introduced a significant overhead that limited the practicality of these systems[Ben+14b]. Recently, starting with the breakthrough of Halo[BGH19], which is built on the batch-verification of Bulletproofs (see Section 3.3.8), a series of papers[Bün+20; Bün+21a; KST22; KS22; KS23] have started to remove these practical limitations. The key insight is that it is unnecessary to prove that the entire SNARK was correct. Instead, we can batch or accumulate the verification checks and only prove that this accumulation

was performed correctly. This has led to IVC constructions with significantly improved practical efficiency and additional greatly reduced the cryptographic requirements for constructing IVC. It is now possible to construct IVC without a trusted setup, simply from the discrete logarithm assumption, without the need for FFTs[Bün+21a]. In Chapter 5 we introduce ProtoStar, a new accumulation-based IVC scheme that advances the state of the art in several significant dimensions.

General Recipe ProtoStar is built on a general recipe for constructing such accumulation schemes that unifies the previous, seemingly ad-hoc constructions. We show that it is possible to build secure and efficient accumulation schemes from very simple building blocks, namely any special-sound protocol with an algebraic verifier.

High degree circuits Unlike prior work, which was only able to handle addition and multiplication gates, ProtoStar is able to handle computations that are of high degree. Both the recursive circuit and the provers work have only a minimal dependence on the degree of the gate. This makes ProtoStar significantly more expressive and suitable for complex statements, such as virtual machine computations.

Lookup Gates ProtoStar is able to handle lookup gates, i.e. checking that some value is in some pre-computed table, such that the provers work is linear in the number of lookups and independent of the table size. This is the most efficient lookup argument of any proof system (IVC or SNARK) to date.

Non-uniform IVC ProtoStar natively and efficiently supports non-uniform circuits. That is, each computation step can execute one of a fixed number of circuits. This efficiently implements the control unit of a virtual machine. The prover's cost is only linear in the executed circuit, not all circuits.

Introduction Notes

[1] https://www.nytimes.com/2019/02/23/opinion/sunday/venezuela-bitcoin-inflation-cryptocurrencies.html

[2] https://thedefiant.io/stablecoin-volume-hits-record-high-of-7-4t-in-2022

[3] https://ycharts.com/indicators/ethereum_average_transaction_fee

[4] https://www.chainalysis.com/,https://www.trmlabs.com/

[5] The guarantee is similar to the level of guarantee delivered by TLS and other cryptography securing the internet today.

[6] https://www.vdfalliance.org/

[7] https://bitcoinmagazine.com/culture/monero-transaction-fees-reduced-97-after-bulletproofs-upgrade

[8] https://www.coindesk.com/business/2019/05/28/jpmorgan-adds-privacy-features-to-ethereum-based-quorum-blockchain/

[9] https://electriccoin.co/blog/explaining-halo-2/

[10] https://vitalik.ca/general/2022/08/04/zkevm.html

[11] https://scroll.io/,https://polygon.technology/polygon-zkevm

[12] https://github.com/han0110/plonkish

[13]The computational puzzle finds partial pre-images of a cryptographic hash function.
[14]https://digiconomist.net/bitcoin-energy-consumption
[15]https://ethereum.org/en/roadmap/merge/
[16]https://docs.chia.net/proof-of-time/
[17]https://spec.filecoin.io
[18]https://solana.com/news/proof-of-history

1.6 Preliminaries and Notation

Bibliographical Notes

This dissertation is based on the following jointly authored publications:

Chapter 2 is based on "Bulletproofs: Short Proofs for Confidential Transactions and More," jointly authored with Dan Boneh, Jonathan Bootle, Andrew Poelstra, Pieter Wuille, and Greg Maxwell and published at the IEEE Symposium on Security and Privacy ("Oakland") in 2018[Bün+18].

Chapter 3 is based on "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates," jointly authored with Binyi Chen, Dan Boneh and Zhenfei Zhang, published at the Annual International Conference on the Theory and Applications of Cryptology and Information Security ("EUROCRYPT") in 2023[Che+23].

Chapter 4 is based on "Verifiable Delay Functions," jointly authored with Dan Boneh, Joseph Bonneau, and Ben Fisch, published at the International Cryptology Conference ("CRYPTO") in 2018 [Bon+18].

Chapter 5 is based on "ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols," jointly authored with Binyi Chen, available as a manuscript on EPRINT as document 2023/620 [BC23].

Chapter 2

Bulletproofs: Privacy through Zero-Knowledge

2.1 Introduction

Blockchain-based cryptocurrencies enable peer-to-peer electronic transfer of value by maintaining a global distributed but synchronized ledger, the *blockchain*. Any independent observer can verify both the current state of the blockchain as well as the validity of all transactions on the ledger. In Bitcoin, this innovation requires that all details of a transaction are public: the sender, the receiver, and the amount transferred. In general, we separate privacy for payments into two properties: (1) anonymity, hiding the identities of sender and receiver in a transaction and (2) confidentiality, hiding the amount transferred. While Bitcoin provides some weak anonymity through the unlinkability of Bitcoin addresses to real world identities, it lacks any confidentiality. This is a serious limitation for Bitcoin and could be prohibitive for many use cases. Would employees want to receive their salaries in bitcoin if it meant that their salaries were published on the public blockchain?

To address the confidentiality of transaction amounts, Maxwell [Max16] introduced *confidential transactions* (CT), in which every transaction amount involved is hidden from public view using a commitment to the amount. This approach seems to prevent public validation of the blockchain; an observer can no longer check that the sum of transaction inputs is greater than the sum of transaction outputs, and that all transaction values are positive. This can be addressed by including in every transaction a zero-knowledge proof of validity of the confidential transaction.

Current proposals for CT zero-knowledge proofs [Poe+19] have either been prohibitively large or required a trusted setup. Neither is desirable. While one could use succinct zero-knowledge proofs (SNARKs) [Ben+13; Gen+13] they require a trusted setup, which means that everyone needs to trust that the setup was performed correctly. One could avoid trusted setup by using

a STARK [Ben+19a], but the resulting range proofs while asymptotically efficient are practically larger than even the currently proposed solutions.

Short non-interactive zero-knowledge proofs without a trusted setup, as described in this paper, have many applications in the realm of cryptocurrencies. In any distributed system where proofs are transmitted over a network or stored for a long time, short proofs reduce overall cost.

2.1.1 Our Contributions

We present Bulletproofs, a new zero-knowledge argument of knowledge[1] system, to prove that a secret committed value lies in a given interval. Bulletproofs do not require a trusted setup. They rely only on the discrete logarithm assumption, and are made non-interactive using the Fiat-Shamir heuristic.

Bulletproofs builds on the techniques of Bootle et al. [Boo+16], which yield communication-efficient zero-knowledge proofs. We present a replacement for their inner-product argument that reduces overall communication by a factor of 3. We make Bulletproofs suitable for proving statements on committed values. Examples include a range proof, a verifiable shuffle, and other applications discussed below. We note that a range proof using the protocol of [Boo+16] would have required implementing the commitment opening algorithm as part of the verification circuit, which we are able to eliminate.

Distributed Bulletproofs generation. We show that Bulletproofs support a simple and efficient multi-party computation (MPC) protocol that allows multiple parties with secret committed values to jointly generate a single small range proof for all their values, without revealing their secret values to each other. One version of our MPC protocol is constant-round but with linear communication. Another variant requires only logarithmic communication, but uses a logarithmic number of rounds. When a confidential transaction has inputs from multiple parties (as in the case of CoinJoin), this MPC protocol can be used to aggregate all the proofs needed to construct the transaction into a single short proof.

Proofs for arithmetic circuits. While we focus on confidential transactions (CT), where our work translates to significant practical savings, we stress that the improvements are not limited to CT. We present Bulletproofs for general NP languages. The proof size is *logarithmic* in the number of multiplication gates in the arithmetic circuit for verifying a witness. The proofs are much shorter than [Boo+16] and allow inputs to be Pedersen commitments to elements of the witness. We also give a proof system for R1CS that allows proving statements about a committed vector of values.

Optimizations and evaluation. We provide a complete implementation of Bulletproofs that includes many further optimizations described in Section 2.7. For example, we show how to batch the verification of multiple Bulletproofs so that the cost of verifying every additional proof is significantly

[1] Proof systems with computational soundness like Bulletproofs are sometimes called argument systems. We will use the terms proof and argument interchangeably.

reduced. We also provide efficiency comparisons with the range proofs currently used for confidential transactions [Max16; ano] and with other proof systems. Our implementation includes a general tool for constructing Bulletproofs for any NP language. The tool reads in arithmetic circuits in the Pinocchio [Par+13] format which lets users use their toolchain. This toolchain includes a compiler from C to the circuit format. We expect this to be of great use to implementers who want to use Bulletproofs.

2.1.2 Applications

We first discuss several applications for Bulletproofs along with related work specific to these applications. Additional related work is discussed in Section 2.1.3.

Confidential Transactions and Mimblewimble

Bitcoin and other similar cryptocurrencies use a transaction-output-based system where each transaction fully spends the outputs of previously unspent transactions. These unspent transaction outputs are called UTXOs. Bitcoin allows a single UTXO to be spent to many distinct outputs, each associated with a different address. To spend a UTXO a user must provide a signature, or more precisely a *scriptSig*, that enables the transaction SCRIPT to evaluate to true [Bon+15]. Apart from the validity of the *scriptSig*, miners verify that the transaction spends previously unspent outputs, and that the sum of the inputs is greater than the sum of the outputs.

Maxwell [Max16] introduced the notion of a *confidential transaction*, where the input and output amounts in a transaction are hidden in Pedersen commitments [Ped92]. To enable public validation, the transaction contains a zero-knowledge proof that the sum of the committed inputs is greater than the sum of the committed outputs, and that all the outputs are positive, namely they lie in the interval $[0, 2^n]$, where 2^n is much smaller than the group size. All current implementations of confidential transactions [Max16; MP15; Poe+19; NM+16] use range proofs over committed values, where the proof size is linear in n. These range proofs are the main contributor to the size of a confidential transaction. In current implementations[Max16], a confidential transaction with only two outputs and 32 bits of precision is 5.4 KB bytes, of which 5 KB are allocated to the range proof. Even with recent optimizations the range proofs would still take up 3.8 KB.

We show in Section 2.7 that Bulletproofs greatly improve on this, even for a single range proof while simultaneously doubling the range proof precision at marginal additional cost (64 bytes). The logarithmic proof size additionally enables the prover to aggregate multiple range proofs, e.g. for transactions with multiple outputs, into a single short proof. With Bulletproofs, m range proofs are merely $O(\log(m))$ additional group elements over a single range proof. This is already useful for confidential transactions in their current form as most Bitcoin transactions have two or more outputs. It also presents an intriguing opportunity to aggregate multiple range proofs from different parties into one proof, as would be needed, for example, in a CoinJoin transaction [Max13]. In

Section 2.4.5, we present a simple and efficient MPC protocol that allows multiple users to generate a single transaction with a single aggregate range proof. The users do not have to reveal their secret transaction values to any of the other participants.

Confidential transaction implementations are available in side-chains[Poe+19], private blockchains[And17], and in the popular privacy-focused cryptocurrency Monero [NM+16]. All these implementations would have subsequently switched to using Bulletproofs. In Monero this led to a 97% reduction in transaction fees[2].

Mimblewimble. Recently an improvement was proposed to confidential transactions, called Mimblewimble [Jed16; ano], which provides further savings.

Jedusor [Jed16] realized that a Pedersen commitment to 0 can be viewed as an ECDSA public key, and that for a valid confidential transaction the difference between outputs, inputs, and transaction fees must be 0. A prover constructing a confidential transaction can therefore sign the transaction with the difference of the outputs and inputs as the public key. This small change removes the need for a scriptSig which greatly simplifies the structure of confidential transactions. Poelstra [ano] further refined and improved Mimblewimble and showed that these improvements enable a greatly simplified blockchain in which all spent transactions can be pruned and new nodes can efficiently validate the entire blockchain without downloading any old and spent transactions. Along with further optimizations, this results in a highly compressed blockchain. It consists only of a small subset of the block-headers as well as the remaining unspent transaction outputs and the accompanying range proofs plus an un-prunable 32 bytes per transaction. Mimblewimble also allows transactions to be aggregated before sending them to the blockchain.

A Mimblewimble blockchain grows with the size of the UTXO set. Using Bulletproofs, it would only grow with the number of transactions that have unspent outputs, which is much smaller than the size of the UTXO set. Overall, Bulletproofs can not only act as a drop-in replacement for the range proofs in confidential transactions, but it can also help make Mimblewimble a practical scheme with a blockchain that is significantly smaller than the current Bitcoin blockchain. Subsequent, to the initial publication of Bulletproofs both public Mimblewimble implementations, Grin and Beam, adopted Bulletproof's range proofs.

Provisions

Dagher et al. [Dag+15] introduced the Provisions protocol which allows Bitcoin exchanges to prove that they are solvent without revealing any additional information. The protocol crucially relies on range proofs to prevent an exchange from inserting fake accounts with negative balances. These range proofs, which take up over 13GB, are the main contributors to the proof sizes of almost 18GB for a large exchange with 2 million customers. The proof size is in fact linear in the number of customers. Since in this protocol, one party (the exchange) has to construct many range proofs at

[2] `https://bitcoinmagazine.com/culture/monero-transaction-fees-reduced-97-after-bulletproofs-upgrade`

once, the general Bulletproofs protocol from Section 2.4.3 is a natural replacement for the NIZK proof used in Provisions. With the proof size listed in Section 2.7, we obtain that the range proofs would take up less than 2 KB with our protocol. Additionally, the other parts of the proof could be similarly compressed using the protocol from Section 2.5. The proof would then be dominated by one commitment per customer, with size 62 MB. This is roughly 300 times smaller then the current implementation of Provisions.

Verifiable shuffles

Consider two lists of committed values $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. The goal is to prove that the second list is a permutation of the first. This problem is called a *verifiable shuffle*. It has many applications in voting [Fur+03; Nef01], mix-nets [Cha82], and solvency proofs [Dag+15]. Neff [Nef01] gave a practical implementation of a verifiable shuffle and later work improved on it [Gro10a; GI08]. Currently the most efficient shuffle [BG12] has size $O(\sqrt{n})$.

Bulletproofs can be used to create a verifiable shuffle of size $O(\log n)$. The two lists of commitments are given as inputs to the circuit protocol from Section 2.5. The circuit can implement a shuffle by sorting the two lists and then checking that they are equal. A sorting circuit can be implemented using $O(n \cdot \log(n))$ multiplications which means that the proof size will be only $O(\log(n))$. This is much smaller than previously proposed protocols. Given the concrete efficiency of Bulletproofs, a verifiable shuffle using Bulletproofs would be very efficient in practice. Constructing the proof and verifying it takes linear time in n.

NIZK Proofs for Smart Contracts

The Ethereum [Woo14] system uses highly expressive smart contracts to enable complex transactions. Smart contracts, like any other blockchain transaction, are public and provide no inherent privacy. To bring privacy to smart contracts, non-interactive zero-knowledge (NIZK) proofs have been proposed as a tool to enable complex smart contracts that do not leak the user inputs [Kos+16; MSH17; Cam+17]. However, these protocols are limited as the NIZK proof itself is not suitable for verification by a smart contract. The reason is that communication over the blockchain with a smart contract is expensive, and the smart contract's own computational power is highly limited. SNARKs, which have succinct proofs and efficient verifiers, seem like a natural choice, but current practical SNARKs [Ben+13] require a complex trusted setup. The resulting common reference strings (CRS) are long, specific to each application, and possess trapdoors. In Hawk [Kos+16], for instance, a different CRS is needed for each smart contract, and either a trusted party is needed to generate it, or an expensive multi-party computation is needed to distribute the trust among a few parties. On the other hand, for small applications like boardroom voting, one can use classical sigma protocols [MSH17], but the proof-sizes and expensive verification costs are prohibitive for more complicated applications. Recently, Campanelli et al. [Cam+17] showed how to securely perform zero-knowledge

contingent payments (ZKCPs) in Bitcoin, while attacking and fixing a previously proposed protocol [Max]. ZKCPs enable the trustless, atomic and efficient exchange of a cryptocurrency vs. some digital good. While ZKCPs support a wide area of applications they fundamentally work for only a single designated verifier and do not allow for public verification. For some smart contracts that have more than two users, public verification is often crucial. In an auction, for example, all bidders need to be convinced that all bids are well formed.

Bulletproofs improves on this by enabling small proofs that do not require a trusted setup. The Bulletproofs verifier is not cheap, but there are multiple ways to work around this. First, a smart contract may act optimistically and only verify a proof if some party challenges its validity. Incentives can be used to ensure that rational parties never create an incorrect proof nor challenge a correct proof. This can be further improved by using an interactive referee delegation model [CRR11], previously proposed for other blockchain applications [BGB17; TR]. In this model, the prover provides a proof along with a succinct commitment to the verifier's execution trace. A challenger that disagrees with the computation also commits to his computation trace and the two parties engage in an interactive binary search to find the first point of divergence in the computation. The smart contract can then execute this single computation step and punish the party which provided a faulty execution trace. The intriguing property of this protocol is that even when a proof is challenged, the smart contract only needs to verify a single computation step, i.e. a single gate of the verification circuit. In combination with small Bulletproofs, this can enable more complex but privacy preserving smart contracts. Like in other applications, these NIZK proofs would benefit from the MPC protocol that we present in Section 2.4.5 to generate Bulletproofs distributively. Consider an auction smart contract where bidders in the first round submit commitments to bids and in the second round open them. A NIZK can be used to prove properties about the bids, e.g. they are in some range, without revealing them. Using Bulletproofs' MPC multiple bidders can combine their Bulletproofs into a single proof. Furthermore, the proof will hide which bidder submitted which bid.

Short Non-Interactive Proofs for Arithmetic Circuits without a Trusted Setup

Non-interactive zero-knowledge protocols for general statements are not possible without using a common reference string, which should be known by both the prover and the verifier. Many efficient non-interactive zero-knowledge proofs and arguments for arithmetic circuit satisfiability have been developed [Mic94; KP98; GS08; Gen+13; Ben+13; Ben+18b], and highly efficient protocols are known. However, aside from their performance, these protocols differ in the complexity of their common reference strings. Some, such as those in[Ben+13], are highly structured, and sometimes feature a trapdoor, while some are simply chosen uniformly at random. Security proofs assume that the common reference string was honestly generated. In practice, the common reference string can be generated by a trusted third party, or using a secure multi-party computation protocol. The latter helps to alleviate concerns about embedded trapdoors, as with the trusted setup ceremony

used to generate the public parameters for [Ben+14a].

Zero-knowledge SNARKs have been the subject of extensive research [Gro10b; Bit+12a; Gen+13; Bit+13a; Par+13; Ben+13; Gro16]. They generate constant-sized proofs for any statement, and have extremely fast verification time. However, they have highly complex common reference strings which require lengthy and computationally intensive protocols [BGG19] to generate distributively. They also rely on strong unfalsifiable assumptions such as the knowledge-of-exponent assumption.

A uniformly-random common reference string, on the other hand, can be derived from common random strings, like the digits of π or by assuming that hash functions behave like a random oracle. Examples of non-interactive protocols that do not require a trusted setup include [Mic94; Boo+16; Boo+17; Ben+17a; Ben+18b].

Ben-Sasson et al. present a proof system [Ben+17b] and implementation [Ben+17a] called Scalable Computational Integrity (SCI). While SCI has a simple setup, and relies only on collision-resistant hash functions, the system is not zero-knowledge and still experiences worse performance than [Ben+13; Boo+16]. The proof sizes are roughly 42 MB large in practice for a reasonable circuit. In subsequent work Ben-Sasson et al. presented STARKs [Ben+18b], which are zero-knowledge and more efficient than SCI. However even with these improvements the proof size is still over 200 KB (and grows logarithmically) at only 60-bit security for a circuit of size 2^{17}. A Bulletproof for such a circuit at twice the security would be only about 1 KB. Constructing STARKs is also costly in terms of memory requirements because of the large FFT that is required to make proving efficient.

Ames et al. [Ame+17] presented a proof system with linear verification time but only square root proof size building on the MPC in the head technique. Wahby [Wah+18] recently present a cryptographic zero-knowledge proof system which achieves square root verifier complexity and proof size based on the proofs for muggles [GKR08] techniques in combination with a sub-linear polynomial commitment scheme.

2.1.3 Additional Related Work

Much of the research related to electronic payments that predates Bitcoin [Nak08] focused on efficient anonymous and confidential payments [CHL05; Cha82] . With the advent of blockchain-based cryptocurrencies, the question of privacy and confidentiality in transactions has gained a new relevance. While the original Bitcoin paper [Nak08] claimed that Bitcoin would provide anonymity through pseudonymous addresses early work on Bitcoin showed that the anonymity is limited [Mei+13; And+13]. Given these limitations, various methods have been proposed to help improve the privacy of Bitcoin transactions. CoinJoin [Max13], proposed by Maxwell, allows users to hide information about the amounts of transactions by merging two or more transactions. This ensures that among the participants who join their transactions, it is impossible to tell which transaction inputs correspond to which transaction outputs. However, users do require some way of searching for other users, and furthermore, should be able to do so without relying on a trusted third party.

ESORICS:RufMorKat14 [RMK14] tried to fulfill this requirement by taking developing the ideas of CoinJoin and proposing a new Bitcoin mixing protocol which is completely decentralized. Monero [Mon] is a cryptocurrency which employs cryptographic techniques to achieve strong privacy guarantees. These include stealth addresses, ring-signatures [Sab13], and ring confidential transactions [NM+16]. ZeroCash [Ben+14a] offers optimal privacy guarantees but comes at the cost of expensive transaction generation and the requirement of a trusted setup.

Range proofs. Range proofs are proofs that a secret value, which has been encrypted or committed to, lies in a certain interval. Range proofs do not leak any information about the secret value, other than the fact that they lie in the interval. Lipmaa [Lip03] presents a range proof which uses integer commitments, and Lagrange's four-square theorem which states that every positive integer y can be expressed as a sum of four squares. Groth [Gro05] notes that the argument can be optimized by considering $4y + 1$, since integers of this form only require three squares. The arguments require only a constant number of commitments. However, each commitment is large, as the security of the argument relies on the Strong RSA assumption. Additionally, a trusted setup is required to generate the RSA modulus or a prohibitively large modulus needs to be used [San99]. Camenisch et al. [CCs08] use a different approach. The verifier provides signatures on a small set of digits. The prover commits to the digits of the secret value, and then proves in zero-knowledge that the value matches the digits, and that each commitment corresponds to one of the signatures. They show that their scheme can be instantiated securely using both RSA accumulators [Bd94] and the Boneh-Boyen signature scheme [BB04]. However, these range proofs require a trusted setup. Approaches based on the n-ary digits of the secret value are limited to proving that the secret value is in an interval of the form $\left[0, n^k - 1\right]$. One can produce range proofs for more general intervals by using homomorphic commitments to translate intervals, and by using a combination of two different range proofs to conduct range proofs for intervals of different widths. However, [CLs10] presented an alternative digital decomposition which enables an interval of general width to be handled using a single range proof.

2.2 Preliminaries

Before we present Bulletproofs, we first review some of the underlying tools. In what follows, a PPT adversary $\mathcal{A}$ is a probabilistic interactive Turing Machine that runs in polynomial time in the security parameter λ. We will drop the security parameter λ from the notation when it is implicit.

2.2.1 Assumptions

Definition 2.1 (Discrete Log Relation)**.** *For all PPT adversaries $\mathcal{A}$ and for all $n \geq 2$ there exists a negligible function $\mu(\lambda)$ such that*

$$P\left[\begin{array}{l}(\mathbb{G},+) = \mathsf{Setup}(1^\lambda),\ G_1,\ldots,G_n \leftarrow\!\$\ \mathbb{G};\\ a_1,\ldots,a_n \in \mathbb{Z}_p \leftarrow \mathcal{A}(\mathbb{G},G_1,\ldots,G_n)\end{array} : \exists a_i \neq 0 \wedge \prod_{i=1}^{n} a_i \cdot G_i = \mathbf{0}_\mathbb{G} \in \mathbb{G}\right] \leq \mu(\lambda)$$

We say $\sum_{i=1}^{n} a_i \cdot G_i = \mathbf{0}_\mathbb{G}$ is a non trivial discrete log relation between $g_1,\ldots,g_n$. The Discrete Log Relation assumption states that an adversary can't find a non-trivial relation between randomly chosen group elements. For $n \geq 1$ this assumption is equivalent to the discrete-log assumption.

2.2.2 Commitments

Definition 2.2 (Commitment)**.** *A non-interactive commitment scheme consists of a pair of probabilistic polynomial time algorithms* $(\mathsf{Setup}, \mathsf{Commit})$*. The setup algorithm* $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$ *generates public parameters* p *for the scheme, for security parameter λ. The commitment algorithm* $\mathsf{Commit}_\mathsf{p}$ *defines a function $\mathcal{M} \times \mathcal{R} \to \mathcal{C}$ for message space $\mathcal{M}$, randomness space $\mathcal{R}$ and commitment space $\mathcal{C}$ determined by* p*. For a message $x \in \mathcal{M}$, the algorithm draws $r \leftarrow\!\$\ \mathcal{R}$ uniformly at random, and computes commitment* $\mathbf{com} = \mathsf{Commit}_\mathsf{p}(x;r)$*.*

For ease of notation we write $\mathsf{Commit} = \mathsf{Commit}_\mathsf{p}$.

Definition 2.3 (Homomorphic Commitments)**.** *A homomorphic commitment scheme is a non-interactive commitment scheme such that $\mathcal{M},\mathcal{R}$ and $\mathcal{C}$ are all abelian groups, and for all $x_1,x_2 \in \mathcal{M}$, $r_1,r_2 \in \mathcal{R}$, we have*

$$\mathsf{Commit}(x_1;r_1) + \mathsf{Commit}(x_2;r_2) = \mathsf{Commit}(x_1+x_2;r_1+r_2)$$

Definition 2.4 (Hiding Commitment)**.** *A commitment scheme is said to be hiding if for all PPT adversaries $\mathcal{A}$ there exists a negligible function $\mu(\lambda)$ such that.*

$$\left|\mathrm{P}\left[b = b' \;\middle|\; \begin{array}{l}\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda);\\ (x_0,x_1) \in \mathcal{M}^2 \leftarrow \mathcal{A}(\mathsf{p}), b \leftarrow\!\$\ \{0,1\}, r \leftarrow\!\$\ \mathcal{R},\\ \mathbf{com} = \mathsf{Commit}(x_b;r), b' \leftarrow \mathcal{A}(\mathsf{p},\mathbf{com})\end{array}\right] - \frac{1}{2}\right| \leq \mu(\lambda)$$

where the probability is over b,r,Setup and $\mathcal{A}$. If $\mu(\lambda) = 0$ then we say the scheme is perfectly hiding*.*

Definition 2.5 (Binding Commitment)**.** *A commitment scheme is said to be binding if for all PPT*

adversaries $\mathcal{A}$ there exists a negligible function μ such that.

$$\mathrm{P}\left[\mathsf{Commit}(x_0; r_0) = \mathsf{Commit}(x_1; r_1) \wedge x_0 \neq x_1 \;\middle|\; \begin{array}{l} \mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda), \\ x_0, x_1, r_0, r_1 \leftarrow \mathcal{A}(\mathsf{p}) \end{array}\right] \leq \mu(\lambda)$$

where the probability is over Setup *and* $\mathcal{A}$*. If* $\mu(\lambda) = 0$ *then we say the scheme is* perfectly binding*.*

In what follows, the order p of the groups used is implicitly dependent on the security parameter λ to ensure that discrete log in these groups is intractable for PPT adversaries.

Definition 2.6 (Pedersen Commitment)**.** $\mathcal{M}, \mathcal{R} = \mathbb{Z}_p$, $\mathcal{C} = (\mathbb{G}, +)$ *of order* p*.*
$\mathsf{Setup} : G, H \leftarrow\!\!\$\, \mathbb{G}$
$\mathsf{Commit}(x; r) = (x \cdot G + r \cdot H)$

Definition 2.7 (Pedersen Vector Commitment)**.** $\mathcal{M} = \mathbb{Z}_p^n$, $\mathcal{R} = \mathbb{Z}_p$, $\mathcal{C} = (\mathbb{G}, +)$ *with* $\mathbb{G}$ *of order* p
$\mathsf{Setup} : \mathbf{G} = (G_1, \ldots, G_n), H \leftarrow\!\!\$\, \mathbb{G}$
$\mathsf{Commit}(\mathbf{x} = (x_1, \ldots, x_n); r) = r \cdot H + \langle \mathbf{x}, \mathbf{G} \rangle = r \cdot H + \sum_i x_i \cdot G_i \in \mathbb{G}$

The Pedersen vector commitment is perfectly hiding and computationally binding under the discrete logarithm assumption. We will often set $r = 0$, in which case the commitment is binding but not hiding.

2.2.3 Zero-Knowledge Arguments of Knowledge

Bulletproofs are zero-knowledge arguments of knowledge. A zero-knowledge proof of knowledge is a protocol in which a prover can convince a verifier that some statement holds without revealing any information about why it holds. A prover can for example convince a verifier that a confidential transaction is valid without revealing why that is the case, i.e. without leaking the transacted values. An argument is a proof which holds only if the prover is computationally bounded and certain computational hardness assumptions hold. We now give formal definitions.

We will consider arguments consisting of three interactive algorithms $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$, all running in probabilistic polynomial time. These are the common reference string generator Setup, the prover P, and the verifier V. On input 1^λ, algorithm Setup produces a common reference string σ. The transcript produced by P and V when interacting on inputs s and t is denoted by $tr \leftarrow \langle \mathsf{P}(s), \mathsf{V}(t) \rangle$. We write $\langle \mathsf{P}(s), \mathsf{V}(t) \rangle = b$ depending on whether the verifier rejects, $b = 0$, or accepts, $b = 1$.

Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time-decidable ternary relation. Given σ, we call w a witness for a statement u if $(\sigma, u, w) \in \mathcal{R}$, and define the CRS-dependent language

$$\mathcal{L}_\sigma = \{x \mid \exists w : (\sigma, x, w) \in \mathcal{R}\}$$

as the set of statements x that have a witness w in the relation $\mathcal{R}$.

Definition 2.8 (Argument of Knowledge)**.** *The triple* $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *is called an* argument of knowledge *for relation* $\mathcal{R}$ *if it satisfies the following two definitions.*

Definition 2.9 (Perfect completeness)**.** $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *has perfect completeness if for all non-uniform polynomial time adversaries* $\mathcal{A}$

$$\mathrm{P}\left[(\sigma, u, w) \notin \mathcal{R} \text{ or } \langle \mathsf{P}(\sigma, u, w), \mathsf{V}(\sigma, u) \rangle = 1 \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda) \\ (u, w) \leftarrow \mathcal{A}(\sigma) \end{array} \right] = 1$$

Definition 2.10 (Computational Witness-Extended Emulation)**.** $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *has* witness-extended emulation *if for all deterministic polynomial time* P^* *there exists an expected polynomial time emulator* Ext *such that for all pairs of interactive adversaries* $\mathcal{A}_1, \mathcal{A}_2$ *there exists a negligible function* $\mu(\lambda)$ *such that*

$$\left| \begin{array}{l} \mathrm{P}\left[\mathcal{A}_1(tr) = 1 \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, s) \leftarrow \mathcal{A}_2(\sigma), \\ tr \leftarrow \langle \mathsf{P}^*(\sigma, u, s), \mathsf{V}(\sigma, u) \rangle \end{array} \right] - \\ \\ \mathrm{P}\left[\begin{array}{l} \mathcal{A}_1(tr) = 1 \\ \wedge (tr \text{ is accepting} \implies (\sigma, u, w) \in \mathcal{R}) \end{array} \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), \\ (u, s) \leftarrow \mathcal{A}_2(\sigma), \\ (tr, w) \leftarrow \mathsf{Ext}^{\mathcal{O}}(\sigma, u) \end{array} \right] \end{array} \right| \leq \mu(\lambda)$$

where the oracle is given by $\mathcal{O} = \langle \mathsf{P}^*(\sigma, u, s), \mathsf{V}(\sigma, u) \rangle$*, and permits rewinding to a specific point and resuming with fresh randomness for the verifier from this point onwards. We can also define* computational *witness-extended emulation by restricting to non-uniform polynomial time adversaries* $\mathcal{A}_1$ *and* $\mathcal{A}_2$*.*

We use witness-extended emulation to define knowledge-soundness as used for example in [Boo+16] and defined in [GI08; Lin03]. Informally, whenever an adversary produces an argument which satisfies the verifier with some probability, then there exists an emulator producing an identically distributed argument with the same probability, but also a witness. The value s can be considered to be the internal state of P^*, including randomness. The emulator is permitted to rewind the interaction between the prover and verifier to any move, and resume with the same internal state for the prover, but with fresh randomness for the verifier. Whenever P^* makes a convincing argument when in state s, Ext can extract a witness, and therefore, we have an argument of knowledge of w such that $(\sigma, u, w) \in \mathcal{R}$.

Definition 2.11 (Public Coin)**.** *An argument of knowledge* $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *is called* public coin *if all messages sent from the verifier to the prover are chosen uniformly at random and independently of the prover's messages, i.e., the challenges correspond to the verifier's randomness* ρ*.*

An argument of knowledge is zero knowledge if it does not leak information about w apart from

what can be deduced from the fact that $(\sigma, x, w) \in \mathcal{R}$. We will present arguments of knowledge that have special honest-verifier zero-knowledge. This means that given the verifier's challenge values, it is possible to efficiently simulate the entire argument without knowing the witness.

Definition 2.12 (Perfect Special Honest-Verifier Zero-Knowledge)**.** *A public coin argument of knowledge* $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *is a* perfect special honest verifier zero-knowledge (SHVZK) *argument of knowledge for* $\mathcal{R}$ *if there exists a probabilistic polynomial time simulator* Sim *such that for all pairs of interactive adversaries* $\mathcal{A}_1, \mathcal{A}_2$

$$\Pr\left[\begin{array}{l|l} (\sigma, u, w) \in \mathcal{R} \text{ and } \mathcal{A}_1(tr) = 1 & \begin{array}{l}\sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma), \\ tr \leftarrow \langle \mathsf{P}(\sigma, u, w), \mathsf{V}(\sigma, u; \rho)\rangle\end{array}\end{array}\right]$$

$$= \Pr\left[\begin{array}{l|l} (\sigma, u, w) \in \mathcal{R} \text{ and } \mathcal{A}_1(tr) = 1 & \begin{array}{l}\sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma), \\ tr \leftarrow \mathcal{S}(u, \rho)\end{array}\end{array}\right]$$

where ρ *is the public coin randomness used by the verifier.*

In this definition the adversary chooses a distribution over statements and witnesses but is still not able to distinguish between the simulated and the honestly generated transcripts for valid statements and witnesses.

We now define range proofs, which are proofs that the prover knows an opening to a commitment, such that the committed value is in a certain range. Range proofs can be used to show that an integer commitment is to a positive number or that two homomorphic commitments to elements in a field of prime order will not overflow modulo the prime when they are added together.

Definition 2.13 (Zero-Knowledge Range Proof)**.** *Given a commitment scheme* $(\mathsf{Setup}, \mathsf{Commit})$ *over a message space* $\mathcal{M}$ *which is a set with a total ordering, a Zero-Knowledge Range Proof is a SHVZK argument of knowledge for the relation* $\mathcal{R}_{\mathsf{Range}}$*:*

$$\mathcal{R}_{\mathsf{Range}} : (\mathsf{p}, (\mathbf{com}, l, r), (x, \rho)) \in \mathcal{R}_{\mathsf{Range}} \leftrightarrow \mathbf{com} = \mathsf{Commit}(x; \rho) \wedge l \leq x < r$$

2.2.4 Notation

Let $\mathbb{G}$ denote a cyclic additive group of prime order p and let $\mathbb{Z}_p$ denote the ring of integers modulo p. Let $\mathbb{G}^n$ and $\mathbb{Z}_p^n$ be vector spaces of dimension n over $\mathbb{G}$ and $\mathbb{Z}_p$ respectively. Let $\mathbb{Z}_p^\star$ denote $\mathbb{Z}_p \setminus \{0\}$. Generators of $\mathbb{G}$ are denoted by $G, H, V, U \in \mathbb{G}$. Group elements are capitalized and blinding factors are denoted by Greek letters, i.e. $C = a \cdot G + \alpha H \in \mathbb{G}$ is a Pedersen commitment to a. If not otherwise clear from context $x, y, z \in \mathbb{Z}_p^\star$ are uniformly distributed challenges. $x \leftarrow\!\$\ \mathbb{Z}_p^\star$ denotes the uniform sampling of an element from $\mathbb{Z}_p^\star$. Throughout the paper, we will also be using vector notations defined as follows. Bold font denotes vectors, i.e. $\mathbf{a} \in \mathbb{F}^n$ is a vector with elements $a_1, \ldots, a_n \in \mathbb{F}$. Capitalized bold font denotes matrices, i.e. $\mathbf{A} \in \mathbb{F}^{n \times m}$ is a matrix with n rows and m columns such

that $a_{i,j}$ is the element of $\mathbf{A}$ in the ith row and jth column. For a scalar $c \in \mathbb{Z}_p$ and a vector $\mathbf{a} \in \mathbb{Z}_p^n$, we denote by $\mathbf{b} = c \cdot \mathbf{a} \in \mathbb{Z}_p^n$ the vector where $b_i = c \cdot a_i$. Furthermore, let $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i \cdot b_i$ denotes the inner product between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$ and $\mathbf{a} \circ \mathbf{b} = (a_1 \cdot b_1, \ldots, a_n \cdot b_n) \in \mathbb{F}^n$ the Hadamard product or entry wise multiplication of two vectors.

We also define vector polynomials $p(X) = \sum_{i=0}^{d} \mathbf{p_i} \cdot X^i \in \mathbb{Z}_p^n[X]$ where each coefficient $\mathbf{p_i}$ is a vector in $\mathbb{Z}_p^n$. The inner product between two vector polynomials $l(X), r(X)$ is defined as

$$\langle l(X), r(X) \rangle = \sum_{i=0}^{d} \sum_{j=0}^{i} \langle \mathbf{l_i}, \mathbf{r_j} \rangle \cdot X^{i+j} \in \mathbb{Z}_p[X] \tag{2.1}$$

Let $t(X) = \langle \mathbf{l}(X), \mathbf{r}(X) \rangle$, then the inner product is defined such that $t(x) = \langle l(x), r(x) \rangle$ holds for all $x \in \mathbb{Z}_p$, i.e. evaluating the polynomials at x and then taking the inner product is the same as evaluating the inner product polynomial at x.

For a vector $\mathbf{G} = (G_1, \ldots, G_n) \in \mathbb{G}^n$ and $\mathbf{a} \in \mathbb{Z}_p^n$ we write $C = \langle \mathbf{a}, \mathbf{G} \rangle = \sum_{i=1}^{n} a_i \cdot G_i \in \mathbb{G}$. This quantity is a binding (but not hiding) commitment to the vector $\mathbf{a} \in \mathbb{Z}_p^n$. Given such a commitment C and a vector $\mathbf{b} \in \mathbb{Z}_p^n$ with non-zero entries, we can treat C as a new commitment to $\mathbf{a} \circ \mathbf{b}$. To so do, define $G'_i = (b_i^{-1}) \cdot G_i$ such that $C = \sum_{i=1}^{n} (a_i \cdot b_i) \cdot G'_i$. The binding property of this new commitment is inherited from the old commitment.

Let $\mathbf{a} \parallel \mathbf{b}$ denote the concatenation of two vectors: if $\mathbf{a} \in \mathbb{Z}_p^n$ and $\mathbf{b} \in \mathbb{Z}_p^m$ then $\mathbf{a} \parallel \mathbf{b} \in \mathbb{Z}_p^{n+m}$. For $0 \leq \ell \leq n$, we use Python notation to denote slices of vectors:

$$\mathbf{a}_{[:\ell]} = (a_1, \ldots, a_\ell) \in \mathbb{F}^\ell, \qquad \mathbf{a}_{[\ell:]} = (a_{\ell+1}, \ldots, a_n) \in \mathbb{F}^{n-\ell}.$$

For $k \in \mathbb{Z}_p^\star$ we use $\mathbf{k}^n$ to denote the vector containing the first n powers of k, i.e.

$$\mathbf{k}^n = (1, k, k^2, \ldots, k^{n-1}) \in (\mathbb{Z}_p^\star)^n.$$

For example, $\mathbf{2}^n = (1, 2, 4, \ldots, 2^{n-1})$. Equivalently $\mathbf{k}^{-n} = (\mathbf{k^{-1}})^n = (1, k^{-1}, \ldots, k^{-n+1})$.

Finally, we write $\{(\text{Public Input}; \text{Witness}) : \text{Relation}\}$ to denote the relation Relation using the specified Public Input and Witness.

2.3 Improved Inner-Product Argument

Bootle et al. [Boo+16] introduced a communication efficient inner-product argument and show how it can be leveraged to construct zero-knowledge proofs for arithmetic circuit satisfiability with low communication complexity. The argument is an argument of knowledge that the prover knows the openings of two binding Pedersen vector commitments that satisfy a given inner product relation.

We reduce the communication complexity of the argument from $6 \log_2(n)$ in [Boo+16] to only

$2\log_2(n)$, where n is the dimension of the two vectors. We achieve this improvement by modifying the relation being proved. Our argument is sound, but is not zero-knowledge. We then show that this protocol gives a public-coin, communication efficient, zero-knowledge range proof on a set of committed values, and a zero-knowledge proof system for arbitrary arithmetic circuits (Sections 2.4 and 2.5). By applying the Fiat-Shamir heuristic we obtain short non-interactive proofs (Section 2.4.4).

Overview. The inputs to the inner-product argument are independent generators $\mathbf{G}, \mathbf{H} \in \mathbb{G}^n$, a scalar $c \in \mathbb{Z}_p$, and $P \in \mathbb{G}$. The argument lets the prover convince a verifier that the prover knows two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ such that

$$P = \langle \mathbf{a}, \mathbf{G} \rangle + \langle \mathbf{b}, \mathbf{H} \rangle \qquad \text{and} \qquad c = \langle \mathbf{a}, \mathbf{b} \rangle.$$

We refer to P as a binding vector commitment to $\mathbf{a}, \mathbf{b}$. Throughout the section we assume that the dimension n is a power of 2. If need be, one can easily pad the inputs to ensure that this holds.

More precisely, the inner product argument is an efficient proof system for the following relation:

$$\mathcal{R}_{\mathsf{ipa}} = \big\{ (\mathbf{G}, \mathbf{H} \in \mathbb{G}^n,\ P \in \mathbb{G},\ c \in \mathbb{Z}_p\ ;\ \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n) :\quad P = \langle \mathbf{a}, \mathbf{G} \rangle + \langle \mathbf{b}, \mathbf{H} \rangle \wedge c = \langle \mathbf{a}, \mathbf{b} \rangle \big\}. \tag{2.2}$$

The simplest proof system for (2.2) is one where the prover sends the vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ to the verifier. The verifier accepts if these vectors are a valid witness for (2.2). This is clearly sound, however, it requires sending $2n$ elements to the verifier. Our goal is to send only $2\log_2(n)$ elements.

We show how to do this when the inner product $c = \langle \mathbf{a}, \mathbf{b} \rangle$ is given as part of the vector commitment P. That is, for a given $P \in \mathbb{G}$, the prover proves that it has vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ for which $P = \langle \mathbf{a}, \mathbf{G} \rangle + \langle \mathbf{b}, \mathbf{H} \rangle + \langle \mathbf{a}, \mathbf{b} \rangle \cdot U$. More precisely, we design a proof system for the relation:

$$\big\{ (\mathbf{G}, \mathbf{H} \in \mathbb{G}^n,\ U, P \in \mathbb{G}\ ;\ \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n\ ) :\ P = \langle \mathbf{a}, \mathbf{G} \rangle + \langle \mathbf{b}, \mathbf{H} \rangle + \langle \mathbf{a}, \mathbf{b} \rangle \cdot U \big\}. \tag{2.3}$$

We show in Protocol 2.1 below that a proof system for (2.3) gives a proof system for (2.2) with the same complexity. Hence, it suffices to give a proof system for (2.3).

To give some intuition for how the proof system for the relation (2.3) works let us define a hash function $H : \mathbb{Z}_p^{2n+1} \to \mathbb{G}$ as follows. First, set $n' = n/2$ and fix generators $\mathbf{G}, \mathbf{H} \in \mathbb{G}^n, U \in \mathbb{G}$. Then the hash function H takes as input $\mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}' \in \mathbb{Z}_p^{n'}$ and $c \in \mathbb{Z}_p$, and outputs

$$H(\mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}', c) = \langle \mathbf{a}, \mathbf{G}_{[:n']} \rangle + \langle \mathbf{a}', \mathbf{G}_{[n':]} \rangle + \langle \mathbf{b}, \mathbf{H}_{[:n']} \rangle + \langle \mathbf{b}', \mathbf{H}_{[n':]} \rangle + c \cdot U \quad \in \mathbb{G}.$$

Now, using the setup in (2.3), we can write P as $P = H\left(\mathbf{a}_{[:n']}, \mathbf{a}_{[n':]},\ \mathbf{b}_{[:n']}, \mathbf{b}_{[n':]},\ \langle \mathbf{a}, \mathbf{b} \rangle\right)$. Note that

H is additively homomorphic in its inputs, i.e.

$$H(\mathbf{a}_1, \mathbf{a'}_1, \mathbf{b}_1, \mathbf{b'}_1, c_1) \cdot H(\mathbf{a}_2, \mathbf{a'}_2, \mathbf{b}_2, \mathbf{b'}_2, c_2) = H(\mathbf{a}_1 + \mathbf{a}_2,\ \mathbf{a'}_1 + \mathbf{a'}_2,\ \mathbf{b}_1 + \mathbf{b}_2,\ \mathbf{b'}_1 + \mathbf{b'}_2,\ c_1 + c_2).$$

Consider the following protocol for the relation (2.3), where $P \in \mathbb{G}$ is given as input:

- The prover computes $L, R \in \mathbb{G}$ as follows:

$$\begin{aligned} L &= H\big(\ \mathbf{0}^{n'},\ \ \mathbf{a}_{[:n']},\ \mathbf{b}_{[n':]},\ \ \mathbf{0}^{n'},\ \ \langle \mathbf{a}_{[:n']}, \mathbf{b}_{[n':]} \rangle\ \big) \\ R &= H\big(\ \mathbf{a}_{[n':]},\ \ \mathbf{0}^{n'},\ \ \mathbf{0}^{n'},\ \ \mathbf{b}_{[:n']},\ \langle \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']} \rangle\ \big) \\ \text{and recall that}\quad P &= H\big(\ \mathbf{a}_{[:n']},\ \mathbf{a}_{[n':]},\ \mathbf{b}_{[:n']},\ \mathbf{b}_{[n':]},\ \ \langle \mathbf{a}, \mathbf{b} \rangle\ \ \big). \end{aligned}$$

  It sends $L, R \in \mathbb{G}$ to the verifier.

- The verifier chooses a random $x \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends x to the prover.

- The prover computes $\mathbf{a'} = x\mathbf{a}_{[:n']} + x^{-1}\mathbf{a}_{[n':]} \in \mathbb{Z}_p^{n'}$ and $\mathbf{b'} = x^{-1}\mathbf{b}_{[:n']} + x\mathbf{b}_{[n':]} \in \mathbb{Z}_p^{n'}$ and sends $\mathbf{a'}, \mathbf{b'} \in \mathbb{Z}_p^{n'}$ to the verifier.

- Given $(L, R, \mathbf{a'}, \mathbf{b'})$, the verifier computes $P' = x^2 \cdot L + P + x^{-2} \cdot R$ and outputs "accept" if

$$P' = H\left(x^{-1}\mathbf{a'},\ x\mathbf{a'},\ x\mathbf{b'},\ x^{-1}\mathbf{b'},\ \langle \mathbf{a'}, \mathbf{b'} \rangle\right). \tag{2.4}$$

It is easy to verify that a proof from an honest prover will always be accepted. Indeed, the left hand side of (2.4) is

$$x^2 \cdot L + P + x^{-2} \cdot R = H\left(\mathbf{a}_{[:n']} + x^{-2}\mathbf{a}_{[n':]},\ \ x^2\mathbf{a}_{[:n']} + \mathbf{a}_{[n':]},\ \ x^2\mathbf{b}_{[n':]} + \mathbf{b}_{[:n']},\ \ \mathbf{b}_{[n':]} + x^{-2}\mathbf{b}_{[:n']},\ \ \langle \mathbf{a'}, \mathbf{b'} \rangle\right)$$

which is the same as the right hand side of (2.4).

In this proof system, the proof sent from the prover is the four tuple $(L, R, \mathbf{a'}, \mathbf{b'})$ and contains only $n + 2$ elements. This is about half the length of the trivial proof where the prover sends the complete $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ to the verifier.

To see why this protocol is a proof system for (2.3) we show how to extract a valid witness $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$ from a successful prover. After the prover sends L, R we rewind the prover three times to obtain three tuples $(x_i, \mathbf{a'}_i, \mathbf{b'}_i)$ for $i = 1, \ldots, 3$, where each tuple satisfies (2.4), namely

$$L^{(x_i^2)} \cdot P \cdot R^{(x_i^{-2})} = H(x_i^{-1}\mathbf{a'}_i,\ x_i\mathbf{a'}_i,\ x_i\mathbf{b'}_i,\ x_i^{-1}\mathbf{b'}_i,\ \langle \mathbf{a'}_i, \mathbf{b'}_i \rangle). \tag{2.5}$$

Assuming $x_i \neq \pm x_j$ for $1 \leq i < j \leq 3$, we can find $\nu_1, \nu_2, \nu_3 \in \mathbb{Z}_p$ such that

$$\sum_{i=1}^{3} x_i^2 \nu_i = 0 \quad \text{and} \quad \sum_{i=1}^{3} \nu_i = 1 \quad \text{and} \quad \sum_{i=1}^{3} x_i^{-2} \nu_i = 0.$$

Then setting

$$\mathbf{a} = \sum_{i=1}^{3} (\nu_i \cdot x_i^{-1} \mathbf{a}'_i, \quad \nu_i \cdot x_i \mathbf{a}'_i) \in \mathbb{Z}_p^n \quad \text{and} \quad \mathbf{b} = \sum_{i=1}^{3} (\nu_i \cdot x_i \mathbf{b}'_i, \quad \nu_i \cdot x_i^{-1} \mathbf{b}'_i) \in \mathbb{Z}_p^n$$

we obtain that $P = H\left(\mathbf{a}_{[:n']}, \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']}, \mathbf{b}_{[n':]}, c\right)$ where $c = \sum_{i=1}^{3} \nu_i \cdot \langle \mathbf{a}'_i, \mathbf{b}'_i \rangle$. We will show in the proof of Theorem 2.1 below that with one additional rewinding, to obtain a fourth relation satisfying (2.5), we must have $c = \langle \mathbf{a}, \mathbf{b} \rangle$ with high probability. Hence, the extracted $\mathbf{a}, \mathbf{b}$ are a valid witness for the relation (2.3), as required.

Shrinking the proof by recursion. Observe that the test in (2.4) is equivalent to testing that

$$P' = \langle \mathbf{a}', \left(x^{-1} \cdot \mathbf{G}_{[:n']} + x \cdot \mathbf{G}_{[n':]}\right) \rangle + \langle \mathbf{b}', \left(x \cdot \mathbf{H}_{[:n']} + x^{-1} \cdot \mathbf{H}_{[n':]}\right) \rangle + \langle \mathbf{a}', \mathbf{b}' \rangle \cdot U.$$

Hence, instead of the prover sending the vectors $\mathbf{a}', \mathbf{b}'$ to the verifier, they can recursively engage in an inner-product argument for P' with respect to generators $(x^{-1} \cdot \mathbf{G}_{[:n']} + x \cdot \mathbf{G}_{[n':]},\ x \cdot \mathbf{H}_{[:n']} + x^{-1} \cdot \mathbf{H}_{[n':]},\ U)$. The dimension of this problem is only $n' = n/2$.

The resulting $\log_2 n$ depth recursive protocol is shown in Protocol 2.2. This $\log_2 n$ round protocol is public coin and can be made non-interactive using the Fiat-Shamir heuristic. The total communication of Protocol 2.2 is only $2\lceil \log_2(n) \rceil$ elements in $\mathbb{G}$ plus 2 elements in $\mathbb{Z}_p$. Specifically, the prover sends the following terms:

$$(L_1, R_1),\ \ldots,\ (L_{\log_2 n}, R_{\log_2 n}), \quad a, b$$

where $a, b \in \mathbb{Z}_p$ are sent at the tail of the recursion. The prover's work is dominated by $8n$ group exponentiations and the verifier's work by $4n$ exponentiations. In Section 2.3.1 we present a more efficient verifier that performs only 1 multi-exponentiation of size $2n + 2\log(n)$. In Section 2.7 we present further optimizations.

Proving security. The inner product protocol for the relation (2.2) is presented in Protocol 2.1. This protocol uses internally a fixed group element $u \in \mathbb{G}$ for which there is no known discrete-log relation among $\mathbf{G}, \mathbf{H}, u$. The heart of Protocol 2.1 is Protocol 2.2 which is a proof system for the relation (2.3). In Protocol 2.1 the element u is raised to a verifier chosen power x to ensure that the extracted vectors $\mathbf{a}, \mathbf{b}$ from Protocol 2.2 satisfy $\langle \mathbf{a}, \mathbf{b} \rangle = c$.

P_{IP}'s input: $(\mathbf{G}, \mathbf{H}; P, c; \mathbf{a}, \mathbf{b})$
V_{IP}'s input: $(\mathbf{G}, \mathbf{H}; P; c)$

$$\mathsf{V}_{\mathsf{IP}} : X \leftarrow\!\!\$\ \mathbb{Z}_p^\star \tag{2.6}$$

$$\mathsf{V}_{\mathsf{IP}} \rightarrow \mathsf{P}_{\mathsf{IP}} : x \tag{2.7}$$

$$P' \leftarrow P + (x \cdot c) \cdot U \tag{2.8}$$

$$\text{Run Protocol 2.2 on Input } (\mathbf{G}, \mathbf{H}, c \cdot U, P'\ ;\ \mathbf{a}, \mathbf{b}) \tag{2.9}$$

Protocol 2.1: Proof system for Relation (2.2) using Protocol 2.2. Here $u \in \mathbb{G}$ is a fixed group element with an unknown discrete-log relative to $\mathbf{G}, \mathbf{H} \in \mathbb{G}^n$.

The following theorem shows that Protocol 2.1 is a proof system for (2.2).

Theorem 2.1 (Inner-Product Argument)**.** *The argument presented in Protocol 2.1 for the relation* (2.2) *has perfect completeness and statistical witness-extended-emulation for either extracting a non-trivial discrete logarithm relation between* $\mathbf{G}, \mathbf{H}, u$ *or extracting a valid witness* $\mathbf{a}, \mathbf{b}$*.*

The proof for Theorem 2.1 is given in Section 2.9.

2.3.1 Inner-Product Verification through Multi-Exponentiation

Protocol 2.2 has a logarithmic number of rounds and in each round the prover and verifier compute a new set of generators $\mathbf{G}', \mathbf{H}'$. This requires a total of $4n$ exponentiations: $2n$ in the first round, n in the second and $\frac{n}{2^{j-3}}$ in the jth. We can reduce the number of exponentiations to a single multi-exponentiation of size $2n$ by delaying all the exponentiations until the last round. This technique provides a significant speed-up if the proof is compiled to a non interactive proof using the Fiat-Shamir heuristic (as in Section 2.4.4).

Let g and h be the generators used in the final round of the protocol and x_j be the challenge from the jth round. In the last round the verifier checks that $a \cdot G + b \cdot H + (a \cdot b) \cdot U = P$, where $a, b \in \mathbb{Z}_p$ are given by the prover. By unrolling the recursion we can express these final g and h in terms of the input generators $\mathbf{G}, \mathbf{H} \in \mathbb{G}^n$ as:

$$g = \sum_{i=1}^{n} s_i \cdot G_i \in \mathbb{G}, \qquad h = \sum_{i=1}^{n} s_i^{-1} \cdot H_i \in \mathbb{G}$$

where $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_p^n$ only depends on the challenges $(x_1, \ldots, x_{\log_2(n)})$. The scalars $s_1, \ldots, s_n \in$

Improved inner product argument Π_{ipa}:
$(\mathsf{pp} = (\mathbf{G}, \mathbf{H}, U); P; (\mathbf{a}, \mathbf{b})) \in \mathcal{R}_{\mathsf{ipa}} : P = \langle \mathbf{a}, \mathbf{G} \rangle + \langle \mathbf{b}, \mathbf{H} \rangle + \langle \mathbf{a}, \mathbf{b} \rangle \cdot U$

	Prover $\mathsf{P}_{\mathsf{ipa}}(\mathsf{pp}; P \in \mathbb{G}; \mathbf{a}, \mathbf{b})$		**Verifier** $\mathsf{V}_{\mathsf{ipa}}(\mathsf{pp}; P)$
	..... **if** $n = 1$		
2 :		$\xrightarrow{a, b}$	Check $a \cdot G + b \cdot H + (a \cdot b) \cdot U \stackrel{?}{=} P$
	..... **else**		
3 :	$n' \leftarrow \frac{n}{2}$		
4 :	$c_L \leftarrow \langle \mathbf{a}_{[:n']}, \mathbf{b}_{[n':]} \rangle \in \mathbb{Z}_p$		
5 :	$c_R \leftarrow \langle \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']} \rangle \in \mathbb{Z}_p$		
6 :	$L \leftarrow \langle \mathbf{a}_{[:n']} \vert\vert \mathbf{b}_{[n':]}, \mathbf{G}_{[n':]} \vert\vert \mathbf{H}_{[:n']} \rangle + c_L \cdot U \in \mathbb{G}$		
7 :	$R \leftarrow \langle \mathbf{a}_{[n':]} \vert\vert \mathbf{b}_{[:n']}, \mathbf{G}_{[:n']} \vert\vert \mathbf{H}_{[n':]} \rangle + c_R \cdot U \in \mathbb{G}$	$\xrightarrow{L, R}$	
8 :		$\xleftarrow{x}$	$x \leftarrow\!\!\$\ \mathbb{Z}_p^*$
9 :	$\mathbf{G}' \leftarrow x^{-1} \cdot \mathbf{G}_{[:n']} + x \cdot \mathbf{G}_{[n':]}$		$\mathbf{G}' \leftarrow x^{-1} \cdot \mathbf{G}_{[:n']} + x \cdot \mathbf{G}_{[n':]}$
10 :	$\mathbf{H}' \leftarrow x \cdot \mathbf{H}_{[:n']} + x^{-1} \cdot \mathbf{H}_{[n':]}$		$\mathbf{H}' \leftarrow x \cdot \mathbf{H}_{[:n']} + x^{-1} \cdot \mathbf{H}_{[n':]}$
11 :	$P' \leftarrow x^2 \cdot L + P + x^{-1} \cdot R$		$P' \leftarrow x^2 \cdot L + P + x^{-1} \cdot R$
12 :	$\mathbf{a}' \leftarrow x \cdot \mathbf{a}_{[:n']} + x^{-1} \mathbf{a}_{[n':]}$		
13 :	$\mathbf{b}' \leftarrow x^{-1} \cdot \mathbf{b}_{[:n']} + x \mathbf{b}_{[n':]}$		
	..... Run Π_{ipa} on input $(\mathsf{pp} = (\mathbf{G}', \mathbf{H}', U); P'; \mathbf{a}'\mathbf{b}'$		
14 :			

Protocol 2.2: Improved Inner-Product Argument

$\mathbb{Z}_p$ are calculated as follows:

$$\text{for } i = 1, \ldots, n: \quad s_i = \prod_{j=1}^{\log_2(n)} x_j^{b(i,j)} \qquad \text{where} \qquad b(i,j) = \begin{cases} 1 & \text{the } j\text{th bit of } i-1 \text{ is 1} \\ -1 & \text{otherwise} \end{cases}$$

Now the entire verification check in the protocol reduces to the following single multi-exponentiation of size $2n + 2\log_2(n) + 1$:

$$\langle a \cdot \mathbf{s}, \mathbf{G} \rangle + \langle b \cdot \mathbf{s}^{-1}, \mathbf{H} \rangle + (a \cdot b) \cdot U \stackrel{?}{=} P + \sum_{j=1}^{\log_2(n)} x_j^2 \cdot L_j + x_j^{-2} \cdot R_j.$$

Because a multi-exponentiation can be done much faster than n separate exponentiations, as we discuss in Section 2.7, this leads to a significant savings.

2.4 Range Proof Protocol with Logarithmic Size

We now present a novel protocol for conducting short and aggregatable range proofs. The protocol uses the improved inner product argument from Protocol 2.1. First, in Section 2.4.1, we describe how to construct a range proof that requires the verifier to check an inner product between two vectors. Then, in Section 2.4.2, we show that this check can be replaced with an efficient inner-product argument. In Section 2.4.3, we show how to efficiently aggregate m range proofs into one short proof. In Section 2.4.4, we discuss how interactive public coin protocols can be made non-interactive by using the Fiat-Shamir heuristic, in the random oracle model. In Section 2.4.5 we present an efficient MPC protocol that allows multiple parties to construct a single aggregate range proof. Finally, in Section 2.4.6, we discuss an extension that enables a switch to quantum-secure range proofs in the future.

2.4.1 Inner-Product Range Proof

We present a protocol which uses the improved inner-product argument to construct a range proof. The proof convinces the verifier that a commitment V contains a number v that is in a certain range, without revealing v. Bootle et al. [Boo+16] give a proof system for arbitrary arithmetic circuits, and in Section 2.5 we show that our improvements to the inner product argument also transfer to this general proof system. It is of course possible to prove that a commitment is in a given range using an arithmetic circuit, and the work of [Boo+16] could be used to construct an asymptotically logarithmic sized range proof (in the length of v). However, the circuit would need to implement the commitment function, namely a multi-exponentiation for Pedersen commitments, leading to a large complex circuit.

We construct a range proof more directly by exploiting the fact that a Pedersen commitment V is an element in the same group $\mathbb{G}$ that is used to perform the inner product argument. We extend this idea in Section 2.5 to construct a proof system for circuits that operate on committed inputs.

Formally, let $v \in \mathbb{Z}_p$ and let $V \in \mathbb{G}$ be a Pedersen commitment to v using randomness γ. The proof system will convince the verifier that $v \in [0, 2^n - 1]$. In other words, the proof system proves the following relation which is equivalent to the range proof relation in Definition 2.13:

$$\mathcal{R}_{\mathsf{range}} = \big\{(G, H \in \mathbb{G}; V, n \ ; \ v, \gamma \in \mathbb{Z}_p) : V = v \cdot G + \gamma \cdot H \wedge v \in [0, 2^n - 1]\big\}. \tag{2.10}$$

Let $\mathbf{a}_L = (a_1, \ldots, a_n) \in \{0,1\}^n$ be the vector containing the bits of v, so that $\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$. The prover $\mathsf{P}_{\mathsf{range}}$ commits to $\mathbf{a}_L$ using a constant size vector commitment $A \in \mathbb{G}$. It will convince the verifier that v is in $[0, 2^n - 1]$ by proving that it knows an opening $\mathbf{a}_L \in \mathbb{Z}_p^n$ of A and $v, \gamma \in \mathbb{Z}_p$ such that $V = v \cdot G + \gamma \cdot H$ and

$$\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v \quad \text{and} \quad \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{0}^n \quad \text{and} \quad \mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^n \tag{2.11}$$

This proves that $a_1, \ldots, a_n$ are all in $\{0,1\}$, as required and that $\mathbf{a}_L$ is composed of the bits of v. The high level goal of the following protocol is to convert these $2n+1$ constraints as a single inner-product constraint. This will allow us to use Protocol 2.1 to efficiently argue that an inner-product relation holds. To do this we take a random linear combination (chosen by the verifier) of the constraints. If the original constraints were not satisfied then it is inversely proportional in the challenge space unlikely that the combined constraint holds.

Concretley, we use the following observation: to prove that a committed vector $\mathbf{b} \in \mathbb{Z}_p^n$ satisfies $\mathbf{b} = \mathbf{0}^n$ it suffices for the verifier to send a random $y \in \mathbb{Z}_p$ to the prover and for the prover to prove that $\langle \mathbf{b}, \mathbf{y}^n \rangle = 0$. If $\mathbf{b} \neq \mathbf{0}^n$ then the equality will hold with at most negligible probability n/p. Hence, if $\langle \mathbf{b}, \mathbf{y}^n \rangle = 0$ the verifier is convinced that $\mathbf{b} = \mathbf{0}^n$.

Using this observation, and using a random $y \in \mathbb{Z}_p$ from the verifier, the prover can prove that (2.11) holds by proving that

$$\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v \quad \text{and} \quad \langle \mathbf{a}_L \ , \ \mathbf{a}_R \circ \mathbf{y}^n \rangle = 0 \quad \text{and} \quad \langle \mathbf{a}_L - \mathbf{1}^n - \mathbf{a}_R \ , \ \mathbf{y}^n \rangle = 0. \tag{2.12}$$

We can combine these three equalities into one using the same technique: the verifier chooses a random $z \in \mathbb{Z}_p$ and then the prover proves that

$$z^2 \cdot \langle \mathbf{a}_L, \mathbf{2}^n \rangle + z \cdot \langle \mathbf{a}_L - \mathbf{1}^n - \mathbf{a}_R \ , \ \mathbf{y}^n \rangle + \langle \mathbf{a}_L \ , \ \mathbf{a}_R \circ \mathbf{y}^n \rangle = z^2 \cdot v.$$

This equality can be re-written as:

$$\Big\langle \mathbf{a}_L - z \cdot \mathbf{1}^n \ , \ \mathbf{y}^n \circ (\mathbf{a}_R + z \cdot \mathbf{1}^n) + z^2 \cdot \mathbf{2}^n \Big\rangle = z^2 \cdot v + \delta(y, z) \tag{2.13}$$

where $\delta(y,z) = (z-z^2)\cdot\langle \mathbf{1}^n, \mathbf{y}^n\rangle - z^3\langle \mathbf{1}^n, \mathbf{2}^n\rangle \in \mathbb{Z}_p$ is a quantity that the verifier can easily calculate. We thus reduced the problem of proving that (2.11) holds to proving a single inner-product identity.

If the prover could send to the verifier the two vectors in the inner product in (2.13) then the verifier could check (2.13) itself, using the commitment V to v, and be convinced that (2.11) holds. However, these two vectors reveal information about $\mathbf{a}_L$ and therefore the prover cannot send them to the verifier. We solve this problem by introducing two additional blinding terms $\mathbf{s}_L, \mathbf{s}_R \in \mathbb{Z}_p^n$ to blind these vectors.

Specifically, to prove the statement (2.10), $\mathsf{P}_{\mathsf{range}}$ and $\mathsf{V}_{\mathsf{range}}$ engage in the following zero knowledge protocol:

$\mathsf{P}_{\mathsf{range}}$ on input v, γ computes: (2.14)

$\mathbf{a}_L \in \{0,1\}^n$ s.t.$\langle \mathbf{a}_L, \mathbf{2}^n\rangle = v$ (2.15)

$\mathbf{a}_R \leftarrow \mathbf{a}_L - \mathbf{1}^n \in \mathbb{Z}_p^n$ (2.16)

$\alpha \leftarrow\!\$\ \mathbb{Z}_p$ (2.17)

$A \leftarrow \alpha\cdot H + \langle \mathbf{a}_L, \mathbf{G}\rangle + \langle \mathbf{a}_R, \mathbf{H}\rangle \in \mathbb{G}$ // *commitment to* $\mathbf{a}_L$ *and* $\mathbf{a}_R$ (2.18)

$\mathbf{s}_L, \mathbf{s}_R \leftarrow\!\$\ \mathbb{Z}_p^n$ // *choose blinding vectors* $\mathbf{s}_L, \mathbf{s}_R$ (2.19)

$\rho \leftarrow\!\$\ \mathbb{Z}_p$ (2.20)

$S \leftarrow \rho\cdot H + \langle \mathbf{s}_L, \mathbf{G}\rangle + \langle \mathbf{s}_R, \mathbf{H}\rangle \in \mathbb{G}$ // *commitment to* $\mathbf{s}_L$ *and* $\mathbf{s}_R$ (2.21)

$\mathsf{P}_{\mathsf{range}} \rightarrow \mathsf{V}_{\mathsf{range}} : A, S$ (2.22)

$\mathsf{V}_{\mathsf{range}} : y, z \leftarrow\!\$\ \mathbb{Z}_p^\star$ // *challenge points* (2.23)

$\mathsf{V}_{\mathsf{range}} \rightarrow \mathsf{P}_{\mathsf{range}} : y, z$ (2.24)

With this setup, let us define two linear vector polynomials $l(X), r(X)$ in $\mathbb{Z}_p^n[X]$, and a quadratic polynomial $t(X) \in \mathbb{Z}_p[X]$ as follows:

$$\begin{aligned} l(X) &= (\mathbf{a}_L - z\cdot\mathbf{1}^n) + \mathbf{s}_L\cdot X && \in \mathbb{Z}_p^n[X] \\ r(X) &= \mathbf{y}^n \circ (\mathbf{a}_R + z\cdot\mathbf{1}^n + \mathbf{s}_R\cdot X) + z^2\cdot\mathbf{2}^n && \in \mathbb{Z}_p^n[X] \\ t(X) &= \langle l(X), r(X)\rangle = t_0 + t_1\cdot X + t_2\cdot X^2 && \in \mathbb{Z}_p[X] \end{aligned}$$

where the inner product in the definition of $t(X)$ is as in (2.1). The constant terms of $l(X)$ and $r(X)$ are the inner product vectors in (2.13). The blinding vectors $\mathbf{s}_L$ and $\mathbf{s}_R$ ensure that the prover can publish $l(x)$ and $r(x)$ for one $x \in \mathbb{Z}_p^\star$ without revealing any information about $\mathbf{a}_L$ and $\mathbf{a}_R$.

The constant term of $t(x)$, denoted t_0, is the result of the inner product in (2.13). The prover

needs to convince the verifier that this t_0 satisfies (2.13), namely

$$t_0 = v \cdot z^2 + \delta(y, z).$$

To so do, the prover commits to the remaining coefficients of $t(X)$, namely $t_1, t_2 \in \mathbb{Z}_p$. It then convinces the verifier that it has a commitment to the coefficients of $t(X)$ by checking the value of $t(X)$ at a random point $x \in \mathbb{Z}_p^\star$. Specifically, they do:

$\mathsf{P}_{\mathsf{range}}$ computes: (2.25)

$\tau_1, \tau_2 \leftarrow\!\!\$\ \mathbb{Z}_p$ (2.26)

$T_i \leftarrow t_i \cdot G + \tau_i \cdot H \in \mathbb{G}, \quad i = \{1, 2\}$ // *commit to* t_1, t_2 (2.27)

$\mathsf{P}_{\mathsf{range}} \rightarrow \mathsf{V}_{\mathsf{range}} : T_1, T_2$ (2.28)

$\mathsf{V}_{\mathsf{range}} : x \leftarrow\!\!\$\ \mathbb{Z}_p^\star$ (2.29)

$\mathsf{V}_{\mathsf{range}} \rightarrow \mathsf{P}_{\mathsf{range}} : x$ // *a random challenge* (2.30)

$\mathsf{P}_{\mathsf{range}}$ computes: (2.31)

$\mathbf{l} \leftarrow l(x) = \mathbf{a}_L - z \cdot \mathbf{1}^n + \mathbf{s}_L \cdot x \in \mathbb{Z}_p^n$ (2.32)

$\mathbf{r} \leftarrow r(x) = \mathbf{y}^n \circ (\mathbf{a}_R + z \cdot \mathbf{1}^n + \mathbf{s}_R \cdot x) + z^2 \cdot \mathbf{2}^n \in \mathbb{Z}_p^n$ (2.33)

$\hat{t} \leftarrow \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p$ // $\hat{t} = t(x)$ (2.34)

$\tau_x \leftarrow \tau_2 \cdot x^2 + \tau_1 \cdot x + z^2 \cdot \gamma \in \mathbb{Z}_p$ // *blinding value for* $\hat{t}$ (2.35)

$\mu \leftarrow \alpha + \rho \cdot x \in \mathbb{Z}_p$ // α, ρ *blind* A, S (2.36)

$\mathsf{P}_{\mathsf{range}} \rightarrow \mathsf{V}_{\mathsf{range}} : \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r}$ (2.37)

The verifier checks that $\mathbf{l}$ and $\mathbf{r}$ are in fact $l(x)$ and $r(x)$ and checks that $t(x) = \langle \mathbf{l}, \mathbf{r} \rangle$. In order to construct a commitment to $\mathbf{a}_R \circ \mathbf{y}^n$ the verifier switches the generators of the commitment from $\mathbf{H} \in \mathbb{G}^n$ to $\mathbf{H}' = (\mathbf{y}^{-n}) \circ \mathbf{H}$. This has the effect that A is now a vector commitment to $(\mathbf{a}_L,\ \mathbf{a}_R \circ \mathbf{y}^n)$ with respect to the new generators $(\mathbf{G}, \mathbf{H}', h)$. Similarly S is now a vector commitment

to $(\mathbf{s}_L,\ \mathbf{s}_R \circ \mathbf{y}^n)$. The remaining steps of the protocol are:

$$H'_i \leftarrow (y^{-i+1}) \cdot H_i \in \mathbb{G}, \quad \forall i \in [1,n] \qquad // \quad \mathbf{H}' = \mathbf{y}^{-\mathbf{1}^n} \circ \mathbf{H}$$

$$\hat{t}G + \tau_x H \stackrel{?}{=} z^2 V + \delta(y,z)G \cdot xT_1 + x^2 T_2 \qquad // \quad \textit{check that } \hat{t} = t(x) = t_0 + t_1 x + t_2 x^2 \tag{2.38}$$

$$P \leftarrow A + x \cdot S \tag{2.39}$$

$$- z \cdot \langle \mathbf{1}, \mathbf{G} \rangle$$

$$+ \langle (z \cdot \mathbf{y}^n + z^2 \cdot \mathbf{2}^n), \mathbf{H}' \rangle \in \mathbb{G} \qquad // \quad \textit{compute a commitment to } l(x), r(x)$$

$$P \stackrel{?}{=} \mu \cdot H + \langle \mathbf{l}, \mathbf{G} \rangle + \langle \mathbf{r}, \mathbf{H}' \rangle \qquad // \quad \textit{check that } \mathbf{l}, \mathbf{r} \textit{ are correct} \tag{2.40}$$

$$\hat{t} \stackrel{?}{=} \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p \qquad // \quad \textit{check that } \hat{t} \textit{ is correct} \tag{2.41}$$

Equation (2.38) is the only place where the verifier uses the given Pedersen commitment V to v.

Corollary 2.2 (Range Proof)**.** *The range proof presented in Section 2.4.1 has perfect completeness, perfect special honest verifier zero-knowledge, and computational witness extended emulation.*

Proof. The range proof is a special case of the aggregated range proof from section 2.4.3 with $m = 1$. This is therefore a direct corollary of Theorem 2.3. □

2.4.2 Logarithmic Range Proof

Finally, we can describe the efficient range proof that uses the improved inner product argument.

In the range proof protocol from Section 2.4.1, $\mathsf{P}_{\mathsf{range}}$ transmits $\mathbf{l}$ and $\mathbf{r}$, whose size is linear in n. Our goal is a proof whose size is logarithmic in n.

We can eliminate the transfer of $\mathbf{l}$ and $\mathbf{r}$ using the inner-product argument from Section 2.3. These vectors are not secret, and hence a protocol that only provides soundness is sufficient.

To use the inner-product argument, observe that verifying (2.40) and (2.41) is the same as verifying that the witness $\mathbf{l}, \mathbf{r}$ satisfies the inner product relation (2.2) on public input $\big(\mathbf{G},\ \mathbf{H}', P - \mu \cdot H,\ \hat{t}\big)$. That is, $P \in \mathbb{G}$ is a commitment to two vectors $\mathbf{l}, \mathbf{r} \in \mathbb{Z}_p^n$ whose inner product is $\hat{t}$. We can therefore replace (2.37) with a transfer of $(\tau_x, \mu, \hat{t})$, as before, and an execution of an inner product argument. Then instead of transmitting $\mathbf{l}$ and $\mathbf{r}$, which has a communication cost of $2 \cdot n$ elements, the inner-product argument transmits only $2 \cdot \lceil \log_2(n) \rceil + 2$ elements. In total, the prover sends only $2 \cdot \lceil \log_2(n) \rceil + 4$ group elements and 5 elements in $\mathbb{Z}_p$.

2.4.3 Aggregating Logarithmic Proofs

In many of the range proof applications described in Section 2.1.2, a single prover needs to perform multiple range proofs at the same time. For example, a confidential transaction often contains multiple outputs, and in fact, most transactions require a so-called *change output* to send any unspent funds back to the sender. In Provisions [Dag+15] the proof of solvency requires the exchange

to conduct a range proof for every single account. Given the logarithmic size of the range proof presented in Section 2.4.2, there is some hope that we can perform a proof for m values which is more efficient than conducting m individual range proofs. In this section, we show that this can be achieved with a slight modification to the proof system from Section 2.4.1.

Concretely, we present a proof system for the following relation:

$$\left\{(G, H \in \mathbb{G},\ \ \mathbf{V} \in \mathbb{G}^m\ \ ;\ \ \mathbf{v}, \boldsymbol{\gamma} \in \mathbb{Z}_p^m) : V_j = \gamma_j \cdot H + v_j \cdot G \wedge v_j \in [0, 2^n - 1]\ \ \forall j \in [1, m]\right\} \quad (2.42)$$

The prover is very similar to the prover for a simple range proof with $n \cdot m$ bits, with the following slight modifications. In line (2.15), the prover should compute $\mathbf{a}_L \in \mathbb{Z}_p^{n \cdot m}$ such that $\langle \mathbf{2}^n, \mathbf{a}_{L\,[(j-1)\cdot n\,:\,j\cdot n-1]} \rangle = v_j$ for all j in $[1, m]$, i.e. $\mathbf{a}_L$ is the concatenation of all of the bits for every v_j. We adjust $l(X)$ and $r(X)$ accordingly so that

$$l(X) = (\mathbf{a}_L - z \cdot \mathbf{1}^{n \cdot m}) + \mathbf{s}_L \cdot X \in \mathbb{Z}_p^{n \cdot m}[X] \quad (2.43)$$

$$r(X) = \mathbf{y}^{n \cdot m} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{n \cdot m} + \mathbf{s}_R \cdot X) + \sum_{j=1}^{m} z^{1+j} \cdot \left(\mathbf{0}^{(j-1)\cdot n} \parallel \mathbf{2}^n \parallel \mathbf{0}^{(m-j)\cdot n}\right) \in \mathbb{Z}_p^{n \cdot m} \quad (2.44)$$

In the computation of τ_x, we need to adjust for the randomness of each commitment V_j, so that $\tau_x = \tau_1 \cdot x + \tau_2 \cdot x^2 + \sum_{j=1}^{m} z^{1+j} \cdot \gamma_j$. Further, $\delta(y, z)$ is updated to incorporate more cross terms.

$$\delta(y, z) = (z - z^2) \cdot \langle \mathbf{1}^{n \cdot m}, \mathbf{y}^{n \cdot m} \rangle - \sum_{j=1}^{m} z^{j+2} \cdot \langle \mathbf{1}^n, \mathbf{2}^n \rangle$$

The verification check (2.38) needs to be updated to include all the V_j commitments.

$$\hat{t} \cdot G + \tau_x \cdot H \stackrel{?}{=} \delta(y, z) \cdot G + z^2 \cdot \langle \mathbf{z}^m, \mathbf{V} \rangle + x \cdot T_1 + x^2 \cdot T_2 \quad (2.45)$$

Finally, we change the definition of P (2.39) such that it is a commitment to the new $\mathbf{r}$.

$$P = A + x \cdot S - z \cdot \langle \mathbf{1}, \mathbf{G} \rangle + z \cdot \langle \mathbf{y}^{n \cdot m}, \mathbf{H}' \rangle + \sum_{j=1}^{m} z^{j+1} \cdot \langle \mathbf{2}^n, \mathbf{H}'_{[(j-1)\cdot n\,:\,j\cdot n-1]} \rangle$$

The aggregated range proof which makes use of the inner product argument uses $2 \cdot \lceil \log_2(n \cdot m) \rceil + 4$ group elements and 5 elements in $\mathbb{Z}_p$. Note that the proof size only grows by an additive term of $2 \cdot \log_2(m)$ when conducting multiple range proofs as opposed to a multiplicative factor of m when creating m independent range proofs.

Theorem 2.3. *The aggregate range proof presented in Section 2.4.3 has perfect completeness, perfect honest verifier zero-knowledge and computational witness extended emulation.*

The proof for Theorem 2.3 is presented in Section 2.10. It mirrors the proof of Theorem 2.4,

which is described in greater detail in Section 2.11.

2.4.4 Non-Interactive Proof through Fiat-Shamir

So far we presented the proof as an interactive protocol with a logarithmic number of rounds. The verifier is a public coin verifier, as all the honest verifier's messages are random elements from $\mathbb{Z}_p^\star$. We can therefore convert the protocol into a non-interactive protocol that is secure and full zero-knowledge in the random oracle model using the Fiat-Shamir transform[BR93]. All random challenges are replaced by hashes of the transcript up to that point, including the statement itself. Subsequent works have shown that this approach is secure, even for multi-round protocols[Wik21; AFK21].

For example, one could set $y = \mathsf{H}(\mathsf{st}, A, S)$ and $z = \mathsf{H}(A, S, y)$, where st is the statement. For a range proof st would be $\{V, n\}$, and for a circuit proof it would be the description of the circuit. It is very important to include the statement st in the hash as otherwise an adversary can prove invalid statements, as shown by Dao et al.[Dao+23]. Since implementing Fiat-Shamir can be error-prone, we recommend using an established library to do so, such as Merlin[3], which was developed as part of an implementation of Bulletproofs in Rust[4].

To avoid a trusted setup we can use a hash function to generate the public parameters $\mathbf{G}, \mathbf{H}, G, H$ from a small seed. The hash function needs to map $\{0,1\}^*$ to $\mathbb{G}$, which can be built as in [BLS04]. This also makes it possible to provide random access to the public parameters. Alternatively, a common random string can be used.

2.4.5 A Simple MPC Protocol for Bulletproofs

In several of the applications described in Section 2.1.2, the prover could potentially consist of multiple parties who each want to generate a single range proof. For instance, multiple parties may want to create a single joined confidential transaction, where each party knows some of the inputs and outputs and needs to create range proofs for their known outputs. The joint transaction would not only be smaller than the sum of multiple transactions, it would also hide which inputs correspond to which outputs and provide some level of anonymity. These kinds of transactions are called CoinJoin transactions [Max13]. In Provisions, an exchange may distribute the private keys to multiple servers and split the customer database into separate chunks, but it still needs to produce a single short proof of solvency. Can these parties generate one Bulletproof without sharing the entire witness with each other? The parties could certainly use generic multi-party computation techniques to generate a single proof, but this might be too expensive and incur significant communication costs. This motivates the need for a simple MPC protocol specifically designed for Bulletproofs which requires little modification to the prover and is still efficient.

[3] `https://github.com/zkcrypto/merlin`
[4] `https://github.com/zkcrypto/bulletproofs`

Note that for aggregate range proofs, the inputs of one range proof do not affect the output of another range proof. Given the composable structure of Bulletproofs, it turns out that m parties each having a Pedersen commitment $(V_k)_{k=1}^m$ can generate a single Bulletproof that each V_k commits to a number in some fixed range. The protocol either uses a constant number of rounds but communication that is linear in both m and the binary encoding of the range, or it uses a logarithmic number of rounds and communication that is only linear in m. We assume for simplicity that m is a power of 2, but the protocol could be easily adapted for other m. We use the same notation as in the aggregate range proof protocol, but use k as an index to denote the kth party's message. That is $A^{(k)}$ is generated just like A but using only the inputs of party k.

The MPC protocol works as follows, we assign a set of distinct generators $(\mathbf{G}^{(k)}, \mathbf{H}^{(k)})_{k=1}^m$ to each party and define $\mathbf{G}$ as the interleaved concatenation of all $\mathbf{G}^{(k)}$ such that $g_i = g_{\lceil \frac{i}{m} \rceil}^{((i-1) \mod m+1)}$. Define $\mathbf{H}$ and $\mathbf{H}^{(k)}$ in an analogous way.

We first describe the protocol with linear communication. In each of the 3 rounds of the protocol, the ones that correspond to the rounds of the range proof in Section 2.4.1, each party simply generates its part of the proof, i.e. the $A^{(k)}, S^{(k)}; T_1^{(k)}, T_2^{(k)}; \tau_x^{(k)}, \mu^{(k)}, \hat{t}^{(k)}, \mathbf{l}^{(k)}, \mathbf{r}^{(k)}$ using its inputs and generators. These shares are then sent to a dealer (which could be one of the parties), who simply adds them homomorphically to generate the respective proof component, e.g. $A = \sum_{k=1}^l A^{(k)}$ and $\tau_x = \sum_{k=1}^l \tau_x^{(k)}$. In each round, the dealer generates the challenges using the Fiat-Shamir heuristic and the combined proof components and sends them to each party. Finally, each party sends $\mathbf{l}^{(k)}, \mathbf{r}^{(k)}$ to the dealer who computes $\mathbf{l}, \mathbf{r}$ as the interleaved concatenation of the shares. The dealer runs the inner product argument and generates the final proof. The protocol is complete as each proof component is simply the (homomorphic) sum of each parties' proof components, and the challenges are generated as in the original protocol. It is also secure against honest but curious adversaries as each share constitutes part of a separate zero-knowledge proof.

The communication can be reduced by running a second MPC protocol for the inner product argument. The generators were selected in such a way that up to the last $\log_2(l)$ rounds each parties' witnesses are independent and the overall witness is simply the interleaved concatenation of the parties' witnesses. Therefore, parties simply compute $L^{(k)}, R^{(k)}$ in each round and a dealer computes L, R as the homomorphic sum of the shares. The dealer then again generates the challenge and sends it to each party. In the final round the parties send their witness to the dealer who completes Protocol 2.2. A similar protocol can be used for arithmetic circuits if the circuit is decomposable into separate independent circuits. Constructing an efficient MPC protocol for more complicated circuits remains an open problem.

2.4.6 Perfectly Binding Commitments and Proofs

Bulletproofs, like the range proofs currently used in confidential transactions, are computationally binding. An adversary that could break the discrete logarithm assumption could generate acceptable

range proofs for a value outside the correct range. On the other hand, the commitments are perfectly hiding and Bulletproofs are perfect zero-knowledge, so that even an all powerful adversary cannot learn which value was committed to. Commitment schemes which are simultaneously perfectly-binding and perfectly-hiding commitments are impossible, so when designing commitment schemes and proof systems, we need to decide which properties are more important. For cryptocurrencies, the binding property is more important than the hiding property [Fcw]. An adversary that can break the binding property of the commitment scheme or the soundness of the proof system can generate coins out of thin air and thus create uncontrolled but undetectable inflation rendering the currency useless. Giving up the privacy of a transaction is much less harmful as the sender of the transaction or the owner of an account is harmed at worst. Unfortunately, it seems difficult to create Bulletproofs from binding commitments. The efficiency of the system relies on vector commitments which allow the commitment to a long vector in a single group element. By definition, for perfectly binding commitment schemes, the size of the commitment must be at least the size of the message and compression is thus impossible. The works [GSV98; GVW01] show that in general, interactive proofs cannot have communication costs smaller than the witness size, unless some very surprising results in complexity theory hold.

While the discrete logarithm assumption is believed to hold for classical computers, it does not hold against a quantum adversary. It is especially problematic that an adversary can create a perfectly hiding UTXO at any time, planning to open to an arbitrary value later when quantum computers are available. To defend against this, we can use the technique from Ruffing and Malavolta [Fcw] to ensure that even though the proof is only computationally binding, it is later possible to switch to a proof system that is perfectly binding and secure against quantum adversaries. In order to do this, the prover simply publishes $\gamma \cdot G$, which turns the Pedersen commitment to v into an ElGamal commitment. Ruffing and Malavolta also show that given a small message space, e.g. numbers in the range $[0, 2^n]$, it is impossible for a computationally bounded prover to construct a commitment that an unbounded adversary could open to a different message in the small message space.

Note that the commitment is now only computationally hiding, but that switching to quantum-secure range proofs is possible. Succinct quantum-secure range proofs remain an open problem, but with a slight modification, the scheme from Poelstra et al. [Poe+19] can achieve statistical soundness. Instead of using Pedersen commitments, we propose using ElGamal commitments in every step of the protocol. An ElGamal commitment is a Pedersen commitment with an additional commitment g^r to the randomness used. The scheme can be improved slightly if the same g^r is used in multiple range proofs. In order to retain the hiding property, a different h must be used for every proof.

2.5 Zero-Knowledge Proof for Arithmetic Circuits

Bootle et al. [Boo+16] present an efficient zero-knowledge argument for arbitrary arithmetic circuits using $6\log_2(n)+13$ elements, where n is the multiplicative complexity of the circuit. We can use our improved inner product argument to get a proof of size $2\log_2(n)+13$ elements, while simultaneously generalizing to include committed values as inputs to the arithmetic circuit. Including committed input wires is important for many applications (notably range proofs) as otherwise the circuit would need to implement a commitment algorithm. Concretely a statement about Pedersen commitments would need to implement the group exponentiation for the group that the commitment is an element of.

Following [Boo+16], we present a proof for a Hadamard-product relation. A multiplication gate of fan-in 2 has three wires; 'left' and 'right' for the input wires, and 'output' for the output wire. In the relation, $\mathbf{a}_L$ is the vector of left inputs for each multiplication gate. Similarly, $\mathbf{a}_R$ is the vector of right inputs, and $\mathbf{a}_O = \mathbf{a}_L \circ \mathbf{a}_R$ is the vector of outputs. [Boo+16] shows how to convert an arbitrary arithmetic circuit with n multiplication gates into a relation containing a Hadamard-product as above, with an additional $Q \leq 2 \cdot n$ linear constraints of the form

$$\langle \mathbf{w}_{L,q}, \mathbf{a}_L \rangle + \langle \mathbf{w}_{R,q}, \mathbf{a}_R \rangle + \langle \mathbf{w}_{O,q}, \mathbf{a}_O \rangle = c_q$$

for $1 \leq q \leq Q$, with $\mathbf{w}_{L,q}, \mathbf{w}_{R,q}, \mathbf{w}_{O,q} \in \mathbb{Z}_p^n$ and $c_q \in \mathbb{Z}_p$.

We include additional commitments V_i as part of our statement, and give a protocol for a more general relation, where the linear consistency constraints include the openings v_j of the commitments V_j. For simplicity and efficiency we present the scheme with V_i being Pedersen commitments. The scheme can be trivially adapted to work with other additively homomorphic schemes by changing the commitments to $t(X)$ and adapting the verification in line (2.50).

2.5.1 Inner-Product Proof for Arithmetic Circuits

The high level idea of the protocol is to convert the Hadamard-product relation along with the linear constraints into a single inner product relation. Similar to the range proof protocol the prover verifiably produces a random linear combination of the Hadamard and the linear constraints to form a single inner product constraint. If the combination is chosen randomly by the verifier, as in our protocol, then with overwhelming probability the inner-product constraint implies the other constraints.

In Section 2.5.2 we show that the inner product relation can be replaced with an efficient inner product argument which yields short proofs for arbitrary circuits where input wires can come from

Pedersen commitments. Formally we present a proof system for the following relation.

$$\{(G, H \in \mathbb{G}, \mathbf{G}, \mathbf{H} \in \mathbb{G}^n, \mathbf{V} \in \mathbb{G}^m, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}, \mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}, \mathbf{c} \in \mathbb{Z}_p^Q; \mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O \in \mathbb{Z}_p^n, \mathbf{v}, \boldsymbol{\gamma} \in \mathbb{Z}_p^m) : V_j = v_j \cdot G + \gamma_j \cdot H \, \forall j \in [1, m] \wedge \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \wedge \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c}\} \tag{2.46}$$

Let $\mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}$ be the weights for a commitment V_j. The presented proof system only works for relations where $\mathbf{W}_V$ is of rank m, i.e. the columns of the matrix are all linearly independent. This restriction is minor as we can construct commitments that fulfill these linearly dependent constraints as a homomorphic combination of other commitments. Consider a vector $\mathbf{w}'_V = \mathbf{a} \cdot \mathbf{W}_V \in \mathbb{Z}_p^m$ for a vector of scalars $\mathbf{a} \in \mathbb{Z}_p^Q$ then we can construct commitment $V' = \langle \mathbf{a} \cdot \mathbf{W}_V, \mathbf{V} \rangle$. Note that if the relation holds, then we can conclude that $\langle \mathbf{w}_{L,j}, \mathbf{a}_L \rangle + \langle \mathbf{w}_{R,j}, \mathbf{a}_R \rangle + \langle \mathbf{w}_{O,j}, \mathbf{a}_O \rangle = \langle \mathbf{w}'_V, \mathbf{v} \rangle + \mathbf{c}$. The protocol is presented in Protocol 2.3. It is split into two parts. In the first part P commits to $l(X), r(X), t(X)$ in the second part P convinces V that the polynomials are well formed and that $\langle l(X), r(X) \rangle = t(X)$.

Theorem 2.4. *The proof system presented in Protocol 2.3 has perfect completeness, perfect honest verifier zero-knowledge and computational witness extended emulation.*

The proof of Theorem 2.4 is presented in Section 2.11.

2.5.2 Logarithmic-Sized Protocol

As for the range proof, we can reduce the communication cost of the protocol by using the inner product argument. Concretely transfer (2.47) is altered to simply $\tau_x, \mu, \hat{t}$ and additionally P and V engage in an inner product argument on public input $(\mathbf{G}, \mathbf{H}', P - \mu \cdot H, \hat{t})$. Note that the statement proven is equivalent to the verification equations (2.52) and (2.49). The inner product argument has only logarithmic communication complexity and is thus highly efficient. Note that instead of transmitting $\mathbf{l}, \mathbf{r}$ the inner product argument only requires communication of $2 \cdot \lceil \log_2(n) \rceil + 2$ elements instead of $2 \cdot n$. In total the prover sends $2 \cdot \lceil \log_2(n) \rceil + 8$ group elements and 5 elements in $\mathbb{Z}_p$. Using the Fiat-Shamir heuristic as in 2.4.4 the protocol can be turned into an efficient non interactive proof. We report implementation details and evaluations in Section 2.7.

Theorem 2.5. *The arithmetic circuit protocol using the improved inner product argument (Protocol 2.2) has perfect completeness, statistical zero-knowledge and computational soundness under the discrete logarithm assumption.*

Proof. Completeness follows from the completeness of the underlying protocols. Zero-knowledge follows from the fact that $\mathbf{l}$ and $\mathbf{r}$ can be efficiently simulated, and because the simulator can simply run Protocol 2.2 given the simulated witness $(\mathbf{l}, \mathbf{r})$. The protocol also has a knowledge-extractor, as the extractor of the range proof can be extended to extract $\mathbf{l}$ and $\mathbf{r}$ by calling the extractor of Protocol 2.2. The extractor uses $O(n^3)$ valid transcripts in total, which is polynomial in λ if

Input: $(G, H \in \mathbb{G}, \mathbf{G}, \mathbf{H} \in \mathbb{G}^n; \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n},$
$\mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}, \mathbf{c} \in \mathbb{Z}_p^Q; \mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O \in \mathbb{Z}_p^n, \boldsymbol{\gamma} \in \mathbb{Z}_p^m)$
P's input: $(G, H, \mathbf{G}, \mathbf{H}; \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \mathbf{W}_V, \mathbf{c}; \mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O, \boldsymbol{\gamma})$
V's input: $(G, H, \mathbf{G}, \mathbf{H}; \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \mathbf{W}_V, \mathbf{c})$
Output: {V accepts,V rejects }
P computes:
$\alpha, \beta, \rho \leftarrow\!\!\$\ \mathbb{Z}_p$
$A_I = \alpha \cdot H + \langle \mathbf{a}_L, \mathbf{G} \rangle + \langle \mathbf{a}_R, \mathbf{H} \rangle \in \mathbb{G}$ // *commit to* $\mathbf{a}_L, \mathbf{a}_R$
$A_O = \beta \cdot H + \langle \mathbf{a}_O, \mathbf{G} \rangle \in \mathbb{G}$ // *commitment to* $\mathbf{a}_O$
$\mathbf{s}_L, \mathbf{s}_R \leftarrow\!\!\$\ \mathbb{Z}_p^n$ // *choose blinding vectors* $\mathbf{s}_L, \mathbf{s}_R$
$S = \rho \cdot H + \langle \mathbf{s}_L, \mathbf{G} \rangle + \langle \mathbf{s}_R, \mathbf{H} \rangle \in \mathbb{G}$ // *commitment to* $\mathbf{s}_L, \mathbf{s}_R$
P $\rightarrow$ V : A_I, A_O, S
V : $y, z \leftarrow\!\!\$\ \mathbb{Z}_p^\star$
V $\rightarrow$ P : y, z
P and V compute:
$\mathbf{y}^n = (1, y, y^2, \ldots, y^{n-1}) \in \mathbb{Z}_p^n$ // *challenge per witness*
$\mathbf{z}_{[1:]}^{Q+1} = (z, z^2, \ldots, z^Q) \in \mathbb{Z}_p^Q$ // *challenge per constraint*
$\delta(y, z) = \langle \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R), \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L \rangle$ // *independent of the witness*
P computes:
$l(X) = \mathbf{a}_L \cdot X + \mathbf{a}_O \cdot X^2 + \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R) \cdot X$
$+ \mathbf{s}_L \cdot X^3 \in \mathbb{Z}_p^n[X]$
$r(X) = \mathbf{y}^n \circ \mathbf{a}_R \cdot X - \mathbf{y}^n + \mathbf{z}_{[1:]}^{Q+1} \cdot (\mathbf{W}_L \cdot X + \mathbf{W}_O)$
$+ \mathbf{y}^n \circ \mathbf{s}_R \cdot X^3 \in \mathbb{Z}_p^n[X]$
$t(X) = \langle l(X), r(X) \rangle = \sum_{i=1}^{6} t_i \cdot X^i \in \mathbb{Z}_p[X]$
$\mathbf{w} = \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O$
$t_2 = \langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{y}^n \rangle - \langle \mathbf{a}_O, \mathbf{y}^n \rangle + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{w} \rangle + \delta(y, z) \in \mathbb{Z}_p$ // $t_2 = d(y,z) + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} + \mathbf{W}_V \cdot \mathbf{v} \rangle$
$\tau_i \leftarrow\!\!\$\ \mathbb{Z}_p \quad \forall i \in [1, 3, 4, 5, 6]$
$T_i = t_i \cdot G + \tau_i \cdot H \quad \forall i \in [1, 3, 4, 5, 6]$
P $\rightarrow$ V : T_1, T_3, T_4, T_5, T_6 // *commitments to* t_1, t_3, t_4, t_5, t_6

Protocol 2.3: Part 1: Computing commitments to $l(X), r(X)$ and $t(X)$

$\mathsf{V} : x \leftarrow\!\!\$\ \mathbb{Z}_p^*$ // *Random challenge*

$\mathsf{V} \rightarrow \mathsf{P} : x$

P computes:

$\mathbf{l} = l(x) \in \mathbb{Z}_p^n$

$\mathbf{r} = r(x) \in \mathbb{Z}_p^n$

$\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p$

$\tau_x = \sum_{i=1, i\neq 2}^{6} \tau_i \cdot x^i + x^2 \cdot \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \boldsymbol{\gamma} \rangle \in \mathbb{Z}_p$ // *blinding value for* $\hat{t}$

$\mu = \alpha \cdot x + \beta \cdot x^2 + \rho \cdot x^3 \in \mathbb{Z}_p$ // *Blinding value for* P

$\mathsf{P} \rightarrow \mathsf{V} : \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r}$ (2.47)

V computes and checks:

$H_i' = y^{-i+1} \cdot H_i \quad \forall i \in [1, n]$ // $\mathbf{H}' = \langle \mathbf{y}^{-\mathbf{1}^n} n, \mathbf{H} \rangle$

$W_L = \langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L, \mathbf{H}' \rangle$ // *Weights for* $\mathbf{a}_L$

$W_R = \langle \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R), \mathbf{G} \rangle$ // *Weights for* $\mathbf{a}_R$ (2.48)

$W_O = \langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_O, \mathbf{H}' \rangle$ // *Weights for* $\mathbf{a}_O$

$\hat{t} \stackrel{?}{=} \langle \mathbf{l}, \mathbf{r} \rangle$ // *Check that* $\hat{t}$ *is correct* (2.49)

$\hat{t} \cdot G + \tau_x \cdot H \stackrel{?}{=} x^2 \cdot (\delta(y, z) + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} \rangle) \cdot G$

$+ \langle x^2 \cdot (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V), \mathbf{V} \rangle + x \cdot T_1 + \sum_{i=3}^{6} x^i \cdot T_i$ // $\hat{t} = t(x) = \sum_{i=1}^{6} t_i \cdot x^i$ (2.50)

$P = x \cdot A_I + (x^2) \cdot A_O - \langle \mathbf{y}^n, \mathbf{H}' \rangle + x \cdot W_L$

$+ x \cdot W_R + W_O + (x^3) \cdot S$ // *commitment to* $l(x), r(x)$ (2.51)

$P \stackrel{?}{=} \mu \cdot H + \langle \mathbf{l}, \mathbf{G} \rangle + \langle \mathbf{r}, \mathbf{H}' \rangle$ // *Check that* $\mathbf{l} = l(x)$ *and* $\mathbf{r} = r(x)$ (2.52)

if all checks succeed: V accepts

else: V rejects

Protocol 2.3: Part 2: Polynomial identity check for $\langle l(x), r(x) \rangle = t(x)$

$n = O(\lambda)$. The extractor is thus efficient and either extracts a discrete logarithm relation or a valid witness. However, if the generators $\mathbf{G}, \mathbf{H}, G, H$ are independently generated, then finding a discrete logarithm relation between them is as hard as breaking the discrete log problem. If the discrete log assumption holds in $\mathbb{G}$ then a computationally bounded P cannot produce discrete-logarithm relations between independent generators. The proof system is therefore computationally sound. □

2.6 Bulletproofs for R1CS with committed witness

In Section 2.5, we introduce an argument for arithmetic circuits using a Hadamard relation. This argument enables proving properties about Pedersen committed values. However, it does not support proving statements about values inside a vector commitment. Also, the characterization of NP is slightly less standard than the commonly use R1CS representation[Gen+13; Chi+20], which enjoys wide library support[con22]. Additionally a recent result showed that proof systems for R1CS can be transformed into proof systems for higher degree statements, through a generalization called CCS[STW23].

We, therefore, will use the succinct argument of knowledge Π_{ipa} to build a succinct argument of knowledge for the following R1CS relation, where a prefix of the witness is committed in a vector commitment T. Formally we define this prefixed R1CS for some parameters $n, m, r \in \mathbb{N}$:

$$\begin{aligned}\mathcal{R}_{\mathrm{R1CS}} := \Big\{ (A, B, C, T) \; ; \; (\mathbf{x}, \boldsymbol{a}) \Big\} \text{ where} \\ (1) \quad A, B, C \in \mathbb{Z}_p^{m \times n}, \quad T \in \mathbb{G}, \quad \mathbf{x} \in \mathbb{Z}_p^r, \quad \boldsymbol{a} \in \mathbb{Z}_p^{n-r}, \quad \mathbf{z} := (\mathbf{x}, \boldsymbol{a}) \in \mathbb{Z}_p^n, \\ (2) \quad (A\mathbf{z}) \circ (B\mathbf{z}) = (C\mathbf{z}), \\ (3) \quad T = \left(\!\!\left(\begin{bmatrix} \mathbf{x} \parallel \mathbf{u} \\ \mathbf{v} \end{bmatrix} \right)\!\!\right) \quad \text{(Note that } \mathbf{u} \in \mathbb{Z}_p^{n-r} \text{ and } \mathbf{v} \in \mathbb{Z}_p^n \text{ can be set to 0..}\end{aligned} \tag{2.53}$$

where $P \leftarrow \left(\!\!\left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \right)\!\!\right) \in \mathbb{G}$ is defined such that $P = \langle \mathbf{a}, \mathbf{G} \rangle + \langle \mathbf{b}, \mathbf{G} \rangle$. We also make use of the following homomorphic operations.

Additive Homomorphism For $P \leftarrow \left(\!\!\left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \right)\!\!\right)$ and $P' \leftarrow \left(\!\!\left(\begin{bmatrix} \mathbf{a}' \\ \mathbf{b}' \end{bmatrix} \right)\!\!\right)$, let $P + P' = \left(\!\!\left(\begin{bmatrix} \mathbf{a} + \mathbf{a}' \\ \mathbf{b} + \mathbf{b}' \end{bmatrix} \right)\!\!\right) = \langle \mathbf{a} + \mathbf{a}', \mathbf{G} \rangle + \langle \mathbf{b} + \mathbf{b}', \mathbf{G} \rangle$

Generator Transformation We also enable multiplying the committed values by constants, through re-defining the generators. This is indicated thorugh multiplication by a matrix $M = \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix}$ with non-zero entries. The transformation is denoted by $M \circ P := \left(\!\!\left(\begin{bmatrix} \mathbf{a} \circ \mathbf{y} \\ \mathbf{b} \circ \mathbf{z} \end{bmatrix} \right)\!\!\right) = \langle \mathbf{a} \circ \mathbf{y}, \mathbf{G}' \rangle + \mathbf{b} \circ \mathbf{z}\mathbf{H}'$, such that $\mathbf{G}' = \mathbf{y}^-1 \circ \mathbf{G}$, i.e. $G'_i = y_i^{-1} \cdot G_i$ and analogously for H_i.

Setup: The prover and verifier have an $\mathcal{R}_{\mathrm{R1}CS}$ statement (A, B, C, T) and the commitment key $\mathbf{G}, \mathbf{H} \in \mathbb{G}^{n+m}$ and $u \in \mathbb{G}$. The prover has a witness $\mathbf{z} := (\mathbf{x}, \boldsymbol{a}) \in \mathbb{Z}_p^n$.

- *step 1:* The prover sends to the verifier

$$S' \leftarrow \left(\!\!\left(\begin{bmatrix} (0^r, \boldsymbol{a}) & \| & A\mathbf{z} \\ 0^n & \| & B\mathbf{z} \end{bmatrix}\right)\!\!\right) \in \mathbb{G}$$

- *step 2:* The verifier samples $\alpha, \beta, \gamma, \delta \leftarrow\!\!\$\ \mathbb{Z}_p$ and sends them to the prover.
- *step 3:* Both the prover and verifier locally compute

$$\mu \leftarrow \alpha\gamma \in \mathbb{Z}_p, w \leftarrow \langle \boldsymbol{\alpha}^m, \boldsymbol{\beta}^m \rangle + \delta^2 \cdot \langle \boldsymbol{\alpha}^n, \mathbf{c} \circ \delta \rangle$$

$$\mathbf{c} \leftarrow \boldsymbol{\mu}^m A + \boldsymbol{\beta}^m B - \boldsymbol{\gamma}^m C \in \mathbb{Z}_p^n, \qquad \text{(encodes the R1CS program)}$$

$$\boldsymbol{\delta} \leftarrow (\delta, \ldots, \delta, 1^{n-r}) \in \mathbb{Z}_p^n, \qquad \boldsymbol{\delta}^{-1} \leftarrow (1/\delta, \ldots, 1/\delta, 1^{n-r}) \in \mathbb{Z}_p^n,$$

$$S'' \leftarrow D \circ ((1/\delta) \cdot T + S') = (1/\delta) \cdot T + D \circ S' = \left(\!\!\left(\begin{bmatrix} \mathbf{z} \circ \boldsymbol{\delta}^{-1} & \| & \boldsymbol{\gamma}^n \circ A\mathbf{z} \\ 0^n & \| & B\mathbf{z} \end{bmatrix}\right)\!\!\right) \in \mathbb{G},$$

where $D := \begin{pmatrix} \mathbf{1} & \boldsymbol{\gamma}^m \\ \mathbf{1} & \mathbf{1} \end{pmatrix}$

$$P \leftarrow S'' + \left(\!\!\left(\begin{bmatrix} \delta^2 \cdot \boldsymbol{\alpha}^n & \| & -\boldsymbol{\beta}^m \\ \mathbf{c} \circ \boldsymbol{\delta} & \| & -\boldsymbol{\alpha}^m \end{bmatrix}\right)\!\!\right) \in \mathbb{G}.$$

The role of $\delta \in \mathbb{Z}_p$ is to defend against spurious non-zero entries in T and S'.

- *step 4:* The prover computes

$$\begin{aligned} \mathbf{u} &\leftarrow (\ \mathbf{z} \circ \boldsymbol{\delta}^{-1} + \delta^2 \cdot \boldsymbol{\alpha}^n, \quad (\boldsymbol{\gamma}^m \circ A\mathbf{z}) - \boldsymbol{\beta}^m \quad) \in \mathbb{Z}_p^{n+m}, \\ \mathbf{v} &\leftarrow (\ \mathbf{c} \circ \boldsymbol{\delta}, \qquad\qquad B\mathbf{z} - \boldsymbol{\alpha}^m \quad) \in \mathbb{Z}_p^{n+m}. \end{aligned}$$

If $\mathbf{z} \in \mathbb{Z}_p^n$ is a valid witness then $P = \left(\!\!\left(\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}\right)\!\!\right)$ and $\langle \mathbf{u}, \mathbf{v} \rangle = w$.

That is, $(\mathbf{u}, \mathbf{v})$ is a valid witness for the $\mathcal{R}_{\mathsf{ipa}}$ statement (P, w).

- *step 5:* The prover and verifier compute $\mathbf{H}' \in \mathbb{G}^{n+m}$ such that,

$$H'_i = H \forall i \in [1, n] \wedge H'_{n+j} = \gamma^{-j} H_{n+j} \forall j \in [1, m]$$

and compute $\mathsf{pp} = (\mathbf{G} \in \mathbb{G}^{n+m}, \mathbf{H}' \in \mathbb{G}^{n+m})$. Then they run Protocol 2.1 on input $(\mathsf{pp}; P, w; \mathbf{u}, \mathbf{v})$

The proof system requires committing to two vectors of length $n+m$ and the prover sends just one commitment before the inner-product argument. Thus, the resulting communication complexity is just $2\log_2(n+m)+3$ elements.

Note that the proof system itself is not zero-knowledge. It can be made zero-knowledge using hiding commitments and the zero-knowledge variant of the inner product argument, presented in [Chu+20].

Theorem 2.6. *The proof system presented in this section has perfect completeness and knowledge-soundness for $\mathcal{R}_{R1CS}$ against polynomial time adversaries under the discrete logarithm assumption.*

Proof. For completeness observe that

$$\begin{aligned}\langle \mathbf{u}, \mathbf{v}\rangle &= \langle \boldsymbol{\delta}^{-1} \circ \mathbf{z} + \delta^2 \boldsymbol{\alpha}^n, \boldsymbol{\delta} \circ (\boldsymbol{\mu}^m A + \boldsymbol{\beta}^m B - \boldsymbol{\gamma}^m C)\rangle + \langle \boldsymbol{\gamma}^m \circ Az - \boldsymbol{\beta}^m, B\mathbf{z} - \boldsymbol{\alpha}^m\rangle \\ &= \boldsymbol{\mu}^m A\mathbf{z} + \boldsymbol{\beta}^m B\mathbf{z} - \boldsymbol{\gamma}^m C\mathbf{z} + \boldsymbol{\gamma}^m(A\mathbf{z} \circ B\mathbf{z}) - \boldsymbol{\alpha}^m A\mathbf{z} - \boldsymbol{\beta}^m B\mathbf{z} + \langle \boldsymbol{\alpha}^m, \boldsymbol{\beta}^m\rangle + \langle \boldsymbol{\alpha}^n, \delta^2\boldsymbol{\delta} \circ \mathbf{c}\rangle \\ &= \langle \boldsymbol{\alpha}^m, \boldsymbol{\beta}^m\rangle + \delta^2\langle \boldsymbol{\alpha}^n, \boldsymbol{\delta} \circ \mathbf{c}\rangle\end{aligned}$$

The last equation holds if $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z}$ which is precisely the requirement for a valid witness.

To show that the protocol has witness-extended emulation, we show that it has $(\max(n+1, m+1), m+1, m+1, 6, 3^{\log_2(n+m)})$- special soundness (see Definiton 5.3) for the relation $\mathcal{R}'_{\mathrm{R1}CS}(x) = (x, \mathbf{w}) \in \mathcal{R}_{\mathrm{R1}CS} \vee (\mathsf{pp}, \mathbf{w}) \in \mathcal{R}_{\mathsf{dlog}}$, where $\mathcal{R}_{\mathsf{dlog}}$ is the relation that describes non-trivial discrete logarithm relation for the generators in pp. The first 4 levels of the transcript tree correspond to α, β, γ and δ respectively, whereas the last $\log_2(n+m)$ correspond to the inner product argument. In order to show that the protocol has special soundness, we show that we can construct an extractor Ext that computes a witness from any appropriately sized transcript trees. Ext first computes witnesses for the inner product relation $\mathcal{R}_{\mathsf{ipa}}$ using the lower $\log_2(n+m)$ levels. This yields a $(\max(n+1, m+1), m+1, m+1, 6)$-tree where each leaf ℓ is labeled with a commitment $P^{(\ell)} \in \mathbb{G}$ a scalar $w^{(\ell)} \in \mathbb{F}_p$ and vectors $\mathbf{v}^{(\ell)}, \mathbf{u}^{(\ell)}$, such that $P^{(\ell)} = \left(\!\left(\begin{bmatrix}\mathbf{v}^{(\ell)} \\ \mathbf{u}^{(\ell)}\end{bmatrix}\right)\!\right)$ and $\langle \mathbf{v}^{(\ell)}, \mathbf{u}^{(\ell)}\rangle = w^{(\ell)}$. $P^{(\ell)}$ and $w^{(\ell)}$ are derived from the transcript, where as $\mathbf{u}^{(\ell)}$ and $\mathbf{v}^{(\ell)}$ are computed by calling the inner-product-argument's extractor. From the verification equation, we can write.

$$S' + \delta^{-1}T = (P - \left(\!\left(\begin{bmatrix}\delta^2\boldsymbol{\alpha}^n & \| & -\boldsymbol{\beta}^m \\ \mathbf{c} \circ \boldsymbol{\delta} & \| & -\boldsymbol{\alpha}^m\end{bmatrix}\right)\!\right)) \circ D^{-1}$$

Now given two transcripts labled with distinct challenges δ and δ' we can extract openings for S' and T, such that

$$S' = \left(\!\left(\begin{bmatrix}s_1 \in \mathbb{F}^r || s_2 \in \mathbb{F}^{n-r} || s_3 \in \mathbb{F}^m \\ s'_1 \in \mathbb{F}^r || s'_2 \in \mathbb{F}^{n-r} || s'_3 \in \mathbb{F}^m\end{bmatrix}\right)\!\right)$$

and

$$T = \left\langle\!\!\left\langle \begin{bmatrix} t_1 \in \mathbb{F}^r || t_2 \in \mathbb{F}^{n-r} || t_3 \in \mathbb{F}^m \\ t_1' \in \mathbb{F}^r || t_2' \in \mathbb{F}^{n-r} || t_3' \in \mathbb{F}^m \end{bmatrix} \right\rangle\!\!\right\rangle$$

If for any transcript leaf ℓ these openings are not consistent with the extracted $\mathbf{v}_\ell$, $\mathbf{u}_\ell$, then we can compute break of the binding property of the commitment, i.e. a non-trivial discrete logarithm between the generators. This happens with negligible probability by assumption. Otherwise, given these witnesses we get that

$$\mathbf{u} = t_1 \cdot \delta^{-1} + s_1 + \delta^2 \boldsymbol{\alpha}^r || t_2 \cdot \delta^{-1} + s_2 + \delta^2 \boldsymbol{\alpha}^{n-r} \cdot \alpha^r || (\delta^{-1} t_3 + s_3) \cdot \boldsymbol{\gamma}^m - \boldsymbol{\beta}^m$$

and

$$\mathbf{v} = t_1' \cdot \delta^{-1} + s_1' + \mathbf{c}_{[1:r]} \cdot \delta || t_2' \cdot \delta^{-1} + s_2' + \mathbf{c}_{[r+1,n]} || t_3' \cdot \delta^{-1} + s_3' - \boldsymbol{\alpha}^m$$

, for each leaf ℓ with $w = \langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle$. This implies that

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{j=-2}^{3} \delta^j d_j = \langle \boldsymbol{\alpha}^m, \boldsymbol{\beta}^m \rangle + \delta^2 \langle \boldsymbol{\alpha}^n, \boldsymbol{\delta} \circ \mathbf{c} \rangle \tag{2.54}$$

Focussing on the constant coefficient in δ we get that

$$d_0 = \langle t_1, \mathbf{c}_{[1:r]} \rangle + \langle s_1, s_1' \rangle + \langle s_2, s_2' + \mathbf{c}[r+1, n] \rangle + \langle \boldsymbol{\gamma}^m \circ s_3, s_3' \rangle - \langle \boldsymbol{\alpha}^m, \boldsymbol{\gamma}^m \circ s_3 \rangle - \langle \boldsymbol{\beta}^m, s_3' \rangle$$

and

$$d_2 = \langle \boldsymbol{\alpha}^n, s_1' || s_2' \rangle + \langle \boldsymbol{\alpha}^{n-r} \cdot \alpha^r, \mathbf{c}_{[r+1,n]} \rangle$$

Since for any given α, β, γ (2.54) holds for 6 distinct δ, we have that $d_0 = \langle \boldsymbol{\alpha}^m, \boldsymbol{\beta}^m \rangle$ and $d_2 = \langle \boldsymbol{\alpha}^{n-r} \cdot \alpha^r, \mathbf{c}_{[r+1,n]} \rangle$. This implies that $\langle \boldsymbol{\alpha}^n, s_1' || s_2' \rangle = 0$. Since this holds for $n+1$ different values of α we have that $s_1' || s_2' = 0^n$. Plugging this in we get that

$$d_2 = \langle t_1 || s_2, \mathbf{c} \rangle + \langle \boldsymbol{\gamma}^m \circ s_3, s_3' \rangle - \langle \boldsymbol{\alpha}^m, \boldsymbol{\gamma}^m \circ s_3 \rangle - \langle \boldsymbol{\beta}^m, s_3' \rangle = \langle \boldsymbol{\alpha}^m, \boldsymbol{\beta}^m \rangle$$

Expanding $\mathbf{c} = \boldsymbol{\mu}^m A + \boldsymbol{\beta}^m B - \boldsymbol{\gamma}^m C$, and using that $\mu = \alpha\beta$ we get that,

$$\langle t_1 || s_2, \boldsymbol{\mu}^m A \rangle - \langle \boldsymbol{\mu}^m, s_3 \rangle + \langle t_1 || s_2, \boldsymbol{\beta}^m B \rangle - \langle \boldsymbol{\beta}^m, s_3' \rangle - \langle t_1 || s_2, \boldsymbol{\gamma}^m C \rangle + \langle \boldsymbol{\gamma}^m s_3, s_3' \rangle = 0 \tag{2.55}$$

By construction of the transcript tree (2.55) holds for $m+1$ values of α, β, γ. Given that (2.55)

is a tri-variate degree m polynomial in α, β, γ the equality must hold everywhere and we get that,

$$A(t_1||s_2) = s_3$$
$$B(t_1||s_2) = s'_3$$
$$C(t_1||s_2) = s_3 \circ s'_3$$

This shows that $\mathbf{z} = (t_1||s_2)$ is a valid witness. As the transcript tree size, $\max(m+1, n+1) \cdot (m+1)^3 \cdot 3^{\log_2(n+m)}$, is polynomial in λ we can use Lemma 3 [AC20] to show that the protocol has knowledge soundness, with negligible knowledge error. □

2.7 Performance

2.7.1 Theoretical Performance

In Table 2.1 we give analytical measurements for the proof size of different range proof protocols. We compare both the proof sizes for a single proof and for m proofs for the range $[0, 2^n - 1]$. We compare Bulletproofs against [Poe+19] and a Σ-protocol range proof where the proof commits to each bit and then shows that the commitment is to 0 or 1. The table shows that Bulletproofs have a

Table 2.1: Range proof size for m proofs. $m = 1$ is the special case of a single range proof

m range proofs for range $[0, 2^n - 1]$		
	# $\mathbb{G}$ elements	# $\mathbb{Z}_p$ elements
Σ Protocol [CD98]	mn	$3mn + 1$
Poelstra et al. [Poe+19]	$0.63 \cdot mn$	$1.26 \cdot mn + 1$
Bulletproofs	$2\big(\log_2(n) + \log_2(m)\big) + 4$	5

significant advantage when providing multiple range proofs at once. The proof size for the protocol presented in Section 2.4.3 only grows by an additive logarithmic factor when conducting m range proofs, while all other solutions grow multiplicatively in m.

2.7.2 An Optimized Verifier Using Multi-Exponentiation and Batch Verification

In many of the applications discussed in Section 2.1.2 the verifier's runtime is of particular interest. For example, with confidential transactions every full node needs to check all confidential transactions and all associated range proofs. We therefore now present a number of optimizations for the non-interactive verifier. We present the optimizations for a single range proof but they all carry over to aggregate range proofs and the arithmetic circuit protocol.

Single multi-exponentiation. In Section 2.3.1 we showed that the verification of the inner product can be reduce to a single multi-exponentiation. We can further extend this idea to verify the whole range proof using a single multi-exponentiation of size $2n + 2\log_2(n) + 7$. Notice that the Bulletproofs verifier only performs two checks (2.41) and (Protocol 2.2, line 2). The idea is to delay exponentiation until those checks are actually performed and then to combine them into a single check. We, therefore, unroll the inner product argument as described in Section 2.3.1 using the input from the range proof. The resulting protocol is presented below with x_u being the challenge from Protocol 2.1, and x_j being the challenge from round j of Protocol 2.2. L_j and R_j are the L, R values from round j of Protocol 2.2. The verifier runs the following verification procedure:

$$\text{input: proof } \pi = \left\{A, S, T_1, T_2, (L_j, R_j)_{j=1}^{\log(n)} \in \mathbb{G}, \quad \tau, \hat{t}, \mu, a, b \in \mathbb{Z}_p\right\} \tag{2.56}$$

$$\text{compute challenges from } \pi: \ \{y, z, x, x_u, (x_j)_{j=1}^{\log_2(n)}\} \tag{2.57}$$

$$\delta(y,z) = (z - z^2) \cdot \langle \mathbf{1}^n, \mathbf{y}^n \rangle - z^3 \langle \mathbf{1}^n, \mathbf{2}^n \rangle \tag{2.58}$$

$$(\hat{t} - \delta(y,z)) \cdot G + \tau_x \cdot H - z^2 \cdot V - x \cdot T_1 - x^2 \cdot T_2 \stackrel{?}{=} \mathbf{0}_{\mathbb{G}} \in \mathbb{G} \tag{2.59}$$

$$b(i,j) = \begin{cases} 1 & \text{if the } j\text{th bit of } i-1 \text{ is 1} \\ -1 & \text{otherwise} \end{cases} \tag{2.60}$$

$$\text{for } i = 1, \ldots, n: \tag{2.61}$$

$$l_i = \prod_{j=1}^{\log_2 n} x_j^{b(i,j)} \cdot a + z \quad \in \mathbb{Z}_p, \quad r_i = y^{1-i} \cdot \left(\prod_{j=1}^{\log_2 n} x_j^{-b(i,j)} \cdot b - z^2 \cdot 2^{i-1}\right) - z \quad \in \mathbb{Z}_p \tag{2.62}$$

$$\mathbf{l} = (l_1, \ldots, l_n) \in \mathbb{Z}_p^n \tag{2.63}$$

$$\mathbf{r} = (r_1, \ldots, r_n) \in \mathbb{Z}_p^n \tag{2.64}$$

$$\langle \mathbf{l}, \mathbf{G} \rangle + \langle \mathbf{r}, \mathbf{H} \rangle + x_u \cdot (a \cdot b - \hat{t}) \cdot G + \mu \cdot H - A - x \cdot S \tag{2.65}$$

$$- \sum_{j=1}^{\log_2(n)} x_j^2 \cdot L_j - \sum_{j=1}^{\log_2(n)} x_j^{-2} \cdot R_j \stackrel{?}{=} \mathbf{0}_{\mathbb{G}} \in \mathbb{G} \tag{2.66}$$

Here $\mathbf{0}_{\mathbb{G}}$ is the identity in $\mathbb{G}$. We can combine the two multi-exponentiations in line (2.59) and (2.66) by using a random value $c \leftarrow\!\!\$\ \mathbb{Z}_p$. This is because if $c \cdot A + B = \mathbf{0}_{\mathbb{G}} \in \mathbb{G}$ for a random c then with high probability $A = 1 \wedge B = 1$.

Various algorithms are known to compute the multi-exponentiations (2.66) and (2.59) efficiently. As explained in [Ber+12], algorithms like Pippenger's [Pip80] perform a number of group operations that scales with $O\frac{n}{\log(n)}$, i.e. sub-linearly. For realistic problem sizes these dominate verification time.

Computing scalars. A further optimization concerns the computation of the l_i and r_i values. Instead of computing $x^{(i)} = \prod_{j=1}^{\log_2 n} x_j^{b(i,j)}$ for each i, we can compute each challenge product using

only one multiplication in $\mathbb{Z}_p$ by applying batch division. First we compute $x^{(1)} = (\prod_{j=1}^{\log_2 n} x_j)^{-1}$ to get the first challenge value using a single inversion. Then computing $x^{(2)} = x^{(1)}x_1^2, \quad x^3 = x^{(1)}x_2^2$, and for example $x^{(7)} = x^{(3)}x_5^2$. In general in order to compute $x^{(i)}$ we let k be the next lower power of 2 of $i-1$ and compute $x^{(i)} = x^{(i-k)} \cdot x_{k+1}^2$ which takes only one additional multiplication in $\mathbb{Z}_p$ and no inversion. Further, note that the squares of the challenges are computed anyway in order to check equation (2.66).

Batch verification. A further important optimization concerns the verification of multiple proofs. In many applications described in Section 2.1.2 the verifier needs to verify multiple (separate) range proofs at once. For example a Bitcoin node receiving a block of transactions needs to verify all transactions and thus range proofs in parallel. As noted above, verification boils down to a large multi-exponentiation. In fact, $2n+2$ of the generators only depend on the public parameters, and only $2\log(n)+5$ are proof-dependent. We can therefore apply batch verification[BGR98] in order to reduce the number of expensive exponentiations. Batch verification is based on the observation that checking $x \cdot G = \mathbf{0}_{\mathbb{G}} \wedge y \cdot G = \mathbf{0}_{\mathbb{G}}$ can be checked by drawing a random scalar α from a large enough domain and checking $(\alpha x + y) \cdot G = \mathbf{0}_{\mathbb{G}}$. With high probability, the latter equation implies that $x \cdot G = \mathbf{0}_{\mathbb{G}} \wedge y \cdot G = \mathbf{0}_{\mathbb{G}}$, but the latter is more efficient to check. The same trick applies to multi-exponentiations and can save $2n$ exponentiations per additional proof. This is equivalent to the trick that is used for combining multiple exponentiations into one with the difference that the bases are equivalent. Verifying m distinct range proofs of size n now only requires a single multi-exponentiation of size $2n + 2 + m \cdot (2 \cdot \log(n) + 5)$ along with $O(m \cdot n)$ scalar operations.

Note that this optimization can even be applied for circuits and proofs for different circuits if the same public parameter are used.

Even for a single verification we can take advantage of the fact that most generators are fixed in the public parameters. Both the verifier and the prover can used fast fixed-base exponentiation with precomputation [Gor98] to speed-up all the multi-exponentiations.

2.7.3 Implementation and Performance

To evaluate the performance of Bulletproofs in practice we give a reference implementation in C and integrate it into the popular library libsecp256k1 which is used in many cryptocurrency clients. libsecp256k1 uses the elliptic curve secp256k1[5] which has 128 bit security.

In their compressed form, secp256k1 points can be stored as 32 bytes plus one bit. We use all of the optimizations described above, except the pre-computation of generators. The prover uses constant time operations until the computation of $\mathbf{l}$ and $\mathbf{r}$. By Theorem 2.2, the inner product argument does not need to hide $\mathbf{l}$ and $\mathbf{r}$ and can therefore use variable time operations. The verifier has no secrets and can therefore safely use variable time operations like the multi-exponentiations.

[5] `http://www.secg.org/SEC2-Ver-1.0.pdf`

All experiments were performed on an Intel i7-6820HQ system throttled to 2.00 GHz and using a single thread. Less than 100 MB of memory was used in all experiments. For reference, verifying an ECDSA signature takes 86 μs on the same system. Table 2.2 shows that in terms of proof size Bulletproofs bring a significant improvement over the 3.8 KB proof size in [Poe+19]. A single 64-bit range proof is 688 bytes. An aggregated proof for 32 ranges is still just 1 KB whereas 32 proofs from [Poe+19] would have taken up 121 KB. The cost to verify a single 64-bit range proof is 3.9 ms but using batch verification of many proofs the amortized cost can be brought down to 450 μs or 5.2 ECDSA verifications. Verifying an aggregated proof for 64 ranges takes 61 ms or 1.9 ms per range. The marginal cost of verifying an additional proof is 2.67 ms or 83 μs per range. This is less than verifying an ECDSA signature, which cannot take advantage of the same batch validation.

To aid future use of Bulletproofs we also implemented Protocol 2.3 for arithmetic circuits and provide a parser for circuits in the Pinocchio [Par+13] format to the Bulletproofs format. This hooks Bulletproofs up to the Pinocchio toolchain which contains a compiler from a subset of C to the circuit format. To evaluate the implementation we analyze several circuits for hash preimages in Table 2.3 and Figure 2.3.

Specifically, a SHA256 circuit generated by jsnark[6] and a Pedersen hash function over an embedded elliptic curve similar to Jubjub[7] are benchmarked. A Bulletproof for knowing a 384-bit Pedersen hash preimage is about 1 KB and takes 61 ms to verify. The marginal cost of verifying an additional proof is 2.1 ms. The SHA256 preimage proof is 1.4 KB and takes 750 ms to verify. The marginal cost of verifying additional proofs is 41.5 ms. Figure 2.3 shows that the proving and verification time grow linearly. The batch verification first grows logarithmically and then linearly. For small circuits the logarithmic number of exponentiations dominate the cost while for larger circuits the linear scalar operations do.

2.8 A General Forking Lemma

We briefly describe the forking lemma of [Boo+16] that will be needed in the proofs.

Suppose that we have a $(2\mu + 1)$-move public-coin argument with μ challenges, $x_1, \ldots, x_\mu$ in sequence. Let $n_i \geq 1$ for $1 \leq i \leq \mu$. Consider $\prod_{i=1}^{\mu} n_i$ accepting transcripts with challenges in the following tree format. The tree has depth μ and $\prod_{i=1}^{\mu} n_i$ leaves. The root of the tree is labeled with the statement. Each node of depth $i < \mu$ has exactly n_i children, each labeled with a distinct value of the ith challenge x_i.

This can be referred to as an $(n_1, \ldots, n_\mu)$-tree of accepting transcripts. Given a suitable tree of accepting transcripts, one can compute a valid witness for our inner-product argument, range proof, and argument for arithmetic circuit satisfiability. This is a natural generalization of special-soundness for Sigma-protocols, where $\mu = 1$ and $n = 2$. Combined with Theorem 2.7, this shows that

[6] See `https://github.com/akosba/jsnark`.
[7] See https://z.cash/technology/jubjub.html.

Figure 2.1: Sizes for range proofs

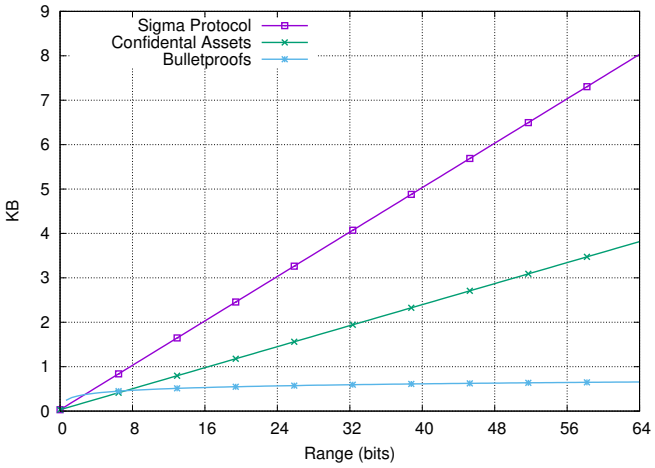


Figure 2.2: Timings for range proofs

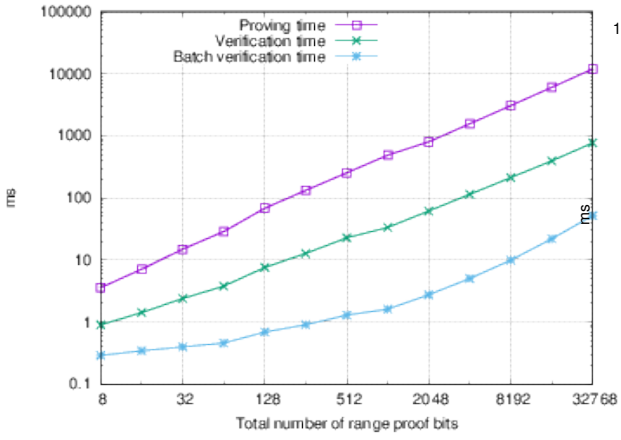


Figure 2.3: Timings for arithmetic circuits (Pedersen Hash)

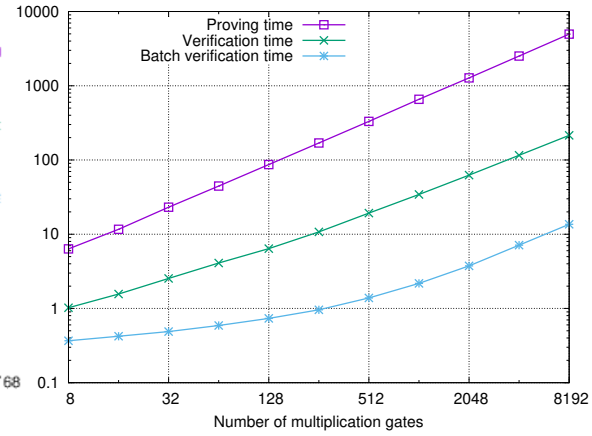


Table 2.2: Range proofs: performance and proof size

Problem size	Gates	π Size (bytes)	Timing (ms) prove	verify	batch
Range proofs (range × aggregation size)					
8 bit	8	482	3.7	0.9	0.28
16 bit	16	546	7.2	1.4	0.33
32 bit	32	610	15	2.4	0.38
64 bit	64	675	29	3.9	0.45
64 bit × 2	128	739	57	6.2	0.55
per range	64	370	29	3.1	0.28
64 bit × 4	256	803	111	10.4	0.71
per range	64	201	28	2.6	0.18
64 bit × 8	512	932	213	18.8	1.08
per range	64	117	27	2.4	0.13
64 bit × 16	1024	932	416	33.2	1.58
per range	64	59	26	2.1	0.10
64 bit × 32	2048	996	812	61.0	2.67
per range	64	32	25	1.9	0.083
64 bit × 64	4096	1060	1594	114	4.91
per range	64	17	25	1.8	0.077
64 bit × 128	8192	1124	3128	210	9.75
per range	64	8.8	25	1.6	0.076
64 bit × 256	16384	1189	6171	392	21.03
per range	64	4.6	24	1.5	0.082
64 bit × 512	32768	1253	12205	764	50.7
per range	64	2.5	24	1.5	0.10

The first 4 instances are n-bit range proofs and the later ones are m aggregated 64-bit proofs and the normalized costs per range. "Batch" is the marginal cost of verifying an additional proof, computed by batch-verifying 100 proofs, subtracting the cost to verify one, and dividing by 99.

Table 2.3: Protocol 2.3: Performance numbers and proof sizes

Input size	Gates	π Size (bytes)	Timing (ms) prove	verify	batch
Pedersen hash preimage (input size)					
48 bit	128	864	88	6.4	0.72
96 bit	256	928	172	10.6	0.93
192 bit	512	992	335	19.1	1.33
384 bit	1024	1056	659	33.6	2.12
768 bit	2048	1120	1292	61.6	3.66
1536 bit	4096	1184	2551	114.9	6.93
3072 bit	8192	1248	5052	213.4	13.20
Unpadded SHA256 preimage					
512 bit	25400	1376	19478	749.9	41.52

Bulletproofs for proving knowledge of x s.t. $H(x) = y$ for different sized x's. The first 7 rows are for the Pedersen hash function and the final row is for SHA256. "Batch" is the marginal cost of verifying an additional proof, computed by batch-verifying 100 proofs, subtracting the cost to verify one, and dividing by 99.

the protocols have witness-extended emulation, and hence, the prover cannot produce an accepting transcript unless they know a witness. For simplicity in the following lemma, we assume that the challenges are chosen uniformly from $\mathbb{Z}_p$ where $|p| = \lambda$, but any sufficiently large challenge space would suffice. The success probability of a cheating prover scales inversely with the size of the challenge space and linearly with the number of accepting transcripts that an extractor needs. Therefore if $\prod_{i=1}^{\mu} n_i$ is negligible in 2^λ, then a cheating prover can create a proof that the verifier accepts with only negligible probability.

Theorem 2.7 (Forking Lemma, [Boo+16])**.** *Let* $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *be a* $(2k+1)$*-move, public coin interactive protocol. Let* χ *be a witness extraction algorithm that succeeds with probability* $1 - \mu(\lambda)$ *for some negligible function* $\mu(\lambda)$ *in extracting a witness from an* $(n_1, \ldots, n_k)$*-tree of accepting transcripts in probabilistic polynomial time. Assume that* $\prod_{i=1}^{k} n_i$ *is bounded above by a polynomial in the security parameter* λ*. Then* $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *has witness-extended emulation.*

The theorem is slightly different than the one from [Boo+16]. We allow the extractor χ to fail with negligible probability. Whenever this happens the Emulator Ext as defined by Definition 2.10 also simply fails. Even with this slight modification, this slightly stronger lemma still holds as Ext overall still only fails with negligible probability.

2.9 Proof of Theorem 2.1

Proof. Perfect completeness follows directly because Protocol 2.1 converts an instance for relation (2.2) into an instance for relation (2.3). Protocol 2.2 is trivially complete. For witness extended emulation we show that there exists an efficient extractor χ that uses n^2 transcripts, as needed by Theorem 2.7.

First we show how to construct an extractor χ_1 for Protocol 2.2 which on input $(\mathbf{G}, \mathbf{H}, U, P)$, either extracts a witness $\mathbf{a}, \mathbf{b}$ such that relation (2.3) holds, or discovers a non-trivial discrete logarithm relation between $\mathbf{G}, \mathbf{H}, U$. Note that the hardness of computing a discrete log relation between $\mathbf{G}', \mathbf{H}', U$ implies the hardness of computing one between $\mathbf{G}, \mathbf{H}, U$ as defined in Protocol 2.2. We will, therefore, use an inductive argument showing that in each step we either extract a witness or a discrete log relation.

If $n = |\mathbf{G}| = 1$, then the prover reveals the witness (a, b) in the protocol and the relation $P = g^a h^b u^{a \cdot b}$ can simply be checked directly.

Next, we show that for each recursive step that on input $(\mathbf{G}, \mathbf{H}, U, P)$, we can efficiently extract from the prover a witness $\mathbf{a}, \mathbf{b}$ or a non-trivial discrete logarithm relation between $\mathbf{G}, \mathbf{H}, U$. The extractor runs the prover to get L and R. Then, by rewinding the prover four times and giving it four challenges x_1, x_2, x_3, x_4, such that $x_i \neq \pm x_j$ for $1 \leq i < j \leq 4$, the extractor obtains four pairs $\mathbf{a'}_i, \mathbf{b'}_i \in \mathbb{Z}_p^{n'}$ such that

$$x_i^2 L + P + x_i^{-2} R = \begin{array}{c} \langle \mathbf{a'}_i, \left(x_i^{-1} \cdot \mathbf{G}_{[:n']} + x_i \cdot \mathbf{G}_{[n':]}\right) \rangle \\ + \langle \mathbf{b'}_i, \left(x_i \cdot \mathbf{H}_{[:n']} + x_i^{-1} \cdot \mathbf{H}_{[n':]}\right) \rangle \\ + \langle \mathbf{a'}_i, \mathbf{b'}_i \rangle \cdot U \end{array} \qquad \text{for } i = 1, \ldots, 4. \tag{2.67}$$

We can use the first three challenges x_1, x_2, x_3, to compute $\nu_1, \nu_2, \nu_3 \in \mathbb{Z}_p$ such that

$$\sum_{i=1}^{3} \nu_i \cdot x_i^2 = 1, \quad \sum_{i=1}^{3} \nu_i = 0, \quad \sum_{i=1}^{3} \nu_i \cdot x_i^{-2} = 0.$$

Then taking a linear combination of the first three equalities in (2.67), with ν_1, ν_2, ν_3 as the coefficients, we can compute $\mathbf{a}_L, \mathbf{b}_L \in \mathbb{Z}_p^n$ and $c_L \in \mathbb{Z}_p$ such that $L = \langle \mathbf{a}_L, \mathbf{G} \rangle + \langle \mathbf{b}_L, \mathbf{H} \rangle + c_L \cdot U$. Repeating this process with different combinations, we can also compute $\mathbf{a}_P, \mathbf{a}_R, \mathbf{b}_P, \mathbf{b}_R \in \mathbb{Z}_p^n$ and $c_P, c_R \in \mathbb{Z}_p$ such that

$$R = \langle \mathbf{a}_R, \mathbf{G} \rangle + \langle \mathbf{b}_R, \mathbf{H} \rangle + c_R \cdot U, \qquad P = \langle \mathbf{a}_P, \mathbf{G} \rangle + \langle \mathbf{b}_P, \mathbf{H} \rangle + c_P \cdot U.$$

Now, for each $x \in \{x_1, x_2, x_3, x_4\}$ and the corresponding $\mathbf{a}', \mathbf{b}' \in \mathbb{Z}_p^{n'}$ we can rewrite (2.67) as:

$$\begin{aligned}
&\langle \mathbf{a}_L \cdot x^2 + \mathbf{a}_P + \mathbf{a}_R \cdot x^{-2}, \mathbf{G}\rangle + \langle \mathbf{b}_L \cdot x^2 + \mathbf{b}_P + \mathbf{b}_R \cdot x^{-2}, \mathbf{H}\rangle + (c_L \cdot x^2 + c_P + c_R \cdot x^{-2}) \cdot U \\
&= x^2 \cdot L + P + x^{-2} \cdot R \\
&= \langle \mathbf{a}' \cdot x^{-1}, \mathbf{G}_{[:n']}\rangle + \langle \mathbf{a}' \cdot x, \mathbf{G}_{[n':]}\rangle + \langle \mathbf{b}' \cdot x, \mathbf{H}_{[:n']}\rangle + \langle \mathbf{b}' \cdot x^{-1}, \mathbf{H}_{[n':]}\rangle + (\langle \mathbf{a}', \mathbf{b}'\rangle) \cdot U.
\end{aligned}$$

This implies that

$$\begin{aligned}
\mathbf{a}' \cdot x^{-1} &= \mathbf{a}_{L,[:n']} \cdot x^2 + \mathbf{a}_{P,[:n']} + \mathbf{a}_{R,[:n']} \cdot x^{-2} \\
\mathbf{a}' \cdot x &= \mathbf{a}_{L,[n':]} \cdot x^2 + \mathbf{a}_{P,[n':]} + \mathbf{a}_{R,[n':]} \cdot x^{-2} \\
\mathbf{b}' \cdot x &= \mathbf{b}_{L,[:n']} \cdot x^2 + \mathbf{b}_{P,[:n']} + \mathbf{b}_{R,[:n']} \cdot x^{-2} \\
\mathbf{b}' \cdot x^{-1} &= \mathbf{b}_{L,[n':]} \cdot x^2 + \mathbf{b}_{P,[n':]} + \mathbf{b}_{R,[n':]} \cdot x^{-2} \\
\langle \mathbf{a}', \mathbf{b}'\rangle &= c_L \cdot x^2 + c_P + c_R \cdot x^{-2}
\end{aligned} \tag{2.68}$$

If any of these equalities do not hold, we directly obtain a non-trivial discrete logarithm relation between the generators $(G_1, \ldots, G_n, H_1, \ldots, H_n, U)$.

If the equalities hold, we can deduce that for each challenge $x \in \{x_1, x_2, x_3, x_4\}$

$$\mathbf{a}_{L,[:n']} \cdot x^3 + (\mathbf{a}_{P,[:n']} - \mathbf{a}_{L,[n':]}) \cdot x + (\mathbf{a}_{R,[:n']} - \mathbf{a}_{P,[n':]}) \cdot x^{-1} - \mathbf{a}_{R,[n':]} \cdot x^{-3} = 0 \tag{2.69}$$

$$\mathbf{b}_{L,[n':]} \cdot x^3 + (\mathbf{b}_{P,[n':]} - \mathbf{b}_{L,[:n']}) \cdot x + (\mathbf{b}_{R,[n':]} - \mathbf{b}_{P,[:n']}) \cdot x^{-1} - \mathbf{b}_{R,[:n']} \cdot x^{-3} = 0 \tag{2.70}$$

The equality (2.69) follows from the first two equations in (2.68). Similarly, (2.70) follows from the third and fourth equations in (2.68).

The only way (2.69) and (2.70) hold for all 4 challenges $x_1, x_2, x_3, x_4 \in \mathbb{Z}_p$ is if

$$\begin{aligned}
&\mathbf{a}_{L,[:n']} = \mathbf{a}_{R,[n':]} = \mathbf{b}_{R,[:n']} = \mathbf{b}_{L,[n':]} = 0, \\
&\mathbf{a}_{L,[n':]} = \mathbf{a}_{P,[:n']}, \qquad \mathbf{a}_{R,[:n']} = \mathbf{a}_{P,[n':]} \\
&\mathbf{b}_{L,[:n']} = \mathbf{b}_{P,[n':]}, \qquad \mathbf{b}_{R,[n':]} = \mathbf{b}_{P,[:n']}.
\end{aligned} \tag{2.71}$$

Plugging these relations into (2.68) we obtain that for every $x \in \{x_1, x_2, x_3, x_4\}$ we have that

$$\mathbf{a}' = \mathbf{a}_{P,[:n']} \cdot x + \mathbf{a}_{P,[n':]} \cdot x^{-1} \qquad \text{and} \qquad \mathbf{b}' = \mathbf{b}_{P,[:n']} \cdot x^{-1} + \mathbf{b}_{P,[n':]} \cdot x.$$

Now, using these values we can see that the extracted c_L, c_P and c_R have the expected form:

$$\begin{aligned}
c_L \cdot x^2 + c_P + c_R \cdot x^{-2} &= \langle \mathbf{a}', \mathbf{b}' \rangle \\
&= \langle \mathbf{a}_{P,[:n']} \cdot x + \mathbf{a}_{P,[n':]} \cdot x^{-1} \ , \ \mathbf{b}_{P,[:n']} \cdot x^{-1} + \mathbf{b}_{P,[n':]} \cdot x \rangle \\
&= \langle \mathbf{a}_{P,[:n']}, \mathbf{b}_{P,[n':]} \rangle \cdot x^2 + \langle \mathbf{a}_{P,[:n']}, \mathbf{b}_{P,[:n']} \rangle + \langle \mathbf{a}_{P,[n':]}, \mathbf{b}_{P,[n':]} \rangle + \langle \mathbf{a}_{P,[n':]}, \mathbf{b}_{P,[:n']} \rangle \cdot x^{-2} \\
&= \langle \mathbf{a}_{P,[:n']}, \mathbf{b}_{P,[n':]} \rangle \cdot x^2 + \langle \mathbf{a}_P, \mathbf{b}_P \rangle + \langle \mathbf{a}_{P,[n':]}, \mathbf{b}_{P,[:n']} \rangle \cdot x^{-2}.
\end{aligned}$$

Since this relation holds for all $x \in \{x_1, x_2, x_3, x_4\}$ it must be that

$$\langle \mathbf{a}_P, \mathbf{b}_P \rangle = c_P.$$

The extractor, thus, either extracts a discrete logarithm relation between the generators, or the witness $(\mathbf{a}_P, \mathbf{b}_P)$ for the relation (2.3).

Using Theorem 2.7 we can see that the extractor uses $4^{\log_2(n)} = n^2$ transcripts in total and thus runs in expected polynomial time in n and λ.

We now show that using Protocol 2.1 we can construct an extractor χ that extracts a valid witness for relation (2.3). The extractor uses the extractor χ_1 of Protocol 2.2. On input $(\mathbf{G}, \mathbf{H}, u, P, c)$ χ runs the prover with on a challenge x and uses the extractor χ_1 to obtain a witness $\mathbf{a}, \mathbf{b}$ such that: $P + (x \cdot c) \cdot U = \langle \mathbf{a}, \mathbf{G} \rangle + \langle \mathbf{b}, \mathbf{H} \rangle + (x \cdot \langle \mathbf{a}, \mathbf{b} \rangle) \cdot U$. Rewinding P, supplying him with a different challenge x' and rerunning the extractor χ_1 yields a second witness $(\mathbf{a}', \mathbf{b}')$. Again the soundness of Protocol 2.2 implies that $P + x' \cdot c \cdot U = \langle \mathbf{a}', \mathbf{G} \rangle + \langle \mathbf{b}', \mathbf{H} \rangle + (x' \cdot \langle \mathbf{a}', \mathbf{b}' \rangle) \cdot U$. From the two witnesses, we can compute:

$$((x - x') \cdot c) \cdot U = \langle \mathbf{a} - \mathbf{a}', \mathbf{G} \rangle + \langle \mathbf{b} - \mathbf{b}', \mathbf{H} \rangle + (x \cdot (\langle \mathbf{a}, \mathbf{b} \rangle) - x' \cdot (\langle \mathbf{a}', \mathbf{b}' \rangle)) \cdot U$$

Unless $\mathbf{a} = \mathbf{a}'$ and $\mathbf{b} = \mathbf{b}'$ we get a not trivial discrete log relation between $\mathbf{G}, \mathbf{H}$ and U. Otherwise we get $((x - x') \cdot c) \cdot U = ((x - x') \cdot \langle \mathbf{a}, \mathbf{b} \rangle) \cdot U \implies c = \langle \mathbf{a}, \mathbf{b} \rangle$. Thus, $(\mathbf{a}, \mathbf{b})$ is a valid witness for relation (2.3). Since χ forks the prover once, and uses the efficient extractor χ_1 twice, it is also efficient. Using the forking lemma (Theorem 2.7) we conclude that the protocol has witness extended emulation. □

2.10 Proof of Theorem 2.3

Proof. Perfect completeness follows from the fact that $t_0 = \delta(y, z) + z^2 \cdot \langle \mathbf{z}^m, \mathbf{v} \rangle$ for all valid witnesses. To prove perfect honest-verifier zero-knowledge we construct a simulator that produces a distribution of proofs for a given statement $(G, H \in \mathbb{G}, \mathbf{G}, \mathbf{H} \in \mathbb{G}^{n \cdot m}, \mathbf{V} \in \mathbb{G}^m)$ that is indistinguishable from valid proofs produced by an honest prover interacting with an honest verifier. The simulator chooses all proof elements and challenges according to the randomness supplied by the adversary from their

respective domains or computes them directly as described in the protocol. S and T_1 are computed according to the verification equations, i.e.:

$$S = -x^{-1} \cdot (-\mu \cdot H + A - \langle z \cdot \mathbf{1}^{n \cdot m} + \mathbf{l}, \mathbf{G} \rangle + \langle z \cdot \mathbf{y}^{n \cdot m} - \mathbf{r}, \mathbf{H}' \rangle + \sum_{j=1}^{m} \langle z^{j+1} \cdot \mathbf{2}^n, \mathbf{H}'_{[(j-1) \cdot m : j \cdot m]} \rangle)$$

$$T_1 = -x^{-1} \cdot (-\tau_x \cdot H + \langle \delta(y,z) - \hat{t}, G \rangle + \langle z^2 \cdot \mathbf{z}^m, \mathbf{V} \rangle + x^2 \cdot T_2)$$

Finally, the simulator runs the inner-product argument with the simulated witness $(\mathbf{l}, \mathbf{r})$ and the verifier's randomness. All elements in the proof are either independently randomly distributed or their relationship is fully defined by the verification equations. The inner product argument remains zero knowledge as we can successfully simulate the witness, thus revealing the witness or leaking information about it does not change the zero-knowledge property of the overall protocol. The simulator runs in time $O(\mathsf{V} + \mathsf{P}_{\mathsf{InnerProduct}})$ and is thus efficient.

In order to prove computational witness extended emulation, we construct an extractor χ as follows. The extractor χ runs the prover with $n \cdot m$ different values of y, $(m+2)$ different values of z, and 3 different values of the challenge x. Additionally it invokes the extractor for the inner product argument on each of the transcripts. This results in $3 \cdot (m+2) \cdot n \cdot m \cdot O(n^2)$ valid proof transcripts.

For each transcript the extractor χ first runs the extractor $\chi_{\mathsf{InnerProduct}}$ for the inner-product argument to extract a witness $\mathbf{l}, \mathbf{r}$ to the inner product argument such that $\mu \cdot H + \langle \mathbf{l}, \mathbf{G} \rangle + \langle \mathbf{r}, \mathbf{H} \rangle = P \wedge \langle \mathbf{l}, \mathbf{r} \rangle = \hat{t}$. Using 2 valid transcripts and extracted inner product argument witnesses for different x challenges, we can compute linear combinations of (2.40) such that in order to compute $\alpha, \rho, \mathbf{a}_L, \mathbf{a}_R, \mathbf{s}_L, \mathbf{s}_R$ such that $A = \alpha \cdot H + \langle \mathbf{a}_L, \mathbf{G} \rangle + \langle \mathbf{a}_R, \mathbf{H} \rangle$, as well as $S = \rho \cdot H + \langle \mathbf{s}_L, \mathbf{G} \rangle + \langle \mathbf{s}_R, \mathbf{H} \rangle$.

If for any other set of challenges (x, y, z) the extractor can compute a different representation of A or S, then this yields a non-trivial discrete logarithm relation between independent generators $H, \mathbf{G}, \mathbf{H}$ which contradicts the discrete logarithm assumption.

Using these representations of A and S, as well as $\mathbf{l}$ and $\mathbf{r}$, we then find that for all challenges x, y and z

$$\mathbf{l} = \mathbf{a}_L - z \cdot \mathbf{1}^{n \cdot m} + \mathbf{s}_L \cdot x$$

$$\mathbf{r} = \mathbf{y}^{n \cdot m} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{n \cdot m} + \mathbf{s}_R \cdot x) + \sum_{j=1}^{m} z^{1+j} \cdot \left(0^{(j-1) \cdot n} || \mathbf{2}^n || 0^{(m-j) \cdot n}\right)$$

If these equalities do not hold for all challenges and $\mathbf{l}, \mathbf{r}$ from the transcript, then we have two distinct representations of the same group element using a set of independent generators. This would be a non-trivial discrete logarithm relation.

For given values of y and z, we now takes 3 transcripts with different x's and uses linear combinations of equation (2.45) to compute τ_1, τ_2, t_1, t_2 such that

$$T_1 = t_1 \cdot G + \tau_1 \cdot H \wedge T_2 = t_1 \cdot G + \tau_2 \cdot H$$

Additionally we can compute a v, γ such that $v \cdot G + \gamma \cdot H = \sum_{j=1}^{m} z^{j+1} \cdot V_j$ Repeating this for m different z challenges, we can compute $(v_j, \gamma_j)_{j=1}^{m}$ such that $v_j \cdot G + \gamma_j \cdot H = V_j \; \forall j \in [1, m]$. If for any transcript $\delta(y, z) + \sum_{j=1}^{m} z^{j+2} \cdot v_j + t_1 \cdot x + t_2 \cdot x^2 \neq \hat{t}$ then this directly yields a discrete log relation between G and H, i.e. a violation of the binding property of the Pedersen commitment. If not, then for all y, z challenges and 3 distinct challenges $X = x_j, j \in [1, 3]$:

$$\sum_{i=0}^{2} t_i \cdot X^i - p(X) = 0$$

with $t_0 = \delta(y, z) + \sum_{j=1}^{m} z^{j+2} \cdot \langle \mathbf{v}_j, \mathbf{2}^n \rangle$ and $p(X) = \sum_{i=0}^{2} p_i \cdot X^i = \langle l(X), r(X) \rangle$. Since the polynomial $t(X) - p(X)$ is of degree 2, but has at least 3 roots (each challenge x_j), it is necessarily the zero polynomial, i.e. $t(X) = \langle l(X), r(X) \rangle$.

Since this implies that $t_0 = p_0$, the following holds for all y, z challenges:

$$\begin{gathered}
\textstyle\sum_{j=1}^{m} z^{j+2} \cdot \langle \mathbf{v}_j, \mathbf{2}^n \rangle + \delta(y, z) \\
= \\
\langle \mathbf{a}_L, \mathbf{y}^{n \cdot m} \circ \mathbf{a}_R \rangle + z \cdot \langle \mathbf{a}_L - \mathbf{a}_R, \mathbf{y}^{n \cdot m} \rangle + \textstyle\sum_{j=1}^{m} z^{j+1} \langle \mathbf{a}_{L,[(j-1) \cdot n : j \cdot n]}, \mathbf{2}^n \rangle \\
-z^2 \cdot \langle \mathbf{1}^{n \cdot m}, \mathbf{y}^{n \cdot m} \rangle - \textstyle\sum_{j=1}^{m} z^{j+2} \cdot \langle \mathbf{1}^n, \mathbf{2}^n \rangle \in \mathbb{Z}_p
\end{gathered}$$

If this equality holds for $n \cdot m$ distinct y challenges and $m + 2$ distinct z challenges, then we can infer the following.

$$\begin{aligned}
\mathbf{a}_L \circ \mathbf{a}_R &= \mathbf{0}^{n \cdot m} && \in \mathbb{Z}_p^{n \cdot m} \\
\mathbf{a}_R &= \mathbf{a}_L - \mathbf{1}^{n \cdot m} && \in \mathbb{Z}_p^{n \cdot m} \\
v_j &= \langle \mathbf{a}_{L,[(j-1) \cdot n : j \cdot n]}, \mathbf{2}^n \rangle && \in \mathbb{Z}_p \forall j \in [1, m]
\end{aligned}$$

The first two equations imply that $\mathbf{a}_L \in \{0, 1\}^{n \cdot m}$. The last equation imply that $v_j \in [0, 2^{n-1}]$ for all $j \in [1, m]$. Since $G^{v_i} H^{\gamma_i} = V_i \quad \forall j \in [1, m]$ we have that $(\mathbf{v}, \boldsymbol{\gamma})$ is valid witness for relation (2.42). The extractor rewinds the prover $3 \cdot (m + 2) \cdot n \cdot m \cdot O(n^2)$ times. Extraction is efficient and the number of transcripts is polynomial in λ because $n, m = O(\lambda)$. Note that extraction either returns a valid witness or a discrete logarithm relation between independently chosen generators. We define χ' being equal to χ but failing whenever χ extracts a discrete log relation. By the Discrete Log Relation assumption this happens with at most negligible probability. We can, therefore, apply the forking lemma and see that computational witness emulation holds.

□

2.11 Proof of Theorem 2.4

Proof. Perfect completeness follows from the fact that

$$t_2 = \delta(y,z) + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O \rangle = \delta(y,z) + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c} \rangle \quad (2.72)$$

whenever the prover knows a witness to the relation and is honest.

To prove perfect honest-verifier zero-knowledge we construct an efficient simulator that produces a distribution of proofs for a given statement

$$\left(\; G, H \in \mathbb{G}, \mathbf{G}, \mathbf{H} \in \mathbb{G}^n, \mathbf{V} \in \mathbb{G}^m, (\mathbf{w}_{L,q}, \mathbf{w}_{R,q}, \mathbf{w}_{O,q})_{q=1}^{Q} \in \mathbb{Z}_p^{n \times 3}, (\mathbf{w}_{V,q})_{q=1}^{Q} \in \mathbb{Z}_p^m, \mathbf{c} \in \mathbb{Z}_p^Q \; \right)$$

and the verifier's randomness that is indistinguishable from valid proofs produced by an honest prover interacting with an honest verifier. The simulator acts as follows:

Compute x, y, z using V's randomness

$$\mu, \tau_x \leftarrow\!\!\$\; \mathbb{Z}_p$$
$$\mathbf{l}, \mathbf{r} \leftarrow\!\!\$\; \mathbb{Z}_p^n$$
$$\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$$
$$A_I, A_O \leftarrow\!\!\$\; \mathbb{G}$$
$$S = -x^{-3} \cdot \left(\; x \cdot A_I + x^2 \cdot A_O - \langle \mathbf{l}, \mathbf{G} \rangle - \langle \mathbf{y}^n + \mathbf{r}, \mathbf{H}' \rangle + x \cdot W_L + x \cdot W_R + W_O - \mu \cdot H \; \right)$$
$$T_3, T_4, T_5, T_6 \leftarrow\!\!\$\; \mathbb{G}$$
$$T_1 = -x^{-1} \cdot \left(\; -\tau_x \cdot H + x^2 \cdot (\delta(y,z) + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} \rangle) - \hat{t} \cdot G + \langle x^2 \cdot (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V), \mathbf{V} \rangle + \textstyle\sum_{i=3}^{6} x^i \cdot T_i \; \right)$$

Output: $(A_I, A_O, S; y, z; T_1, (T_i)_3^6; x; \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r})$

The values $A_I, A_O, \mathbf{l}, \mathbf{r}, \mu, \tau_x$ produced by an honest prover interacting with an honest verifier are random independent elements, i.e. if $\mathbf{s}, \rho, \alpha, \tau_1, (\tau_i)_3^6, \rho$ as well as x, y, z are chosen independently and randomly. $\hat{t}$ is the inner product of $\mathbf{l}, \mathbf{r}$ as in any verifying transcript. The simulated S is fully defined by equations (2.52). The honestly produced T are perfectly hiding commitments and as such random group elements. Their internal relation given $\hat{t}$ and τ_x is fully defined by equation (2.50), which is ensured by computing T_1 accordingly. Therefore, the transcript of the proof is identically distributed to an honestly computed proof with uniformly selected challenges. The simulator runs in time $O(\mathsf{V})$ and is thus efficient.

In order to prove computational witness extended emulation we construct an extractor χ as follows. The χ runs the prover with n different y, $(Q+1)$ different z and 7 different x challenges.

This results in $7 \cdot (Q+1) \cdot n$ valid proof transcripts. We takes 3 valid transcripts for $x \in \{x_1, x_2, x_3\}$ and fixed y and z. From the transmitted $\mathbf{l}, \mathbf{r}, \hat{t}$ for each combination of challenges, we compute ν_1, ν_2, ν_3 such that

$$\sum_{i=1}^{3} \nu_i \cdot x_i = 1 \wedge \sum_{i=1}^{3} \nu_i \cdot x^2 = \sum_{i=1}^{3} \nu_i \cdot x_i^3 = 0$$

Taking the linear combinations of equation (2.52) with (ν_1, ν_2, ν_3) as coefficients, we compute $\alpha \in \mathbb{Z}_p, \mathbf{a}_L, \mathbf{a}_R \in \mathbb{Z}_p^n$ such that $\alpha \cdot H + \langle \mathbf{a}_L, \mathbf{G} \rangle + \langle \mathbf{a}_R, \mathbf{H} \rangle = A_I$. If for any other set of challenges we can compute different $\alpha', \mathbf{a}'_L, \mathbf{a}'_R$ such that

$$\alpha' \cdot H + \langle \mathbf{a}'_L, \mathbf{G} \rangle + \langle \mathbf{a}'_R, \mathbf{H} \rangle = A_I = \alpha H + \langle \mathbf{a}_L, \mathbf{G} \rangle + \langle \mathbf{a}_R, \mathbf{H} \rangle,$$

then this yields a non-trivial discrete log relation between independent generators $H, \mathbf{G}, \mathbf{H}$ which contradicts the discrete log relation assumption. Similarly, we can use the same challenges and equation (2.52) to compute unique $\beta, \rho \in \mathbb{Z}_p, \mathbf{a}_{O,L}, \mathbf{a}_{O,R}, \mathbf{s}_L, \mathbf{s}_R \in \mathbb{Z}_p^n$ such that $\beta \cdot H + \langle \mathbf{a}_{O,L}, \mathbf{G} \rangle + \langle \mathbf{a}_{O,R}, \mathbf{H} \rangle = A_O$ and $\rho \cdot H + \langle \mathbf{s}_L, \mathbf{G} \rangle + \langle \mathbf{s}_R, \mathbf{H} \rangle = S$.

Using Equation (2.52), we can replace A_I, A_O, S with the computed representations and read $\mathbf{l}, \mathbf{r}, \hat{t}$ from the transcripts. We then find that for all challenges x, y, z:

$$\begin{aligned}
\mathbf{l} &= \mathbf{a}_L \cdot x + \mathbf{a}_{O,L} \cdot x^2 + \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R) \cdot X + \mathbf{s}_L \cdot x^3 \\
\mathbf{r} &= \mathbf{y}^n \circ \mathbf{a}_R \cdot x - \mathbf{y}^n + \mathbf{z}_{[1:]}^{Q+1} \cdot (\mathbf{W}_L \cdot x + \mathbf{W}_O) + \mathbf{y}^n \circ \mathbf{a}_{O,R} \cdot x^2 + \mathbf{y}^n \circ \mathbf{s}_R \cdot x^3 \\
\hat{t} &= \langle \mathbf{l}, \mathbf{r} \rangle
\end{aligned}$$

If these equalities do not hold for all challenges and $\mathbf{l}, \mathbf{r}$ from the transcript, then we necessarily have a non-trivial discrete log relation between the generators $\mathbf{G}, \mathbf{H}$ and h.

We now show that t_2 indeed has the form described in (2.72). For a given y, z the extractor takes 6 transcripts with different x's and uses linear combinations of equation (2.50) to compute $(\tau_i, t_i), i \in [1, 3, \ldots, 6]$ such that $T_i = t_i \cdot G + \tau_i \cdot H$. Note that the linear combinations have to cancel out the other $x^i \cdot T_i$ terms as well as $x^2 \cdot \langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V, \mathbf{V} \rangle$. Using these (τ_i, t_i) we can compute v, γ such that $v \cdot G + \gamma \cdot H = \langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V, \mathbf{V} \rangle$. Repeating this for m different z challenges, we can compute $(v_j, \gamma_j)_{j=1}^m$ using linear combinations of $v \cdot G + \gamma \cdot H = \langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V, \mathbf{V} \rangle$ such that $v_j \cdot G + \gamma_j \cdot H = V_j \ \forall j \in [1, m]$. This will however only succeed if the weight vectors $\mathbf{w}_{V,j}$ are linearly independent, i.e if the matrix $\mathbf{W}_V$ has rank m. This necessarily implies that $Q \geq m$. If for any transcript $t_1 \cdot x + \sum_{i=3}^{6} t_i \cdot x^i + x^2 \cdot (\langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c} \rangle + \delta(y, z)) \neq \hat{t}$ then this directly yields a a discrete log relation between G and H.

If not, then for all y, z challenges and 7 distinct challenges $x = x_j, j \in [1, 7]$:

$$\sum_{i=1}^{6} t_i \cdot x - p(x) = 0 \tag{2.73}$$

with $t_2 = \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c} \rangle + \delta(y,z)$ and $p(x) = \sum_{i=1}^{6} p_i \cdot x^i = \langle \mathbf{l}(x), \mathbf{r}(x) \rangle$. Since the polynomial $t(x) - p(x)$ is of degree 6, but has at least 7 roots (each challenge x_j), it is necessarily the zero polynomial, i.e. $t(x) = \langle \mathbf{l}(x), \mathbf{r}(x) \rangle$. Finally, we show that this equality implies that we can extract a witness $(\mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O \in \mathbb{Z}_p^n, \mathbf{v}, \boldsymbol{\gamma} \in \mathbb{Z}_p^m)$ which satisfies the relation.

The quadratic coefficient of p is:

$$p_2 = \langle \mathbf{a}_L, \mathbf{y}^n \circ \mathbf{a}_R \rangle - \langle \mathbf{a}_{O,L}, \mathbf{y}^n \rangle + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_{R,q} \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_{O,L} \rangle + \delta(y,z) \in \mathbb{Z}_p$$

The polynomial equality implies that any challenge y, z, $p_2 = t_2$. Using a fixed y and $(Q+1)$ different z challenges we can infer that all coefficients of $p_2(z) - t_2(z)$ have to be zero. Using n different y challenges, i.e. $n \cdot (Q+1)$ total transcripts we can infer the following equalities:

$$\mathbf{a}_L \circ \mathbf{a}_R - \mathbf{a}_{O,L} = \mathbf{0}^n \in \mathbb{Z}_p^n \tag{2.74}$$

$$\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_{O,L} = \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c} \in \mathbb{Z}_p^Q \tag{2.75}$$

From equation (2.74) we can directly infer that $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_{O,L}$. Equations (2.75) are exactly the linear constraints on the circuit gates.

Defining $\mathbf{a}_O = \mathbf{a}_{O,L}$, we can conclude that $(\mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O, \mathbf{v}, \boldsymbol{\gamma})$ is indeed a valid witness. Extraction is efficient and the number of transcripts is polynomial in λ because $n, m = O(\lambda)$. Note that extraction either returns a valid witness or a non-trivial discrete logarithm relation between independently chosen generators. We define χ' being equal to χ but failing whenever χ extracts a discrete log relation. By the discrete log relation assumption, this happens with at most negligible probability. We can, therefore, apply the forking lemma and see that computational witness emulation holds. $\square$

Chapter 3

HyperPlonk: A proof system for the zkEVM

3.1 Introduction

Proof systems [GMR89; BM88] have a long and rich history in cryptography and complexity theory. In recent years, the efficiency of proof systems has dramatically improved and this has enabled a multitude of new real-world applications that were not previously possible. In this paper, we focus on succinct non-interactive arguments of knowledge, also called SNARKs [Bit+12a]. Here, succinct refers to the fact that the proof is short and verification time is fast, as explained below. Recent years have seen tremendous progress in improving the efficiency of the prover [Wah+18; Mal+19; Xie+19; Ame+17; Ben+19b; Zha+20; Chi+20; Bün+18; GW20a; Set20; Boo+22b; Gol+21; XZS22a].

Let us briefly review what a (preprocessing) SNARK is. We give a precise definition in Section 3.2. Fix a finite field $\mathbb{F}$, and consider the relation $\mathcal{R}(\mathcal{C}, \mathbb{x}, \mathbb{w})$ that is true whenever $\mathbb{x} \in \mathbb{F}^n$, $\mathbb{w} \in \mathbb{F}^m$, and $\mathcal{C}(\mathbb{x}, \mathbb{w}) = 0$, where $\mathcal{C}$ is the description of an arithmetic circuit over $\mathbb{F}$ that takes $n + m$ inputs. A SNARK enables a prover P to non-interactively and succinctly convince a verifier V that P knows a witness $\mathbb{w} \in \mathbb{F}^m$ such that $\mathcal{R}(\mathcal{C}, \mathbb{x}, \mathbb{w})$ holds, for some public circuit $\mathcal{C}$ and $\mathbb{x} \in \mathbb{F}^n$.

In more detail, a SNARK is a tuple of four algorithms $(\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$, where $\mathsf{Setup}(1^\lambda)$ is a randomized algorithm that outputs parameters gp, and $\mathcal{I}(\mathsf{gp}, \mathcal{C})$ is a deterministic algorithm that pre-processes the circuit $\mathcal{C}$ and outputs prover parameters pp and verifier parameters vp. The prover $\mathsf{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w})$ is a randomized algorithm that outputs a proof π, and the verifier $\mathsf{V}(\mathsf{vp}, \mathbb{x}, \pi)$ is a deterministic algorithm that outputs 0 or 1. The SNARK must be *complete*, *knowledge sound*, and *succinct*, as defined in Section 3.2. Here *succinct* means that if $\mathcal{C}$ contains s gates, and $\mathbb{x} \in \mathbb{F}^n$, then the size of the proof should be $O_\lambda(\log s)$ and the verifier's running time should be $\tilde{O}_\lambda(n + \log s)$. A SNARK is often set in the random oracle model where all four algorithms can query the oracle. If

the Setup algorithm is randomized, then we say that the SNARK requires a *trusted setup*; otherwise, the SNARK is said to be *transparent* because Setup only has access to public randomness via the random oracle. Optionally, we might want the SNARK to be zero-knowledge, in which case it is called a zkSNARK.

Modern SNARKs are constructed by compiling an information-theoretic object called an Interactive Oracle Proof (IOP) [BCS16] to a SNARK using a suitable cryptographic commitment scheme. There are several examples of this paradigm. Some SNARKs use a univariate polynomial commitment scheme to compile a Polynomial-IOP to a SNARK. Examples include Sonic [Mal+19],Marlin [Chi+20], and Plonk [GW20a]. Other SNARKs use a multivariate linear (multilinear) commitment scheme to compile a multilinear-IOP to a SNARK. Examples include Hyrax [Wah+18], Libra [Xie+19], Spartan [Set20], Quarks [SL20], and Gemini [Boo+22b]. Yet other SNARKs use a vector commitment scheme (such as a Merkle tree) to compile a vector-IOP to a SNARK. The STARK system [Ben+18b] is the prime example in this category, but other examples include Aurora [Ben+19b], Virgo [Zha+20], Brakedown [Gol+21], and Orion [XZS22a]. While STARKs are post-quantum secure, require no trusted setup, and have an efficient prover, they generate a relatively long proof (tens of kilobytes in practice). The paradigm of compiling an IOP to a SNARK using a suitable commitment scheme lets us build *universal* SNARKs where a single trusted setup can support many circuits. In earlier SNARKs, such as [Gro16; Gen+13; Bit+13b], every circuit required a new trusted setup.

The Plonk system. Among the IOP-based SNARKs that use a Polynomial-IOP, the Plonk system [GW20a] has emerged as one of the most widely adopted in industry. This is because Plonk proofs are very short (about 400 bytes in practice) and fast to verify. Moreover, Plonk supports custom gates, as we will see in a minute. An extension of Plonk, called PlonKup [Pea+22], further extends Plonk to incorporate lookup gates using the Plookup IOP of [GW20a].

One difficulty with Plonk, compared to some other schemes, is the prover's complexity. For a circuit $\mathcal{C}$ with s arithmetic gates, the Plonk prover runs in time $O_\lambda(s \log s)$. The primary bottlenecks come from the fact that the prover must commit to and later open several degree $O(s)$ polynomials. When using the KZG polynomial commitment scheme [KZG10], the prover must (i) compute a multi-exponentiation of size $O(s)$ in a pairing-friendly group where discrete log is hard, and (ii) compute several FFTs and inverse-FFTs of dimension $O(s)$. When using a FRI-based polynomial commitment scheme [Ben+18a; KPV19; Zha+20], the prover computes an $O(cs)$-sized FFT and $O(cs)$ hashes, where $1/c$ is the rate of a certain Reed-Solomon code. The performance further degrades for circuits that contain *high-degree* custom gates, as some FFTs and multi-exponentiations have size proportional to the degree of the custom gates.

In practice, when the circuit size s is bigger than 2^{20}, the FFTs become a significant part of the running time. This is due to the quasi-linear running time of the FFT algorithm, while other parts of the prover scale linearly in s. The reliance on FFT is a direct result of Plonk's use of *univariate*

polynomials. We note that some proof systems eliminate the need for an FFT by moving away from Plonk altogether [Set20; Boo+22b; Gol+21; XZS22a; Dra19].

Hyperplonk. In this paper, we introduce HyperPlonk, an adaptation of the Plonk IOP and its extensions to operate over the boolean hypercube $B_\mu := \{0,1\}^\mu$. We present HyperPlonk as a multilinear-IOP, which means that it can be compiled using a suitable multilinear commitment scheme to obtain a SNARK (or a zkSNARK) with an efficient prover.

HyperPlonk inherits the flexibility of Plonk to support circuits with custom gates, but presents several additional advantages. First, by moving to the boolean hypercube we eliminate the need for an FFT during proof generation. We do so by making use of the classic SumCheck protocol [Lun+92], and this reduces the prover's running time from $O_\lambda(s \log s)$ to $O_\lambda(s)$. The efficiency of SumCheck is the reason why many of the existing multilinear SNARKs [Wah+18; Xie+19; Set20; SL20; Boo+22b] use the boolean hypercube. Here we show that Plonk can similarly benefit from the SumCheck protocol.

Second, and more importantly, we show that the hypercube lets us incorporate custom gates more efficiently into HyperPlonk. A custom gate is a function $G : \mathbb{F}^\ell \to \mathbb{F}$, for some ℓ. An arithmetic circuit $\mathcal{C}$ with a custom gate G, denoted $\mathcal{C}[G]$, is a circuit with addition and multiplication gates along with a custom gate G that can appear many times in the circuit. The circuit may contain multiple types of custom gates, but for now, we will restrict to one type to simplify the presentation. These custom gates can greatly reduce the circuit size needed to compute a function, leading to a faster prover. For example, if one needs to implement the S-box in a block cipher, it can be more efficient to implement it as a custom gate.

Custom gates are not free. Let $G : \mathbb{F}^\ell \to \mathbb{F}$ be a custom gate that computes a multivariate polynomial of total degree d. Let $\mathcal{C}[G]$ be a circuit with a total of s gates. In the Plonk IOP, the circuit $\mathcal{C}[G]$ results in a prover that manipulates univariate polynomials of degree $O(s \cdot d)$. Consequently, when compiling Plonk using KZG [KZG10], the prover needs to do a group multi-exponentiation of size $O(sd)$ as well as FFTs of this dimension. This restricts custom gates in Plonk to gates of low degree.

We show that the prover's work in HyperPlonk is much lower. Let $G : \mathbb{F}^\ell \to \mathbb{F}$ be a custom gate that can be evaluated using k arithmetic operations. In HyperPlonk, the bulk of the prover's work when processing $\mathcal{C}[G]$ is only $O(sk \log^2 k)$ field operations. Moreover, when using KZG multilinear commitments [PST13], the total number of group exponentiations is only $O(s + d \log s)$, where d is the total degree of G. This is much lower than Plonk's $O(sd)$ group exponentiations. It lets us use custom gates of much higher degree in HyperPlonk.

Making Plonk and its Plonkup extension work over the hypercube raises interesting challenges, as discussed in Section 3.1.1. In particular, adapting the Plookup IOP [GW20a], used to implement table lookups, requires changing the protocol to make it work over the hypercube (see Section 3.3.7).

The resulting version of HyperPlonk that supports lookup gates is called HyperPlonk+ and is described in Section 3.5. There are also subtleties in making HyperPlonk zero knowledge. In Section 3.8, we describe a general compiler to transform a multilinear-IOP into one that is zero knowledge.

Batch openings and commit-and-prove SNARKs. The prover in HyperPlonk needs to open several multilinear polynomials at random points. We present a new sum-check-based batch-opening protocol (Section 3.3.8) that can batch many openings into one, significantly reducing the prover time, proof size, and verifier time. Our protocol takes $O(k \cdot 2^\mu)$ field operations for the prover for batching k of μ-variate polynomials, compared to $O(k^2\mu \cdot 2^\mu)$ for the previously best protocol [Tha20]. Under certain conditions, we also obtain a more efficient batching scheme with complexity $O(2^\mu)$, which yields a very efficient commit-and-prove protocol.

Improved multilinear commitments. Since HyperPlonk relies on a multilinear commitment scheme, we revisit two approaches to constructing multilinear commitments and present significant improvements to both.

First, in Section 3.7 we use our commit-and-prove protocol to improve the Orion multilinear commitment scheme [XZS22a]. Orion is highly efficient: the prover time is strictly linear, taking only $O(2^\mu)$ field operations and hashes for a multilinear polynomial in μ variables (no group exponentiations are used). The proof size is $O(\lambda\mu^2)$ hash and field elements, and the verifier time is proportional to the proof size. In Section 3.7 we describe Orion+, that has the same prover complexity, but has $O(\mu)$ proof size and $O(\mu)$ verifier time, with good constants. In particular, for security parameter $\lambda = 128$ and $\mu = 25$ the proof size with Orion+ is only about 7 KBs, compared with 5.5 MB with Orion, a nearly 1000x improvement. Using Orion+ in HyperPlonk gives a strictly linear time prover.

Second, in Section 3.9, we show how to generically transform a univariate polynomial commitment scheme into a multilinear commitment scheme using the tensor-product *univariate* Polynomial-IOP from [Boo+22b]. This yields a new construction for multilinear commitments from FRI [Ben+18a] by applying the transformation to the univariate FRI-based commitment scheme from [KPV19]. This approach leads to a more efficient FRI-based multilinear commitment scheme compared to the prior construction in [Zha+20], which uses recursive techniques. Using this commitment scheme in HyperPlonk gives a quantum-resistant quasilinear-time prover.

Another permutation PIOP for small fields. Looking ahead, the HyperPlonk IOP builds upon a linear-time polynomial IOP for the permutation check relation. However, for μ-variate polynomials, the linear-time permutation PIOP has soundness error $O(\frac{2^\mu}{|\mathbb{F}|})$, which limits its usage with polynomial-sized fields [Boo+22a]. To remedy this, we also propose another permutation PIOP with a much smaller soundness error, that is, $O(\frac{\mu^2}{|\mathbb{F}|})$. The tradeoff is that the PIOP has quasi-linear (rather than linear) prover time.

Application	$\mathcal{R}_{\text{R1}CS}$	Spartan	$\mathcal{R}_{\text{PLONK+}}$	Jellyfish	HyperPlonk
3-to-1 Rescue Hash	288 [Aly+20]	422 ms	144 [Sys22]	40 ms	88 ms
Zexe's recursive circuit	2^{22} [Xio+22]	6 min	2^{17} [Xio+22]	13.1s	5.1s
Rollup of 50 private tx	2^{25}	39 min	2^{20} [Sys22]	110 s	38.2 s

Table 3.1: The prover runtime of Hyperplonk, Spartan [Set20], and Jellyfish Plonk, for popular applications. The first column (next to the column of the applications) shows the number of R1CS constraints for each application. The third column shows the corresponding number of constraints in HyperPlonk+. Note that the Zexe and the Rollup applications are using the BW6-761 curve. For more detail see Section 3.6.5.

Evaluation results. After applying the optimizations in Appendix 3.10, when instantiated with the pairing-based multilinear commitment scheme of [PST13], the proof size of Hyperplonk is $\mu + 5$ group elements and $4\mu + 29$ field elements[1]. Using BLS12-381 as the pairing group, we obtain $4.7KB$ proofs for $\mu = 20$ and $5.5KB$ proofs for $\mu = 25$. For comparison, Kopis [SL20] and Gemini [Boo+22b], which also have linear-time provers, report proofs of size 39KB and 18KB respectively for $\mu = 20$. In Table 3.1 and Table 3.6 we show that our prototype HyperPlonk implementation outperforms an optimized commercial-strength Plonk system for circuits with more than 2^{14} gates. It also shows the effects of PLONK arithmetization compared to R1CS by comparing the prover runtime for several important applications. Hyperplonk outperforms Spartan [Set20] for these applications by a factor of over 60. We discuss the evaluation further in Section 3.6.

3.1.1 Technical overview

In this section we give a high level overview of how to make Plonk and its extensions work over the hybercube. We begin by describing Plonk in a modular way, breaking it down into a sequence of elementary components shown in Figure 3.1. In Section 3.3 we show how to instantiate each component over the hybercube.

Some components of Plonk in Figure 3.1 rely on the simple linear ordering of the elements of a finite cyclic group induced by the powers of a generator. On the hypercube there is no natural simple ordering, and this causes a problem in the Plookup protocol [GW20a] that is used to implement a lookup gate. To address this we modify the Plookup argument in Section 3.5 to make it work over the hypercube. We give an overview of our approach below.

A review of Plonk. Let us briefly review the Plonk SNARK. Let $\mathcal{C}[G] : \mathbb{F}^{n+m} \to \mathbb{F}$ be a circuit with a total of s gates, where each gate has fan-in two and can be one of addition, multiplication, or a custom gate $G : \mathbb{F}^2 \to \mathbb{F}$. Let $\mathbb{x} \in \mathbb{F}^n$ be a public input to the circuit. Plonk represents the

[1] The constants depend linearly on the degree of the custom gates. These numbers are for simple degree 2 arithmetic circuits.

resulting computation as a sequence of $n + s + 1$ triples[2]: =

$$\hat{M} := \Big\{ \big(L_i, R_i, O_i\big) \in \mathbb{F}^3 \Big\}_{i=0,\ldots,n+s}. \tag{3.1}$$

This $\hat{M}$ is a matrix with three columns and $n+s+1$ rows. The first n rows encode the n public input; the next s rows represent the left and right inputs and the output for each gate; and the final row enforces that the final output of the circuit is zero. We will see how in a minute.

In basic (univariate) Plonk, the prover encodes the cells of $\hat{M}$ using a cyclic subgroup $\Omega \subseteq \mathbb{F}$ of order $3(n+s+1)$. Specifically, let $\omega \in \Omega$ be a generator. Then the prover interpolates and commits to a polynomial $M \in \mathbb{F}[X]$ such that

$$M(\omega^{3i}) = L_i, \quad M(\omega^{3i+1}) = R_i, \quad M(\omega^{3i+2}) = O_i \quad \text{for } i = 0, \ldots, n+s.$$

Now the prover needs to convince the verifier that the committed M encodes a valid computation of the circuit $\mathcal{C}$. This is the bulk of Plonk system.

Hyperplonk. In HyperPlonk we instead use the boolean hypercube to encode $\hat{M}$. From now on, suppose that $n+s+1$ is a power of two, so that $n+s+1 = 2^\mu$. The prover interpolates and commits to a multilinear polynomial $M \in \mathbb{F}[X^{\mu+2}] = \mathbb{F}[X_1, \ldots, X_{\mu+2}]$ such that

$$M\big(0, 0, \langle i \rangle\big) = L_i, \quad M\big(0, 1, \langle i \rangle\big) = R_i, \quad M\big(1, 0, \langle i \rangle\big) = O_i, \quad \text{for } i = 0, \ldots, n+s. \tag{3.2}$$

Here $\langle i \rangle$ is the μ-bit binary representation of i. Note that a multilinear polynomial on $\mu+2$ variables is defined by a vector of $2^{\mu+2} = 4 \times 2^\mu$ coefficients. Hence, it is always possible to find a multilinear polynomial that satisfies the $3 \times 2^\mu$ constraints in (3.2). Next, the prover needs to convince the verifier that the committed M encodes a valid computation of the circuit $\mathcal{C}$. To do so, we need to adapt Plonk to work over the hypercube.

Let us start with the pre-processing algorithm $\mathcal{I}(\mathsf{gp}, \mathcal{C})$ that outputs prover and verifier parameters pp and vp. The verifier parameters vp encode the circuit $\mathcal{C}[G]$ as a commitment to four multilinear polynomials (S_1, S_2, S_3, σ), where $S_1, S_2, S_3 \in \mathbb{F}[X^\mu]$ and $\sigma \in \mathbb{F}[X^{\mu+2}]$. The first three are called *selector polynomials* and σ is called the *wiring polynomial*. We will see how they are defined in a minute. There is one more auxiliary multilinear polynomial $I \in \mathbb{F}[X^\mu]$ that encodes the input $\mathbb{x} \in \mathbb{F}^n$. This polynomial is defined as $I(\langle i \rangle) = \mathbb{x}_i$ for $i = 0, \ldots, n-1$, and is zero on the rest of the boolean cube B_μ. The verifier, on its own, computes a commitment to the polynomial I to ensure that the correct input $\mathbb{x} \in \mathbb{F}^n$ is being used in the proof. Computing a commitment to I can be done in time $O_\lambda(n)$, which is within the verifier's time budget.

With this setup, the Plonk prover P convinces the verifier that the committed M satisfies two

[2] A more general Plonkish arithmetization [Zca22] supports wider tuples, but triples are sufficient here.

polynomial identities:

The gate identity: Let $S_1, S_2, S_3 : \mathbb{F}^\mu \to \{0,1\}$ be the three selector polynomials that the pre-processing algorithm $\mathcal{I}(\mathsf{gp}, \mathcal{C})$ committed to in vp. To prove that all gates were evaluated correctly, the prover convinces the verifier that the following identity holds for all $\mathbf{x} \in B_\mu := \{0,1\}^\mu$:

$$\begin{aligned} 0 = S_1(\mathbf{x}) \cdot \Big(\underbrace{M(0,0,\mathbf{x})}_{L_{[\mathbf{x}]}} + \underbrace{M(0,1,\mathbf{x})}_{R_{[\mathbf{x}]}}\Big) \;\; + \;\; S_2(\mathbf{x}) \cdot \underbrace{M(0,0,\mathbf{x})}_{L_{[\mathbf{x}]}} \cdot \underbrace{M(0,1,\mathbf{x})}_{R_{[\mathbf{x}]}} \\ + \;\; S_3(\mathbf{x}) \cdot \boldsymbol{G}\Big(\underbrace{M(0,0,\mathbf{x})}_{L_{[\mathbf{x}]}}, \underbrace{M(0,1,\mathbf{x})}_{R_{[\mathbf{x}]}} \Big) \;\; - \;\; \underbrace{M(1,0,\mathbf{x})}_{O_{[\mathbf{x}]}} \;\; + \;\; I(\mathbf{x}) \end{aligned} \tag{3.3}$$

where $[\mathbf{x}] = \sum_{i=0}^{\mu-1} \mathbf{x}_i 2^i$ is the integer whose binary representation is $\mathbf{x} \in B_\mu$. For each $i = 0, \ldots, n+s$, the selector polynomials S_1, S_2, S_3 are defined to do the "right" thing:

- for an addition gate: $S_1(\langle i \rangle) = 1, \quad S_2(\langle i \rangle) = S_3(\langle i \rangle) = 0 \quad$ (so $O_i = L_i + R_i$)
- for a multiplication gate: $S_1(\langle i \rangle) = S_3(\langle i \rangle) = 0, \quad S_2(\langle i \rangle) = 1 \quad$ (so $O_i = L_i \cdot R_i$)
- for a G gate: $S_1(\langle i \rangle) = S_2(\langle i \rangle) = 0, \quad S_3(\langle i \rangle) = 1 \quad$ (so $O_i = G(L_i, R_i)$)
- when $i < n$ or $i = n + s$: $S_1(\langle i \rangle) = S_2(\langle i \rangle) = S_3(\langle i \rangle) = 0 \quad$ (so $O_i = I(\langle i \rangle)$).

The last bullet ensures that O_i is equal to the i-th input for $i = 0, \ldots, n-1$, and that the final output of the circuit, O_{n+s}, is equal to zero.

The wiring identity: Every wire in the circuit $\mathcal{C}$ induces an equality constraint on two cells in the matrix $\hat{M}$. In HyperPlonk, the wiring constraints are captured by a permutation $\hat{\sigma} : B_{\mu+2} \to B_{\mu+2}$. The prover needs to convince the verifier that

$$M(\mathbf{x}) = M(\hat{\sigma}(\mathbf{x})) \quad \text{for all } \mathbf{x} \in B_{\mu+2} := \{0,1\}^{\mu+2}. \tag{3.4}$$

To do so, the pre-processing algorithm $\mathcal{I}(\mathsf{gp}, \mathcal{C})$ commits to a multilinear polynomial $\sigma : \mathbb{F}^{\mu+2} \to \mathbb{F}$ that satisfies $\sigma(\mathbf{x}) = [\hat{\sigma}(\mathbf{x})]$ for all $\mathbf{x} \in B_{\mu+2}$ (recall that $[\hat{\sigma}(\mathbf{x})]$ is the integer whose binary representation is $\hat{\sigma}(\mathbf{x}) \in B_{\mu+2}$). The prover then convinces the verifier that the following two sets are equal (both sets are subsets of $\mathbb{F}^2$):

$$\Big\{ \big([\mathbf{x}], M(\mathbf{x})\big) \Big\}_{\mathbf{x} \in B_{\mu+2}} = \Big\{ \big([\hat{\sigma}(\mathbf{x})], M(\mathbf{x})\big) \Big\}_{\mathbf{x} \in B_{\mu+2}}. \tag{3.5}$$

This equality of sets implies that (3.4) holds.

Proving the gate identity. The prover convinces the verifier that the Gate identity holds by proving that the polynomial defined by the right hand side of (3.3) is zero for all $\mathbf{x} \in B_\mu$. This is done using a ZeroCheck IOP, defined in Section 3.3.2. If the custom gate G has total degree d and

there are s gates in the circuit, then the total number of coefficientsx of the polynomial in (3.3) is $(d+1)(s+n+1)$ which is about $(d \cdot s)$. If this were a univariate polynomial, as in Plonk, then a ZeroCheck would require a multi-exponentiation of dimension $(d \cdot s)$ and an FFT of the same dimension. When the polynomial is defined over the hypercube, the ZeroCheck is implemented using the SumCheck protocol in Section 3.3.1, which requires no FFTs. In that section we describe two optimizations to the SumCheck protocol for the settings where the multivariate polynomial has a high degree d in every variable:

- First, in every round of SumCheck the prover sends a polynomial commitment to a univariate polynomial of degree d, instead of sending the polynomial in the clear as in regular SumCheck. This greatly reduces the proof size.
- Second, in standard SumCheck, the prover opens the univariate polynomial commitment at three points: at 0, 1, and at a random $r \in \mathbb{F}$. We optimize this step by showing that opening the commitment at a *single* point is sufficient. This further shortens the final proof.

The key point is that the resulting ZeroCheck requires the prover to do only about $s + d \cdot \mu$ group exponentiations, which is much smaller than $d \cdot s$ in Plonk. The additional arithmetic work that the prover needs to do depends on the number of multiplication gates in the circuit implementing the custom gate G, not on the total degree of G, as in Plonk. As such, we can support much larger custom gates than Plonk.

In summary, proof generation time is reduced for two reasons: (i) the elimination of the FFTs, and (ii) the better handling of high-degree custom gates.

Proving the wiring identity. The prover convinces the verifier that the Wiring identity holds by proving the set equality in (3.5). We describe a set equality protocol over the hypercube in Section 3.3.4. Briefly, we use a technique from Bayer and Groth [BG12], that is also used in Plonk, to reduce this problem to a certain ProductCheck over the hypercube (Section 3.3.3). We then use an idea from Quarks [SL20] to reduce the hypercube ProductCheck to a ZeroCheck, which then reduces to a SumCheck. This sequence of reductions is shown in Figure 3.1. Again, no FFTs are needed.

Table lookups. An important extension to Plonk supports circuits with table lookup gates. The table is represented as a fixed vector $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$. A table lookup gate ensures that a specific cell in the matrix $\hat{M}$ is contained in $\mathbf{t}$. For example, one can set $\mathbf{t}$ to be the field elements in $\{0, 1, \ldots, B\}$ for some B (padding the vector by 0 as needed). Now, checking that a cell in $\hat{M}$ is contained in $\mathbf{t}$ is a simple way to implement a range check.

Let $f, t : B_\mu \to \mathbb{F}$ be two multilinear polynomials. Here the polynomial t encodes the table $\mathbf{t}$, where the table values are $t(B_\mu)$. The polynomial f encodes the cells of $\hat{M}$ that need to be checked. An important step in supporting lookup gates in Plonk is a way for the prover to convince the verifier that $f(B_\mu) \subseteq t(B_\mu)$, when the verifier has commitments to f and t. The Plookup proof

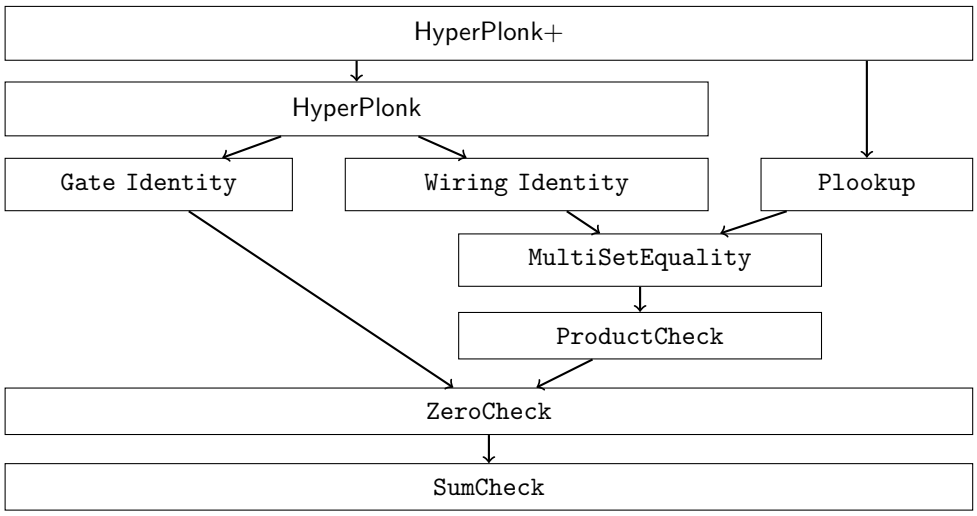


Figure 3.1: The multilinear polynomial-IOPs that make up HyperPlonk.

system by Gabizon and Williamson [GW20a] is a way for the prover to do just that. More recently preprocessed alternatives to lookup have been developed[Zap+22a; PK22]. These perform better if the table is known, e.g. a range of values but are in general orthogonal to Plookup.

The problem is that Plookup is designed to work when the polynomials are defined over a cyclic subgroup $\mathbb{G} \subseteq \mathbb{F}^*$ of order q with generator $\omega \in \mathbb{G}$. In particular, Plookup requires a function $\mathsf{next} : \mathbb{F} \to \mathbb{F}$ that induces an ordering of $\mathbb{G}$. This function must satisfy two properties: (i) the sequence

$$\omega, \quad \mathsf{next}(\omega), \quad \mathsf{next}\big(\mathsf{next}(\omega)\big), \quad \ldots, \quad \mathsf{next}^{(q-1)}(\omega) \tag{3.6}$$

should traverse all of $\mathbb{G}$, and (ii) the function next should be a *linear* function. This is quite easy in a cyclic group: simply define $\mathsf{next}(x) := \omega x$.

To adapt Plookup to the hypercube we need a *linear* function $\mathsf{next} : \mathbb{F}^\mu \to \mathbb{F}^\mu$ that traverses all of B_μ as in (3.6), starting with some element $\mathbf{x}_0 \in B_\mu$. However, such an $\mathbb{F}$-linear function does not exist. Nevertheless, we construct in Section 3.3.7 a quadratic function from $\mathbb{F}^\mu$ to $\mathbb{F}^\mu$ that traverses B_μ. The function simulates B_μ using a binary extension and has a beautiful connection to similar techniques used in early PCP work[BSS08]. We then show how to linearize the function by modifying some of the building blocks that Plookup uses. This gives an efficient Plookup protocol over the hypercube. Finally, in Section 3.5 we use this hypercube Plookup protocol to support lookup gates in HyperPlonk. The resulting protocol is called HyperPlonk+.

3.1.2 Additional related work

The origins of SNARKs date back to the work of Kilian [Kil92] and Micali [Mic94] based on the PCP theorem. Many of the SNARK constructions cited in the previous sections rely on techniques introduced in the proof of the PCP theorem.

Recursive SNARKs [Val08] are an important technique for building a SNARK for a long computation. Early recursive SNARKs [CT10; Bit+12b; Ben+14b; COS20] built a prover for the entire SNARK circuit and then repeatedly used this prover. More recent recursive SNARKs rely on accumulation schemes [BGH19; Bün+20; Bon+21; Bün+21a; KST21] where the bulk of the SNARK verifier runs outside of the prover.

Many practical SNARKs rely on the random oracle model and often use a non-falsifiable assumption. Indeed, a separation result due to Gentry and Wichs [GW11] suggests that a SNARK requires either an idealized model or a non-falsifiable assumption. An interesting recent direction is the construction of *batch proofs* [CJJ21a; CJJ21b; WW22] in the standard model from standard assumptions. These give succinct proofs for computations in $\mathcal{P}$, namely succinct proofs for computations that do not rely on a hidden witness. SNARKs give succinct proofs for computations in $\mathcal{NP}$.

3.2 Preliminaries

Notation: We use λ to denote the security parameter. For $n \in \mathbb{N}$ let $[n]$ be the set $\{1, 2, \ldots, n\}$; for $a, b \in \mathbb{N}$ let $[a, b)$ denote the set $\{a, a+1, \ldots b-1\}$. A function $f(\lambda)$ is $\mathsf{poly}(\lambda)$ if there exists a $c \in \mathbb{N}$ such that $f(\lambda) = O(\lambda^c)$. If for all $c \in \mathbb{N}$, $f(\lambda)$ is $o(\lambda^{-c})$, then $f(\lambda)$ is in $\mathsf{negl}(\lambda)$ and is said to be **negligible**. A probability that is $1 - \mathsf{negl}(\lambda)$ is **overwhelming**. We use $\mathbb{F}$ to denote a field of prime order p such that $\log(p) = \Omega(\lambda)$.

A *multiset* is an extension of the concept of a set where every element has a positive multiplicity. Two finite multisets are equal if they contain the same elements with the same multiplicities.

Recall that a **relation** is a set of pairs $(\mathbb{x}, \mathbb{w})$. An **indexed relation** is a set of triples $(\mathbb{i}, \mathbb{x}; \mathbb{w})$. The index $\mathbb{i}$ is fixed at setup time.

In defining the syntax of the various protocols, we use the following convention concerning public values (known to both the prover and the verifier) and secret ones (known only to the prover). In any list of arguments or returned tuple $(a, b, c; d, e)$, those variables listed before the semicolon are public, and those listed after it are secret. When there is no secret information, the semicolon is omitted.

Useful facts. We next list some facts that will be used throughout the paper.

Lemma 3.1 (Multilinear extensions)**.** *For every function $f : \{0,1\}^\mu \to \mathbb{F}$, there is a unique multilinear polynomial $\tilde{f} \in \mathbb{F}[X_1, \ldots, X_\mu]$ such that $\tilde{f}(\boldsymbol{b}) = f(\boldsymbol{b})$ for all $\boldsymbol{b} \in \{0,1\}^\mu$. We call $\tilde{f}$ the*

multilinear extension of f, and $\tilde{f}$ can be expressed as

$$\tilde{f}(\mathbf{X}) = \sum_{\boldsymbol{b}\in\{0,1\}^\mu} f(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \mathbf{X})$$

where $eq(\boldsymbol{b}, \mathbf{X}) := \prod_{i=1}^{\mu}\big(\boldsymbol{b}_i\mathbf{X}_i + (1-\boldsymbol{b}_i)(1-\mathbf{X}_i)\big)$.

Lemma 3.2 (Schwartz-Zippel Lemma)**.** *Let $f \in \mathbb{F}[X_1, \ldots, X_\mu]$ be a non-zero polynomial of total degree d over field $\mathbb{F}$. Let S be any finite subset of $\mathbb{F}$, and let $r_1, \ldots, r_\mu$ be μ field elements selected independently and uniformly from set S. Then*

$$\Pr\left[f(r_1, \ldots, r_\mu) = 0\right] \leq \frac{d}{|S|}\,.$$

Linear codes. We review the definition of linear code.

Definition 3.1 (Linear Code)**.** *An (n, k, δ)-linear error-correcting code $E : \mathbb{F}^k \to \mathbb{F}^n$ is an injective mapping from $\mathbb{F}^k$ to a linear subspace C in $\mathbb{F}^n$, such that (i) the injective mapping can be computed in linear time in k; (ii) any linear combination of codewords is still a codeword; and (iii) the relative hamming distance $\Delta(u, v)$ between any two different codewords $u, v \in \mathbb{F}^k$ is at least δ. The rate of the code E is defined as k/n.*

3.2.1 Proofs and arguments of knowledge.

We define interactive proofs of knowledge, which consist of a non-interactive *preprocessing* phase run by an indexer as well as an interactive *online* phase between a prover and a verifier.

Definition 3.2 (Interactive Proof and Argument of Knowledge)**.** *An interactive protocol $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathsf{P}, \mathsf{V})$ between a prover P and verifier V is an argument of knowledge for an indexed relation $\mathcal{R}$ with knowledge error $\delta : \mathbb{N} \to [0, 1]$ if the following properties hold, where given an index $\mathtt{i}$, common input $\mathtt{x}$ and prover witness $\mathtt{w}$, the deterministic indexer outputs $(\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathtt{i})$ and the output of the verifier is denoted by the random variable $\langle \mathsf{P}(\mathsf{pp}, \mathtt{x}, \mathtt{w}), \mathsf{V}(\mathsf{vp}, \mathtt{x})\rangle$:*

- *Perfect Completeness: for all $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$*

$$\Pr\left[\, \langle \mathsf{P}(\mathsf{pp}, \mathtt{x}, \mathtt{w}), \mathsf{V}(\mathsf{vp}, \mathtt{x})\rangle = 1 \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathtt{i}) \end{array} \right] = 1$$

- *δ-Soundness (adaptive): Let $\mathcal{L}(\mathcal{R})$ be the language corresponding to the indexed relation $\mathcal{R}$ such that $(\mathtt{i}, \mathtt{x}) \in \mathcal{L}(\mathcal{R})$ if and only if there exists $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$. Π is δ-sound if for every pair of probabilistic polynomial time adversarial prover algorithm $(\mathcal{A}_1, \mathcal{A}_2)$ the following*

holds:

$$\Pr\left[\langle \mathcal{A}_2(\mathbb{i},\mathbb{x},\mathsf{st}),\mathsf{V}(\mathsf{vp},\mathbb{x})\rangle = 1 \wedge (\mathbb{i},\mathbb{x}) \notin \mathcal{L}(\mathcal{R}) \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i},\mathbb{x},\mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathsf{vp},\mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp},\mathbb{i}) \end{array}\right] \leq \delta(|\mathbb{i}| + |\mathbb{x}|)\,.$$

We say a protocol is computationally sound if δ is negligible. If $\mathcal{A}_1, \mathcal{A}_2$ are unbounded and δ is negligible, then the protocol is statistically sound. If $A = (\mathcal{A}_1, \mathcal{A}_2)$ is unbounded, the soundness definition becomes for all $(\mathbb{i},\mathbb{x}) \notin \mathcal{L}(\mathcal{R})$

$$\Pr\left[\langle \mathcal{A}_2(\mathbb{i},\mathbb{x},\mathsf{gp}),\mathsf{V}(\mathsf{vp},\mathbb{x})\rangle = 1 \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{vp},\mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp},\mathbb{i}) \end{array}\right] \leq \delta(|\mathbb{i}| + |\mathbb{x}|)$$

- *δ-Knowledge Soundness: There exists a polynomial $\mathsf{poly}(\cdot)$ and a probabilistic polynomial-time oracle machine $\mathcal{E}$ called the* extractor *such that given oracle access to any pair of probabilistic polynomial time adversarial prover algorithm $(\mathcal{A}_1, \mathcal{A}_2)$ the following holds:*

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_2(\mathbb{i},\mathbb{x},\mathsf{st}),\mathsf{V}(\mathsf{vp},\mathbb{x})\rangle = 1 \\ \wedge \\ (\mathbb{i},\mathbb{x},\mathbb{w}) \notin \mathcal{R} \end{array} \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i},\mathbb{x},\mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathsf{vp},\mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp},\mathbb{i}) \\ \mathbb{w} \leftarrow \mathsf{Ext}^{\mathcal{A}_1,\mathcal{A}_2}(\mathsf{gp},\mathbb{i},\mathbb{x}) \end{array}\right] \leq \delta(|\mathbb{i}| + |\mathbb{x}|)$$

  An interactive protocol is "knowledge sound", or simply an "argument of knowledge", if the knowledge error δ is negligible in λ. If the adversary is unbounded, then the argument is called an interactive proof of knowledge.

- *Public coin An interactive protocol is considered to be public coin if all of the verifier messages (including the final output) can be computed as a deterministic function given a random public input.*

- *Zero knowledge: An interactive protocol $\langle \mathsf{P}, \mathsf{V}\rangle$ is considered to be zero-knowledge if there is a PPT simulator $\mathcal{S}$ such that for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, it holds that*

$$\left|\Pr\left[\langle \mathsf{P}(\mathsf{pp},\mathbb{x},\mathbb{w}),\mathcal{A}_2(\mathsf{st},\mathbb{i},\mathbb{x})\rangle = 1 \wedge (\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R} \;\middle|\; \begin{array}{c} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{i},\mathbb{x},\mathbb{w},\mathsf{st}) \leftarrow \mathcal{A}_1(z,\mathsf{gp}) \\ (\mathsf{vp},\mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp},\mathbb{i}) \end{array}\right] - \right.$$
$$\left.\Pr\left[\langle \mathcal{S}(\sigma,z,\mathsf{pp},\mathbb{x}),\mathcal{A}_2(\mathsf{st},\mathbb{i},\mathbb{x})\rangle = 1 \wedge (\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R} \;\middle|\; \begin{array}{c} (\mathsf{gp},\sigma) \leftarrow \mathcal{S}(1^\lambda) \\ (\mathbb{i},\mathbb{x},\mathbb{w},\mathsf{st}) \leftarrow \mathcal{A}_1(z,\mathsf{gp}) \\ (\mathsf{vp},\mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp},\mathbb{i}) \end{array}\right]\right| \leq \mathsf{negl}(\lambda)\,.$$

We say that $\langle \mathsf{P}, \mathsf{V} \rangle$ is statistically zero knowledge if $\mathcal{A}$ is unbounded; and say it perfectly zero knowledge if $\mathsf{negl}(\lambda)$ is replaced with zero. $\langle \mathsf{P}, \mathsf{V} \rangle$ is honest-verifier zero knowledge (HVZK) if the adversary $\mathcal{A}_2$ honestly follows the verifier algorithm.

We introduce both notions of soundness and knowledge soundness. Knowledge soundness implies soundness, as the existence of an extractor implies that $(\mathtt{i}, \mathtt{x}) \in \mathcal{L}(\mathcal{R})$. Furthermore, we show in Lemma 3.3 that soundness directly implies knowledge soundness for certain oracle relations and oracle arguments.

PolyIOPs. SNARKs can be constructed from information-theoretic proof systems that give the verifier oracle access to prover messages. The information-theoretic proof is then compiled using a cryptographic tool, such as a polynomial commitment. We now define a specific type of information-theoretic proof system called polynomial interactive oracle proofs.

Definition 3.3. *A polynomial interactive oracle proof (PIOP) is a public-coin interactive proof for a polynomial oracle relation $\mathcal{R} = \{(\mathtt{i}, \mathtt{x}; \mathtt{w})\}$. The relation is an oracle relation in that $\mathtt{i}$, and $\mathtt{x}$ can contain oracles to μ-variate polynomials over some field $\mathbb{F}$. The oracles specify μ and the degree in each variable. These oracles can be queried at arbitrary points in $\mathbb{F}^\mu$ to evaluate the polynomial at these points. The actual polynomials corresponding to the oracles are contained in the pp and the $\mathtt{w}$, respectively. We denote an oracle to a polynomial f by $[[f]]$. In every protocol message, the P sends multi-variate polynomial oracles. The verifier in every round sends a random challenge.*

We measure the following parameters for the complexity of a PIOP:

- *The prover time measures the runtime of the prover.*
- *The verifier time measures the runtime of the verifier.*
- *The query complexity is the number of queries the verifier performs to the oracles.*
- *The round complexity measures the number of rounds. In our protocols, it is always equivalent to the number of oracles sent.*
- *The size of the proof oracles is the length of the transmitted polynomials.*
- *The size of the witness is the length of the witness polynomial.*

Proof of Knowledge. As a proof system, the PIOP satisfies perfect completeness and unbounded knowledge-soundness with knowledge-error δ. Note that the extractor can query the oracle at arbitrary points to efficiently recover the entire polynomial.

Non-interactive arguments. Interactive public-coin arguments can be made non-interactive using the Fiat-Shamir transform. The Fiat-Shamir transform replaces the verifier challenges with hashes of the transcript up to that point. The works by [AFK21; Wik21] show that this is secure for multi-round special-sound protocols and multi-round oracle proofs.

Soundness and knowledge soundness.

Lemma 3.3 (Sound PIOPs are knowledge sound)**.** *Consider a δ-sound PIOP for oracle relations $\mathcal{R}$ such that for all $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, $\mathbb{w}$ consists only of polynomials such that the instance contains oracles to these polynomials. The PIOP has δ knowledge-soundness, and the extractor runs in time $O(|\mathbb{w}|)$*

Proof. We will show that we can construct an extractor Ext that can produce $\mathbb{w}^*$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}^*) \in \mathcal{R}$ if and only if $(\mathbb{i}, \mathbb{x}) \in \mathcal{L}(\mathcal{R})$. This implies that the soundness error exactly matches the knowledge soundness error. For each oracle of a μ-variate polynomial with degree d in each variable, the extractor queries the polynomial at $(d+1)^\mu$ distinct points to extract the polynomial inside the oracle and thus w^*. If $(\mathbb{i}, \mathbb{x}, \mathbb{w}^*) \in \mathcal{R}$ then by definition $(\mathbb{i}, \mathbb{x}) \in \mathcal{L}(\mathcal{R})$. Additionally assume that $(\mathbb{i}, \mathbb{x}) \in \mathcal{L}(\mathcal{R})$ but $(\mathbb{i}, \mathbb{x}, \mathbb{w}^*) \notin \mathcal{R}$. Then there must exists $\mathbb{w}' \neq \mathbb{w}^*$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}') \in \mathcal{R}$. Since the relation only admits polynomials as witnesses and these polynomials are degree d and μ-variate, then there cannot be two distinct witnesses that agree on $(d+1)^\mu$ oracle queries. Therefore $\mathbb{w}' = \mathbb{w}^*$ which leads to a contradiction. The extractor, therefore, outputs the unique, valid witness for every $(\mathbb{i}, \mathbb{x})$ in the language, and thus, the soundness and knowledge soundness error are the same. □

3.2.2 Multilinear polynomial commitments.

Definition 3.4 (Commitment scheme)**.** *A commitment scheme Γ is a tuple $\Gamma = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ of PPT algorithms where:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{gp}$ *generates public parameters* gp*;*
- $\mathsf{Commit}(\mathsf{gp}; x) \to (C; r)$ *takes a secret message x and outputs a public commitment C and (optionally) a secret opening hint r (which might or might not be the randomness used in the computation).*
- $\mathsf{Open}(\mathsf{gp}, C, x, r) \to b \in \{0, 1\}$ *verifies the opening of commitment C to the message x provided with the opening hint r.*

*A commitment scheme Γ is **binding** if for all PPT adversaries $\mathcal{A}$:*

$$\Pr\left[b_0 = b_1 \neq 0 \wedge x_0 \neq x_1 \ : \ \begin{array}{l} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (C, x_0, x_1, r_0, r_1) \leftarrow \mathcal{A}(\mathsf{gp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{gp}, C, x_0, r_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{gp}, C, x_1, r_1) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

A commitment scheme Γ *is* ***hiding*** *if for any polynomial-time adversary* $\mathcal{A}$*:*

$$\left|\Pr\left[b = b' \;:\; \begin{array}{l} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x_0, x_1, st) \leftarrow \mathcal{A}(\mathsf{gp}) \\ b \leftarrow\!\!\$\ \{0,1\} \\ (C_b; r_b) \leftarrow \mathsf{Commit}(\mathsf{gp}; x_b) \\ b' \leftarrow \mathcal{A}(\mathsf{gp}, st, C_b) \end{array}\right] - 1/2\right| = \mathsf{negl}(\lambda)\,.$$

If the adversary is unbounded, then we say the commitment is statistically hiding. We additionally define polynomial commitment schemes for multi-variate polynomials.

Definition 3.5. *(Polynomial commitment) A polynomial commitment scheme is a tuple of protocols* $\Gamma = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *where* $(\mathsf{Setup},\ \mathsf{Commit}, \mathsf{Open})$ *is a binding commitment scheme for a message space* $R[X]$ *of polynomials over some ring* R*, and*

- $\mathsf{Eval}((\mathsf{vp}, \mathsf{pp}), C, \mathbf{z}, y, d, \mu; f) \to b \in \{0,1\}$ *is an interactive public-coin protocol between a PPT prover* P *and verifier* V*. Both* P *and* V *have as input a commitment* C*, points* $\mathbf{z} \in \mathbb{F}^\mu$ *and* $y \in \mathbb{F}$*, and a degree* d*. The prover has prover parameters* pp*, and the verifier has verifier parameters* vp*. The prover additionally knows the opening of* C *to a secret polynomial* $f \in \mathcal{F}_\mu^{(\leq d)}$*. The protocol convinces the verifier that* $f(\mathbf{z}) = y$*.*

A polynomial commitment scheme is ***correct*** *if an honest committer can successfully convince the verifier of any evaluation. Specifically, if the prover is honest, then for all polynomials* $f \in \mathcal{F}_\mu^{(\leq d)}$ *and all points* $z \in \mathbb{F}^\mu$*,*

$$\Pr\left[b = 1 \;:\; \begin{array}{l} \mathsf{gp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (C; r) \leftarrow \mathsf{Commit}(\mathsf{gp}, f) \\ y \leftarrow f(\mathbf{z}) \\ b \leftarrow \mathsf{Eval}(\mathsf{gp}, c, z, y, d, \mu; f, r) \end{array}\right] = 1\ .$$

We require that Eval *is an interactive argument of knowledge and has* knowledge soundness*, which ensures that we can extract the committed polynomial from any evaluation.*

Multi-variate polynomial commitments can be instantiated from random oracles using the FRI protocol [Zha+20], bilinear groups [PST13], groups of unknown order [BFS20] and discrete logarithm groups. We give a table of polynomial commitments with their different properties in Table 3.2:

Virtual oracles and commitments. Given multiple polynomial oracles, we can construct virtual oracles to the functions of these polynomials. An oracle to $g([[f_1]], \ldots, [[f_k]])$ for some function g is simply the list of oracles $\{[[f_1]], \ldots, [[f_k]]\}$ as well as a description of g. In order to evaluate $g([[f_1]], \ldots, [[f_k]])$ at some point $\mathbf{x}$ we compute $y_i = f_i(\mathbf{x}) \forall i \in [k]$ and output $g(y_1, \ldots y_k)$. Equivalently given commitments to polynomials, we can construct a virtual commitment to a function of

Scheme		Prover time: Commit+ Eval	Verifier time	Proof size	$n = 2^{25}$	Setup	Add.
KZG-based [PST13]	BL	n $\mathbb{G}_1$	$\log(n)$ P	$\log(n)$ $\mathbb{G}_1$	0.8KB	Univ.	Yes
Dory [Lee21]	BL	$n\mathbb{G}_1 + \sqrt{n}P$	$\log(n)$ $\mathbb{G}_T$	$6\log(n)$ $\mathbb{G}_T$	30KB	Trans.	Yes
Bulletproofs [Bün+18]	DL	n $\mathbb{G}$	n $\mathbb{G}$	$2\log(n)$ $\mathbb{G}$	1.6KB	Trans.	Yes
FRI-based (§3.9)	RO	$n\log(n)/\rho\mathbb{F} + n/\rho H$	$\log^2(n)\frac{\lambda}{-\log\rho}$ H	$\log^2(n)\frac{\lambda}{-\log\rho}$ H	250KB	Trans.	No
Orion	RO	$nH + \frac{n}{k} + k$ rec.	$\lambda\log^2 nH$	$\lambda\log^2 n$ H	5.5MB	Trans.	No
Orion + (§3.7)	BL	$n/k\mathbb{G}_1 + nH +$ $(k\lambda H + \frac{n}{k}\mathbb{F})$ rec.	$\log(n)P$	$4\log n$ $\mathbb{G}_1$	7KB	Univ.	No

Table 3.2: Multi-linear polynomial commitment schemes for μ-variate linear polynomials and $n = 2^\mu$. The prover time measures the complexity of committing to a polynomial and evaluating it once. The commitment size is constant for all protocols. Unless constants are mentioned, the metrics are assumed to be asymptotic. In the 4th row, ρ denotes the rate of Reed-Solomon codes. In the 5th and 6th rows, k denotes the number of rows of the matrix that represents the polynomial coefficients. The 6th column measures the concrete proof size for $n = 2^{25}$, i.e. $\mu = 25$ and 128-bit security. Legend: BL=Bilinear Group, DL=Discrete Logarithm, RO=Random Oracle, H= Hashes, P= pairings, $\mathbb{G}$= group scalar multiplications, rec.= Recursive circuit size, univ.= universal setup, trans.= transparent setup, Add.=Additive

these polynomials in the same manner. If g is an additive function and the polynomial commitment is additively homomorphic, then we can use the homomorphism to do the evaluation. A common example is that given additive commitments C_f, C_g to polynomials $f(\mathbf{X})$, $g(\mathbf{X})$, we want to construct a commitment to $(1-Y)f + Yg$. Then (C_f, C_g) serves as such a commitment and we can evaluate it at $(y, \mathbf{x})$ by evaluating $(1-y)C_f + y \cdot C_g$ at $\mathbf{x}$.

3.2.3 PIOP compilation

PIOP compilation transforms the interactive oracle proof into an interactive argument of knowledge (without oracles) Π. The compilation replaces the oracles with polynomial commitments. Every query by the verifier is replaced with an invocation of the Eval protocol at the query point $\mathbf{z}$. The compiled verifier accepts if the PIOP verifier accepts and if the output of all Eval invocations is 1. If Π is public-coin, then it can further be compiled to a non-interactive argument of knowledge (or NARK) using the Fiat-Shamir transform.

Theorem 3.1 (PIOP Compilation [BFS20; Chi+20])**.** *If the polynomial commitment scheme Γ has witness-extended emulation, and if the t-round Polynomial IOP for $\mathcal{R}$ has negligible knowledge error, then Π, the output of the PIOP compilation, is a secure (non-oracle) argument of knowledge for $\mathcal{R}$. The compilation also preserves zero knowledge. If Γ is hiding and Eval is honest-verifier zero-knowledge, then Π is honest-verifier zero-knowledge. The efficiency of the resulting argument of knowledge Π depends on the efficiency of both the PIOP and Γ:*

- *Prover time The prover time is equal to the sum of (i) prover time of the PIOP, (ii) the oracle length times the commitment time, and (iii) the query complexity times the prover time of Γ.*

- *Verifier time* The verifier time is equal to the sum of (i) the verifier time of the PIOP and (ii) the verifier time for Γ times the query complexity of the PIOP.
- *Proof size* The proof size is equal to sum of (i) the message complexity of the PIOP times the commitment size and (ii) the query complexity times the proof size of Γ. If the proof size is $O(\log^c(|\mathbb{w}|))$, then we say the proof is succinct.

Batching. *The prover time, verifier time, and proof size can be significantly reduced using batch openings of the polynomial commitments. After batching, the proof size only depends on the number of oracles plus a single polynomial commitment opening.*

3.3 A toolbox for multivariate polynomials

We begin by reviewing several important PolyIOPs that will serve as building blocks for HyperPlonk. Some are well-known, and some are new. Figure 3.1 serves as a guide for this section: we define the PolyIOPs listed in the figure following the dependency order.

Notation. From here on, we let $B_\mu := \{0,1\}^\mu \subseteq \mathbb{F}^\mu$ be the boolean hypercube. We use $\mathcal{F}_\mu^{(\leq d)}$ to denote the set of multivariate polynomials in $\mathbb{F}[X_1, \ldots, X_\mu]$ where the degree in each variable is at most d; moreover, we require that each polynomial in $\mathcal{F}_\mu^{(\leq d)}$ can be expressed as a virtual oracle to $c = O(1)$ multilinear polynomials. that is, with the form $f(\mathbf{X}) := g(h_1(\mathbf{X}), \ldots, h_c(\mathbf{X}))$ where $h_i \in \mathcal{F}_\mu^{(\leq 1)}$ $(1 \leq i \leq c)$ is multilinear and g is a c-variate polynomial of total degree at most d. Looking ahead, we restrict ourselves to this kind of polynomials so that we can have sumchecks for the polynomials with linear-time provers.
For polynomials $f, g \in \mathcal{F}_\mu^{(\leq d)}$, we denote $\mathsf{merge}(f,g) \in \mathcal{F}_{\mu+1}^{(\leq d)}$ as

$$\mathsf{merge}(f,g) := h(\mathbf{X}_0, \ldots, \mathbf{X}_\mu) := (1 - \mathbf{X}_0) \cdot f(\mathbf{X}_1, \ldots, \mathbf{X}_\mu) + \mathbf{X}_0 \cdot g(\mathbf{X}_1, \ldots, \mathbf{X}_\mu) \tag{3.7}$$

so that $h(0, \mathbf{X}) = f(\mathbf{X})$ and $h(1, \mathbf{X}) = g(\mathbf{X})$. In the following definitions, we omit the public parameters $\mathsf{gp} := (\mathbb{F}, \mu, d)$ when the context is clear. We use $\delta_{\text{check}}^{d,\mu}$ to denote the soundness error of the PolyIOP for relation $\mathcal{R}_{\text{check}}$ with public parameter $(\mathbb{F}, d, \mu)$, where check $\in \{\text{sum}, \text{zero}, \text{prod}, \text{mset}, \text{perm}, \text{lkup}\}$.

3.3.1 SumCheck PIOP for high degree polynomials

In this section, we describe a PIOP for the sumcheck relation using the classic sumcheck protocol [Lun+92]. However, we modify the protocol and adapt it to our setting of high-degree polynomials.

Definition 3.6 (SumCheck relation)**.** *The relation $\mathcal{R}_{SUM}$ is the set of all tuples $(\mathbb{x}; \mathbb{w}) = \big((v, [[f]]); f\big)$ where $f \in \mathcal{F}_\mu^{(\leq d)}$ and $\sum_{\boldsymbol{b} \in B_\mu} f(\boldsymbol{b}) = v$.*

Scheme	P time	V time	Num of queries	Num of rounds	Proof oracle size	Witness size
SumCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu+1$	μ	$d\mu$	$O(2^\mu)$
ZeroCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu+1$	μ	$d\mu$	$O(2^\mu)$
ProdCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu+2$	$\mu+1$	$O(2^\mu)$	$O(2^\mu)$
MsetEqChk	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu+2$	$\mu+1$	$O(2^\mu)$	$O(k2^\mu)$
PermCheck	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu+2$	$\mu+1$	$O(2^\mu)$	$O(2^\mu)$
Plookup	$O(2^\mu d \log^2 d)$	$O(\mu)$	$\mu+3$	$\mu+2$	$O(2^\mu)$	$O(2^\mu)$
BatchEval	$O(2^\mu k)$	$O(k\mu)$	1	$\mu+\log k$	$O(\mu+\log k)$	$O(k2^\mu)$

Table 3.3: The complexity of PIOPs. d and μ denote the degree and the number of variables of the multivariate polynomials; k in MsetCheck is the length of each element in the multisets; k in BatchEval is the number of evaluations.

Construction. The classic SumCheck protocol [Lun+92] is a PolyIOP for the relation $\mathcal{R}_{\text{SUM}}$. When applying the protocol to a polynomial $f \in \mathcal{F}_\mu^{(\leq d)}$, the protocol runs in μ rounds where in every round, the prover sends a univariate polynomial of degree at most d to the verifier. The verifier then sends a random challenge point for the univariate polynomial. At the end of the protocol, the verifier checks the consistency between the univariate polynomials and the multi-variate polynomial using a single query to f.

Given a tuple $(\mathbb{x}; \mathbb{w}) = (v, [[f]]; f)$ for μ-variate degree d polynomial f such that $\sum_{\boldsymbol{b}\in B_\mu} f(\boldsymbol{b}) = v$:

- For $i = \mu, \mu-1, \ldots, 1$:
  - The prover computes $r_i(X) := \sum_{\boldsymbol{b}\in B_{i-1}} f(\boldsymbol{b}, X, \alpha_{i+1}, \ldots, \alpha_\mu)$ and sends the oracle $[[r_i]]$ to the verifier. r_i is univariate and of degree at most d.
  - The verifier checks that $v = r_i(0) + r_i(1)$, samples $\alpha_i \leftarrow \mathbb{F}$, sends α_i to the prover, and sets $v \leftarrow r_i(\alpha_i)$.
- Finally, the verifier accepts if $f(\alpha_1, \ldots, \alpha_\mu) = v$.

Theorem 3.2. *The PIOP for $\mathcal{R}_{SUM}$ is perfectly complete and has knowledge error $\delta_{sum}^{d,\mu} := d\mu/|\mathbb{F}|$.*

We refer to [Tha20] for the proof of the theorem.

Sending r as an oracle. Unlike in the classic sumcheck protocol, we send an oracle to r_i, in each round, instead of the actual polynomial. This does not change the soundness analysis, as the soundness is still proportional to the degree of the univariate polynomials sent in each round. However, it reduces the communication and verifier complexity, especially if the degree of r is large, as in our application of Hyperplonk with custom gates.

Moreover, the verifier has to evaluate r_i at three points: 0, 1, and α_i. As a useful optimization, the prover can instead send an oracle for the degree $d-2$ polynomial

$$r_i'(X) := \frac{r_i(X) - (1-X)\cdot r_i(0) - X \cdot r_i(1)}{X \cdot (1-X)},$$

along with $r_i(0)$. The verifier then computes $r_i(1) \leftarrow v - r_i(0)$ and

$$r_i(\alpha_i) \leftarrow r'_i(\alpha_i) \cdot (1-\alpha_i) \cdot \alpha_i + (1-\alpha_i) \cdot r_i(0) + \alpha_i \cdot r_i(1).$$

This requires only one query to the oracle of r'_i at α_i and one field element per round.

Computing sumcheck for high-degree polynomials. Consider a multi-variate polynomial $f(\mathbf{X}) := h(g_1(\mathbf{X}), \ldots, g_c(\mathbf{X}))$ such that h is degree d and can be evaluated through an arithmetic circuit with $O(d)$ gates. In the sumcheck protocol, the prover has to compute a univariate polynomial $r_i(X)$ in each round using the previous verifier messages $\alpha_1, \ldots, \alpha_{i-1}$. We adapt the algorithm by [Tha13; Xie+19] that showed how the sumcheck prover can be run in time linear in 2^μ using dynamic programming. The algorithm takes as input a description of f as well as the sumcheck round challenges $\alpha_1, \ldots, \alpha_\mu$. It outputs the round polynomials $r_1, \ldots, r_\mu$. The sumcheck prover runs the algorithm in parallel to the sumcheck protocol, taking each computed r_i as that rounds message:

Algorithm 1 Computing $r_1, \ldots, r_\mu$ [Tha13; Xie+19]

procedure SUMCHECK PROVER($h, g_1(\mathbf{X}), \ldots, g_c(\mathbf{X})$)
  For each g_j build table $A_j : \{0,1\}^\mu \to \mathbb{F}$ of all evaluations over B_μ
  for $i \leftarrow \mu \ldots 1$ **do**
    For each $\boldsymbol{b} \in B_{i-1}$ and each $j \in [c]$, define $r^{(j,\boldsymbol{b})}(X) := (1-X)A_j[\boldsymbol{b}, 0] + XA_j[\boldsymbol{b}, 1]$.
    Compute $r^{(\boldsymbol{b})}(X) \leftarrow h(r^{(1,\boldsymbol{b})}(X), \ldots, r^{(c,\boldsymbol{b})}(X))$ for all $\boldsymbol{b} \in B_{i-1}$ using Algorithm 2 .
    $r_i(X) \leftarrow \sum_{\boldsymbol{b} \in B_{i-1}} r_{\boldsymbol{b}}(X)$.
    Send $r_i(X)$ to V.
    Receive α_i from V.
    Set $A_j[\boldsymbol{b}] \leftarrow r^{(j,\boldsymbol{b})}(\alpha_i)$ for each $\boldsymbol{b} \in B_{i-1}$.
  end for
end procedure

In [Tha13; Xie+19], $r^{(\boldsymbol{b})}(X) := h(r^{(1,\boldsymbol{b})}(X), \ldots, r^{(c,\boldsymbol{b})}(X))$ is computed by evaluating h on d distinct values for X, e.g. $X \in \{0, \ldots, d\}$ and interpolating the output. This works as h is a degree d polynomial and each $r^{j,\boldsymbol{b}}$ is linear. Evaluating $r^{j,\boldsymbol{b}}$ on d points can be done in d steps. So the total time to evaluate all $r^{j,\boldsymbol{b}}$ for $j \in [c]$ is $c \cdot d$. Furthermore, the circuit has $O(d)$ gates, and evaluating it on d inputs, takes time $O(d^2)$. Assuming that $c \approx d$ the total time to compute $r^{(\boldsymbol{b})}$ with this algorithm is $O(d^2)$ and the time to run Algorithm 1 is $O(2^\mu d^2)$.

We show how this can be reduced to $O(2^\mu \cdot d \log^2 d)$ for certain low depth circuits, such as $h := \prod_c r_c(\mathbf{X})$. The core idea is that evaluating the circuit for h *symbolically*, instead of at d individual points, is faster if fast polynomial multiplication algorithms are used.

We will present the algorithm for computing $h(X) := \prod_{j=1}^{d} r_j(X)$, then we will discuss how to extend this for more general h. Assume w.l.o.g. that d is a power of 2.

Algorithm 2 Evaluating $h := \prod_{j=1}^{d} r_j$

Require: $r_1, \ldots, r_d$ are linear functions
1: **procedure** $h(r_1(X), \ldots, r_d(X))$
2: $t_{1,j} \leftarrow r_j$ for all $j \in [d]$.
3: **for** $i \leftarrow 1 \ldots \log d$ **do**
4: **for** $j \in [d/2^i]$ **do**
5: $t_{i+1,j}(X) \leftarrow t_{i,2j-1}(X) \cdot t_{i,2j}(X)$ ▷ Using fast polynomial multiplication
6: **end for**
7: **end for**
8: **return** $h = t_{\log_2(d),1}$
9: **end procedure**

In round i there are $d/2^i$ polynomial multiplications for polynomials of degree 2^{i-1}. In FFT-friendly[3] fields, polynomial multiplication can be performed in time $O(d \log(d))$.[4] The total running time of the algorithm is therefore $\sum_{i=1}^{\log_2(d)} \frac{d}{2^i} 2^{i-1} \log(2^{i-1}) = \sum_{i=1}^{\log_2(d)} O(d \cdot i) = O(d \log^2(d))$.

Algorithm 2 naturally extends to more complicated, low-depth circuits. Addition gates are performed directly through polynomial addition, which takes $O(d)$ time for degree d polynomials. As long as the circuit is low-depth and has $O(d)$ multiplication gates, the complexity remains $O(d \log^2(d))$. Furthermore, we can compute $r^k(X)$ for $k \leq d$ using only a single FFT of length $\deg(r) \cdot k$ for an input polynomial r. The FFT evaluates r at $\deg(r) \cdot k$ points. Then we raise each point to the power of k. This takes time $O(\deg(r) \cdot k(\log(\deg(r)) + \log(k)))$ and saves a factor of $\log(k)$ over a repeated squaring implementation.

Batching. Multiple sumcheck instances, e.g. $(s, [[f]])$ and $(s', [[g]])$ can easily be batched together. This is done using a random-linear combination, i.e. showing that $(s + \alpha s', [[f]] + \alpha[[g]]) \in \mathcal{L}(\mathcal{R}_{\text{SUM}})$ for a random verifier-generated α [Wah+18; CFS17]. The batching step has soundness $\frac{1}{\mathbb{F}}$.

Complexity. Overall, Algorithm 1 calls Algorithm 2 for each point in the boolean hypercube and then on each point in a cube of half the size. The total runtime of Algorithm 1 is, therefore, $O(2^\mu d \log^2 d)$ if h is degree d and low-depth. We summarize the complexity of the PIOP for $\mathcal{R}_{\text{SUM}}$ with respect to $f \in \mathcal{F}_\mu^{(\leq d)}$, below:

- The prover time is $\mathsf{tp}^f_{\text{sum}} = \mathcal{O}(2^\mu \cdot d \log^2 d)$ $\mathbb{F}$-ops (for low-depth f that can be evaluated in time $O(d)$).
- The verifier time is $\mathsf{tv}^f_{\text{sum}} = \mathcal{O}(\mu)$.
- The query complexity is $\mathsf{q}^f_{\text{sum}} = \mu + 1$, μ queries to univariate oracles, one to multi-variate f.
- The round complexity and the number of proof oracles is $\mathsf{rc}^f_{\text{sum}} = \mu$.

[3] These are fields where there exists an element that has a smooth order of at least d.

[4] Recent breakthrough results have shown that polynomial multiplication is $O(d \log(d))$ over arbitrary finite fields [HVDH22] and there have been efforts toward building practical, fast multiplication algorithms for arbitrary fields [BS+22]. In practice, and especially for low-degree polynomials, using Karatsuba multiplication might be faster.

- The number of field elements sent by P is μ.
- The size of the proof oracles is $\mathsf{pl}^f_{\text{sum}} = d \cdot \mu$; the size of the witness is $c \cdot 2^\mu$.

3.3.2 ZeroCheck PIOP

In this section, we describe a PIOP showing that a multivariate polynomial evaluates to zero everywhere on the boolean hypercube. The PIOP builds upon the sumcheck PIOP in Section 3.3.1 and is a key building block for product-check PIOP in Section 3.3.3. The zerocheck PIOP is also helpful in HyperPlonk for proving the gate identity.

Definition 3.7 (ZeroCheck relation)**.** *The relation $\mathcal{R}_{ZERO}$ is the set of all tuples $(\mathbb{x}; \mathbb{w}) = \big(([[f]]); f\big)$ where $f \in \mathcal{F}_\mu^{(\leq d)}$ and $f(\mathbf{x}) = 0$ for all $\mathbf{x} \in B_\mu$.*

We use an idea from [Set20] to reduce a ZeroCheck to a SumCheck.

Construction. Given a tuple $(\mathbb{x}; \mathbb{w}) = \big(([[f]]); f\big)$, the protocol is the following:

- V sends P a random vector $\mathbf{r} \leftarrow\$ \mathbb{F}^\mu$
- Let $\hat{f}(\mathbf{X}) := f(\mathbf{X}) \cdot eq(\mathbf{X}, \mathbf{r})$ where $eq(\mathbf{x}, \mathbf{y}) := \prod_{i=1}^{\mu} \big(x_i y_i + (1 - x_i)(1 - y_i)\big)$.
- Run a sumcheck PolyIOP to convince the verifier that $\big((0, [[\hat{f}]]); \hat{f}\big) \in \mathcal{R}_{\text{SUM}}$.

Batching. It is possible to batch two instances $\big(([[f]]); f\big) \in \mathcal{R}_{\text{ZERO}}$ and $\big(([[g]]); g\big) \in \mathcal{R}_{\text{ZERO}}$ by running a zerocheck on $\big(([[f + \alpha g]]); f + \alpha g\big)$ for a random $\alpha \in \mathbb{F}$. The soundness error of the batching protocol $\frac{1}{\mathbb{F}}$.

Theorem 3.3. *The PIOP for $\mathcal{R}_{ZERO}$ is perfectly complete and has knowledge error $\delta^{d,\mu}_{zero} := d\mu/|\mathbb{F}| + \delta^{d+1,\mu}_{sum} = \mathcal{O}(d\mu/|\mathbb{F}|)$.*

Proof. **Completeness.** For every $\big(([[f]]); f\big) \in \mathcal{R}_{\text{ZERO}}$, $\hat{f}$ is also zero everywhere on the boolean hypercube, thus the sumcheck of $\hat{f}$ is zero, and completeness follows from sumcheck's completeness.

Knowledge soundness. By Lemma 3.3, it is sufficient to argue the soundness error of the protocol. We note that $[[f]] \in \mathcal{L}(\mathcal{R}_{\text{ZERO}})$ (i.e., $\big(([[f]]); f\big) \in \mathcal{R}_{\text{ZERO}}$) if and only if the following auxiliary polynomial

$$g(\mathbf{Y}) := \sum_{\mathbf{x} \in B_\mu} f(\mathbf{x}) \cdot eq(\mathbf{x}, \mathbf{Y})$$

is identically zero. This is because $eq(x, y)$ for a $\mathbf{x}, \mathbf{y} \in B_\mu$ is 1 if $\mathbf{x} = \mathbf{y}$ and 0 otherwise. So $g(\mathbf{y}) = f(\mathbf{y})$ for all $\mathbf{y} \in B_\mu$. Therefore, for any $[[f]] \notin \mathcal{L}(\mathcal{R}_{\text{ZERO}})$, the corresponding g is a non-zero polynomial and by Lemma 3.2,

$$g(\mathbf{r}) = \sum_{\mathbf{x} \in B_\mu} f(\mathbf{x}) \cdot eq(\mathbf{x}, \mathbf{r}) = 0$$

with probability $d\mu/|\mathbb{F}|$ over the choice of $\mathbf{r}$, thus the probability that the verifier accepts is at most $d\mu/|\mathbb{F}|$ plus the probability that the SumCheck PIOP verifier accepts when $\big((0, [[\hat{f}]]); \hat{f}\big) \notin \mathcal{R}_{\text{SUM}}$, which is $d\mu/|\mathbb{F}| + \delta_{\text{sum}}^{d+1,\mu}$ as desired. □

Complexity. We analyze the complexity of the PIOP for $\mathcal{R}_{\text{ZERO}}$ with respect to $f \in \mathcal{F}_\mu^{(\leq d)}$.

- The prover time is $\mathsf{tp}_{\text{zero}}^f = \mathsf{tp}_{\text{sum}}^{\hat{f}} = \mathcal{O}(d \log^2 d \cdot 2^\mu)$ $\mathbb{F}$-ops.
- The verifier time is $\mathsf{tv}_{\text{zero}}^f = \mathcal{O}(\mu)$.
- The query complexity is $\mathsf{q}_{\text{zero}}^f = \mathsf{q}_{\text{sum}}^{\hat{f}} = \mu + 1$.
- The round complexity and the number of proof oracles is $\mathsf{rc}_{\text{zero}}^f = \mathsf{rc}_{\text{sum}}^{\hat{f}} = \mu$.
- The number of field elements sent by P is $\mathsf{nf}_{\text{zero}}^f = \mathsf{nf}_{\text{sum}}^{\hat{f}} = \mu$.
- The size of the proof oracles is $\mathsf{pl}_{\text{zero}}^f = \mathsf{pl}_{\text{sum}}^{\hat{f}} = d\mu$; the size of the witness is $\mathcal{O}(2^\mu)$.

3.3.3 ProductCheck PIOP

We describe a PIOP for the product check relation, that is, for a rational polynomial (where both the nominator and the denominator are multivariate polynomials), the product of the evaluations on the boolean hypercube is a claimed value s. The PIOP uses the idea from the Quark system [SL20, §5], we adapt it to build upon the zerocheck PIOP in Section 3.3.2. Product check PIOP is a key building block for the multiset equality check PIOP in Section 3.3.4.

Definition 3.8 (ProductCheck relation)**.** *The relation $\mathcal{R}_{PROD}$ is the set of all tuples $(\mathbb{x}; \mathbb{w}) = \big((s, [[f_1]], [[f_2]]); f_1, f_2\big)$ where $f_1 \in \mathcal{F}_\mu^{(\leq d)}$, $f_2 \in \mathcal{F}_\mu^{(\leq d)}$, $f_2(b) \neq 0$ $\forall b \in B_\mu$ and $\prod_{\mathbf{x} \in B_\mu} f'(\mathbf{x}) = s$, where f' is the rational polynomial $f' := f_1/f_2$. In the case that $f_2 = c$ is a constant polynomial, we directly set $f := f_1/c$ and write $(\mathbb{x}; \mathbb{w}) = \big((s, [[f]]); f\big)$.*

Construction. The Quark system [SL20, §5] constructs a proof system for the $\mathcal{R}_{\text{PROD}}$ relation. The proof system uses an instance of the $\mathcal{R}_{\text{ZERO}}$ PolyIOP on $\mu + 1$ variables. Given a tuple $(\mathbb{x}; \mathbb{w}) = \big((s, [[f_1]], [[f_2]]); f_1, f_2\big)$, we denote by $f' := f_1/f_2$. The protocol is the following:

- P sends an oracle $\tilde{v} \in \mathcal{F}_{\mu+1}^{(\leq 1)}$ such that for all $\mathbf{x} \in B_\mu$,

$$\tilde{v}(0, \mathbf{x}) = f'(\mathbf{x}), \qquad \tilde{v}(1, \mathbf{x}) = \tilde{v}(\mathbf{x}, 0) \cdot \tilde{v}(\mathbf{x}, 1), \qquad \tilde{v}(\mathbf{1}) = 0.$$

- Define $\hat{h} := \mathsf{merge}(\hat{f}, \hat{g}) \in \mathcal{F}_{\mu+1}^{(\leq \max(2, d+1))}$ where

$$\hat{f}(\mathbf{X}) := \tilde{v}(1, \mathbf{X}) - \tilde{v}(\mathbf{X}, 0) \cdot \tilde{v}(\mathbf{X}, 1), \qquad \hat{g}(\mathbf{X}) := f_2(\mathbf{X}) \cdot \tilde{v}(0, \mathbf{X}) - f_1(\mathbf{X}).$$

  Run a ZeroCheck PolyIOP for $\big([[\hat{h}]]; \hat{h}\big) \in \mathcal{R}_{\text{ZERO}}$, i.e., the polynomial $\tilde{v}$ is computed correctly.
- V queries $[[\tilde{v}]]$ at point $(1, \ldots, 1, 0) \in \mathbb{F}^{\mu+1}$, and checks that the evaluation is s.

Theorem 3.4. *Let $d' := \max(2, d + 1)$. The PIOP for $\mathcal{R}_{PROD}$ is perfectly complete and has knowledge error $\delta_{prod}^{d,\mu} := \delta_{zero}^{d',\mu+1} = \mathcal{O}(d'\mu/|\mathbb{F}|)$.*

Proof. **Completeness.** First, if the prover honestly generates $\tilde{v}$, it holds that $\big(([[\hat{h}]]); \hat{h}\big) \in \mathcal{R}_{\text{ZERO}}$, and the verifier accepts in the sub-PIOP, given that ZeroCheck is complete. Second, if $\big((s, [[f_1]], [[f_2]]); f_1, f_2\big) \in \mathcal{R}_{\text{PROD}}$, the evaluation $\tilde{v}(1, \ldots, 1, 0)$ is exactly the product of f's evaluations on the boolean hypercube B_μ (c.f. [SL20, §5]), which is s as desired.

Knowledge soundness. By Lemma 3.3, it is sufficient to argue the soundness error of the protocol. For any $(s, [[f_1]], [[f_2]]) \notin \mathcal{L}(\mathcal{R}_{\text{PROD}})$ and any $\tilde{v}$ sent by a malicious prover, it holds that either $\tilde{v}$ is not computed correctly (i.e., $\big(([[\hat{h}]]); \hat{h}\big) \notin \mathcal{R}_{\text{ZERO}}$), or the evaluation $\tilde{v}(1, \ldots, 1, 0) \neq s$ and V rejects. Hence the probability that V accepts is at most $\max\big(\delta_{\text{zero}}^{d',\mu+1}, 0\big) = \delta_{\text{zero}}^{d',\mu+1}$ as claimed. □

Complexity. Let $\hat{h}$ be the polynomials described in the construction, we analyze the complexity of the PIOP for $\mathcal{R}_{\text{PROD}}$ with respect to $f' := f_1/f_2$ where $f_1, f_2 \in \mathcal{F}_\mu^{(\leq d)}$.

- The prover time is $\mathsf{tp}_{\text{prod}}^{f'} = \mathsf{tp}_{\text{zero}}^{\hat{h}} + 2^\mu = \mathcal{O}(d \log^2 d \cdot 2^\mu)$ $\mathbb{F}$-ops. The term 2^μ is for computing the product polynomial $\tilde{v}$.
- The verifier time is $\mathsf{tv}_{\text{prod}}^{f'} = \mathsf{tv}_{\text{zero}}^{\hat{h}} = \mathcal{O}(\mu)$.
- The query complexity is $\mathsf{q}_{\text{prod}}^{f'} = \mathsf{q}_{\text{zero}}^{\hat{h}} + 1 = \mu + 2$, the additional query is for $\tilde{v}(1, \ldots, 1, 0)$.
- The round complexity and the number of proof oracles is $\mathsf{rc}_{\text{prod}}^{f'} = \mathsf{rc}_{\text{zero}}^{\hat{h}} + 1 = \mu + 1$.
- The number of field elements sent by P is $\mathsf{nf}_{\text{prod}}^{f'} = \mathsf{nf}_{\text{zero}}^{\hat{h}} = \mu$.
- The size of the proof oracles is $\mathsf{pl}_{\text{prod}}^{f} = 2^\mu + \mathsf{pl}_{\text{zero}}^{\hat{h}} = \mathcal{O}(2^\mu)$; the size of the witness is $\mathcal{O}(2^\mu)$.

3.3.4 Multiset Check PIOP

We describe a multivariate PIOP checking that two multisets are equal. The PIOP builds upon the product-check PIOP in Section 3.3.3. The multiset check PIOP is a key building block for the permutation PIOP in Section 3.3.5 and the lookup PIOP in Section 3.3.7. A similar idea has been proposed in the univariate polynomial setting by Gabizon in a blogpost [Gab].

Definition 3.9 (Multiset Check relation)**.** *For any $k \geq 1$, the relation $\mathcal{R}_{MSET}^k$ is the set of all tuples*

$$(\mathbb{x}; \mathbb{w}) = \big(([[f_1]], \ldots, [[f_k]], [[g_1]], \ldots, [[g_k]]); (f_1, \ldots, f_k, g_1, \ldots, g_k)\big)$$

where $f_i, g_i \in \mathcal{F}_\mu^{(\leq d)}$ $(1 \leq i \leq k)$ and the following two multisets of tuples are equal:

$$\Big\{\mathbf{f_x} := \big[f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})\big]\Big\}_{\mathbf{x} \in B_\mu} = \Big\{\mathbf{g_x} := \big[g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})\big]\Big\}_{\mathbf{x} \in B_\mu}$$

Basic construction. We start by describing a PolyIOP for $\mathcal{R}_{\text{MSET}}^1$. The protocol can be obtained from a protocol for $\mathcal{R}_{\text{PROD}}$. Given a tuple $\big(([[f]], [[g]]); (f, g)\big)$, the protocol is the following:

- V samples and sends P a challenge $r \leftarrow\!\!\$\, \mathbb{F}$.
- Set $f' := r + f$ and $g' := r + g$

- If $g' \neq 0 \forall b \in B_\mu$ run a ProductCheck PolyIOP for $\big((1, [[f']], [[g']]); f', g'\big) \in \mathcal{R}_{\text{PROD}}$.
- Else the prover sends b such that $g'(b) = 0$ and the verifier accepts if $g(b) = -r$ (this case happens with negligible probability).

Theorem 3.5. *The PIOP for $\mathcal{R}^1_{MSET}$ has perfect completeness and has knowledge error $\delta^{d,\mu}_{mset,1} := 2^{\mu+1}/|\mathbb{F}| + \delta^{d,\mu}_{prod} = \mathcal{O}\big((2^\mu + d\mu)/|\mathbb{F}|\big)$.*

Proof. **Completeness.** For any $\big(([[f]], [[g]]); (f, g)\big) \in \mathcal{R}^1_{\text{MSET}}$, it holds that

$$\prod_{\mathbf{x} \in B_\mu} \big(r + f(\mathbf{x})\big) = \prod_{\mathbf{x} \in B_\mu} \big(r + g(\mathbf{x})\big)\,,$$

. If $g(b) = -r$ then the prover will just open g at that point and the verifier accepts. Otherwise $r + g(b) \neq 0, \forall b \in B_\mu$, thus $\prod_{\mathbf{x} \in B_\mu} \big(r + f(\mathbf{x})\big)/\big(r + g(\mathbf{x})\big) = 1$, i.e., $\big((1, [[r + f]], [[r + g]]); r + f, r + g\big) \in \mathcal{R}_{\text{PROD}}$. Therefore completeness holds given that the PolyIOP for $\mathcal{R}_{\text{PROD}}$ is complete.

Knowledge soundness. By Lemma 3.3, it is sufficient to argue the soundness error of the protocol. For any $([[f]], [[g]]) \notin \mathcal{L}(\mathcal{R}^1_{\text{MSET}})$ (i.e., $\big(([[f]], [[g]]); (f, g)\big) \notin \mathcal{R}^1_{\text{MSET}}$), it holds that

$$F(Y) := \prod_{\mathbf{x} \in B_\mu} (Y + f(\mathbf{x})) \neq G(Y) := \prod_{\mathbf{x} \in B_\mu} (Y + g(\mathbf{x}))\,.$$

By Lemma 3.2, $F(r) \neq G(r)$ and $G(r) = 0$ with probability at least $1 - 2 * (2^\mu/|\mathbb{F}|)$. Conditioned on $F(r) \neq G(r)$ and $G(r) \neq 0$, it holds that $\big((1, [[r + f]], [[r + g]]); r + f, r + g\big) \notin \mathcal{R}_{\text{PROD}}$. Hence the probability that V accepts conditioned on $F(r) \neq G(r)$ and $G(r) \neq 0$ is at most $\delta^{d,\mu}_{\text{prod}}$. In summary, the probability that V accepts is at most $2^{\mu+1}/|\mathbb{F}| + \delta^{d,\mu}_{\text{prod}}$ as claimed. □

The final construction. Next we describe the protocol for $\mathcal{R}^k_{\text{MSET}}$ for any $k \geq 1$. Given a tuple

$$\big(([[f_1]], \ldots, [[f_k]], [[g_1]], \ldots, [[g_k]]); (f_1, \ldots, f_k, g_1, \ldots, g_k)\big)\,,$$

the protocol is the following:

- V samples and sends P challenges $r_2, \ldots, r_k \leftarrow\!\!\$\, \mathbb{F}$.
- Run a Multiset Check PolyIOP for $\big(([[\hat{f}]], [[\hat{g}]]); (\hat{f}, \hat{g})\big) \in \mathcal{R}^1_{\text{MSET}}$, where $\hat{f}, \hat{g} \in \mathcal{F}^{(\leq d)}_\mu$ are defined as $\hat{f} := f_1 + r_2 \cdot f_2 + \cdots + r_k \cdot f_k$ and $\hat{g} := g_1 + r_2 \cdot g_2 + \cdots + r_k \cdot g_k$.

Theorem 3.6. *The PIOP for $\mathcal{R}^k_{MSET}$ is perfectly complete and has knowledge error $\delta^{d,\mu}_{mset,k} := 2^\mu/|\mathbb{F}| + \delta^{d,\mu}_{mset,1} = \mathcal{O}\big((2^\mu + d\mu)/|\mathbb{F}|\big)$.*

Proof. **Completeness.** Completeness holds since the PolyIOP for $\big(([[f]], [[g]]); (f, g)\big) \in \mathcal{R}^1_{\text{MSET}}$ is complete.

Knowledge soundness. By Lemma 3.3, it is sufficient to argue the soundness error of the protocol. Given any

$$\left([[f_1]], \ldots, [[f_k]], [[g_1]], \ldots, [[g_k]]\right) \notin \mathcal{L}\left(\mathcal{R}^k_{\text{MSET}}\right) ,$$

let

$$U := \left\{\mathbf{f_x} := [f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})]\right\}_{\mathbf{x}\in B_\mu} , \qquad V := \left\{\mathbf{g_x} := [g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})]\right\}_{\mathbf{x}\in B_\mu}$$

denote the corresponding multisets. Let W be the maximal multiset such that $W \subseteq U$ and $W \subseteq V$. We set $U' := U \setminus W$, $V' := V \setminus W$.[5] We observe that $|U'| = |V'| > 0$ as $U \neq V$, and $U' \cap V' = \emptyset$ by definition of W. Thus there exists an element $\mathbf{x} \in \mathbb{F}^k$ where $\mathbf{x} \in U'$ but $\mathbf{x} \notin V'$. It is well-known that the map $\phi_{\mathbf{r}} : (x_1, \ldots, x_k) \to x_1 + r_2 x_2 + \cdots + r_k x_k$ is a universal hash family [CW77; WC81; Sti94], that is, for any $\mathbf{x}, \mathbf{y} \in \mathbb{F}^k$, $\mathbf{x} \neq \mathbf{y}$, it holds that

$$\Pr_{\mathbf{r}}\left[\phi_{\mathbf{r}}(\mathbf{x}) = \phi_{\mathbf{r}}(\mathbf{y})\right] \leq \frac{1}{|\mathbb{F}|} .$$

Thus by union bound, the probability (over the choice of $\mathbf{r}$) that

$$\phi_{\mathbf{r}}(\mathbf{x}) \in \left\{\phi_{\mathbf{r}}(\mathbf{y}) : \mathbf{y} \in V'\right\}$$

is at most $|V'|/|\mathbb{F}| \leq 2^\mu/|\mathbb{F}|$. Conditioned on that above does not happen, we have that $\left(([[\hat{f}]], [[\hat{g}]])\right) \notin \mathcal{L}(\mathcal{R}^1_{\text{MSET}})$ and the probability that V accepts in the PolyIOP for $\mathcal{R}^1_{\text{MSET}}$ is at most $\delta^{d,\mu}_{\text{mset},1}$. In summary, the soundness error is at most $2^\mu/|\mathbb{F}| + \delta^{d,\mu}_{\text{mset},1}$ as claimed. □

Complexity. We analyze the complexity of the PIOP for $\mathcal{R}_{\text{MSET}}$ with respect to

$$\mathbf{F} := (f_1, \ldots, f_k, g_1, \ldots, g_k) \in \left(\mathcal{F}^{(\leq d)}_\mu\right)^{2k} .$$

- The prover time is $\mathsf{tp}^{\mathbf{F}}_{k,\text{mset}} = \mathsf{tp}^{\hat{f},\hat{g}}_{1,\text{mset}} = \mathsf{tp}^{f'/g'}_{\text{prod}} = \mathcal{O}\left(d \log^2 d \cdot 2^\mu\right)$ $\mathbb{F}$-ops (for k where $\hat{f} := f_1 + r_2 \cdot f_2 + \cdots + r_k \cdot f_k$ and $\hat{g} := g_1 + r_2 \cdot g_2 + \cdots + r_k \cdot g_k$ can be evaluated in time $O(d)$).
- The verifier time is $\mathsf{tv}^{\mathbf{F}}_{\text{mset}} = \mathsf{tv}^{f'/g'}_{\text{prod}} = \mathcal{O}(\mu)$.
- The query complexity is $\mathsf{q}^{\mathbf{F}}_{\text{mset}} = \mathsf{q}^{f'/g'}_{\text{prod}} = \mu + 2$.
- The round complexity and the number of proof oracles is $\mathsf{rc}^{\mathbf{F}}_{\text{mset}} = \mathsf{rc}^{f'/g'}_{\text{prod}} = \mu + 1$.
- The number of field elements sent by P is $\mathsf{nf}^{\mathbf{F}}_{\text{mset}} = \mathsf{nf}^{f'/g'}_{\text{prod}} = \mu$.
- The size of the proof oracles is $\mathsf{pl}^{\mathbf{F}}_{\text{mset}} = \mathsf{pl}^{f'/g'}_{\text{prod}} = \mathcal{O}(2^\mu)$; the size of the witness is $\mathcal{O}(k \cdot 2^\mu)$.

3.3.5 Permutation PIOP

We describe a multivariate PIOP showing that for two multivariate polynomials $f, g \in \mathcal{F}^{(\leq d)}_\mu$, the evaluations of g on the boolean hypercube is a predefined permutation σ of f's evaluations on the

[5] E.g., if $k = 1$ and $U = \{1, 1, 1, 2\}$ and $V = \{1, 1, 2, 2\}$, then $W = \{1, 1, 2\}$, $U' = \{1\}$ and $V' = \{2\}$.

boolean hypercube. The permutation PIOP is a key building block of HyperPlonk for proving the wiring identity.

Definition 3.10 (Permutation relation)**.** *The indexed relation $\mathcal{R}_{PERM}$ is the set of tuples*

$$(\mathbb{i}; \mathbb{x}; \mathbb{w}) = \big(\sigma; ([[f]], [[g]]); (f, g)\big)\,,$$

where $\sigma : B_\mu \to B_\mu$ is a permutation, $f, g \in \mathcal{F}_\mu^{(\leq d)}$, and $g(\mathbf{x}) = f(\sigma(\mathbf{x}))$ for all $\mathbf{x} \in B_\mu$.

Construction. Gabizon et. al. [GWC19] construct a permutation argument. We adapt their scheme into a multivariate PolyIOP. The construction uses a PolyIOP instance for $\mathcal{R}_{\text{MSET}}$. Given a tuple $\big(\sigma; ([[f]], [[g]]); (f, g)\big)$ where σ is the predefined permutation, the indexer generates two oracles $[[s_{\text{id}}]]$, $[[s_\sigma]]$ such that $s_{\text{id}} \in \mathcal{F}_\mu^{(\leq 1)}$ maps each $\mathbf{x} \in B_\mu$ to $[\mathbf{x}] := \sum_{i=1}^{\mu} \mathbf{x}_i \cdot 2^{i-1} \in \mathbb{F}$, and $s_\sigma \in \mathcal{F}_\mu^{(\leq 1)}$ maps each $\mathbf{x} \in B_\mu$ to $[\sigma(\mathbf{x})]$.[6] The PolyIOP is the following:

- Run a Multiset Check PolyIOP for

$$\big(([[s_{\text{id}}]], [[f]], [[s_\sigma]], [[g]]); (s_{\text{id}}, f, s_\sigma, g)\big) \in \mathcal{R}^2_{\text{MSET}}\,.$$

Theorem 3.7. *The PIOP for $\mathcal{R}_{PERM}$ is perfectly complete and has knowledge error $\delta_{perm}^{d,\mu} := \delta_{mset,2}^{d,\mu} = \mathcal{O}\big((2^\mu + d\mu)/|\mathbb{F}|\big)$.*

Proof. **Completeness.** For any $\big(\sigma; ([[f]], [[g]]); (f, g)\big) \in \mathcal{R}_{\text{PERM}}$, it holds that the multiset $\{([\mathbf{x}], f(\mathbf{x}))\}_{\mathbf{x} \in B_\mu}$ is identical to the multiset $\{([\sigma(\mathbf{x})], g(\mathbf{x}))\}_{\mathbf{x} \in B_\mu}$. Thus

$$\big(([[s_{\text{id}}]], [[f]], [[s_\sigma]], [[g]]); (s_{\text{id}}, f, s_\sigma, g)\big) \in \mathcal{R}^2_{\text{MSET}}$$

and completeness follows from the completeness of the PolyIOP for $\mathcal{R}^2_{\text{MSET}}$.

Knowledge soundness. By Lemma 3.3, it is sufficient to argue the soundness error of the protocol. The PolyIOP has soundness error $\delta_{\text{mset},2}^{d,\mu}$ as the permutation relation holds if and only if the above multiset check relation holds. □

Complexity. The complexity of the PIOP for $\mathcal{R}_{\text{PERM}}$ with respect to $f, g \in \mathcal{F}_\mu^{(\leq d)}$ is identical to the complexity of the PIOP for $\mathcal{R}^2_{\text{MSET}}$ with respect to $(s_{\text{id}}, f, s_\sigma, g)$.

3.3.6 Another permutation PIOP for small fields

We describe a different multivariate PIOP for $\mathcal{R}_{\text{PERM}}$. This PIOP directly reduces to $\mathcal{R}_{\text{SUM}}$ and has soundness $O(\mu^2/|\mathbb{F}|)$ instead of $O(2^\mu/|\mathbb{F}|)$. This enables using polynomial-sized fields such as

[6] Here we further require $|\mathbb{F}| \geq 2^\mu$ so that $[\mathbf{x}]$ never overflow.

in [Boo+22a]. The downside is that the PIOP has quasi-linear (rather than linear) prover time. This comes from splitting up the permutation into μ multi-linear polynomials. We emphasize that to obtain linear-time provers, the PIOPs used in Section 3.4 and Section 3.5 are the version in Section 3.3.5, rather than in this section.

For simplicity, we only describe the construction for multi-linear polynomials f, g. It can be easily extended to higher degree polynomials by adding an additional ZeroCheck and proving equivalence between a multi-variate and a multi-linear polynomial on the boolean hypercube.

Construction. Our core idea is similar to the protocol in $\mathcal{R}_{\text{BATCH}}$. Given a permutation $\sigma : B_\mu \to B_\mu$, we reduce the equality check $f(b') = g(\sigma(b'))$ for a given point $b' \in B_\mu$ to a sum-check via multilinear extension. We then use a zero-check over all b' to prove the equality for all $b' \in B_\mu$, which in turn can be reduced to another sumcheck. The resulting sumcheck is over a polynomial with 2μ variables and individual degree $\mu+1$, and naively running the protocol takes time quadratic in 2^μ. Fortunately, we can utilize that σ is a permutation on B_μ and reduce the prover computation to $O(2^\mu \mu \log^2(\mu))$.

We begin by defining a multi-linear version of the permutation σ. $\tilde{\sigma} = (\sigma_1(\mathbf{X}), \ldots, \sigma_\mu(\mathbf{X})) : \mathbb{F}^\mu \to \mathbb{F}^\mu$, such that for all $i \in [\mu]$, σ_i is the multi-linear extension of the ith binary digit of σ. Note that $\sigma_i(\mathbf{x}) \in \{0,1\}$ for all $\mathbf{x} \in B_\mu$ and $i \in [\mu]$. We now rewrite the permutation relation as an $\mathcal{R}_{\text{ZERO}}$ relation for a polynomial equality equation.

$$f(\tilde{\sigma}(\mathbf{x})) - g(\mathbf{x}) = 0 \forall \mathbf{x} \in B_\mu \,.$$

We can now expand $f(\tilde{\sigma}(\mathbf{x})) - g(\mathbf{x})$ to its multilinear extension form just as in the protocol for $\mathcal{R}_{\text{BATCH}}$. This becomes

$$\sum_{\mathbf{y} \in B_\mu} (f(\mathbf{y}) eq(\tilde{\sigma}(\mathbf{x}), \mathbf{y}) - g(\mathbf{y}) eq(\mathbf{x}, \mathbf{y})) = 0 \forall \mathbf{x} \in B_\mu \,.$$

Next, we use the standard trick in Section 3.3.2 to reduce the zerocheck to another sumcheck. Namely, for a random challenge $\mathbf{t} \in \mathbb{F}^\mu$, we check that

$$\sum_{\mathbf{x} \in B_\mu} eq(\mathbf{t}, \mathbf{x}) \cdot \sum_{\mathbf{y} \in B_\mu} (f(\mathbf{y}) eq(\tilde{\sigma}(\mathbf{x}), \mathbf{y}) - g(\mathbf{y}) eq(\mathbf{x}, \mathbf{y})) = 0 \,. \tag{3.8}$$

This is a 2μ-round sumcheck. Unfortunately, we cannot directly evaluate this sumcheck efficiently. To see this, consider the second round of the sumcheck after the verifier has sent a challenge α_1. The prover needs to evaluate

$$eq(\tilde{\sigma}(\alpha_1, x_2, \ldots, x_\mu), \mathbf{y}) = eq((\sigma_1(\alpha_1, x_2, \ldots, x_\mu) \ldots, \sigma_\mu(\alpha_1, x_2, \ldots), x_\mu), \mathbf{y})$$

for all $(x_2, \ldots, x_\mu) \in B_{\mu-1}$ and all $\mathbf{y} \in B_\mu$. This takes time $O(2^{2\mu})$, i.e. quadratic in the size of the permutation.

To remedy this we take advantage of the fact that σ is a permutation and has an inverse permutation σ^{-1} such that $\sigma(\sigma^{-1}(\mathbf{x})) = \mathbf{x} \forall \mathbf{x} \in B_\mu$. We similarly define $\tilde{\sigma}^{-1} : \mathbb{F}^\mu \to \mathbb{F}^\mu$ as the multi-linear version of the inverse permutation σ^{-1}. More precisely, $\tilde{\sigma}^{-1} = (\sigma_1^{-1}(\mathbf{X}), \ldots, \sigma_\mu^{-1}(\mathbf{X})) : \mathbb{F}^\mu \to \mathbb{F}^\mu$, such that for all $i \in [\mu]$, σ_i^{-1} is the multi-linear extension of the ith binary digit of σ^{-1}. We can then rewrite the sumcheck from Equation 3.8 as

$$\sum_{\mathbf{x} \in B_\mu} eq(\mathbf{t}, \mathbf{x}) \cdot \sum_{\mathbf{y} \in B_\mu} (f(\mathbf{y}) eq(\mathbf{x}, \tilde{\sigma}^{-1}(\mathbf{y})) - g(\mathbf{y}) eq(\mathbf{x}, \mathbf{y})) = 0 \tag{3.9}$$

For any $\mathbf{x}$ and $\mathbf{y} \in B_\mu$ the inner sum is $f(\mathbf{y})$ if $\mathbf{x} = \sigma^{-1}(\mathbf{y})$ or equivalently $f(\sigma(\mathbf{x}))$ and $-g(\mathbf{x})$ if $\mathbf{x} = \mathbf{y}$ and 0 otherwise. Since we sum over the entire boolean hypercube for both $\mathbf{x}$ and $\mathbf{y}$ and σ is a permutation on B_μ, (3.9) is equivalent to $\sum_{\mathbf{x} \in B_\mu} eq(\mathbf{t}, \mathbf{x})(f(\sigma(\mathbf{x})) - g(\mathbf{x}))$

The prover can run the sumcheck from (3.9) efficiently by treating $\tilde{\sigma}^{-1}$ as a permutation in the first μ rounds of the sumcheck, where we only consider $\mathbf{x}$ values within the hypercube. In the latter μ rounds $\mathbf{x}$ has been replaced with verifier challenges $\alpha_1, \ldots, \alpha_\mu$ and we can now treat $\tilde{\sigma}^{-1}$ as a collection of μ multi-linear functions. We present the PIOP and the corresponding prover algorithm in Algorithm 3. The verifier is identical to the $\mathcal{R}_{\text{SUM}}$ verifier for the sumcheck equation (3.9).

Algorithm 3 Permutation PIOP with better soundness.

1: **procedure** PERM2 PROVER($f \in \mathcal{F}_\mu^{(\leq 1)}, g \in \mathcal{F}_\mu^{(\leq 1)}, \sigma : B_\mu \to B_\mu$)
2: V sends P a random vector $\mathbf{t} \leftarrow\$ \mathbb{F}^\mu$.
3: Run the sumcheck for μ rounds on the outer sum as described in Algorithm 1. Note that in the ith round, for any given value of $\mathbf{x} \in B_{\mu-i}$, $eq((\alpha_1, \ldots, \alpha_i, \mathbf{x}), \tilde{\sigma}^{-1}(\mathbf{y}))$ is non-zero for at most 2^i values of $\mathbf{y} \in B_\mu$. This is because $\tilde{\sigma}^{-1}$ is a permutation on the boolean hypercube, thus the eq value is non-zero only if the last $\mu - i$ values of $\tilde{\sigma}^{-1}(\mathbf{y})$ are identical to $\mathbf{x}$. Similarly $eq((\alpha_1, \ldots, \alpha_i, \mathbf{x}), \mathbf{y}) \neq 0$ for at most 2^i values of $\mathbf{y}$ for any given $\mathbf{x}$. The prover can, therefore, evaluate all inner sumchecks in each round in time $O(2^\mu)$. The prover runs in time $O(\mu 2^\mu)$.
4: Run the inner sumcheck with $\mathbf{x} = \boldsymbol{\alpha}$ as described in Algorithm 1 and treating $\tilde{\sigma}_1^{-1}, \ldots, \tilde{\sigma}_\mu^{-1}$ as multi-linear polynomials. The prover runs in time $O(2^\mu \mu \log^2(\mu))$ as the sumcheck has μ rounds and degree μ.
5: **end procedure**

Theorem 3.8. *The PERM2 PIOP described by the sumcheck for Equation* (3.9) *and the corresponding prover from Algorithm 3 for* $\mathcal{R}_{PERM}$ *is perfectly complete and has knowledge error* $\delta_{perm2}^{1,\mu} := \delta_{sum}^{\mu+1,2\mu} = O(\mu^2/|\mathbb{F}|)$.

Proof. **Completeness.** For any $\big(\sigma; ([[f]], [[g]]); (f, g)\big) \in \mathcal{R}_{\text{PERM}}$, it holds that $f(\sigma(\mathbf{x})) - g(\mathbf{x}) = 0$ for all $\mathbf{x} \in B_\mu$. Moreover, for all $\mathbf{x}, \mathbf{y} \in B_\mu$, it holds that $eq(\mathbf{x}, \tilde{\sigma}^{-1}(\mathbf{y})) = 1$ if $\sigma(\mathbf{x}) = \mathbf{y}$ and $eq(\mathbf{x}, \tilde{\sigma}^{-1}(\mathbf{y})) = 0$ otherwise; and $eq(\mathbf{x}, \mathbf{y}) = 1$ if $\mathbf{x} = \mathbf{y}$ and $eq(\mathbf{x}, \mathbf{y}) = 0$ otherwise. Thus for

$h(\mathbf{x}) = \sum_{\mathbf{y}\in B_\mu} f(\mathbf{y}) \cdot eq(\mathbf{x}, \tilde{\sigma}^{-1}(\mathbf{y})) - g(\mathbf{y}) eq(\mathbf{x}, \mathbf{y})$

$$\big([[h]]; h\big) \in \mathcal{R}_{\text{ZERO}}$$

and completeness follows from the completeness of the PolyIOP for $\mathcal{R}_{\text{ZERO}}$.

Knowledge soundness. By Lemma 3.3, it is sufficient to argue the soundness error of the protocol. The permutation relation holds if and only if the above zerocheck relation holds, which reduces to a sumcheck for Equation (3.9). The sumcheck PolyIOP is over a virtual polynomial that has 2μ variables and individual degree $\mu + 1$. Thus the soundness error is $\delta^{\mu+1,2\mu}_{\mathcal{R}_{\text{SUM}}} = O(\mu^2/|\mathbb{F}|)$. □

Complexity. The prover complexity of the PERM2 PIOP with respect to $f, g \in \mathcal{F}_\mu^{(\leq 1)}$ is $O(2^\mu \mu \log^2(\mu))$ as the prover uses $O(\mu 2^\mu)$ in step 3 and $O(2^\mu \mu \log^2(\mu))$ in step 4 of the algorithm. The verifier is simply the sumcheck verifier for a 2μ round sumcheck and runs in time $O(\mu)$. The verifier queries 2μ univariate polynomials and $2 + \mu$ multi-linear polynomials $(f, g, \tilde{\sigma}^{-1})$ each at one point. The PIOP proof size consists of 2μ univariate oracles (one per round of the sumcheck).

3.3.7 Lookup PIOP

This section describes a multivariate PIOP checking the table lookup relation. The PIOP builds upon the multiset check PIOP (Section 3.3.4) and is a key building block for HyperPlonk+ (Section 3.5). Our construction is inspired by a univariate PIOP for the table lookup relation called Plookup [GW20a]. However, it is non-trivial to adapt Plookup to the multivariate setting because their scheme requires the existence of a subdomain of the polynomial that is a cyclic subgroup $\mathbb{G}$ with a generator $\omega \in \mathbb{G}$. Translating to the multilinear case, we need to build an efficient function g that generates the entire boolean hypercube; moreover, g has to be linear so that the degree of the polynomial does not blow up. However, such a linear function does not exist. Fortunately, we can construct a quadratic function from $\mathbb{F}^\mu$ to $\mathbb{F}^\mu$ that traverses B_μ. We then show how to linearize it by modifying some of the building blocks that Plookup uses. This gives an efficient Plookup protocol over the hypercube.

Definition 3.11 (Lookup relation)**.** *The indexed relation $\mathcal{R}_{LOOKUP}$ is the set of tuples*

$$(\mathbb{i}; \mathbb{x}; \mathbb{w}) = \big(\mathbf{t}; [[f]]; (f, \mathsf{addr})\big)$$

where $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$, $f \in \mathcal{F}_\mu^{(\leq d)}$, and $\mathsf{addr} : B_\mu \to [1, 2^\mu)$ is a map such that $f(\mathbf{x}) = \mathbf{t}_{\mathsf{addr}(\mathbf{x})}$ for all $\mathbf{x} \in B_\mu$.

Before presenting the PIOP for $\mathcal{R}_{\text{LOOKUP}}$, we first show how to build a *quadratic* function that generates the entire boolean hypercube.

A quadratic generator in $\mathbb{F}_{2^\mu}$. For every $\mu \in \mathbb{N}$, we fix a *primitive polynomial* $p_\mu \in \mathbb{F}_2[X]$ where $p_\mu := X^\mu + \sum_{s\in S} X^s + 1$ for some set $S \subseteq [\mu - 1]$, so that $\mathbb{F}_2[X]/(p_\mu) \cong \mathbb{F}_2^\mu[X] \cong \mathbb{F}_{2^\mu}$. By definition of primitive polynomials, $X \in \mathbb{F}_2^\mu[X]$ is a generator of $\mathbb{F}_2^\mu[X]\setminus\{0\}$. This naturally defines a generator function $g_\mu : B_\mu \to B_\mu$ as

$$g_\mu(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_\mu) = (\boldsymbol{b}_\mu, \boldsymbol{b}'_1, \ldots, \boldsymbol{b}'_{\mu-1}),$$

where $\boldsymbol{b}'_i = \boldsymbol{b}_i \oplus \boldsymbol{b}_\mu$ $(i \le 1 < \mu)$ if $i \in S$, and $\boldsymbol{b}'_i = \boldsymbol{b}_i$ otherwise. Essentially, for a polynomial $f \in \mathbb{F}_2^\mu[X]$ with coefficients $\boldsymbol{b}$, $g_\mu(\boldsymbol{b})$ is the coefficient vector of $X \cdot f(X)$. Hence the following lemma is straightforward.

Lemma 3.4. *Let $g_\mu : B_\mu \to B_\mu$ be the generator function defined above. For every $\mathbf{x} \in B_\mu \setminus \{0^\mu\}$, it holds that $\{g_\mu^{(i)}(\mathbf{x})\}_{i\in[2^\mu-1]} = B_\mu \setminus \{0^\mu\}$, where $g_\mu^{(i)}(\cdot)$ denotes i repeated application of g_μ.*

Directly composing a polynomial f with the generator g will blow up the degree of the resulting polynomial; moreover, the prover needs to send the composed oracle $f(g(\cdot))$. Both of which affect the efficiency of the PIOP. We address the issue by describing a trick that manipulates f in a way that simulates the behavior of $f(g(\cdot))$ on the boolean hypercube, but without blowing up the degree.

Linearizing the generator. For a multivariate polynomial $f \in \mathcal{F}_\mu^{(\le d)}$, we define $f_{\Delta_\mu} \in \mathcal{F}_\mu^{(\le d)}$ as

$$f_{\Delta_\mu}(\mathbf{X}_1, \ldots, \mathbf{X}_\mu) := \mathbf{X}_\mu \cdot f(1, \mathbf{X}'_1, \ldots, \mathbf{X}'_{\mu-1}) + (1 - \mathbf{X}_\mu) \cdot f(0, \mathbf{X}_1, \ldots, \mathbf{X}_{\mu-1})$$

where $\mathbf{X}'_i := 1 - \mathbf{X}_i$ $(i \le 1 < \mu)$ if $i \in S$, and $\mathbf{X}'_i := \mathbf{X}_i$ otherwise.

Lemma 3.5. *For every $\mu \in \mathbb{N}$, let $g_\mu : B_\mu \to B_\mu$ be the generator function defined in Lemma 3.4. For every $d \in \mathbb{N}$ and polynomial $f \in \mathcal{F}_\mu^{(\le d)}$, it holds that $f_{\Delta_\mu}(\mathbf{x}) = f(g_\mu(\mathbf{x}))$ for every $\mathbf{x} \in B_\mu$. Moreover, f_{Δ_μ} has individual degree d and one can evaluate f_{Δ_μ} from 2 evaluations of f.*

Proof. By definition, f_{Δ_μ} has individual degree d and an evaluation of f_{Δ_μ} can be derived from 2 evaluations of f. Next, we argue that $f_{\Delta_\mu}(\mathbf{x}) = f(g_\mu(\mathbf{x}))$ for every $\mathbf{x} \in B_\mu$.

First, $f_{\Delta_\mu}(0^\mu) = f(g_\mu(0^\mu))$ because $f_{\Delta_\mu}(0^\mu) = f(0^\mu)$ and $g_\mu(0^\mu) = 0^\mu$ by definition of f_{Δ_μ}, g_μ. Second, for every $\mathbf{x} \in B_\mu \setminus \{0^\mu\}$, by definition of g_μ,

$$f(g_\mu(\mathbf{x}_1, \ldots, \mathbf{x}_\mu)) = f(\mathbf{x}_\mu, \mathbf{x}'_1, \ldots, \mathbf{x}'_{\mu-1}),$$

where $\mathbf{x}'_i = \mathbf{x}_i \oplus \mathbf{x}_\mu$ $(i \le 1 < \mu)$ for every i in the fixed set S, and $\mathbf{x}'_i = \mathbf{x}_i$ otherwise. We observe that $\mathbf{x}_i \oplus \mathbf{x}_\mu = 1 - \mathbf{x}_i$ when $\mathbf{x}_\mu = 1$ and $\mathbf{x}_i \oplus \mathbf{x}_\mu = \mathbf{x}_i$ when $\mathbf{x}_\mu = 0$, thus we can rewrite

$$\begin{aligned} f(\mathbf{x}_\mu, \mathbf{x}'_1, \ldots, \mathbf{x}'_{\mu-1}) &= \mathbf{x}_\mu \cdot f(1, \mathbf{x}^*_1, \ldots, \mathbf{x}^*_{\mu-1}) + (1 - \mathbf{x}_\mu) \cdot f(0, \mathbf{x}_1, \ldots, \mathbf{x}_{\mu-1}) \\ &= f_{\Delta_\mu}(\mathbf{x}_1, \ldots, \mathbf{x}_\mu) \end{aligned}$$

where $\mathbf{x}_i^* = 1 - \mathbf{x}_i$ ($i \le 1 < \mu$) for every i in the fixed set S, and $\mathbf{x}_i^* = \mathbf{x}_i$ otherwise. The last equality holds by definition of f_{Δ_μ}. In summary, $f(g_\mu(\mathbf{x}_1, \ldots, \mathbf{x}_\mu)) = f_{\Delta_\mu}(\mathbf{x}_1, \ldots, \mathbf{x}_\mu)$ for every B_μ and the lemma holds. □

Construction. Now we are ready to present the PIOP for $\mathcal{R}_{\text{LOOKUP}}$, which is an adaptation of Plookup [GW20a] in the multivariate setting. The PIOP invokes a protocol for $\mathcal{R}^2_{\text{MSET}}$. We introduce a notation that embeds a vector to the hypercube while still preserving the vector order with respect to the generator function. For a vector $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$, we denote by $t \leftarrow \mathsf{emb}(\mathbf{t}) \in \mathcal{F}_\mu^{(\le 1)}$ the multilinear polynomial such that $t(0^\mu) = 0$ and $t\big(g_\mu^{(i)}(1, 0^{\mu-1})\big) = \mathbf{t}_i$ for every $i \in [2^\mu - 1]$. By Lemma 3.4, t is well-defined and embeds the entire vector $\mathbf{t}$ onto $B_\mu \setminus \{0^\mu\}$.

For an index $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$, the indexer generates an oracle $[[t]]$ where $t \leftarrow \mathsf{emb}(\mathbf{t})$. For a tuple $\big(\mathbf{t}; [[f]]; (f, \mathsf{addr})\big)$ where $f(B_\mu) \subseteq t(B_\mu) \setminus \{0\}$, let $(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{2^\mu - 1})$ be the vector where $\boldsymbol{a}_i \in \mathbb{N}$ is the number of appearance of $\mathbf{t}_i$ in $f(B_\mu)$. Note that $\sum_{i=1}^{2^\mu - 1} \boldsymbol{a}_i = 2^\mu$. Denote by $\mathbf{h} \in \mathbb{F}^{2^{\mu+1}-1}$ the vector

$$\mathbf{h} := \big(\underbrace{\mathbf{t}_1, \ldots, \mathbf{t}_1}_{1+\boldsymbol{a}_1}, \mathbf{t}_2, \ldots, \mathbf{t}_{i-1}, \underbrace{\mathbf{t}_i, \ldots, \mathbf{t}_i}_{1+\boldsymbol{a}_i}, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{2^\mu - 2}, \underbrace{\mathbf{t}_{2^\mu - 1}, \ldots \mathbf{t}_{2^\mu - 1}}_{1+\boldsymbol{a}_{2^\mu - 1}}\big) .$$

We present the protocol below:

- P sends V oracles $[[h]]$, where $h \leftarrow \mathsf{emb}(\mathbf{h}) \in \mathcal{F}_{\mu+1}^{(\le 1)}$.
- Define $g_1 := \mathsf{merge}(f, t) \in \mathcal{F}_{\mu+1}^{(\le d)}$ and $g_2 := \mathsf{merge}(f, t_{\Delta_\mu}) \in \mathcal{F}_{\mu+1}^{(\le d)}$, where merge is defined in equation (3.7). Run a multiset check PIOP (Section 3.3.4) for

$$\big(([[g_1]], [[g_2]], [[h]], [[h_{\Delta_{\mu+1}}]])\,; (f, t, h)\big) \in \mathcal{R}^2_{\text{MSET}} .$$

- V queries $h(0^{\mu+1})$ and checks that the answer equals 0.

Theorem 3.9. *The PIOP for $\mathcal{R}_{LOOKUP}$ is perfectly complete and has knowledge error $\delta_{lkup}^{d,\mu} := \delta_{mset,2}^{d,\mu+1} = \mathcal{O}\big((2^\mu + d\mu)/|\mathbb{F}|\big)$.*

Proof. **Completeness.** Denote by $n := 2^\mu$. For any $\big(\mathbf{t}; [[f]]; (f, \mathsf{addr})\big) \in \mathcal{R}_{\text{LOOKUP}}$, let $\mathbf{h} \in \mathbb{F}^{2n-1}$ be the vector defined in the construction. Gabizon and Williamson [GW20a] observed that

$$\big\{\mathbf{f_i}, \mathbf{f_i}\big\}_{i \in [n]} \cup \big\{\mathbf{t_i}, \mathbf{t}_{(\mathbf{i} \bmod (\mathbf{n-1}))+\mathbf{1}}\big\}_{i \in [n-1]} = \big\{\mathbf{h_i}, \mathbf{h}_{(\mathbf{i} \bmod (\mathbf{2n-1}))+\mathbf{1}}\big\}_{i \in [2n-1]} ,$$

equivalently, by definition of t, h and by Lemma 3.5, the following two multisets of tuples are equal

$$\big\{\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x})\big\}_{\mathbf{x} \in B_\mu} \cup \big\{\mathbf{t}(\mathbf{x}), \mathbf{t}_{\boldsymbol{\Delta}_\mu}(\mathbf{x})\big\}_{\mathbf{x} \in B_\mu \setminus \{0^\mu\}} = \big\{\mathbf{h}(\mathbf{x}), \mathbf{h}_{\boldsymbol{\Delta}_{\mu+1}}(\mathbf{x})\big\}_{\mathbf{x} \in B_{\mu+1} \setminus \{0^{\mu+1}\}} .$$

By adding element $\mathbf{0}, \mathbf{0} = \mathbf{t}(\mathbf{0}^\mu), \mathbf{t}_{\mathbf{\Delta}_\mu}(\mathbf{0}^\mu) = \mathbf{h}(\mathbf{0}^{\mu+\mathbf{1}}), \mathbf{h}_{\mathbf{\Delta}_{\mu+\mathbf{1}}}(\mathbf{0}^{\mu+\mathbf{1}})$ on both sides, we have

$$\left\{\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x})\right\}_{\mathbf{x}\in B_\mu} \cup \left\{\mathbf{t}(\mathbf{x}), \mathbf{t}_{\mathbf{\Delta}_\mu}(\mathbf{x})\right\}_{\mathbf{x}\in B_\mu} = \left\{\mathbf{h}(\mathbf{x}), \mathbf{h}_{\mathbf{\Delta}_{\mu+\mathbf{1}}}(\mathbf{x})\right\}_{\mathbf{x}\in B_{\mu+1}}.$$

Hence the verifier accepts in the multiset check by completeness of the PIOP for $\mathcal{R}^2_{\text{MSET}}$.

Knowledge soundness. By Lemma 3.3, to argue knowledge soundness, it is sufficient to argue the soundness error of the protocol. Fix $n := 2^\mu$, for any $\left(\mathbf{t}; [[f]]\right) \notin \mathcal{L}(\mathcal{R}_{\text{LOOKUP}})$, denote by $\mathbf{f} \in \mathbb{F}^n$ the evaluations of f on B_μ. Gabizon et. al. [GW20a] showed that for any $\mathbf{h} \in \mathbb{F}^{2n-1}$, it holds that

$$\left\{\mathbf{f_i}, \mathbf{f_i}\right\}_{i\in[n]} \cup \left\{\mathbf{t_i}, \mathbf{t}_{(\mathbf{i} \bmod (\mathbf{n-1}))+\mathbf{1}}\right\}_{i\in[n-1]} \neq \left\{\mathbf{h_i}, \mathbf{h}_{(\mathbf{i} \bmod (\mathbf{2n-1}))+\mathbf{1}}\right\}_{i\in[2n-1]},$$

since $t(0^\mu) = 0$ and V checks that $h(0^{\mu+1}) = 0$, with a similar argument as in the completeness proof, we have

$$\left\{\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x})\right\}_{\mathbf{x}\in B_\mu} \cup \left\{\mathbf{t}(\mathbf{x}), \mathbf{t}_{\mathbf{\Delta}_\mu}(\mathbf{x})\right\}_{\mathbf{x}\in B_\mu} \neq \left\{\mathbf{h}(\mathbf{x}), \mathbf{h}_{\mathbf{\Delta}_{\mu+\mathbf{1}}}(\mathbf{x})\right\}_{\mathbf{x}\in B_{\mu+1}}$$

and the multiset check relation does not hold. Therefore, the probability that V accepts is at most $\delta^{d,\mu+1}_{\text{mset},2}$ as claimed. □

Complexity. Let f, $\mathbf{F} := (g_1, g_2, h, h_{\Delta_{\mu+1}}) \in \left(\mathcal{F}^{(\leq d)}_{\mu+1}\right)^2 \times \left(\mathcal{F}^{(\leq 1)}_{\mu+1}\right)^2$ be the polynomials defined in the construction. We analyze the complexity of the PIOP for $\mathcal{R}_{\text{LOOKUP}}$ with respect to $f \in \mathcal{F}^{(\leq d)}_\mu$.

- The prover time is $\mathsf{tp}^f_{\text{lkup}} = \mathsf{tp}^{\mathbf{F}}_{\text{mset}} = \mathcal{O}\left(d \log^2 d \cdot 2^\mu\right)$ $\mathbb{F}$-ops.
- The verifier time is $\mathsf{tv}^f_{\text{lkup}} = \mathsf{tv}^{\mathbf{F}}_{\text{mset}} = \mathcal{O}(\mu)$.
- The query complexity is $\mathsf{q}^f_{\text{lkup}} = 1 + \mathsf{q}^{\mathbf{F}}_{\text{mset}} = \mu + 3$.
- The round complexity and the number of proof oracles is $\mathsf{rc}^f_{\text{lkup}} = 1 + \mathsf{rc}^{\mathbf{F}}_{\text{mset}} = \mu + 2$.
- The number of field elements sent by P is $\mathsf{nf}^f_{\text{lkup}} = \mathsf{nf}^{\mathbf{F}}_{\text{mset}} = \mu$.
- The size of the proof oracles is $\mathsf{pl}^f_{\text{lkup}} = 2^{\mu+1} + \mathsf{pl}^{\mathbf{F}}_{\text{mset}} = \mathcal{O}(2^\mu)$ where $2^{\mu+1}$ is the oracle size of h. The size of the witness is $\mathcal{O}(2^\mu)$.

3.3.8 Batch openings

This section describes a batching protocol proving the correctness of multiple multivariate polynomial evaluations. Essentially, the protocol reduces multiple oracle queries to different polynomials into a *single* query to a multivariate oracle. The batching protocol is helpful for HyperPlonk to enable efficient batch evaluation openings. In particular, the SNARK prover only needs to compute a single multilinear PCS evaluation proof, even if there are multiple PCS evaluations.

We note that Thaler [Tha20, §4.5.2] shows how to batch two evaluations of a *single* multilinear polynomial. The algorithm can be generalized for multiple evaluations of *different* multilinear polynomials. However, the prover time complexity is $O(k^2\mu \cdot 2^\mu)$ where k is the number of evaluations,

and μ is the number of variables. In comparison, our algorithm achieves complexity $O(k \cdot 2^\mu)$ which is $k\mu$-factor faster. Note that $O(k \cdot 2^\mu)$ is already optimal as the prover needs to take $O(k \cdot 2^\mu)$ time to evaluate $\{f_i(\mathbf{z}_i)\}_{i\in[k]}$ before batching.

Definition 3.12 (BatchEval relation)**.** *The relation $\mathcal{R}^k_{BATCH}$ is the set of all tuples $(\mathbb{x}; \mathbb{w}) = \big((\mathbf{z}_i)_{i\in[k]}, (y_i)_{i\in[k]}, \big([[f_i]]\big)_{i\in[k]}; (f_i)_{i\in[k]}\big)$ where $\mathbf{z}_i \in \mathbb{F}^\mu$, $y_i \in \mathbb{F}$, $f_i \in \mathcal{F}_\mu^{(\leq d)}$ and $f_i(\mathbf{z}_i) = y_i$ for all $i \in [k]$.*

Remark 3.3.1. *The polynomials $\{f_i\}_{i\in[k]}$ are not necessarily distinct. E.g., to evaluate a single polynomial f at k distinct points, we can set $f_1 = f_2 = \cdots = f_k = f$.*

Remark 3.3.2. *The polynomials $\{f_i\}_{i\in[k]}$ are all μ-variate. This is without loss of generality. E.g., suppose one of the evaluated polynomial f'_j has only $\mu - 1$ variables, we can define $f_j(Y, \mathbf{X}) = Y \cdot f'_j(\mathbf{X}) + (1-Y) \cdot f'_j(\mathbf{X})$ which is essentially f'_j but with μ variables. The same trick easily extends to f'_j with arbitrary $\mu' < \mu$ variables.*

Construction. For ease of exposition, we consider the case where $f_1, \ldots, f_k$ are *multilinear*. We emphasize that the same techniques can be extended for multi-variate polynomials.

Assume w.l.o.g that $k = 2^\ell$ is a power of 2. We observe that $\mathcal{R}^k_{\text{BATCH}}$ is essentially a ZeroCheck relation over the set $Z := \{\mathbf{z}_i\}_{i\in[k]} \subseteq \mathbb{F}^\mu$, that is, for every $i \in [k]$, $f_i(\mathbf{z}_i) - y_i = 0$. Nonetheless, Z is outside the boolean hypercube, and we cannot directly reuse the ZeroCheck PIOP.

The key idea is to interpret each zero constraint as a sumcheck via multilinear extension, so that we can work on the boolean hypercube later. In particular, for every $i \in [k]$, we want to constrain $f_i(\mathbf{z}_i) - y_i = 0$. Since f_i is multilinear, by definition of multilinear extension, this is equivalent to constraining that

$$c_i := \left(\sum_{\boldsymbol{b}\in B_\mu} f_i(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \mathbf{z}_i)\right) - y_i = 0\,. \tag{3.10}$$

Note that equation (3.10) holds for every $i \in [k]$ if and only if the polynomial

$$\sum_{i\in[k]} eq(\mathbf{Z}, \langle i\rangle) \cdot c_i$$

is identically zero, where $\langle i\rangle$ is ℓ-bit representation of $i - 1$. By Lemma 3.2, it is sufficient to check that for a random vector $\mathbf{t} \leftarrow\!\!\$\, \mathbb{F}^\ell$, it holds that

$$\sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot c_i = \sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot \left[\left(\sum_{\boldsymbol{b}\in B_\mu} f_i(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \mathbf{z}_i)\right) - y_i\right] = 0\,. \tag{3.11}$$

Next, we arithmetize equation (3.11) and make it an algebraic formula. For every $(i, \boldsymbol{b}) \in [k] \times B_\mu$, we set value $g_{i,\boldsymbol{b}} := eq(\mathbf{t}, \langle i\rangle) \cdot f_i(\boldsymbol{b})$, and define an MLE $\tilde{g}$ for $(g_{i,\boldsymbol{b}})_{i\in[k],\, \boldsymbol{b}\in B_\mu}$ such that

$\tilde{g}(\langle i \rangle, \boldsymbol{b}) = g_{i,\boldsymbol{b}} \forall (i, \boldsymbol{b}) \in [k] \times B_\mu$; similarly, we define an MLE $\tilde{eq}$ for $(eq(\boldsymbol{b}, \mathbf{z}_i))_{i\in[k],\, \boldsymbol{b}\in B_\mu}$ where $\tilde{eq}(\langle i \rangle, \boldsymbol{b}) = eq(\boldsymbol{b}, \mathbf{z}_i) \forall (i, \boldsymbol{b}) \in [k] \times B_\mu$. Let $s := \sum_{i\in[k]} eq(\mathbf{t}, \langle i \rangle) \cdot y_i$, then equation (3.11) can be rewritten as

$$\sum_{i\in[k], \boldsymbol{b}\in B_\mu} \tilde{g}(\langle i \rangle, \boldsymbol{b}) \cdot \tilde{eq}(\langle i \rangle, \boldsymbol{b}) = s\,.$$

This is equivalent to prove a sumcheck claim for the degree-2 polynomial $g^* := \tilde{g}(\mathbf{Y}, \mathbf{X}) \cdot \tilde{eq}(\mathbf{Y}, \mathbf{X})$ over set $B_{\ell+\mu}$. Hence we obtain the following PIOP protocol in Algorithm 4. Note that $g^* = \tilde{g} \cdot \tilde{eq}$ is only with degree 2. Thus we can run a classic sumcheck without sending any univariate oracles.

Algorithm 4 Batch evaluation of multi-linear polynomials

procedure BATCHEVAL($[f_i \in \mathcal{F}_\mu^{(\leq 1)}, \mathbf{z}_i \in \mathbb{F}^\mu, y_i \in \mathbb{F}]_{i=1}^k$)
  V sends P a random vector $\mathbf{t} \leftarrow\$ \mathbb{F}^\ell$.
  Define sum $s := \sum_{i\in[k]} eq(\mathbf{t}, \langle i \rangle) \cdot y_i$.
  Let $\tilde{g}$ be the MLE for $(g_{i,\boldsymbol{b}})_{i\in[k],\, \boldsymbol{b}\in B_\mu}$ where

$$g_{i,\boldsymbol{b}} := eq(\mathbf{t}, \langle i \rangle) \cdot f_i(\boldsymbol{b})\,.$$

  Let $\tilde{eq}$ be the MLE for $(eq(\boldsymbol{b}, \mathbf{z}_i))_{i\in[k],\, \boldsymbol{b}\in B_\mu}$ such that $\tilde{eq}(\langle i \rangle, \boldsymbol{b}) = eq(\boldsymbol{b}, \mathbf{z}_i)$.
  P and V run a SumCheck PIOP for $\left(s, [[g^*]]; g^*\right) \in \mathcal{R}_{\text{SUM}}$, where $g^* := \tilde{g} \cdot \tilde{eq}$.
  Let $(\boldsymbol{a}_1, \boldsymbol{a}_2) \in \mathbb{F}^{\ell+\mu}$ be the sumcheck challenge vector. P answers the oracle query $\tilde{g}(\boldsymbol{a}_1, \boldsymbol{a}_2)$.
  V evaluates $\tilde{eq}(\boldsymbol{a}_1, \boldsymbol{a}_2)$ herself, and checks that

$$\tilde{g}(\boldsymbol{a}_1, \boldsymbol{a}_2) \cdot \tilde{eq}(\boldsymbol{a}_1, \boldsymbol{a}_2)$$

  is consistent with the last message of the sumcheck.
end procedure

Remark 3.3.3. *If the SNARK is using a homomorphic commitment scheme, to answer query $\tilde{g}(\boldsymbol{a}_1, \boldsymbol{a}_2)$ the prover only needs to provide a single PCS opening proof for a μ-variate polynomial*

$$g'(\mathbf{X}) := \tilde{g}(\boldsymbol{a}_1, \mathbf{X}) = \sum_{i\in[k]} eq(\langle i \rangle, \boldsymbol{a}_1) \cdot eq(\mathbf{t}, \langle i \rangle) \cdot f_i(\mathbf{X})$$

on point $\boldsymbol{a}_2$. The verifier can evaluate $\{eq(\langle i \rangle, \boldsymbol{a}_1) \cdot eq(\mathbf{t}, \langle i \rangle)\}_{i\in[k]}$ in time $O(k)$, and homomorphically compute g''s commitment from the commitments to $\{f_i\}_{i\in[k]}$, and checks the opening proof against g''s commitment. Finally, the verifier checks that $g'(\boldsymbol{a}_2)$ matches the claimed evaluation $\tilde{g}(\boldsymbol{a}_1, \boldsymbol{a}_2)$.

Analysis. The PIOP for $\mathcal{R}_{\text{BATCH}}$ is complete and knowledge-sound given the completeness and knowledge-soundness of the sumcheck PIOP.

Next, we analyze the complexity of the protocol: The prover time is $O(k \cdot 2^\mu)$ as it runs a sumcheck PIOP for a polynomial $g^* := \tilde{g} \cdot \tilde{eq}$ of degree 2 and $\mu + \log k$ variables, where $\tilde{g}$ and

$\tilde{eq}$ can both be constructed in time $O(k \cdot 2^\mu)$. Note that this is already optimal as the prover anyway needs to take $O(k \cdot 2^\mu)$ time to evaluate $\{f_i(\mathbf{z}_i)\}_{i\in[k]}$ before batching. The verifier takes time $O(\mu + \log k)$ in the sumcheck; the sum s can be computed in time $O(k)$; the evaluation $\tilde{eq}(\boldsymbol{a}_1, \boldsymbol{a}_2) = \sum_{i\in[k]} eq(\boldsymbol{a}_1, \langle i\rangle) \cdot \tilde{eq}(\langle i\rangle, \boldsymbol{a}_2)$ can be derived from $\boldsymbol{a}_1$ and the k evaluations $\{\tilde{eq}(\langle i\rangle, \boldsymbol{a}_2) = eq(\boldsymbol{a}_2, \mathbf{z}_i)\}_{i\in[k]}$ where each evaluation $eq(\boldsymbol{a}_2, \mathbf{z}_i)$ takes time $O(\mu)$. In summary, the verifier time is $O(k\mu)$.

A more efficient batching scheme in a special setting

Sometimes one only needs to open a *single* multilinear polynomial at multiple points, where each point is *in the boolean hypercube*. In this setting, we provide a more efficient algorithm with complexity $O(2^\mu)$ which is k times faster than Algorithm 4. We also note that the technique can be used to construct an efficient Commit-and-Prove SNARK scheme from multilinear commitments.

Recall the sumcheck equation (3.11) in the general batch opening scheme, when there is only one polynomial f and assume for simplicity that $y_i = 0 \forall i \in [k]$[7], we can rewrite it as

$$\sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot \left(\sum_{\boldsymbol{b}\in B_\mu} f(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \mathbf{z}_i) \right) = \sum_{\boldsymbol{b}\in B_\mu} f(\boldsymbol{b}) \left(\sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot eq(\boldsymbol{b}, \mathbf{z}_i) \right).$$

Denote by $d_i = eq(\mathbf{t}, \langle i\rangle)$. The above is essentially a sumcheck for polynomial $f \cdot \tilde{eq^*}$ on set B_μ, where

$$\tilde{eq^*}(\mathbf{X}) := \sum_{i\in[k]} d_i \cdot eq(\mathbf{X}, \mathbf{z}_i).$$

Thus we can reduce the batching argument to a PCS opening on polynomial f.

In the sumcheck protocol, in each round $\mu - i + 1 \in [\mu]$, the prover needs to evaluate a degree-2 polynomial $r_i(X)$ on point $x_i \in \{0, 1, 2\}$, where

$$r_i(X) := \sum_{\boldsymbol{b}\in B_{i-1}} f(\boldsymbol{b}, X, \boldsymbol{\alpha}) \cdot \tilde{eq^*}(\boldsymbol{b}, X, \boldsymbol{\alpha}) \tag{3.12}$$

and $\boldsymbol{\alpha} = (\alpha_{i+1}, \ldots, \alpha_\mu)$ are the round challenges. Note that the evaluation $f(\boldsymbol{b}, x_i, \boldsymbol{\alpha})$ is easy to obtain by maintaining a table $f(B_{i-1}, \{0, 1, 2\}, \boldsymbol{\alpha})$ as in Algorithm 1. Next we argue that the evaluation $r_i(x_i)$ can be computed in time $O(k)$ given the evaluations $f(B_{i-1}, \{0, 1, 2\}, \boldsymbol{\alpha})$. Since there are μ rounds and the complexity for maintaining the table is $O(2^\mu)$, the total complexity is $O(2^\mu + k\mu)$.

[7]The algorithm can be easily extended when y_i are non-zero.

We observe that in equation (3.12), since $\{\mathbf{z}_i\}_{i\in[k]}$ are in the boolean hypercube, and

$$\begin{aligned}\tilde{eq^*}(\boldsymbol{b}, X, \boldsymbol{\alpha}) &= \sum_{j\in[k]} d_i \cdot eq((\boldsymbol{b}, X, \boldsymbol{\alpha}), \mathbf{z}_j) \\ &= \sum_{j\in[k]} d_j \cdot eq(\boldsymbol{b}, \mathbf{z}_j[1..i-1]) \cdot eq(X, \mathbf{z}_j[i]) \cdot eq(\boldsymbol{\alpha}, \mathbf{z}_j[i+1..])\,,\end{aligned}$$

by definition of eq, there are at most k choices of $\boldsymbol{b}$ where $\tilde{eq^*}(\boldsymbol{b}, X, \boldsymbol{\alpha})$ is non-zero. In particular, the ℓ-th ($1 \le \ell \le k$) such vector is $\mathbf{c}_\ell := \mathbf{z}_\ell[1..i-1]$ such that

$$\tilde{eq^*}(\mathbf{c}_\ell, X, \boldsymbol{\alpha}) = \sum_{j\in[k]} d_j \cdot eq(\mathbf{z}_\ell[1..i-1], \mathbf{z}_j[1..i-1]) \cdot eq(X, \mathbf{z}_j[i]) \cdot eq(\boldsymbol{\alpha}, \mathbf{z}_j[i+1..])\,.$$

we note that for each $j \in [k]$, the value $eq(\boldsymbol{\alpha}, \mathbf{z}_j[i+1..])$ can be maintained dynamically; the value $eq(X, \mathbf{z}_j[i])$ can be computed in time $O(1)$. Moreover, $eq(\mathbf{z}_\ell[1..i-1], \mathbf{z}_j[1..i-1])$ equals 1 if $\mathbf{z}_\ell[1..i-1] = \mathbf{z}_j[1..i-1]$ and equals 0 otherwise. In summary, all non-zero values $\{\tilde{eq^*}(\mathbf{c}_\ell, X, \boldsymbol{\alpha})\}_{\ell\in[k]}$ can be computed in a batch in time $O(k)$. Therefore for each $x_i \in \{0, 1, 2\}$, one can evaluate $r_i(x_i)$ from evaluations $\{f(\mathbf{c}_\ell, x_i, \boldsymbol{\alpha})\}_{\ell\in[k]}$ in time $O(k)$, by evaluating $\{\tilde{eq^*}(\mathbf{c}_\ell, x_i, \boldsymbol{\alpha})\}_{\ell\in[k]}$ first and computing the inner product between $(\tilde{eq^*}(\mathbf{c}_\ell, x_i, \boldsymbol{\alpha}))_{\ell\in[k]}$ and $(f(\mathbf{c}_\ell, x_i, \boldsymbol{\alpha}))_{\ell\in[k]}$.

Applications to Commit-and-Prove SNARKs. Our batching scheme is helpful for building Commit-and-Prove SNARKs (CP-SNARKs) from multilinear commitments. In the setting of CP-SNARKs, given two commitments C_f, C_g that commit to vectors $\mathbf{f} \in \mathbb{F}^n$, $\mathbf{g} \in \mathbb{F}^m$ ($m \le n$), and given two sets $I_f \subseteq [n]$, $I_g \subseteq [m]$, one needs to prove that the values of $\mathbf{f}(I_f)$ is consistent with $\mathbf{g}(I_g)$. This problem can be solved using a variant of our special batching scheme with complexity $O(n)$.

For simplicity suppose that $n = m$,[8] and we assume w.l.o.g that $n = 2^\mu$. The idea is to view $\mathbf{f}, \mathbf{g}$ as the evaluations of polynomials $f, g \in \mathcal{F}_\mu^{(\le 1)}$ on the boolean hypercube B_μ. Then the commitments C_f, C_g can be instantiated with multilinear commitments to polynomials f, g respectively. The relation that $\mathbf{f}(I_f) = \mathbf{g}(I_g)$ is a slightly more general version of the batching relation: let $k = |I_f| = |I_g|$, it is equivalent to prove that $f(\mathbf{z}_i) = g(\mathbf{u}_i)$ for all $i \in [k]$, where $\mathbf{z}_i, \mathbf{u}_i \in B_\mu$ map to the i-th index of set I_f, I_g respectively.

Similar to equation (3.11), we can define a sumcheck relation

$$\begin{aligned}&\sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot \left[\left(\sum_{\boldsymbol{b}\in B_\mu} f(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \mathbf{z}_i)\right) - \left(\sum_{\boldsymbol{b}\in B_\mu} g(\boldsymbol{b}) \cdot eq(\boldsymbol{b}, \mathbf{u}_i)\right)\right] \\ &= \sum_{\boldsymbol{b}\in B_\mu} f(\boldsymbol{b}) \left(\sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot eq(\boldsymbol{b}, \mathbf{z}_i)\right) - \sum_{\boldsymbol{b}\in B_\mu} g(\boldsymbol{b}) \left(\sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot eq(\boldsymbol{b}, \mathbf{u}_i)\right) = 0\,,\end{aligned}$$

[8] the same technique applies for $n \ne m$

which is essentially a sumcheck for the degree-2 polynomial $h := f \cdot \tilde{eq}_f - g \cdot \tilde{eq}_g$ on set B_μ, where

$$\tilde{eq}_f(\mathbf{X}) := \sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot eq(\mathbf{X}, \mathbf{z}_i)\,, \qquad \tilde{eq}_g(\mathbf{X}) := \sum_{i\in[k]} eq(\mathbf{t}, \langle i\rangle) \cdot eq(\mathbf{X}, \mathbf{u}_i)\,.$$

We can use the same sumcheck algorithm underlying the special batching scheme. The complexity is $O(2^\mu)$. The CP-SNARK proving is then reduced to two PCS openings, one for commitment C_f and one for C_g.

3.4 HyperPlonk: Scalable SNARKs for scaling Blockchains

Equipped with the building blocks in Section 3.3, we now describe the Polynomial IOP for HyperPlonk. In Section 3.4.1, we introduce $\mathcal{R}_{\text{PLONK}}$ — an indexed relation on the boolean hypercube that generalizes the vanilla Plonk constraint system [GWC19]. We present a Polynomial IOP protocol for $\mathcal{R}_{\text{PLONK}}$ and analyze its security and efficiency in Section 3.4.2.

3.4.1 Constraint systems

Notation. For any $m \in \mathbb{Z}$ and $i \in [0, 2^m)$, we use $\langle i\rangle_m = \mathbf{v} \in B_m$ to denote the m-bit binary representation of i, that is, $i = \sum_{j=1}^m \mathbf{v}_j \cdot 2^{j-1}$.

Definition 3.13 (HyperPlonk indexed relation)**.** *Fix public parameters* $\mathsf{gp} := \big(\mathbb{F}, \ell, n, \ell_w, \ell_q, f\big)$ *where* $\mathbb{F}$ *is the field,* $\ell = 2^\nu$ *is the public input length,* $n = 2^\mu$ *is the number of constraints,* $\ell_w = 2^{\nu_w}, \ell_q = 2^{\nu_q}$ *are the number of witnesses and selectors per constraint*[9]*, and* $f : \mathbb{F}^{\ell_q+\ell_w} \to \mathbb{F}$ *is an algebraic map with degree* d*. The indexed relation* $\mathcal{R}_{PLONK}$ *is the set of all tuples*

$$(\mathbb{i}; \mathbb{x}; \mathbb{w}) = ((q, \sigma); (p, [[w]]); w)\,,$$

where $\sigma : B_{\mu+\nu_w} \to B_{\mu+\nu_w}$ *is a permutation,* $q \in \mathcal{F}^{(\le 1)}_{\mu+\nu_q}$*,* $p \in \mathcal{F}^{(\le 1)}_{\mu+\nu}$*,* $w \in \mathcal{F}^{(\le 1)}_{\mu+\nu_w}$*, such that*

- *the wiring identity is satisfied, that is,* $\big(\sigma; ([[w]], [[w]]); w\big) \in \mathcal{R}_{PERM}$ *(Definition 3.10);*
- *the gate identity is satisfied, that is,* $\big(([[\tilde{f}]]); \tilde{f}\big) \in \mathcal{R}_{ZERO}$ *(Definition 3.7), where the virtual polynomial* $\tilde{f} \in \mathcal{F}^{(\le d)}_\mu$ *is defined as* $\tilde{f}(\mathbf{X}) := f(q(\langle 0\rangle_{\nu_q}, \mathbf{X}), \ldots, q(\langle \ell_q - 1\rangle_{\nu_q}, \mathbf{X}), w(\langle 0\rangle_{\nu_w}, \mathbf{X}), \ldots, w(\langle \ell_w -$
- *the public input is consistent with the witness, that is, the public input polynomial* $p \in \mathcal{F}^{(\le 1)}_\nu$ *is identical to* $w(0^{\mu+\nu_w-\nu}, \mathbf{X}) \in \mathcal{F}^{(\le 1)}_\nu$*.*

$\mathcal{R}_{\text{PLONK}}$ is general enough to capture many computational models. In the introduction, we reviewed how $\mathcal{R}_{\text{PLONK}}$ captures simple arithmetic circuits. $\mathcal{R}_{\text{PLONK}}$ can be used to capture higher degree circuits with higher arity and more complex gates, including state machine computations.

[9] We can pad zeroes if the actual number is not a power of two.

State machines. $\mathcal{R}_{\text{PLONK}}$ can model state machine computations, as shown by Gabizon and Williamson [GW20b]. A state machine execution with $n-1$ steps starts with an initial state $\mathsf{state}_0 \in \mathbb{F}^k$ where k is the width of the state vector. In each step $i \in [0, n-1)$, given input the previous state state_i and an online input $\mathsf{inp}_i \in \mathbb{F}$, the state machine executes a transition function f and outputs $\mathsf{state}_{i+1} \in \mathbb{F}^w$. Let $\mathcal{T} := (\mathsf{state}_0, \ldots, \mathsf{state}_{n-1})$ be the execution trace and define $\mathsf{inp}_{n-1} := \perp$, we say that $\mathcal{T}$ is valid for input $(\mathsf{inp}_0, \ldots, \mathsf{inp}_{n-1})$ if and only if (i) $\mathsf{state}_{n-1}[0] = 0^k$, and (ii) $\mathsf{state}_{i+1} = f(\mathsf{state}_i, \mathsf{inp}_i)$ for all $i \in [0, n-1)$.

We build a HyperPlonk indexed relation that captures the state machine computation. W.l.o.g we assume that $n = 2^\mu$ for some $\mu \in \mathbb{N}$.[10] Let ν_w be the minimal integer such that $2^{\nu_w} > 2k$. We also assume that there is a low-depth algebraic predicate f_* that captures the transition function f, that is, $f_*(\mathsf{state}', \mathsf{state}, \mathsf{inp}) = 0$ if and only if $\mathsf{state}' = f(\mathsf{state}, \mathsf{inp})$. For each $i \in [0, n)$:

- the online input at the i-th step is $\mathsf{inp}_i := w(\langle 0 \rangle_{\nu_w}, \langle i \rangle_\mu)$;
- the input state of step i is $\mathsf{state}_{\text{in},i} := \big[w\left(\langle 1 \rangle_{\nu_w}, \langle i \rangle_\mu\right), \ldots, w\left(\langle k \rangle_{\nu_w}, \langle i \rangle_\mu\right)\big]$;
- the output state of step i is $\mathsf{state}_{\text{out},i} := \big[w\left(\langle k+1 \rangle_{\nu_w}, \langle i \rangle_\mu\right), \ldots, w\left(\langle 2k \rangle_{\nu_w}, \langle i \rangle_\mu\right)\big]$;
- the selector for step i is $\mathbf{q}_i := q\big(\langle i \rangle_\mu\big)$;
- the transition and output correctness are jointly captured by a high-degree algebraic map f',

$$f'(\mathsf{inp}_i, \mathsf{state}_{\text{in},i}, \mathsf{state}_{\text{out},i}; \mathbf{q}_i) := (1 - \mathbf{q}_i) \cdot f_*(\mathsf{state}_{\text{out},i}, \mathsf{state}_{\text{in},i}, \mathsf{inp}_i) + \mathbf{q}_i \cdot \mathsf{state}_{\text{in},i}[0]\,.$$

For all $i \in [0, n-1)$, we set $\mathbf{q}_i = 0$ so that $\mathsf{state}_{i+1} = f_i(\mathsf{state}_i, \mathsf{inp}_i)$ if and only if

$$f'(\mathsf{inp}_i, \mathsf{state}_{\text{in},i}, \mathsf{state}_{\text{out},i}; \mathbf{q}_i) = f_*(\mathsf{state}_{\text{out},i}, \mathsf{state}_{\text{in},i}, \mathsf{inp}_i) = 0\,;$$

we set $\mathbf{q}_{n-1} = 1$ so that $\mathsf{state}_{\text{in},n-1}[0] = 0$ if and only if

$$f'(\mathsf{inp}_{n-1}, \mathsf{state}_{\text{in},n-1}, \mathsf{state}_{\text{out},n-1}; \mathbf{q}_{n-1}) = \mathsf{state}_{\text{in},n-1}[0] = 0\,.$$

Note that we also need to enforce equality between the i-th input state and the $(i-1)$-th output state for all $i \in [n-1]$. We achieve it by fixing a permutation σ and constraining that the witness assignment is invariant after applying the permutation.

Remark 3.4.1. *We can halve the size of the witness and remove the permutation check by using the polynomial shifting technique in Section 3.3.7. Specifically, we can remove output state columns* $\mathsf{state}_{out,i}$ *and replace it with* $\mathsf{state}_{in,i+1}$ *for every* $i \in [0, n)$.

[10] We can pad with dummy states if the number of steps is not a power of two.

3.4.2 The PolyIOP protocol

In this Section, we present a *multivariate* PIOP for $\mathcal{R}_{\text{PLONK}}$ that removes expensive FFTs.

Construction. Intuitively, the PIOP for $\mathcal{R}_{\text{PLONK}}$ builds on a zero-check PIOP (Section 3.3.2) for custom algebraic gates and a permutation-check PIOP (Section 3.3.5) for copy constraints; consistency between the public input and the online witness is achieved via a random evaluation check between the public input polynomial and the witness polynomial.

Let $\mathsf{gp} := \big(\mathbb{F}, \ell, n, \ell_w, \ell_q, f\big)$ be the public parameters and let $d := \deg(f)$. For a tuple $(\mathbb{i}; \mathbb{x}; \mathbb{w}) = \big((q, \sigma); (p, [[w]]); w\big)$, we describe the protocol in Figure 3.2.

Indexer. $\mathcal{I}(q, \sigma)$ calls the permutation PIOP indexer $([[s_{\text{id}}]], [[s_\sigma]]) \leftarrow \mathcal{I}_{\text{perm}}(\sigma)$. The oracle output is $([[q]], [[s_{\text{id}}]], [[s_\sigma]])$, where $q \in \mathcal{F}^{(\leq 1)}_{\mu+\nu_q}$, $s_{\text{id}}, s_\sigma \in \mathcal{F}^{(\leq 1)}_{\mu+\nu_w}$.

The protocol. $\mathsf{P}(\mathsf{gp}, \mathbb{i}, p, w)$ and $\mathsf{V}(\mathsf{gp}, p, [[q]], [[s_{\text{id}}]], [[s_\sigma]])$ run the following protocol.

1. P sends V the witness oracle $[[w]]$ where $w \in \mathcal{F}^{(\leq 1)}_{\mu+\nu_w}$.
2. P and V run a PIOP for the gate identity, which is a zero-check PIOP (Section 3.3.2) for $\big([[\tilde{f}]]; \tilde{f}\big) \in \mathcal{R}_{\text{ZERO}}$ where $\tilde{f} \in \mathcal{F}^{(\leq d)}_{\mu}$ is as defined in Equation 3.13.
3. P and V run a PIOP for the wiring identity, which is a permutation PIOP (Section 3.3.5) for $\big(\sigma; ([[w]], [[w]]); (w, w)\big) \in \mathcal{R}_{\text{PERM}}$.
4. V checks the consistency between witness and public input. It samples $\mathbf{r} \leftarrow\!\$\ \mathbb{F}^\nu$, queries $[[w]]$ on input $(\langle 0\rangle_{\mu+\nu_w-\nu}, \mathbf{r})$, and checks $p(\mathbf{r}) \stackrel{?}{=} w(\langle 0\rangle_{\mu+\nu_w-\nu}, \mathbf{r})$.

Figure 3.2: PIOP for $\mathcal{R}_{\text{PLONK}}$.

Theorem 3.10. *Let $\mathsf{gp} := \big(\mathbb{F}, \ell, n, \ell_w, \ell_q, f\big)$ be the public parameters where $\ell_w, \ell_q = O(1)$ are some constants. Let $d := \deg(f)$. The construction in Figure 3.2 is a multivariate PolyIOP for relation $\mathcal{R}_{PLONK}$ (Definition 3.13) with soundness error $\mathcal{O}\big(\frac{2^\mu + d\mu}{|\mathbb{F}|}\big)$ and the following complexity:*

- *the prover time is $\mathsf{tp}^{\mathsf{gp}}_{plonk} = \mathcal{O}\big(nd \log^2 d\big)$;*
- *the verifier time is $\mathsf{tv}^{\mathsf{gp}}_{plonk} = \mathcal{O}(\mu + \ell)$;*
- *the query complexity is $\mathsf{q}^{\mathsf{gp}}_{plonk} = 2\mu + 4 + \log \ell_w$, that is, $2\mu + \log \ell_w$ univariate oracle queries, 3 multilinear oracle queries, and 1 query to the virtual polynomial $\tilde{f}$.*
- *the round complexity and the number of proof oracles is $\mathsf{rc}^{\mathsf{gp}}_{plonk} = 2\mu + 1 + \nu_w$;*
- *the number of field elements sent by the prover is $\mathsf{nf}^{\mathsf{gp}}_{plonk} = 2\mu$;*
- *the size of the proof oracles is $\mathsf{pl}^{\mathsf{gp}}_{plonk} = \mathcal{O}\big(n\big)$; the size of the witness is $n\ell_w$.*

Remark 3.4.2. *Two separate sumcheck PIOPs are underlying the HyperPlonk PIOP. We can batch the two sumchecks into one by random linear combination. The optimized protocol has round complexity $\mu+1+\log \ell_w$, and the number of field elements sent by the prover is μ. The query complexity $\mu+3+\log \ell_w$, that is, $\mu+\log \ell_w$ univariate queries, 2 multilinear queries, and 1 queries to the virtual polynomial $\tilde{f}$.*

Remark 3.4.3. *The prover's memory consumption is linear to the number of constraints. For space-bounded provers, we can split the proving work to multiple parallel parties or apply the techniques from [Boo+22b] to obtain a space-efficient prover with quasilinear proving time. We leave concrete specifications of space-efficient HyperPlonk provers as future work.*

Lemma 3.6. *The PIOP in Figure 3.2 is perfectly complete.*

Proof. For any $\big((q,\sigma);(p,[[w]]);w\big) \in \mathcal{R}_{\text{PLONK}}$, by Definition 3.13, it holds that

- $\big([[\tilde{f}]];\tilde{f}\big) \in \mathcal{R}_{\text{ZERO}}$, thus V passes the check in Step 2 as the ZeroCheck PIOP is complete;
- $\big(\sigma;([[w]],[[w]]);w\big) \in \mathcal{R}_{\text{PERM}}$, thus V passes the check in Step 3 as the permutation PIOP is complete;
- the public input polynomial $p \in \mathcal{F}_\nu^{(\leq 1)}$ is identical to $w(0^{\mu+\nu_w-\nu},\mathbf{X}) \in \mathcal{F}_\nu^{(\leq 1)}$, thus their evaluations are always the same, and V passes the check in Step 4.

In summary, the lemma holds as desired. □

Lemma 3.7. *Let $\mathsf{gp} := \big(\mathbb{F}, \ell = 2^\nu, n = 2^\mu, \ell_w = 2^{\nu_w}, \ell_q, f\big)$ be the public parameters and let $d := \deg(f)$ The PIOP in Figure 3.2 has soundness error*

$$\delta_{plonk}^{\mathsf{gp}} := \max\left\{\delta_{zero}^{d,\mu}, \delta_{perm}^{1,\mu+\nu_w}, \frac{\nu}{|\mathbb{F}|}\right\}.$$

Proof. For any $\big((q,\sigma);(p,[[w]])\big) \notin \mathcal{L}(\mathcal{R}_{\text{PLONK}})$, that is, $\big((q,\sigma);(p,[[w]]);w\big) \notin \mathcal{R}_{\text{PLONK}}$, at least one of the following conditions holds:

- $\big([[\tilde{f}]];\tilde{f}\big) \notin \mathcal{R}_{\text{ZERO}}$;
- $\big(\sigma;([[w]],[[w]]);w\big) \notin \mathcal{R}_{\text{PERM}}$;
- $p(\mathbf{X}) \neq w(0^{\mu+\nu_w-\nu},\mathbf{X})$;

In the first condition, the probability that V passes the ZeroCheck in Step 2 is at most $\delta_{\text{zero}}^{d,\mu}$; in the second condition, the probability that V passes the permutation check in Step 3 is at most $\delta_{\text{perm}}^{1,\mu+\nu_w}$; in the last condition, by Lemma 3.2, V passes the evaluation check in Step 4 with probability at most $\nu/|\mathbb{F}|$. In summary, for any $\big((q,\sigma);(p,[[w]]);w\big) \notin \mathcal{R}_{\text{PLONK}}$, the probability that V accepts is at most $\max\{\delta_{\text{zero}}^{d,\mu}, \delta_{\text{perm}}^{1,\mu+\nu_w}, \nu/|\mathbb{F}|\}$ as claimed. □

Zero knowledge. We refer to Section 3.8 for the zero-knowledge version of the HyperPlonk PIOP.

3.5 HyperPlonk+: HyperPlonk with Lookup Gates

This section illustrates how to integrate lookup gates into the HyperPlonk constraint system. Then we present and analyze a Polynomial IOP protocol for the extended relation.

3.5.1 Constraint systems

The HyperPlonk+ indexed relation $\mathcal{R}_{\text{PLONK+}}$ is built on $\mathcal{R}_{\text{PLONK}}$ (Definition 3.13). The difference is that $\mathcal{R}_{\text{PLONK+}}$ further enables a set of *non-algebraic* constraints enforcing that some function over the witness values belongs to a preprocessed table. We illustrate via a simple example. Suppose we capture a fan-in-2 circuit with n addition/multiplication gates using relation $\mathcal{R}_{\text{PLONK}}$. We need to further constrain that for a subset of gates, the sum of two input wires should be in the range $[0, \ldots, B)$. What we can do is to set up a preprocessed table $\mathsf{table} = \{0, 1, \ldots, B\}$ and a selector $q_{\text{lk}} \in \mathbb{F}^n$ so that for every $i \in [n]$, $q_{\text{lk}}(i) = 1$ if the i-th gate has a range-check, and $q_{\text{lk}}(i) = 0$ otherwise. Then we prove a lookup relation that for all $i \in [n]$, the value $q_{\text{lk}}(i) \cdot \big(w_1(i) + w_2(i)\big)$ is in table, where $w_1(i), w_2(i)$ are the first and the second input wire of gate i.

We generalize the idea above and enable enforcing *arbitrary algebraic functions* (over the selectors and witnesses) to be in the table. Namely, the index further setups an algebraic functions f_{lk}. Each constraint is of the form

$$f_{\text{lk}}\left(q_{\text{lk}}(\langle 0\rangle, \langle i\rangle),\ \ldots,\ q_{\text{lk}}(\langle \ell_{\text{lk}} - 1\rangle, \langle i\rangle),\ w(\langle 0\rangle, \langle i\rangle),\ \ldots,\ w(\langle \ell_w - 1\rangle, \langle i\rangle)\right) \in \mathsf{table}$$

where ℓ_{lk} is the number of selectors, ℓ_w is the number of witness wires and $\langle i\rangle$ is the binary representation of i. Note that the constraint in the previous paragraph is a special case where $f_{\text{lk}} = q_{\text{lk}}(i) \cdot \big(w_1(i) + w_2(i)\big)$. We formally define the relation below.

Definition 3.14 (HyperPlonk+ indexed relation)**.** *Let $\mathsf{gp}_1 := \big(\mathbb{F}, \ell, n, \ell_w, \ell_q, f\big)$ be the public parameters for relation $\mathcal{R}_{PLONK}$ (Definition 3.13). Let $\mathsf{gp}_2 := (\ell_{lk}, f_{lk})$ be the additional public parameters where $\ell_{lk} = 2^{\nu_{lk}}$ is the number of lookup selectors and $f_{lk} : \mathbb{F}^{\ell_{lk}+\ell_w} \to \mathbb{F}$ is an algebraic map. The indexed relation $\mathcal{R}_{PLONK+}$ is the set of all triples*

$$(\mathbb{i}; \mathbb{x}; \mathbb{w}) = ((\mathbb{i}_1, \mathbb{i}_2); (p, [[w]]); w)$$

where $\mathbb{i}_2 := \big(\mathsf{table} \in \mathbb{F}^{n-1}, q_{lk} \in \mathcal{F}^{(\le 1)}_{\mu+\nu_{lk}}\big)$ such that

- *$(\mathbb{i}_1; \mathbb{x}; \mathbb{w}) \in \mathcal{R}_{PLONK}$;*
- *there exists $\mathsf{addr} : B_\mu \to [1, 2^\mu)$ such that $\big(\mathsf{table}; [[g]]; (g, \mathsf{addr})\big) \in \mathcal{R}_{LOOKUP}$ (Definition 3.11),*

where $g \in \mathcal{F}_\mu^{(\leq \deg(f_{lk}))}$ *is defined as*

$$g(\mathbf{X}) := f_{lk}\left(q_{lk}(\langle 0\rangle_{\nu_{lk}}, \mathbf{X}), \ldots, q_{lk}(\langle \ell_{lk}-1\rangle_{\nu_{lk}}, \mathbf{X}), w(\langle 0\rangle_{\nu_w}, \mathbf{X}), \ldots, w(\langle \ell_w - 1\rangle_{\nu_w}, \mathbf{X})\right). \tag{3.14}$$

Remark 3.5.1 (Supporting vector lookups)**.** *We can generalize* $\mathcal{R}_{PLONK+}$ *to support vector lookups where each "element" in the table is a vector rather than a single field element. Let* $k \in \mathbb{N}$ *be the length of the vector. The lookup table is* $\mathsf{table} \in \mathbb{F}^{k\times(n-1)}$*; the lookup function* $f_{lk} : \mathbb{F}^{2^{\nu_{lk}}+2^{\nu_w}} \to \mathbb{F}^k$ *is an algebraic map that outputs* k *field elements.*

Remark 3.5.2 (Supporting multiple tables)**.** *We can generalize* $\mathcal{R}_{PLONK+}$ *to support multiple lookup tables. In particular, the index* $\mathbb{i}_2$ *can specify* $k > 1$ *lookup tables* $\mathsf{table}_1, \ldots, \mathsf{table}_k$ *and* k *lookup functions* $f_{lk}^{(1)}, \ldots, f_{lk}^{(k)}$*; and we require that all of the* k *lookup relations hold.*

3.5.2 The PolyIOP protocol

Construction. The PIOP for $\mathcal{R}_{\text{PLONK+}}$ is a combination of the PIOP for $\mathcal{R}_{\text{PLONK}}$ and the PIOP for a lookup relation (Section 3.3.7). Let $\mathsf{gp} := (\mathsf{gp}_1, \mathsf{gp}_2)$ be the public parameters where $\mathsf{gp}_1 := \big(\mathbb{F}, \ell, n, \ell_w, \ell_q, f\big)$ and $\mathsf{gp}_2 := (\ell_{\text{lk}}, f_{\text{lk}})$. We denote $d_{\text{lk}} := \deg(f_{\text{lk}})$. For a tuple $(\mathbb{i}; \mathbb{x}; \mathbb{w}) = ((\mathbb{i}_1, \mathbb{i}_2); (p, [[w]]); w)$ where $\mathbb{i}_2 := \big(\mathsf{table} \in \mathbb{F}^{n-1}, q_{\text{lk}} \in \mathcal{F}_{\mu+\nu_{\text{lk}}}^{(\leq 1)}\big)$ we describe the protocol in Figure 3.3.

Indexer. $\mathcal{I}(\mathbb{i}_1, \mathbb{i}_2 = (\mathsf{table}, q_{\text{lk}}))$ calls the HyperPlonk PIOP indexer $\mathsf{vp}_{\text{plonk}} \leftarrow \mathcal{I}_{\text{plonk}}(\mathbb{i}_1)$, and calls the Lookup PIOP indexer $\mathsf{vp}_t \leftarrow \mathcal{I}_{\text{lkup}}(\mathsf{table})$. The oracle output is $\mathsf{vp} := ([[q_{\text{lk}}]], \mathsf{vp}_t, \mathsf{vp}_{\text{plonk}})$.

The protocol. $\mathsf{P}(\mathsf{gp}, \mathbb{i}, p, w)$ and $\mathsf{V}(\mathsf{gp}, p, \mathsf{vp})$ run the following protocol.

1. P sends V the witness oracle $[[w]]$ where $w \in \mathcal{F}_{\mu+\nu_w}^{(\leq 1)}$.
2. Run a HyperPlonk PIOP (Section 3.4.2) for $(\mathbb{i}_1; \mathbb{x}; \mathbb{w}) \in \mathcal{R}_{\text{PLONK}}$.
3. Run a lookup PIOP (Section 3.3.7) for $\big(\mathsf{table}; [[g]]\big) \in \mathcal{L}(\mathcal{R}_{\text{LOOKUP}})$ where $g \in \mathcal{F}_\mu^{(\leq d_{\text{lk}})}$ is as defined in Equation 3.14.

Figure 3.3: PIOP for $\mathcal{R}_{\text{PLONK+}}$.

Theorem 3.11. *Let* $\mathsf{gp} := (\mathsf{gp}_1, \mathsf{gp}_2)$ *be the public parameters, where* $\mathsf{gp}_1 := \big(\mathbb{F}, \ell, n, \ell_w, \ell_q, f\big)$ *and* $\ell_w, \ell_q = O(1)$ *are some constants;* $\mathsf{gp}_2 := (\ell_{lk}, f_{lk})$ *and* $\ell_{lk} = O(1)$ *is some constant. Let* $d' := \max\big(\deg(f), \deg(f_{lk})\big)$ *and let* g *be the polynomial defined in Equation 3.14. The construction in Figure 3.3 is a multivariate PolyIOP for relation* $\mathcal{R}_{PLONK+}$ *with soundness error* $\mathcal{O}\big(\frac{2^\mu + d'\mu}{|\mathbb{F}|}\big)$ *and the following complexity:*

- *The prover time is* $\mathsf{tp}_{plonk+}^{\mathsf{gp}} = \mathsf{tp}_{plonk}^{\mathsf{gp}_1} + \mathsf{tp}_{lkup}^{g} = \mathcal{O}\big(nd' \log^2 d'\big)$ $\mathbb{F}$*-ops.*
- *The verifier time is* $\mathsf{tv}_{plonk+}^{\mathsf{gp}} := \mathsf{tv}_{plonk}^{\mathsf{gp}_1} + \mathsf{tv}_{lkup}^{g} = \mathcal{O}(\mu + \ell)$ $\mathbb{F}$*-ops.*

- *The query complexity is* $\mathsf{q}^{\mathsf{gp}}_{plonk+} = \mathsf{q}^{\mathsf{gp}_1}_{plonk} + \mathsf{q}^{g}_{lkup} = 3\mu + 7 + \log \ell_w$*, that is,* $3\mu + \log \ell_w$ *univariate oracle queries,* 5 *multilinear oracle queries,* 1 *query to the virtual polynomial* $\tilde{f}$*, and* 1 *query to the virtual polynomial* g *defined in Equation 3.14.*
- *The round complexity and the number of proof oracles is* $\mathsf{rc}^{\mathsf{gp}}_{plonk+} = \mathsf{rc}^{\mathsf{gp}_1}_{plonk} + \mathsf{rc}^{g}_{lkup} = 3\mu + 3 + \log \ell_w$*.*
- *The number of field elements sent by* P *is* 3μ*.*
- *The size of the proof oracles is* $\mathcal{O}(n)$*; the size of the witness is* $n\ell_w$*.*

Remark 3.5.3. *Similar to Remark 3.4.2, there are* 3 *separate sumcheck PIOPs underlying the HyperPlonk+ PIOP. By random linear combination, we can batch the* 3 *sumchecks into a single one. The optimized protocol has query complexity* $\mu + 7 + \log \ell_w$*, round complexity* $\mu + 3 + \log \ell_w$*, and the number of field elements sent by the prover is* μ*.*

Remark 3.5.4. *We emphasize that the PolyIOP for* $\mathcal{R}_{PLONK+}$ *naturally works for the more general versions of* $\mathcal{R}_{PLONK+}$ *that involve vector lookups (Remark 3.5.1) or multiple tables (Remark 3.5.2). Because we can transform the problem of building PIOPs for the more general relations to the problem of building PIOPs for* $\mathcal{R}_{PLONK+}$ *by applying the randomization and domain separation techniques in Section 4 of [GW20a].*

Lemma 3.8. *The PIOP in Figure 3.3 is perfectly complete.*

Proof. For any $\big((\mathbb{i}_1, \mathsf{table}, q_{\mathrm{lk}}); (p, [[w]]); w\big) \in \mathcal{R}_{\mathrm{PLONK+}}$, by Definition 3.14, it holds that

- $(\mathbb{i}_1; \mathbb{x}; \mathbb{w}) \in \mathcal{R}_{\mathrm{PLONK}}$, thus V passes the check in Step 2 as the HyperPlonk PIOP is complete;
- $\big(\mathsf{table}; [[g]]\big) \in \mathcal{L}(\mathcal{R}_{\mathrm{LOOKUP}})$, thus V passes the check in Step 3 as the lookup PIOP is complete.

In summary, the lemma holds as desired. □

Lemma 3.9. *Let* $\mathsf{gp} := (\mathsf{gp}_1, \mathsf{gp}_2)$ *be the public parameters. Let* $n = 2^\mu \in \mathsf{gp}_1$ *denote the number of constraints. Let* $f_{lk} \in \mathsf{gp}_2$ *be the lookup gate map and set* $d_{lk} := \deg(f_{lk})$*. The PIOP in Figure 3.3 has soundness error*

$$\delta^{\mathsf{gp}}_{plonk+} := \max\left\{\delta^{\mathsf{gp}_1}_{plonk}, \delta^{d_{lk},\mu}_{lkup}\right\}.$$

Proof. For any $\big((\mathbb{i}_1, \mathsf{table}, q_{\mathrm{lk}}); (p, [[w]])\big) \notin \mathcal{L}(\mathcal{R}_{\mathrm{PLONK+}})$, that is, $\big((\mathbb{i}_1, \mathsf{table}, q_{\mathrm{lk}}); (p, [[w]]); w\big) \notin \mathcal{R}_{\mathrm{PLONK}}$, at least one of the following conditions holds:

- $\big(\mathbb{i}_1; \mathbb{x}; \mathbb{w}\big) \notin \mathcal{R}_{\mathrm{PLONK}}$;
- $\big(\mathsf{table}; [[g]]\big) \notin \mathcal{L}(\mathcal{R}_{\mathrm{LOOKUP}})$, where $g \in \mathcal{F}_\mu^{(\leq d_{\mathrm{lk}})}$ is as defined in Equation 3.14.

For the first case, the probability that V accepts in the HyperPlonk PIOP is at most $\delta^{\mathsf{gp}_1}_{\mathrm{plonk}}$; for the second case, the probability that V passes the lookup check is at most $\delta^{d_{\mathrm{lk}},\mu}_{\mathrm{lkup}}$. Thus for every instance not in $\mathcal{L}(\mathcal{R}_{\mathrm{PLONK+}})$, the probability that V accepts is at most $\max\big(\delta^{\mathsf{gp}_1}_{\mathrm{plonk}}, \delta^{d_{\mathrm{lk}},\mu}_{\mathrm{lkup}}\big)$. □

Zero knowledge. We refer to Section 3.8 for the zero-knowledge version of the HyperPlonk+ PIOP.

3.6 Instantiation and evaluation

3.6.1 Implementation

We implement HyperPlonk as a library using about 5600 lines of RUST. Figure 3.4 highlights the building blocks contributing to our HyperPlonk code base. Our backend is built on top of the Arkworks [con22]. Specifically, we adopted the finite field, elliptic curve, and polynomial libraries from this project. We then build our PIOP libraries, including our core zero and permutation checks, and use merlin transcript [Val22] to turn it into a non-interactive protocol. We also implement a multilinear KZG commitment scheme variant that is compatible with our batch-evaluation PIOP.

Our implementation is highly modular: one may switch between different elliptic curves, other multilinear polynomial commitment schemes and various circuit frontends within our framework.

The current version of our code base has a few limitations, which do not affect the benchmarks reported in this section. Firstly, it is built for benchmarking purposes with mock circuits, but we aim to support Halo2 and Jellyfish arithmetization as frontends. Secondly, we are not yet supporting lookup tables and thus HyperPlonk+.

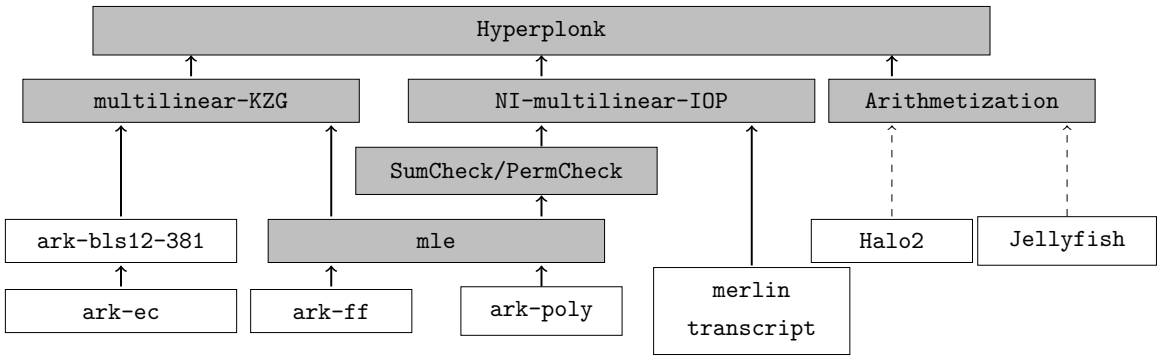


Figure 3.4: Stack of libraries comprising HyperPlonk. The components in grey we implemented ourselves. The arithmetization frontends have not yet been linked to the implementation.

3.6.2 Evaluation

We benchmark HyperPlonk on an AWS EC2 instance running Ubuntu 20.04. The server has 64 cores (AMD EPYC 7R13 at 2.65GHz) and 128 GB of RAM. The hyperplonk benchmarks were run using a rust implementation available online[11]. We utilize a multi-linear KZG commitment built using

[11] `https://github.com/EspressoSystems/hyperplonk`

curve BLS12-381. If not otherwise indicated, we use the same custom gate as the Jellyfish library. The gate has 5 inputs, 13 selectors, and degree 5.

Proof size and verification time. Our implementation is modeled after the unrolled and optimized Hyperplonk scheme described in Section 3.10. The proof size is 176 μ+ 1168 bytes. One caveat is that we do not yet commit to the univariate polynomials in each round of the sum check. This slightly increases the proof size. The verification time for $\mu = 20$ is less than 20ms on a consumer-grade laptop.

Cost breakdown. We present a cost breakdown of HyperPlonk's prover cost. The breakdown is measured on a consumer-grade laptop[12]. As we see in Figure 3.5a, the majority of the computation is spent on committing and (batch) opening the commitment; the actual time spent on the information-theoretic PIOPs (`Perm Check` and `Circuit Check`) is about 30%. The batch opening does not yet take advantage of the fact that many evaluation points and polynomials are identical. This could reduce the complexity of the resulting zero-check.

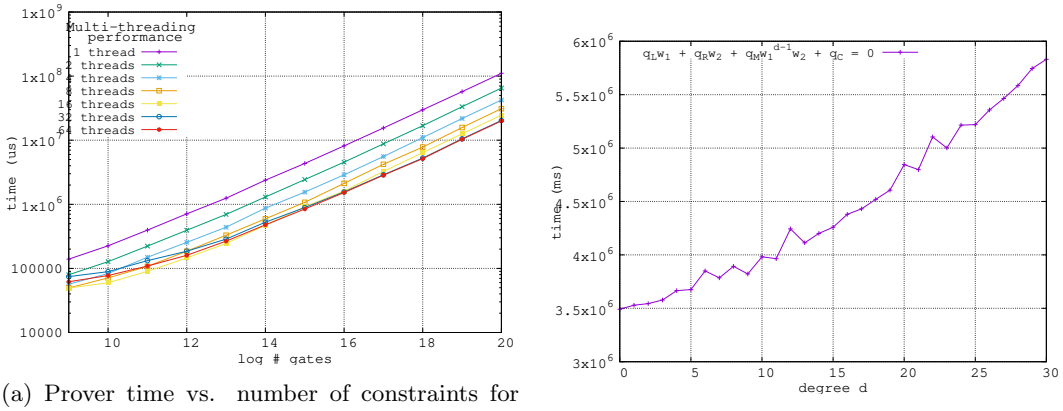
Figure 3.5b gives another breakdown. It shows that the majority of the time (61%) is spent on multi-linear evaluations. This includes the operations performed within the sumcheck protocol. The rest of the time is spent on elliptic curve multi-exponentiations. Batching zero-checks and improving the batch-opening implementation could further reduce the number of MLE operations. We note that both multi-exponentiations and sumchecks are highly parallelizable and hardware-friendly, thus we expect further performance improvement on special-purpose hardware (e.g. GPUs).

It is also worth noting that HyperPlonk never requires the explicit multiplication of polynomials. This enables high-degree custom gates for HyperPlonk.

(a) In terms of building blocks (b) In terms of computations

Figure 3.5: Cost breakdown for vanilla $\mathcal{R}_{\text{PLONK}}$ with 2^{20} constraints.

[12] 2021 Apple MacBook Pro with M1 chip



(a) Prover time vs. number of constraints for different number of threads

(b) Prover time vs. custom gate degree.

Figure 3.6

3.6.3 MultiThreading performance

A key advantage of HyperPlonk is that it does not rely on FFT algorithms that are less parallelizable. Indeed, in Figure 3.6a we observe an almost linear improvement when num of threads is small. We also observe that with low parallelization, the prover's run time is linear in the number of gates. For example, increase from a single thread to two threads, the prover time is reduced by 45% on average. In contrast, from 32 to 64 threads, there is almost no additional speedup. We assume that this is implementation dependent.

3.6.4 High degree gates

It has been shown in VeriZexe [Xio+22] that custom gates, even at degree 5, allow for significant improvement of circuit size and prover time. For example, one may perform an elliptic curve group addition with just two gates; while a naive version may require 10+ gates. The better expressibility of high-degree gates enables VeriZexe to improve 9x of prover time over the previous state-of-the-art [Bow+20].

However, in a univariate Plonk system, such as [GW20a; Pea+22], high-degree custom gates increase the size of the required FFTs as well as the number of group operations. This limits their utility as they get larger. In comparison, in HyperPlonk, high-degree only affects the number of field operations. Our benchmark result in Figure 3.6b validates this observation and shows that the prover time from a degree 2 gate to a degree 32 gate only increases by 30%. These more expressive gates can significantly reduce the number of gates in the circuit which more than offsets the added cost.

3.6.5 Comparisons

We compare our scheme with both Jellyfish Plonk[13], and Spartan [Set20][14]. Jellyfish is a highly optimized implementation of Plonk with lookup arguments. It is the state-of-the-art Plonk prove-system that uses Arkworks as the backend. Spartan is a sumcheck based ZKP system. Spartan's statements are written in Rank-1-Constraint-Systems ($\mathcal{R}_{\text{R1}CS}$), which is simpler but less expressive.

Hyperplonk, Plonk, and Spartan are polynomial IOPs that can be combined with various polynomial commitments. The commitment has a large impact on the performance of the proof system. For sake of comparison, we ensure that all 3 systems use the same elliptic curve and the same implementation. Concretely we use the Arkworks BLS12-381 implementation. Hyperplonk uses the multi-linear KZG commitment, Jellyfish the univariate KZG commitment, and Spartan uses an inner product argument [Boo+16; Bün+18; Wah+18]-based polynomial commitment. We refer to the Spartan fork as Ark-Spartan to highlight the use of the Arkworks BLS12-381 backend.

Comparison by application. We have presented data points for a few typical applications in Table 3.4. The proof systems are evaluated using mock circuits. The circuit sizes for both the $\mathcal{R}_{\text{PLONK+}}$ (using the Jellyfish custom gate) and for the $\mathcal{R}_{\text{R1}CS}$ arithmetization are taken from references and demonstrate the advantages of (Hyper)Plonk. For example, a proof of knowledge of exponent for a 256-bits elliptic curve group element requires 3315 $\mathcal{R}_{\text{R1}CS}$ constraints [MSZ21], while it reduces to 783 for $\mathcal{R}_{\text{PLONK+}}$[Xio+22]. Note that our HyperPlonk implementation does not yet support lookups, but we estimate that the slowdown will only be minor and offset by further optimizations.

Application	$\mathcal{R}_{\text{R1}CS}$	Ark-Spartan	$\mathcal{R}_{\text{PLONK+}}$	Jellyfish	HyperPlonk
3-to-1 Rescue Hash	288 [Aly+20]	422 ms	144 [Sys22]	40 ms	88 ms
PoK of Exponent	3315 [MSZ21]	902 ms	783 [MSZ21]	64 ms	105 ms
ZCash circuit	2^{17} [Hop+22]	8.3 s	2^{15} [Esp22]	0.8 s	0.6 s
Zexe's recursive circuit	2^{22} [Xio+22]	6 min	2^{17} [Xio+22]	13.1 s	5.1 s
Rollup of 50 private tx	2^{25}	39 min[b]	2^{20} [Sys22]	110 s	38.2 s
zkEVM circuit[a]	N/A	N/A	2^{27}	1 hour[b,c]	25 min[b,c]

Table 3.4: Prover runtime of Hyperplonk vs. Spartan[Set20] and the Jellyfish Plonk implementation for popular applications. Column 2 shows the number of $\mathcal{R}_{\text{R1}CS}$ constraints for each application and column 4 shows the corresponding number of constraints in HyperPlonk+/Ultraplonk. We emphasize that the Zexe and the Rollup applications are using the BW6-761 curve because they need to use two-chain curves. The rest of the applications are using the BLS12-381 curve.
[a] So far, there have been no approaches to express zkEVM as an R1CS circuit. Common approaches rely heavily on lookup tables which require plonk+. [b] Estimations. [c] This assumes a linear scaling factor that is in favor of Jellyfish. Note that we observe a super-linear growth for log degree from 20 to 23 in Jellyfish, while a sub-linear growth in HyperPlonk.

[13] `https://github.com/EspressoSystems/jellyfish/tree/hyperplonk-bench`
[14] `https://github.com/zhenfeizhang/ark-spartan`

Comparison by circuit size. We also compare HyperPlonk with Jellyfish and Ark-Spartan. We run both the *vanilla-gate* version of HyperPlonk that supports just additions and multiplication gates as well as the degree 5 Jellyfish gate. Note that n Jellyfish gates are significantly more expressive than n vanilla or R1CS gates. We measure both the single-threaded performance in Table 3.5 and the multi-threaded performance in Table 3.6. Our benchmark shows that multi-threaded HyperPlonk outperforms Jellyfish starting from 2^{14} constraints; the advantage grows when circuit size increases. This is mainly because FFTs scale worse than multi-exponentiations.

HyperPlonk also has slightly better performance than Spartan. The difference is more pronounced in the multi-threaded benchmark which is likely because the Ark-Spartan implementation does not take full advantage of parallelism. We stress again that plonk+ is more expressive than $\mathcal{R}_{\text{R1}CS}$, and thus a fair comparison should be over the same application rather than the same size of constraints. Table 3.4 shows that HyperPlonk is $5 \sim 60$x faster than Spartan in those applications.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
HP (Vanilla Gate)	0.22	0.4	0.7	1.2	2.2	4.1	7.8	14.6	28	53	103
HP (Jellyfish Gate)	0.33	0.6	1.1	1.9	3.6	6.8	13	24.5	49.5	95	185
Jellyfish Plonk	0.49	0.9	1.5	2.8	5.5	10.5	19.4	37.9	74	143	284
Ark-Spartan	0.95	1.6	2.5	4.4	6.4	12.1	20.8	38.7	69.2	135	223

Table 3.5: Single-thread prover's performance (in seconds) for varying number of constraints under different schemes.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
HP (Vanilla Gate)	0.07	0.1	0.14	0.2	0.3	0.5	0.8	1.4	2.5	5.1	9.6
HP (Jellyfish Gate)	0.1	0.13	0.18	0.27	0.4	0.67	1.2	2	3.7	7.3	13.5
Jellyfish Plonk	0.07	0.1	0.15	0.25	0.46	0.78	1.4	2.7	5.5	10.8	22
Ark-Spartan	0.51	0.72	0.9	1.4	1.9	3.1	4.7	8.3	13.7	27	44

Table 3.6: 64-thread prover's performance (in seconds) for varying number of constraints under different schemes.

3.7 Orion+: a linear-time multilinear PCS with constant proof size

Recently, Xie et al. [XZS22a] introduced a highly efficient multilinear polynomial commitment scheme called Orion. The prover time is strictly linear, that is, $O(2^\mu)$ field operations and hashes where μ is the number of variables. For $\mu = 27$, it takes only 115 seconds to commit to a polynomial and compute an evaluation proof using a single thread on a consumer-grade desktop. The verifier time and proof size is $O_\lambda(\mu^2)$, which also improves the state-of-the-art [BCG20; Gol+21]. However,

the concrete proof size is still unsatisfactory, e.g., for $\mu = 27$, the proof size is 6 MBs. In this section, we describe a variant of Orion PCS that enjoys similar proving complexity but has $O(\mu)$ proof size and verifier time, with good constants. In particular, for security parameter $\lambda = 128$ and $\mu = 27$, the proof size is less than 10KBs, which is 600× smaller than Orion for $\mu = 27$.

This section is organized as follows. We start by reviewing the linear-time PCS from tensor product arguments [BCG20; Gol+21], which Orion builds upon, then we describe our techniques for shrinking the proof size. Finally, we analyze the security and complexity of the construction.

Linear-time PCS from tensor-product argument [BCG20; Gol+21]. Bootle, Chiesa, and Groth [BCG20] propose an elegant scheme for building PCS with strictly linear-time provers. Golovnev et al. [Gol+21] later further simplify the scheme. Let $f \in \mathcal{F}_\mu^{(\leq 1)}$ be a multilinear polynomial where $f_{\boldsymbol{b}} \in \mathbb{F}$ is the coefficient of $\mathbf{X}_{\boldsymbol{b}} := \mathbf{X}_1^{\boldsymbol{b}_1} \cdots \mathbf{X}_\mu^{\boldsymbol{b}_\mu}$ for every $\boldsymbol{b} \in B_\mu$. Denote by $n = 2^\mu$, $k = 2^\nu < 2^\mu$ and $m = n/k$, one can view the evaluation of f as a tensor product, that is,

$$f(\mathbf{X}) = \langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle \tag{3.15}$$

where $\mathbf{w} = (f_{\langle 0 \rangle}, \ldots, f_{\langle n-1 \rangle})$, $\mathbf{t}_0 = \left(\mathbf{X}_{\langle 0 \rangle}, \mathbf{X}_{\langle 1 \rangle}, \ldots, \mathbf{X}_{\langle k-1 \rangle}\right)$ and $\mathbf{t}_1 = \left(\mathbf{X}_{\langle 0 \rangle}, \mathbf{X}_{\langle k \rangle}, \ldots, \mathbf{X}_{\langle (m-1)\cdot k \rangle}\right)$. Here $\langle i \rangle$ denotes the μ-bit binary representation of i. Let $E : \mathbb{F}^m \to \mathbb{F}^M$ be a linear encoding scheme, that is, a linear function whose image is a linear code (Definition 3.1). Golovnev et al. [Gol+21, §4.2] construct a PCS scheme as follows:

- **Commitment:** To commit a multilinear polynomial f with coefficients $\mathbf{w} \in \mathbb{F}^n$, the prover P interprets $\mathbf{w}$ as a $k \times m$ matrix, namely $\mathbf{w} \in \mathbb{F}^{k \times m}$, encodes $\mathbf{w}$'s rows, and obtains matrix $W \in \mathbb{F}^{k \times M}$ such that $W[i, :] = E(\mathbf{w}[i, :])$ for every $i \in [k]$. Then P computes a Merkle tree commitment for each column of W and builds another Merkle tree T on top of the column commitments. The polynomial commitment C_f is the Merkle root of T.
- **Evaluation proof:** To prove that $f(\mathbf{z}) = y$ for some point $\mathbf{z} \in \mathbb{F}^\mu$ and value $y \in \mathbb{F}$, the prover P translates $\mathbf{z}$ to vectors $\mathbf{t}_0 \in \mathbb{F}^k$ and $\mathbf{t}_1 \in \mathbb{F}^m$ as above and proves that $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$ (where $\mathbf{w} \in \mathbb{F}^{k \times m}$ is the message encoded and committed in C_f). To do so, P does two things:
  - **Proximity check:** The prover shows that the matrix $W \in \mathbb{F}^{k \times M}$ committed by C_f is close to k codewords. Specifically, the verifier sends a random vector $\mathbf{r} \in \mathbb{F}^k$, the prover replies with a vector $\mathbf{y}_\mathbf{r} := \mathbf{r} \cdot \mathbf{w} \in \mathbb{F}^m$ which is the linear combination of $\mathbf{w}$'s rows according to $\mathbf{r}$. The verifier checks that the encoding of $\mathbf{y}_\mathbf{r}$, namely $E(\mathbf{y}_\mathbf{r}) \in \mathbb{F}^M$, is close to $\mathbf{r} \cdot W$, the linear combination of W's rows. This implies that the k rows of W are all close to codewords [Gol+21, §4.2].
  - **Consistency check:** The prover shows that $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$ where $\mathbf{w} \in \mathbb{F}^{k \times m}$ is the k error-decoded messages from $W \in \mathbb{F}$ committed in C_f. The scheme is similar to the proximity check except that we replace the random vector $\mathbf{r}$ with $\mathbf{t}_0$. After receiving the linearly combined vector $\mathbf{y}_0 \in \mathbb{F}^m$, the verifier further checks that $\langle \mathbf{y}_0, \mathbf{t}_1 \rangle = y$.

We describe the concrete PCS evaluation protocol below.

Protocol 1 (PCS evaluation [Gol+21]): The goal is to check that $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$ (where $\mathbf{w} \in \mathbb{F}^{k \times m}$ is the message encoded and committed in C_f).

1. V sends a random vector $\mathbf{r} \in \mathbb{F}^k$.
2. P sends vector $\mathbf{y_r}, \mathbf{y}_0 \in \mathbb{F}^m$ where

$$\mathbf{y_r} = \sum_{i=1}^{k} \mathbf{r}_i \cdot \mathbf{w}[i,:], \text{ and } \mathbf{y}_0 = \sum_{i=1}^{k} \mathbf{t}_{0,i} \cdot \mathbf{w}[i,:],$$

   where $\mathbf{w} \in \mathbb{F}^{k \times m}$ is the message matrix being encoded and committed.
3. V sends P a random subset $I \subseteq [M]$ with size $|I| = \Theta(\lambda)$.
4. P opens the entire columns $\{W[:,j]\}_{j \in I}$ using Merkle proofs, where $W \in \mathbb{F}^{k \times M}$ is the row-wise encoded matrix. That is, P outputs the column commitment h_j for every column $j \in I$, and provide the Merkle proof for h_j w.r.t. to Merkle root C_f.
5. V checks that (i) the Merkle openings are correct w.r.t. C_f, and (ii) for all $j \in I$, it holds that

$$E(\mathbf{y_r})_j = \langle \mathbf{r}, W[:,j] \rangle \text{ and } E(\mathbf{y}_0)_j = \langle \mathbf{t}_0, W[:,j] \rangle .$$

6. V checks that $\langle \mathbf{y}_0, \mathbf{t}_1 \rangle = y$.

Note that by sampling a subset I with size $\Theta(\lambda)$ and checking that $\mathbf{r} \cdot W$, $\mathbf{t}_0 \cdot W$ are consistent with the encodings $E(\mathbf{y_r})$, $E(\mathbf{y}_0)$ on set I, the verifier is confident that $\mathbf{r} \cdot W$, $\mathbf{t}_0 \cdot W$ are indeed close to the encodings $E(\mathbf{y_r})$, $E(\mathbf{y}_0)$ with high probability. By setting $k = \sqrt{n}$, the prover takes $O(n)$ $\mathbb{F}$-ops and hashes; the verifier time and proof size are both $O_\lambda(\sqrt{n})$. Orion describes an elegant code-switching scheme that reduces the proof size and verifier time down to $O_\lambda(\log^2(n))$. However, the concrete proof size is still large. Next, we describe a scheme that has much smaller proof.

Linear-time PCS with small proofs. Similar to Orion (and more generally, the proof composition technique [Boo+17; BCG20; Gol+21]), instead of letting the verifier check the correctness of $\mathbf{y_r}$, $\mathbf{y}_0$ and the openings of the columns $W[:,j] \forall j \in I$, the prover can compute another (succinct) outer proof validating the correctness of $\mathbf{y_r}, \mathbf{y}_0, W[:,j]$. However, we need to minimize the outer proof's circuit complexity, which is non-trivial. Orion builds an efficient SNARK circuit that removes all of the hashing gadgets, with the tradeoff of larger proof size. We describe a variant of their scheme that minimizes the proof size without significantly increasing the circuit complexity.

Specifically, after receiving challenge vector $\mathbf{r} \in \mathbb{F}^k$, P instead sends V commitments $C_{\mathbf{r}}, C_0$ to the messages $\mathbf{y_r}, \mathbf{y}_0$; after receiving V's random subset $I \subset [M]$, P computes a SNARK proof for the following statement:

Statement 1 (PCS Eval verification):

- Witness: $\mathbf{y_r}, \mathbf{y}_0 \in \mathbb{F}^m$, $\{W[:,j]\}_{j \in I}$.

- Circuit statements:
  - $C_{\mathbf{r}}$, C_0 are the commitments to $\mathbf{y}_{\mathbf{r}}$, $\mathbf{y}_0$ respectively.
  - For all $j \in I$, it holds that
    - $h_j = H(W[:, j])$ where H is a fast hashing scheme;
    - $E(\mathbf{y}_{\mathbf{r}})_j = \langle \mathbf{r}, W[:, j] \rangle$ and $E(\mathbf{y}_0)_j = \langle \mathbf{t}_0, W[:, j] \rangle$.
  - $\langle \mathbf{y}_0, \mathbf{t}_1 \rangle = y$.
- Public output: $\{h_j\}_{j \in I}$, and $C_{\mathbf{r}}, C_0$.

Besides the SNARK proof, the prover also provides the openings of $\{h_j\}_{j \in I}$ with respect to the commitments C_f. Intuitively, the new protocol is "equivalent" to Protocol 1, because the SNARK witness $\{W[:, j]\}_{j \in I}$ and $\mathbf{y}_{\mathbf{r}}, \mathbf{y}_0$ are identical to those committed in $C_f, C_{\mathbf{r}}, C_0$ by the binding property of the commitments; and the SNARK does all of the verifier checks. Unfortunately, the scheme has the following drawbacks:

- Instantiating the commitments with Merkle trees leads to a large overhead on the proof size. In particular, the proof contains $|I|$ Merkle proofs, each with length $O(\log n)$. For 128-bit security, we need to set $|I| = 1568$, and the proof size is at least 1 MBs for $\mu = 20$.
- The random subset I varies for different evaluation instances. It is non-trivial to efficiently lookup the witness $\{E(\mathbf{y}_{\mathbf{r}})_j, E(\mathbf{y}_0)_j\}_{j \in I}$ in the circuit if the set I is dynamic (i.e. we need an efficient random access gadget).
- The circuit complexity is huge. In particular, the circuit is dominated by the commitments to $\mathbf{y}_{\mathbf{r}}, \mathbf{y}_0$ and the hash commitments to $\{W[:, j]\}_{j \in I}$. This leads to $2m + k|I|$ hash gadgets in the circuit. Note that we can't use algebraic hash functions like Rescue [Aly+20] or Poseidon [Gra+21], which are circuit-friendly, but have slow running times. For $\mu = 26$, $k = m = \sqrt{n}$ and 128-bit security (where $|I| = 1568$), this leads to 13 million hash gadgets where each hash takes hundreds to thousands of constraints, which is unaffordable.

We resolve the above issues via the following observations.

First, a large portion of the multilinear PCS evaluation proof is Merkle opening paths. We can shrink the proof size by replacing Merkle trees with multilinear PCS that enable efficient batch openings (Section 3.3.8). Specifically, in the committing phase, after computing the hashes of W's columns, instead of building another Merkle tree T of size $M = O(n/k)$ and set the Merkle root as the commitment, the prover can commit to the column hashes using a multilinear PCS (e.g. KZG). Though the KZG committing is more expensive, *the problem size has been reduced to $O(n/k)$, thus for sufficiently large k, the committing complexity is still approximately $O(n)$ $\mathbb{F}$-ops.* A great advantage is that the batch opening proof for $\{h_j\}_{j \in I}$ consists of only $O(\log n)$ group/field elements, with good constant. Even better, when instantiating the outer proof with HyperPlonk(+), the openings can be batched with those in the outer SNARK and thus incur almost no extra cost in proof size.

Second, with Plookup, we can efficiently simulate random access in arrays in the SNARK circuit. For example, to extract witness $\{\mathbf{Y}_{\mathbf{r},j} = E(\mathbf{y}_\mathbf{r})_j\}_{j\in I}$, we can build an (online) table T where each element of the table is a pair $(i, E(\mathbf{y}_\mathbf{r})_i)$ $(1 \le i \le M)$. Then for every $j \in I$, we build a lookup gate checking that $(j, \mathbf{Y}_{\mathbf{r},j})$ is in the table T, thus guarantee that $\mathbf{Y}_{\mathbf{r},j}$ is identical to $E(\mathbf{y}_\mathbf{r})_j$. The circuit description is now independent of the random set I and we only need to preprocess the circuit once in the setup phase.

Third, with the help of Commit-and-Prove-SNARKs (CP-SNARK) [Cam+21; CFQ19; Ara+21], there is no need to check the consistency between commitments $C_\mathbf{r}, C_0$ and $\mathbf{y}_\mathbf{r}, \mathbf{y}_0$ in the circuit. Instead, we can commit $(\mathbf{y}_\mathbf{r}, \mathbf{y}_0)$ to a multilinear commitment C, and build a CP-SNARK proof showing that the vector underlying C is identical to the witness vector $(\mathbf{y}_\mathbf{r}, \mathbf{y}_0)$ in the circuit. We further observe that C can be a part of the witness polynomials, which further removes the need of an additional CP-SNARK proof.

After applying previous optimizations, the proof size is dominated by the $|I|$ field elements $\{h_j\}_{j\in I}$. We can altogether remove them by applying the CP-SNARK trick again. In particular, since $\{h_j\}_{j\in I}$ are both committed in the polynomial commitment C_f and the SNARK witness commitment, it is sufficient to construct a CP-SNARK proving that they are consistent in the two commitments with respect to set I. We refer to Section 3.3.8 for constructing CP-SNARK proofs from multilinear commitments.

Since the bulk of verification work is delegated to the prover, there is no need to set $k = \sqrt{n}$. Instead, we can set an appropriate $k = \Theta(\lambda/\log n)$ to minimize the outer circuit size. In particular, the circuit is dominated by 2 linear encodings (of length n/k) and $|I|$ hashes (of length k). If we use vanilla HyperPlonk+ as the outer SNARK scheme and use Reinforced Concrete [Bar+21] as the hashing scheme that has a similar running time to SHA-256, for $\mu = 30$, $k = 64$ and 128-bit security (where $|I| = 1568$), the circuit complexity is only $\approx 2^{26}$ constraints. And we can expect the running time of the outer proof to be $O_\lambda(n)$.

The resulting multilinear polynomial commitment scheme is shown in Figure 3.7.

Remark 3.7.1 (CP-SNARKs instantiation.)**.** *We can use the algorithm in Section 3.3.8 to instantiate the CP-SNARK in Figure 3.7 from any multilinear PIOP-based SNARKs with minimal overhead. First, we can split the witness polynomial into two parts: one includes the vector* $(\mathbf{y}_\mathbf{r}, \mathbf{y}_0)$ *while the other includes the rest. The witness polynomial commitment to* $(\mathbf{y}_\mathbf{r}, \mathbf{y}_0)$ *is essentially the commitment* $C_{p_\mathbf{y}}$ *in Figure 3.7, so that we don't need to additionally commit to* $(\mathbf{y}_\mathbf{r}, \mathbf{y}_0)$ *and provide a proof. We emphasize that* $C_{p_\mathbf{y}}$ *is sent before the prover receives the challenge set* I*, which is essential for knowledge soundness.*

Second, the CP-proof generation between the multilinear commitment C_f *and the SNARK witness polynomial commitment (w.r.t. set* I*) consists of a sumcheck with* $O(\log m)$ *rounds and* 2 *PCS openings (one for* C_f *and one for the witness polynomial). If we instantiate the SNARK with*

HyperPlonk+, *we can batch the proving of the CP-proof and the SNARK proof so that the CP-proof adds no extra cost to the proof size beyond the original SNARK proof.*

Building blocks: A CP-SNARK scheme OSNARK; an (extractable) polynomial commitment scheme PC; a hash commitment scheme HCom; and a linear encoding scheme E with minimum distance δ.

$\mathsf{Setup}(1^\lambda, \mu^*) \to \mathsf{gp}$**:** Given security parameter λ, upper bound μ^* on the number of variables, set m^* so that the running time of OSNARK (and PC) is $O_\lambda(2^{\mu^*})$ for circuit size (and degree) m^*. Run $\mathsf{gp}_o \leftarrow \mathsf{OSNARK.Setup}(1^\lambda, m^*)$, $\mathsf{gp}_{pc} \leftarrow \mathsf{PC.Setup}(1^\lambda, m^*)$, run the indexing phase of OSNARK for the circuit statement in Figure 3.8 and obtain $(\mathsf{vp}_o, \mathsf{pp}_o)$. Output $\mathsf{gp} := (\mathsf{gp}_o, \mathsf{gp}_{pc}, \mathsf{vp}_o, \mathsf{pp}_o)$.

$\mathsf{Commit}(\mathsf{gp}; f) \to C_f$**:** Given polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$ with coefficients $\mathbf{w} = (f_{\langle 0\rangle}, \ldots, f_{\langle n-1\rangle})$, set $m = n/k$ so that the running time of OSNARK (and PC) is $O_\lambda(2^\mu)$ for circuit size (and degree) m. Interpret $\mathbf{w}$ as a $k \times m$ matrix (i.e. $\mathbf{w} \in \mathbb{F}^{k\times m}$):

- Compute matrix $W \in \mathbb{F}^{k\times M}$ such that $W[i,:] = E(\mathbf{w}[i,:]) \forall i \in [k]$. Here $E : \mathbb{F}^m \to \mathbb{F}^M$ is the linear encoding.
- For each column $j \in [M]$, compute hash commitment $h_j \leftarrow \mathsf{HCom}(W[:,j])$, where $W[:,j] \in \mathbb{F}^k$ is the j-th column of W.
- Let p_h be the polynomial that interpolates vector $(h_j)_{j\in[M]}$. Output commitment $C_f \leftarrow \mathsf{PC.Commit}(\mathsf{gp}_{pc}, p_h)$.

$\mathsf{Open}(\mathsf{gp}, C_f, f)$**:** Given polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$ with coefficients $\mathbf{w} \in \mathbb{F}^{k\times m}$, run the committing algorithm and check if the output is consistent with C_f.

$\mathsf{Eval}(\mathsf{gp}; C_f, \mathbf{z}, y; f)$**:** Given public parameter gp, point $\mathbf{z} \in \mathbb{F}^\mu$ and commitment C_f to polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$ with coefficients $\mathbf{w} \in \mathbb{F}^{k\times m}$, transform $\mathbf{z}$ to vectors $\mathbf{t}_0 \in \mathbb{F}^k$ and $\mathbf{t}_1 \in \mathbb{F}^m$ as in Equation (3.15) such that $f(\mathbf{z}) = \langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle$. The prover P and the verifier V run the following protocol:

1. V sends P a random vector $\mathbf{r} \in \mathbb{F}^k$.
2. Define vectors
$$\mathbf{y}_\mathbf{r} = \sum_{i=1}^{k} \mathbf{r}_i \cdot \mathbf{w}[i,:], \qquad \mathbf{y}_0 = \sum_{i=1}^{k} \mathbf{t}_{0,i} \cdot \mathbf{w}[i,:].$$
Let $p_\mathbf{r}$ be the polynomial that interpolates $(\mathbf{y}_\mathbf{r}, \mathbf{y}_0)$. P sends V commitment $C_{p_\mathbf{y}} \leftarrow \mathsf{PC.Commit}(\mathsf{gp}_{pc}, p_\mathbf{y})$.
3. V sends a random subset $I \subseteq [M]$ with size $t := \frac{-\lambda}{\log(1-\delta)}$.
4. P sends V, a CP-SNARK proof π_o showing that
   - the statement in Figure 3.8 holds true;
   - the SNARK witness $(\mathbf{y}_\mathbf{r}, \mathbf{y}_0)$ is identical to the vector committed in $C_{p_\mathbf{y}}$;
   - the SNARK witness $(h_j)_{j\in I}$ is consistent with that in the polynomial commitment C_f w.r.t. set I.
5. V checks π_o with public input $(\alpha, \mathbf{r}, y, \mathbf{z})$, and commitments $C_{p_\mathbf{y}}, C_f$.

Figure 3.7: The multilinear polynomial commitment scheme.

Witness:

- messages $\mathbf{y_r}, \mathbf{y}_0 \in \mathbb{F}^m$, encodings $\mathbf{Y_r}, \mathbf{Y}_0 \in \mathbb{F}^M$, and evaluation vectors $\mathbf{t}_0 \in \mathbb{F}^k$, $\mathbf{t}_1 \in \mathbb{F}^m$;
- the columns of W in subset I, that is, $W' = (W[:, j])_{j \in I} \in \mathbb{F}^{k \times |I|}$;
- the values of $\mathbf{Y_r}, \mathbf{Y}_0$ in subset I, that is $\mathbf{Y'_r} = (\mathbf{Y}_{\mathbf{r},j})_{j \in I} \in \mathbb{F}^{|I|}$, and $\mathbf{Y}'_0 = (\mathbf{Y}_{0,j})_{j \in I} \in \mathbb{F}^{|I|}$;
- column hashes $\mathbf{h} = (h_1, \ldots, h_{|I|}) \in \mathbb{F}^{|I|}$.

Public input:

- challenge vector $\mathbf{r} \in \mathbb{F}^k$;
- random subset $I \subseteq [M]$;
- evaluation point $\mathbf{z} \in \mathbb{F}^\mu$ and claimed evaluation $y \in \mathbb{F}$.

Circuit statements:

- $\mathbf{t}_0, \mathbf{t}_1$ is the correct transformation from $\mathbf{z}$ as in Equation (3.15).
- $\mathbf{Y_r} = E(\mathbf{y_r})$ and $\mathbf{Y}_0 = E(\mathbf{y}_0)$.
- For $i = 1 \ldots |I|$, let $j_i \in I$ be the i-th element in I, it holds that
  - $\mathbf{Y}'_{\mathbf{r},i} = \mathbf{Y}_{\mathbf{r},j_i}$, that is, $(j_i, \mathbf{Y}'_{\mathbf{r},i})$ is in the table $\{(k, \mathbf{Y}_{\mathbf{r},k})\}_{k \in [M]}$, and
  - $\mathbf{Y}'_{0,i} = \mathbf{Y}_{0,j_i}$, that is, $(j_i, \mathbf{Y}'_{0,i})$ is in the table $\{(k, \mathbf{Y}_{0,k})\}_{k \in [M]}$.
- For $i = 1 \ldots |I|$, it holds that
  - $h_i = \mathsf{HCom}(W'[:, i])$ where $\mathsf{HCom} : \mathbb{F}^k \to \mathbb{F}$ is the hash commitment scheme;
  - $\mathbf{Y}'_{\mathbf{r},i} = \langle \mathbf{r}, W'[:, i] \rangle$ and $\mathbf{Y}'_{0,i} = \langle \mathbf{t}_0, W'[:, i] \rangle$.
- $\langle \mathbf{y}_0, \mathbf{t}_1 \rangle = y$.

Figure 3.8: The outer SNARK circuit statement. The circuit configuration is independent of the random set I.

Theorem 3.12. *The multilinear polynomial commitment scheme in Figure 3.7 is correct and binding. The PCS evaluation protocol is knowledge-sound.*

Proof. **Correctness and binding.** Correctness holds obviously by inspection of the protocol. We prove the binding property by contradiction. Suppose an adversary finds a commitment C_f and two polynomials f_1, f_2 with different coefficients $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{F}^{k \times m}$ such that C_f can open to both $\mathbf{w}_1$ and $\mathbf{w}_2$. There are two cases:

1. C_f can open to two different vectors of column hash commitments $\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{F}^M$, which contradicts the binding property of the PCS PC.
2. C_f binds to a single vector $\mathbf{h} \in \mathbb{F}^M$, but encoding $\mathbf{w}_1, \mathbf{w}_2$ lead to two different encoded matrices $W_1, W_2 \in \mathbb{F}^{k \times M}$. This contradicts the collision resistance of the hash function.

In summary, the binding property holds.

Knowledge soundness. We use a similar technique as in [Gol+21] that enables extracting polynomials even if the linear code E is not efficiently decodable. For any adversary $\mathcal{A}$ that can pass the PCS evaluation check with probability more than ϵ, the extractor $\mathsf{Ext}^{\mathcal{A}}$ works as follows:

1. Run $\mathcal{A}$ and obtain commitment C_f, point $\mathbf{z} \in \mathbb{F}^\mu$, and evaluation $y \in \mathbb{F}$. Run the extractors of the PCS and the hash function to recover the matrix $W' \in \mathbb{F}^{k \times M}$ underlying C_f. Abort if the extraction fails.

2. Set $S \leftarrow \emptyset$, repeat the following procedures until $|S| \geq k$ or the number of $\mathbf{r}$ being sampled is more than $8k/\epsilon$:

   - Sample and send $\mathcal{A}$ a random vector $\mathbf{r} \leftarrow\!\!\$\ \mathbb{F}^k$.
   - Obtain the PCS commitments $C_{p_{\mathbf{y}}}$. Use the PCS extractor to extract the vector $(\mathbf{y_r}, \mathbf{y}_0) \in \mathbb{F}^{2m}$. Abort and rerun with another $\mathbf{r}$ if the extraction fails.
   - Sample and send $\mathcal{A}$ a random subset $I \subseteq [M]$.
   - Obtain the CP-SNARK proof π_o. Add the pair $(\mathbf{r}, \mathbf{y_r})$ into set S if the proof correctly verifies.

3. If $|S| \geq k$ and the random vectors $\{\mathbf{r}\}$ in S are linearly independent, run the Gaussian elimination algorithm to extract the witness $\mathbf{w}$ from $S = \{(\mathbf{r}, \mathbf{y_r})\}$, otherwise abort.

The extractor runs in polynomial time as Step 2 runs in polynomial time, and the extractor executes Step 2 for at most $8k/\epsilon$ times. Next, we argue that the extractor's success probability is non-negligible. Since $\mathcal{A}$ succeeds with probability at least ϵ, with probability at least $\epsilon/2$ over the choice of $(C_f, \mathbf{z}, y)$, the adversary passes the PCS evaluation protocol $\mathsf{Eval}(\mathsf{gp}; C_f, \mathbf{z}, y)$ with probability at least $\epsilon/2$. We denote by B the event that the above happens.

Conditioned on event B, we first argue that with high probability, Ext can add k pairs to S, and the $\mathbf{r}$'s in S are linearly-independent. Note that for each run of PCS evaluation (with a freshly sampled vector $\mathbf{r}$), the probability that the extractor adds a pair to S is at least $\epsilon/2 - \mathsf{negl}(\lambda) \geq \epsilon/4$. This is because $\mathcal{A}$ passes the checks with probability at least $\epsilon/2$, and thus with probability at least $\epsilon/2 - \mathsf{negl}(\lambda)$, $\mathcal{A}$ passes all the checks, and the PCS extractor suceeds. Therefore, by Chernoff bound, the probability that Ext adds k pairs to S within $8k/\epsilon$ runs of Step 2 is at least $1 - \exp(-k/8)$. Moreover, as noted by Lemma 2 of [Gol+21], the random vectors $\{\mathbf{r}\}$ in set S are linearly independent with overwhelming probability.

Next, still conditioned on event B, we argue that with probability $1 - \mathsf{negl}(\lambda)$, there exists a coefficient matrix $\mathbf{w} \in \mathbb{F}^{k \times m}$ that is consistent with the commitment C_f, such that $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$ (i.e. the evaluation is correct) and $\mathbf{y_r} = \sum_{i=1}^{k} \mathbf{r}_i \cdot \mathbf{w}_i$ for every pair $(\mathbf{r}, \mathbf{y_r})$ in set S. Let $W' \in \mathbb{F}^{k \times M}$ be the matrix extracted by Ext at Step 1, note that W' commits to C_f. Consider each run of PCS evaluation where the extractor adds a pair $(\mathbf{r}, \mathbf{y_r})$ to S. Since $C_f, C_{p_{\mathbf{y}}}$ are binding, and the SNARK proofs verify, it holds that w.h.p over the choice of I, $E(\mathbf{y_r})$ is close to $\sum_{i=1}^{k} \mathbf{r}_i \cdot W'_i$. By Lemma 1

in [Gol+21], w.h.p. over the choice of $\mathbf{r}$, it also holds that W'_i is close to a codeword for all $i \in [k]$. Therefore, there exists a matrix $\mathbf{w} \in \mathbb{F}^{k\times m}$ such that (i) W'_i is close to $E(\mathbf{w}_i)$ for all $i \in [k]$, and (ii) $\mathbf{y_r} = \sum_{i=1}^{k} \mathbf{r}_i \cdot \mathbf{w}_i$. Moreover, by the uniqueness of encoding, $\mathbf{w}$ is identical for every challenge vector $\mathbf{r}$ in set S. Similarly, we can argue that $\mathbf{y}_0 = \sum_{i=1}^{k} \mathbf{t}_{0,i} \cdot \mathbf{w}_i$ and thus $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$.

Given the above, we conclude that with high probability, it holds that (i) Ext adds k pairs to S where the $\mathbf{r}$'s in S are linearly independent; and (ii) there exists $\mathbf{w} \in \mathbb{F}^{k\times m}$ that is consistent with C_f and $\langle \mathbf{w}, \mathbf{t}_0 \otimes \mathbf{t}_1 \rangle = y$ and $\mathbf{y_r} = \sum_{i=1}^{k} \mathbf{r}_i \cdot \mathbf{w}_i$ for every pair $(\mathbf{r}, \mathbf{y_r})$ in set S. In summary, conditioned on event B, the extractor can extract the coefficient matrix $\mathbf{w}$ via Gaussian elimination with high probability, which completes the proof. □

Theorem 3.13. *When instantiating the outer SNARK with* HyperPlonk+*, the multilinear PCS in Figure 3.7 has committing and evaluation opening complexity $O_\lambda(n)$; the proof size and verifier time is $O_\lambda(\log n)$.*

Proof. Fix $k = \Theta(\lambda/\log n)$ and let $m = n/k$. The committing algorithm takes $O(n)$ $\mathbb{F}$-ops to encode the coefficients $\mathbf{w} \in \mathbb{F}^{k\times m}$ to $W \in \mathbb{F}^{k\times M}$, $O(n)$ hashes to compute the column commitments, and an $O(m)$-sized MSM to commit to the vector of column commitments. The total complexity is $O_\lambda(n)$.

The evaluation proving mainly consists of the following steps:

- compute a HyperPlonk+ SNARK proof for the statement in Figure 3.8;
- compute a CP-SNARK proof between the commitment C_f and the SNARK witness polynomial commitment with respect to a set I.

By the linear algorithm specified in Section 3.3.8, the CP-SNARK proof generation is dominated by a multi-group-exponentiations with size s, where s is the circuit size; similarly, the HyperPlonk+ SNARK proving is also dominated by a few multi-group-exponentiations with size s. Next, we prove that the outer circuit complexity is $s = O(m)$, Hence the evaluation opening complexity is also $O_\lambda(n)$.

Lemma 3.10. *The number of constraints in the circuit in Figure 3.8 is $O(m) + |H| \cdot O(k\lambda)$, where $|H|$ is the number of constraints for a hash instance.*

Proof. The circuit for computing $\mathbf{t}_0$, $\mathbf{t}_1$ from $\mathbf{z}$ takes $O(k)$ and $O(m)$ constraints, respectively; the circuit for encoding $\mathbf{y_r}, \mathbf{y}_0$ takes $O(m)$ constraints; the extraction of $\mathbf{Y}'_\mathbf{r}, \mathbf{Y}'_0$ from $\{E(\mathbf{y_r})_j, E(\mathbf{y}_0)\}_{j\in I}$, takes $2|I|$ lookup gates; the computation of $\mathbf{Y}'_\mathbf{r}$, $\mathbf{Y}'_0$ takes $O(k|I|)$ constraints; the computation of y takes $O(m)$ constraints; the computation of hashes $\{h_i\}$ takes $k|I| = O(k\lambda)$ hash gadgets as $|I| = \Theta(\lambda)$, thus the number of constraints is $O(m) + |H| \cdot O(k\lambda)$. □

The evaluation verifier checks the the CP-SNARK proof π_o which takes time $O_\lambda(\log n)$.

The evaluation proof consists of a single CP-SNARK proof π_o. As noted by Remark 3.7.1, the proof size is no more than that of a single HyperPlonk+ proof for circuit size $s = O(m)$. In summary, the proof size is $O_\lambda(\log n)$. □

Remark 3.7.2. *We stress that the CP-SNARK proving time (between C_f and the SNARK witness) for set I is independent of the size of I, as the complexity of the special batching algorithm in Section 3.3.8 is independent of the number of evaluations. This is highly important because $|I|$ can be as large as thousands in practice.*

Remark 3.7.3. *If we instantiate the linear code with the generalized Spielman code proposed in [XZS22a], and instantiate the outer SNARK with vanilla* HyperPlonk+*, for* 128*-bit security and $\mu = 30$, the outer circuit size is approximately 2^{26}, thus the proof is less than* 10 KB*s.*

Remark 3.7.4. *In practice, to minimize the outer circuit complexity, we choose k such that $2 \cdot \ell(n/k) = k|I| \cdot |H|$, where $\ell(n/k)$ is the circuit size for encoding a message with length n/k. Note that $|I| = 1568$ for* 128*-bit security and $|I| = 980$ for* 80*-bit security.*

Remark 3.7.5. *In contrast with* Orion*,* Orion+ *requires using a pairing-friendly field. We leave the construction of linear-time PCS with succinct proofs/verifier that supports arbitrary fields as future work.*

3.8 Zero Knowledge PIOPs and zk-SNARKs

In this Section, we describe a compiler that transforms a class of sumcheck-based multivariate PolyIOPs into ones that are zero knowledge. The general framework consists of two parts. The first part is to mask the oracle polynomials so that their oracle query answers do not reveal the information of the original polynomial; moreover, we require that the masking do not change evaluations over the boolean hypercube, thus the correctness of PIOPs still holds. The second part is making the underlying sumcheck PIOPs zero knowledge. For this we reuse the ZK sumchecks described in [Xie+19].

We note that in contrast with univariate PIOPs, there is a subtlety in compiling multivariate PIOPs: the zero-knowledge property is hard to achieve if the set of query points is highly structural. E.g., suppose f is 2-variate and there are are 4 query points (r_1, r_2), (r_1, r_1), (r_2, r_1), (r_2, r_2). Though all of the 4 points are distinct, each dimension has at least 2 points that share the same value. This makes the adversary much easier to cancel out the masking randomness and obtain a correlation between the evaluations of f on the 4 points. We resolve the issue by restricting the set of query points to be less structured. In particular, we require that there is at least one dimension where each point has a distinct value. We also slightly modify the underlying sumcheck protocols to satisfy the restriction while the soundness is not affected.

The Section is organized as follows. We define zero knowledge PIOPs in Section 3.8.1. In Section 3.8.2, we describe a scheme masking the multivariate polynomials. Section 3.8.3 reviews the ZK sumchecks in [Xie+19]. We describe the ZK compiler for PIOPs in Section 3.8.4 and explain how to obtain a zk-SNARK from a zk-PIOP and a PCS in Section 3.8.5.

3.8.1 Definition

We follow [Chi+20] and define the (honest verifier) zero-knowledge property of PIOPs. Since the provers in sumcheck PIOPs also send field elements, we slightly adapt the definition in [Chi+20].

Definition 3.15. *A PIOP $\langle \mathsf{P}, \mathsf{V} \rangle$ has perfect zero-knowledge with query bound t and query checker C if there is a PPT simulator $\mathcal{S}$ such that for every field $\mathbb{F}$, index $\mathtt{i}$, instance $\mathtt{x}$, witness $\mathtt{w}$, and every (t, C)-admissable verifier V^*, the following transcripts are identically distributed:*

$$\mathtt{View}\left(\mathsf{P}(\mathbb{F}, \mathtt{i}; \mathtt{x}; \mathtt{w}), \mathsf{V}^*\right) \approx \mathcal{S}^{\mathsf{V}^*}(\mathbb{F}, \mathtt{i}; \mathtt{x}) .$$

Here the view consists of V^*'s randomness, the non-oracle messages sent by P, and the list of answers to V^*'s oracle queries. A verifier is (t, C)-admissible if it makes no more than t queries, and each query is accepted by the checker C. We say that $\langle \mathsf{P}, \mathsf{V} \rangle$ is honest-verifier-zero-knowlege *(HVZK) if there is a simulator for V.*

3.8.2 Polynomial masking

Definition 3.16. *A randomized algorithm msk is a (t, C, μ)-masking if*

1. *for every $d \in \mathbb{N}$ and every polynomial $f \in \mathcal{F}_\mu^{(\leq d)}$, the masked polynomial $f^* \leftarrow\!\!\$\ \mathsf{msk}(f, t, C)$ does not change evaluations over the boolean hypercube B_μ;*
2. *for every $d \in \mathbb{N}$ and every polynomials $f \in \mathcal{F}_\mu^{(\leq d)}$, and every list of queries $\mathbf{q} := (q_1, \ldots, q_t)$ that is accepted by the checker C, let $f^* \leftarrow\!\!\$\ \mathsf{msk}(f, t, C)$. It holds that $\big(f^*(q_1), \ldots, f^*(q_t)\big)$ is uniformly distributed over $\mathbb{F}^t$.*

Lemma 3.11. *There is a (t, C_ℓ, μ)-masking algorithm $\mathsf{msk}(f, t, \ell)$ for every $\mu, t \in \mathbb{N}$ and $\ell \in [\mu]$, where checker C_ℓ accepts a list of queries $(q_1, \ldots, q_t)$ if and only if $b_{i,\ell} \notin \{0, 1, b_{1,\ell}, \ldots, b_{i-1,\ell}\}$ for every query $q_i := (b_{i,1}, \ldots, b_{i,\mu}) \in \mathbb{F}^\mu$ $(1 \leq i \leq t)$. For any $f \in \mathcal{F}_\mu^{(\leq d)}$ and $\ell \in [\mu]$, the degree of the masked polynomial $f^* \leftarrow \mathsf{msk}(f, t, \ell)$ is $\max(d, t+1)$.*

Proof. Given a polynomial $f \in \mathcal{F}_\mu^{(\leq d)}$, query bound t, and checker C_ℓ, the algorithm does follow:

- Sample a univariate polynomial $R(X) := c_0 + c_1 X + \ldots c_{t-1} X^{t-1}$ where $c_0, \ldots, c_{t-1} \leftarrow\!\!\$\ \mathbb{F}$.
- Output $f^* := f + Z(X_\ell) \cdot R(X_\ell)$, where $Z(X_\ell) := X_\ell \cdot (1 - X_\ell)$.

It is clear that f^* has degree $\max(d, t+1)$; f^* does not change f's evaluations over B_μ as Z evaluates to zero over B_μ. Next, we argue that $\mathbf{f}^* := \big(f^*(q_1), \ldots, f^*(q_t)\big) \in \mathbb{F}^t$ is uniformly random. Denote query $q_i := (b_{i,1}, \ldots, b_{i,\mu})$ $(1 \le i \le t)$, we define $\mathbf{R}$ to be

$$\mathbf{R} := \big(Z(b_{1,\ell}) \cdot R(b_{1,\ell}),\ \ldots,\ Z(b_{t,\ell}) \cdot R(b_{t,\ell})\big)\,.$$

Since the queries satisfy $b_{i,\ell} \notin \{0,1\}$ for every $i \in [t]$, it holds that $z_i := Z(b_{i,\ell})$ are non-zero and thus invertible. Moreover, since R is a random univariate polynomial with degree $t-1$ and $\{b_{1,\ell}, \ldots, b_{t,\ell}\}$ are distinct, it holds that $\{R(b_{1,\ell}), \ldots, R(b_{t,\ell})\}$ are uniformly random. Therefore $\mathbf{R}$ is uniformly random, and thus $\mathbf{f}^* = \mathbf{f} + \mathbf{q}$ is also uniformly random where $\mathbf{f} := \big(f(q_1), \ldots, f(q_t)\big)$. □

3.8.3 Zero knowledge SumCheck

Construction. Xie et al. [Xie+19] described an efficient ZK compiler for sumchecks. For reader's convenience, we adapt Construction 1 in [Xie+19] to a PIOP.

Zero knowledge SumCheck PIOP $\langle \mathsf{P}, \mathsf{V} \rangle$:

- Input: polynomial $f \in \mathcal{F}_\mu^{(\le d)}$ and claimed sum $H \in \mathbb{F}$.
- P samples a polynomial $g := c_0 + g_1(\mathbf{x}_1) + \cdots + g_\mu(\mathbf{x}_\mu)$ where $g_i(\mathbf{x}_i) := c_{i,1}\mathbf{x}_i + \cdots + c_{i,d}\mathbf{x}_i^d$ and $c_{i,1}, \ldots, c_{i,d}$ are uniformly random. P sends oracle g and a claimed sum $G := \sum_{\mathbf{x} \in B_\mu} g(\mathbf{x})$.
- V sends a challenge $\rho \leftarrow\!\$\ \mathbb{F}^*$.
- P and V run SumCheck PIOP (Section 3.3.1) over polynomial $f + \rho g$ and claimed sum $H + \rho G$.
- V queries g and f at point $\mathbf{r}$ where $\mathbf{r} \in \mathbb{F}^\mu$ is the vector of sumcheck's challenges. V then checks that $f(\mathbf{r}) + \rho g(\mathbf{r})$ is consistent with the last message of the sumcheck.

The completeness of the ZK PIOP holds obviously, it was shown in [CFS17] that the PIOP also preserves soundness. The zero knowledge property is proved in [Xie+19] and we state it below.

Lemma 3.12 (Theorem 3 of [Xie+19])**.** *For every field $\mathbb{F}$, verifier V^* and multivariate polynomial $f \in \mathcal{F}_\mu^{(\le d)}$, there is a simulator $\mathcal{S}_{sum}(\mathbb{F}, \mu, d, H)$ that perfectly simulates P's oracle answers except for $f(\mathbf{r})$. Here $H := \sum_{\mathbf{x} \in B_\mu} f(\mathbf{x})$.*

3.8.4 Zero knowledge compilation for SumCheck-based PIOPs

A general description to the sumcheck-based PIOPs. The multivariate PIOPs considered in this paper can all be adapted to the following format.

General sumcheck-based PIOPs:

1. Both P and V have oracle access to a public multilinear polynomial $p_0 \in \mathcal{F}_{\mu_0}^{(\le 1)}$.

2. For every $i \in [k_1]$, P sends a multilinear polynomial $p_i \in \mathcal{F}_{\mu_i}^{(\leq 1)}$, and V sends some random challenges. p_i is a function of $p_0, \ldots, p_{i-1}$ and verifier's previous challenges.

3. P and V sequentially run k_2 sumcheck PIOPs. The i-th ($1 \leq i \leq k_2$) sumcheck is over a polynomial $f_i := h_i(g_1, \ldots, g_{c_i}) \in \mathcal{F}_{\nu_i}^{(\leq d_i)}$, where h_i is public information and each multilinear polynomial $g_j \in \mathcal{F}_{\nu_i}^{(\leq 1)}$ ($1 \leq j \leq c_i$) is $g_j := v_{|\mathbf{X}_S = \boldsymbol{b}}$ for some boolean vector $\boldsymbol{b}$ and some $v \in \{p_1, \ldots, p_{k_1}\}$, that is, g_j is a partial polynomial of v where the variables in S are set to $\boldsymbol{b}$.

4. For every $i \in [k_2]$, V queries a random point $\mathbf{r}_i \in \mathbb{F}^{\nu_i}$ to the oracle f_i, where $\mathbf{r}_i$ are the round challenges in the i-th sumcheck. V then checks that $f_i(\mathbf{r}_i)$ is consistent with the last message in the i-th sumcheck.

5. For every $i \in [k_3]$, the verifier queries a point $\mathbf{c}_i \in \mathbb{F}^{\mu_{j_i}}$ to an oracle p_{j_i} ($0 \leq j_i \leq k_1$) and checks that the evaluation is y_i. We emphasize that the evaluations $\{y_i\}_{i \in [k_3]}$ can be efficiently and deterministically derived from $\{\mathbf{c}_i, j_i\}_{i \in [k_3]}$ and the public oracle p_0.

We note that the above description captures all of the multivariate PIOPs in this paper because

- for the case where P sends an oracle $f := h(g_1, \ldots, g_c) \in \mathcal{F}_{\mu}^{(\leq d)}$ for $d > 1$, we can instead let P send $g_1, \ldots, g_c \in \mathcal{F}_{\mu}^{(\leq 1)}$ as h is public information;
- for the case where P sends multiple multilinear oracles in a round, we can merge the polynomials into a single polynomial;
- the PIOPs we consider are all finally reduced to one or more sumcheck PIOPs.

Construction. We present a generic framework that transforms any (sumcheck-based) multivariate PIOPs into zero knowledge PIOPs. For a PIOP $\langle \mathsf{P}, \mathsf{V} \rangle$, let $\big(\{p_i\}_{i \in [0,k_1]}, \{f_i\}_{i \in [k_2]}\big)$ be the polynomials denoted in the above protocol. For every $i \in [k_1]$, let $t_i \in \mathbb{N}$ be the number of p_i's partial polynomials that appear in the sumcheck polynomials $f_1, \ldots, f_{k_2}$, and let $t^* := \max\{t_i\}_{i \in [k_1]}$. For every $i \in [k_1]$, we assume that there exists index $\ell_i \in [\mu_i]$ such that for every p_i's partial polynomial $v_{|\mathbf{X}_S = \boldsymbol{b}}$ that appears in some sumcheck (where p_i's variables in set S are boolean), it holds that ℓ_i is not in the set S. Let msk be the masking algorithm described in Lemma 3.11. The compiled zero knowledge PIOP $\langle \hat{\mathsf{P}}, \hat{\mathsf{V}} \rangle$ works as follows.

The ZK-compiler for sumcheck-based PIOPs:

1. For every $i \in [k_1]$, $\hat{\mathsf{P}}$ sends an oracle $[[p_i^*]]$ where $p_i^* \leftarrow\!\$\ \mathsf{msk}(p_i, t_i, \ell_i)$. $\hat{\mathsf{V}}$ sends the same challenges as V does.

2. $\hat{\mathsf{P}}$ and $\hat{\mathsf{V}}$ sequentially run k_2 zero knowledge sumcheck PIOPs (Section 3.8.3). The i-th ($1 \leq i \leq k_2$) sumcheck is over the polynomial $f_i^* := h_i(g_1^*, \ldots, g_{c_i}^*) \in \mathcal{F}_{\nu_i}^{(\leq d_i t^*)}$, where h_i is the same as in $\langle \mathsf{P}, \mathsf{V} \rangle$; each $g_j^* \in \mathcal{F}_{\nu_i}^{(\leq t^*)}$ ($1 \leq j \leq c_i$) is $g_j^* := v^*_{|\mathbf{X}_S = \boldsymbol{b}}$ for some boolean vector $\boldsymbol{b}$ and some $v^* \in \{p_1^*, \ldots, p_{k_1}^*\}$.

3. For every $i \in [k_2]$, $\hat{\mathsf{V}}$ queries a random point $\mathbf{r}_i \in \mathbb{F}^{\nu_i}$ to the oracle f_i, where $\mathbf{r}_i$ are the round challenges in the i-th ZK sumcheck. $\hat{\mathsf{V}}$ then checks that $f_i(\mathbf{r}_i)$ is consistent with the last message of the i-th ZK sumcheck. We emphasize a slight modification over the original PIOP $\langle \mathsf{P}, \mathsf{V} \rangle$: *in the i-th sumcheck, $\hat{\mathsf{V}}$ samples each round challenge $\mathbf{r}_{i,j}$ $(1 \le j \le \mu_i)$ in the set $\mathbb{F} \setminus \{0, 1, \mathbf{r}_{1,j}, \ldots, \mathbf{r}_{i-1,j}\}$ rather than in $\mathbb{F}$.*

4. $\hat{\mathsf{V}}$ simulates V, i.e., for all $i \in [k_3]$, queries points $\mathbf{c}_i$ to oracle $p^*_{j_i}$ and checks the evaluation.

Theorem 3.14. *Given any PIOP $\langle \mathsf{P}, \mathsf{V} \rangle$ for some relation over the boolean hypercube, the compiled PIOP $\langle \hat{\mathsf{P}}, \hat{\mathsf{V}} \rangle$ is HVZK. Moreover, $\langle \hat{\mathsf{P}}, \hat{\mathsf{V}} \rangle$ preserves perfect completeness and negligible soundness.*

Proof. **Completeness.** Completeness holds because the sumcheck relations are over boolean hypercubes and the masked polynomials' evaluations do not change over the boolean hypercubes by the property of msk.

Soundness. Compared to the sumchecks in $\langle \mathsf{P}, \mathsf{V} \rangle$, the following changes of the sumchecks in $\langle \hat{\mathsf{P}}, \hat{\mathsf{V}} \rangle$ affect soundness error:

1. The degrees of the sumcheck polynomials are increased by a factor t^*.
2. The challenge space of j-th round in the i-th $(1 \le i \le k_2)$ sumcheck is $\mathbb{F} \setminus \{0, 1, \mathbf{r}_{1,j}, \ldots, \mathbf{r}_{i-1,j}\}$ rather than $\mathbb{F}$.
3. The sumcheck protocols are replaced with ZK sumchecks.

Since t^* and k_2 are constants and ZK sumchecks preserves soundness [CFS17], the compiled protocol preserves negligible soundness.

HVZK. We describe the simulator as follows.

The simulator $\mathcal{S}^{\hat{\mathsf{V}}}(\mathbb{F}, \mathbb{i}; \mathbb{x})$:

1. Honestly generate the public polynomial $p_0 \in \mathcal{F}^{(\le 1)}_{\mu_0}$.
2. Pick arbitrary polynomial $\{\tilde{p}_i\}_{i \in [k_1]}$ conditioned on that the sumcheck relations over $f_1, \ldots, f_{k_2}$ hold. Send $\hat{\mathsf{V}}$ polynomials $\{\tilde{p}^*_i\}_{i \in [k_1]}$ where $\tilde{p}^*_i \leftarrow\!\!\$\ \mathsf{msk}(\tilde{p}_i, t_i, \ell_i)$, obtain from $\hat{\mathsf{V}}$ the challenges in the first k_1 rounds.
3. Run the next k_2 ZK sumcheck PIOPs using p_0 and the sampled polynomials $\{\tilde{p}^*_i\}_{i \in [k_1]}$.
4. For every $i \in [k_2]$, answer query $f^*_i(\mathbf{r}_i)$ honestly using $\{\tilde{p}^*_i\}_{i \in [k_1]}$.
5. For every $i \in [k_3]$, answer query $\mathbf{c}_i$ with value y_i, where $\{y_i\}_{i \in [k_3]}$ are deterministically derived from $\{\mathbf{c}_i, j_i\}_{i \in [k_3]}$ and the public polynomial p_0.

Next we show that $\mathcal{S}^{\hat{\mathsf{V}}}(\mathbb{F}, \mathbb{i}; \mathbb{x}) \approx \mathtt{View}(\hat{\mathsf{P}}(\mathbb{F}, \mathbb{i}; \mathbb{x}; \mathbb{w}), \hat{\mathsf{V}})$. We set $H_0 := \mathcal{S}^{\hat{\mathsf{V}}}(\mathbb{F}, \mathbb{i}; \mathbb{x})$ and consider following hybrid games.

- Game H_1: identical to H_0 except that step 3 is replaced with the ZK sumcheck simulator's output. We note that $H_1 \approx H_0$ by the ZK property of the ZK sumchecks.
- Game H_2: identical to H_1 except that the queries in step 4 are answered with random values. (Note that $f_i^*(\mathbf{r}_i)$'s answer is a random value consistent with the last message of the i-th sumcheck.) We argue that $H_2 \approx H_1$: for every $i \in [k_1]$, the number of queries to oracle $\tilde{p}_i^* \leftarrow \mathsf{msk}(\tilde{p}_i, t_i, \ell_i)$ is no more than t_i and the ℓ_i-th element in each of the query point are distinct and non-boolean, by Lemma 3.11, the answers to the queries are uniformly random.
- Game H_3: identical to H_2 except that the polynomials $\{\tilde{p}_i\}_{i\in[k_1]}$ in step 2 are replaced with $\{p_i\}_{i\in[k_1]}$. Note that $H_3 \approx H_2$ as the verifier's view does not change at all.
- Game $H_4 := \mathtt{View}(\hat{\mathsf{P}}(\mathbb{F}, \mathbb{i}; \mathbb{x}; \mathbb{w}), \hat{\mathsf{V}})$: identical to H_3 except that the queries in step 4 are answered honestly and the ZK sumchecks are run honestly using p_0 and the sampled polynomials $\{p_i^*\}_{i\in[k_1]}$. With similar arguments (for H_1 and H_2) we have $H_4 \approx H_3$.

Given above, it holds that $\mathcal{S}^{\hat{\mathsf{V}}}(\mathbb{F}, \mathbb{i}; \mathbb{x}) \approx \mathtt{View}(\hat{\mathsf{P}}(\mathbb{F}, \mathbb{i}; \mathbb{x}; \mathbb{w}), \hat{\mathsf{V}})$ and we complete the proof. □

3.8.5 zk-SNARKs from PIOPs

In the ZK PIOP of Section 3.8.4, the masked polynomials sent by the prover are with the form $f^* := f + Z(\mathbf{x}_\ell) \cdot R(\mathbf{x}_\ell)$ where $f \in \mathcal{F}_\mu^{(\leq 1)}$ is multilinear and $Z(\mathbf{x}_\ell) := \mathbf{x}_\ell \cdot (1 - \mathbf{x}_\ell)$ is univariate and with degree $t + 1$. It is shown in Theorem 10 of [Bon+21] that every additive and m-spanning PCS can be compiled into a hiding PCS with a zero-knowledge Eval protocol, where m-spanning means that commitments to polynomials of degree at most m can already generate the commitment space $\mathbb{G}$. Thus we can construct a hiding PCS for f^* with ZK evaluations from any additive and spanning polynomial commitment schemes (e.g., KZG and FRI). In particular, one instantiation is to set the commitment of f^* to be $(C_1, C_2) \in \mathbb{G}$ where C_1 is the multilinear commitment to f and C_2 is the univariate commitment to $Z(X) \cdot R(X)$, then apply the ZK transformation in [Bon+21].

By combining Theorem 3.1 and Theorem 3.14 we obtain the following corollary.

Corollary 3.15. *Given any (non-hiding) additive and spanning polynomial commitment schemes, we can transform any (non-ZK) sumcheck-based PIOP (Section 3.8.4) for relation $\mathcal{R}$ to a zk-SNARK for $\mathcal{R}$.*

3.9 The FRI-based multilinear polynomial commitment

In this Section, we construct a simple multilinear polynomial commitment scheme (PCS) from FRI [Ben+18a]. Along the way, we also show how to generically transform a univariate PCS to a multilinear PCS using the tensor-product *univariate* PIOP from [Boo+22b], which might be of independent interest. We note that Virgo [Zha+20, §3] describes another scheme constructing

multilinear PCS from FRI. The main idea is to build the evaluation opening proof from a univariate sumcheck [Ben+19b], which in turn uses FRI. However, the naive scheme incurs linear-time overhead for the verifier. Virgo [Zha+20, §3] resolves the issue by delegating the verifier computation to the prover. To this end, the prover needs to compute another GKR proof convincing that the linear-time verifier will accept the proof. This complicates the scheme and adds additional concrete overhead on prover time and proof size.

We refer to [Ben+18a; KPV19] and [Hab22a] for background of FRI low-degree testing and the approach to build univariate PCS from FRI. We note that the FRI-based univariate PCS supports batch opening. The evaluation opening protocol for multiple points on multiple polynomials invokes only a *single* call to the FRI protocol. Below we present a generic approach to transforming any univariate PCS into a multilinear PCS.

Generic transformation from univariate PCS to multilinear PCS. Bootle et al. built a *univariate* PIOP for the tensor-product relation in Section 5 of [Boo+22b]. The tensor-product relation $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, n, z_1, \ldots, z_\mu, y), \mathbf{f}\big)$ states that $\mathbf{f} \in \mathbb{F}^n$ satisfies that $\langle \mathbf{f}, \otimes_j (1, z_j) \rangle = y$, where $\langle \cdot, \cdot \rangle$ denotes an inner product, and $\otimes$ denotes a tensor product. The PIOP naturally implies an algorithm that transforms univariate polynomial commitment schemes to multilinear polynomial commitment schemes.

- The commitment to a multilinear polynomial $\tilde{f}$ with monomial coefficients, $\mathbf{f}$ is the commitment to a univariate polynomial f with the same coefficients.
- To open $\tilde{f}$ at point $(z_1, \ldots, z_\mu)$ that evaluates y, the prover and the verifier runs the univariate PIOP for the relation $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, n, z_1, \ldots, z_\mu, y), \mathbf{f}\big)$, which reduces to a batch evaluation on a set of $\mu + 1$ univariate polynomials.

We provide the concrete construction below. Let $\mathsf{PC}_u = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{BatchOpen}, \mathsf{BatchVfy})$ be a univariate PCS, we construct a multilinear PCS PC_m as follows.

- $\mathsf{PC}_m.\mathsf{Setup}(1^\lambda, \mu) \to (\mathsf{ck}, \mathsf{vk})$. On input security parameter λ and the number of variables μ, output $\mathsf{PC}_u.\mathsf{Setup}(1^\lambda, n)$ where $n = 2^\mu$.
- $\mathsf{PC}_m.\mathsf{Commit}(\mathsf{ck}, \tilde{f}) \to c$. On input committer key ck, multilinear polynomial $\tilde{f}$ with coefficients $\mathbf{f} \in \mathbb{F}^n$, output $\mathsf{PC}_u.\mathsf{Commit}(\mathsf{ck}, f)$ where f has the same coefficients as $\mathbf{f}$.
- $\mathsf{PC}_m.\mathsf{Open}(\mathsf{ck}, \tilde{f}, \mathbf{z}, y) \to \pi$. On input committer key ck, multilinear polynomial $\tilde{f}$, point $\mathbf{z} \in \mathbb{F}^\mu$ and evaluation $y \in \mathbb{F}$, the prover computes the proof as follows. Let $f_0(X) := f(X)$ be the committed univariate polynomial that has the same coefficients as $\tilde{f}$, consider the following PIOP for the tensor-product relation $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, n, \mathbf{z}, y), \mathbf{f}\big)$:

– The prover sends the verifier univariate polynomials $f_1, \ldots, f_\mu$ such that for all $i \in [\mu]$,

$$f_i(X) = g_{i-1}(X) + \mathbf{z}_i \cdot h_{i-1}(X)\,,$$

where g_{i-1}, h_{i-1} satisfies that $f_{i-1}(X) = g_{i-1}(X^2) + X \cdot h_{i-1}(X^2)$.

– The verifier samples a random challenge $\beta \leftarrow\!\!\$\ \mathbb{F}^\times$ (where $\mathbb{F}^\times$ is a multiplicative subgroup of $\mathbb{F}$), and queries the oracles to obtain evaluations $\{a_i, b_i, c_i\}_{i\in\{0,\ldots,\mu\}}$ such that

$$a_i := f_i(\beta), \qquad b_i := f_i(-\beta), \qquad c_i := f_{i+1}(\beta^2)\,.$$

Note that we skip $f_{\mu+1}(\beta^2)$ and set $c_\mu := y$.

– The verifier checks that for all $i \in \{0, \ldots, \mu\}$,

$$c_i = \frac{a_i + b_i}{2} + z_i \cdot \frac{a_i - b_i}{2\beta}\,.$$

The opening proof π comprises (i) the univariate commitments to $f_1, \ldots, f_\mu$, (ii) the evaluations $\{a_i, b_i, c_i\}_{i\in\{0,\ldots,\mu\}}$, and (iii) the batch opening proof for polynomials $(f_0, f_1, \ldots, f_\mu)$ at points $(\beta, -\beta, \beta^2)$, where the random challenge β is derived via the Fiat-Shamir transform.

- $\mathsf{PC}_m.\mathsf{Vfy}(\mathsf{vk}, c, \mathbf{z}, y, \pi) \in \{0, 1\}$. On input verifier key vk, commitment c, point $\mathbf{z}$, evaluation y, and proof π, parse π to commitments $(c_1, \ldots, c_\mu)$, evaluations evals, and the batch opening proof π^*. Derive random challenge β via the Fiat-Shamir transform, perform the verification check in the above PIOP, and run $\mathsf{PC}_u.\mathsf{BatchVfy}(\mathsf{vk}, (c, c_1, \ldots, c_\mu), (\beta, -\beta, \beta^2), \mathsf{evals}, \pi^*)$.

Efficiency. We emphasize that when instantiated PC_u with the FRI-based PCS, the multilinear polynomial commitment scheme has approximately the same complexity as that in the univariate setting. In particular, the committing phase takes only a Merkle root computation with tree depth $\log(n)$; the opening phase takes (i) μ Merkle commitment computation where the i-th ($1 \le i \le \mu$) Merkle tree is with size $2^{\mu-i}$, and (ii) a univariate PCS batch evaluation protocol that is simply a *single* call to the FRI protocol.

3.10 Unrolled and optimized Hyperplonk

In Figure 3.9 and Figure 3.10, we present an optimized and batched version of HyperPlonk. The protocol batches the zerochecks and additionally batches all evaluations using $\mathcal{R}_{\text{BATCH}}$. Moreover, the sumcheck has complexity proportional to 2^μ rather than $2^{\mu+\nu_w}$ where ν_w is the logarithm of the number of wires.

Proof size analysis of the compiled protocol. We analyze the concrete proof size of the optimized PIOP. We analyze the proof size after compilation, i.e., where the prover sends commitments and performs evaluation proofs. The prover sends

1. $\ell_w + 2$ of μ-variate multilinear polynomial commitments (ℓ_w for the witness and 2 for the product polynomial),
2. μ of degree $\max(d-2, \ell_w - 1)$ univariate polynomial commitments and 2μ claimed evaluations (in the first batched sumcheck),
3. $8 + 2 \cdot \ell_w + \ell_q$ claimed multilinear evaluations,
4. 1 univariate evaluation of a batched univariate polynomial,
5. 1 multilinear evaluation of a batched multilinear polynomial, and
6. $2 \cdot (\mu + \lceil \log_2(8 + 2 \cdot \ell_w + \ell_q) \rceil)$ field elements for the sumcheck in the PIOP of $\mathcal{R}_{\text{BATCH}}$.

For KZG-based commitments, the proof size is $2 + \ell_w + \mu$ $\mathbb{G}_1$ elements and $4\mu + 10 + 2 \cdot \ell_w + \ell_q + 2\lceil \log_2(8 + 2 \cdot \ell_w + \ell_q) \rceil$ field elements. For the case where $\ell_w = \ell_q = 3$, using BLS12-381, where $\mathbb{G}_1$ elements are 48 bytes and field elements are 32 bytes the proof size becomes $176 \cdot \mu + 1168$ bytes. For $\mu = 20$, this is only 4688 bytes.

Indexer. The indexer $\mathcal{I}$ on an input circuit C calls the permutation indexer $\mathcal{I}_{\text{perm}}(\sigma)$: $([[s_{\text{id}}]], [[s_\sigma]]) \leftarrow \mathcal{I}_{\text{perm}}(\sigma)$ and computes the selector polynomial $q \in \mathcal{F}^{(\leq 1)}_{\mu+\nu_q}$. Let $\ell_w = 2^{\nu_w}$ be the number of wires, denote by

$$\mathbf{S}_\sigma := \big([[s_\sigma(\langle 0 \rangle_{\nu_w}, \mathbf{X})]], [[s_\sigma(\langle 1 \rangle_{\nu_w}, \mathbf{X})]], \ldots, [[s_\sigma(\langle \ell_w - 1 \rangle_{\nu_w}, \mathbf{X})]]\big)$$

the lists of partial polynomials of $s_\sigma \in \mathcal{F}^{(\leq 1)}_{\mu+\nu_w}$. The indexer outputs $([[q]], \mathbf{S}_\sigma)$.
We note that for all $i \in [0, 2^{\mu+\nu_w})$, $s_{\text{id}} \in \mathcal{F}^{(\leq 1)}_{\mu+\nu_w}$ evaluates to i at point $\langle i \rangle_{\mu+\nu_w} \in B_{\mu+\nu_w}$ (where $\langle i \rangle_{\mu+\nu_w}$ denotes $\mu + \nu_w$-bit binary encoding of i). Since multilinear extension is unique, it holds that $s_{\text{id}}(\mathbf{X}) = \sum_{i \in [\mu+\nu_w]} 2^{i-1} \cdot \mathbf{X}_i$ and thus one can evaluate s_{id} at any points in time $O(\mu + \nu_w)$. Hence, there is no need for the indexer to output oracle $[[s_{\text{id}}]]$.

Figure 3.9: The indexer of the optimized PIOP for $\mathcal{R}_{\text{PLONK}}$.

3.10.1 Using only one sumcheck

The protocol described in Figure 3.10 has two sumchecks. The latter is the result of the batch-opening protocol (Section 3.3.8). The precise evaluations that are being batched, are

- $1 + \ell_w$ of $\mathbf{W}$ (ℓ_w from the batched sumcheck, one to check the outputs)
- 3 of $\tilde{v}(0, \mathbf{X})$ and 4 of $\tilde{v}(1, \mathbf{X})$ from the product check
- ℓ_q of q (one per selector)

- ℓ_w of $\mathbf{S}_\sigma$ from the product check.

Note that most of these polynomials are evaluated at the same point. The point is exactly the sumcheck challenges. The only divergence is the evaluation of $\tilde{v}(0, \mathbf{X})$ and $\tilde{v}(1, \mathbf{X})$.

$\mathsf{P}(\mathsf{gp}, \mathbb{i}, p, w)$ and $\mathsf{V}(\mathsf{gp}, p, [[q]], \mathbf{S}_\sigma)$ run the following protocol.

1. P sends V the witness oracles $\mathbf{W} := \big([[w_0]], [[w_1]], \ldots, [[w_{\ell_w - 1}]]\big)$ where $w_i := w(\langle i \rangle_{\nu_w}, \mathbf{X})$ is the ith $(0 \le i < \ell_w)$ partial polynomial of the witness polynomial $w \in \mathcal{F}_{\mu+\nu_w}^{(\le 1)}$.
2. V sends input challenge $\mathbf{r}_{IO} \leftarrow\!\!\$\ \mathbb{F}^\nu$, $\mathcal{R}^1_{\text{MSET}}$ challenge γ and $\mathcal{R}^2_{\text{MSET}}$ challenges β.
3. P computes the product polynomial $\tilde{v} \in \mathcal{F}_{\mu+1}^{(\le 1)}$ from $\mathbf{W}, \mathbf{S}_\sigma, s_{\text{id}}$ and the challenges β, γ (See Section 3.3.3), where for all $\mathbf{x} \in B_\mu$,

$$\tilde{v}(0, \mathbf{x}) = \prod_{i \in [0, \ell_w)} \frac{\mathbf{W}_i(\mathbf{x}) + \beta \cdot s_{\text{id}}(\langle i \rangle_{\nu_w}, \mathbf{x}) + \gamma}{\mathbf{W}_i(\mathbf{x}) + \beta \mathbf{S}_{\sigma,i}(\mathbf{x}) + \gamma}\,.$$

   Here $\mathbf{W}_i, \mathbf{S}_{\sigma,i}$ denotes the ith polynomial in $\mathbf{W}, \mathbf{S}_\sigma$ respectively. P then sends oracles $[[\tilde{v}(0, \mathbf{X})]], [[\tilde{v}(1, \mathbf{X})]]$ to V.
4. Verifier sends challenges α_1, α_2 to batch three zerochecks, one resulting from the gate identity (see Section 3.4.2) and two from the productcheck (see Section 3.3.3). The two zerocheck virtual polynomials $Q_1(\mathbf{X}) \in \mathcal{F}_\mu^{(\le 2)}$, $Q_2(\mathbf{X}) \in \mathcal{F}_\mu^{(\le \ell_w + 1)}$ for the productcheck are $Q_1(\mathbf{X}) := \tilde{v}(1, \mathbf{X}) - \tilde{v}(\mathbf{X}, 0)\tilde{v}(\mathbf{X}, 1)$ and

$$Q_2(\mathbf{X}) := \prod_{i \in [0, \ell_w)} (\mathbf{W}_i(\mathbf{X}) + \beta \cdot s_{\text{id}}(\langle i \rangle_{\nu_w}, \mathbf{X}) + \gamma) - \tilde{v}(0, \mathbf{X}) \prod_{i \in [0, \ell_w)} (\mathbf{W}_i(\mathbf{X}) + \beta \mathbf{S}_{\sigma,i}(\mathbf{X}) + \gamma)\,.$$

   Note that $\tilde{v}(\mathbf{X}, 0), \tilde{v}(\mathbf{X}, 1)$ can be simulated given the oracle accesses to $[[\tilde{v}(0, \mathbf{X})]], [[\tilde{v}(1, \mathbf{X})]]$, because for all $b \in \{0, 1\}$, $\tilde{v}(\mathbf{X}, b) = (1 - \mathbf{X}_1) \cdot \tilde{v}(0, \mathbf{X}_2, \ldots, \mathbf{X}_\mu, b) + \mathbf{X}_1 \cdot \tilde{v}(1, \mathbf{X}_2, \ldots, \mathbf{X}_\mu, b)$.
5. V send zerocheck challenge $r_Z \leftarrow\!\!\$\ \mathbb{F}^\mu$
6. P and V run sumcheck resulting from batched zerocheck. The sumcheck is of size μ and has degree $\max(d + 1, \ell_w + 2)$. In each round, the prover sends an oracle to the univariate round polynomial as well as the claimed evaluation. The verifier delays querying the oracles. Similarly, in the last round, the verifier receives the claimed evaluations of all the multilinear polynomials. There are $8 + 2 \cdot \ell_w + \ell_q$ total evaluations:
   - $1 + \ell_w$ of $\mathbf{W}$ (ℓ_w from the batched sumcheck, one to check the outputs)
   - 3 of $\tilde{v}(0, \mathbf{X})$ and 4 of $\tilde{v}(1, \mathbf{X})$ from the product check
   - ℓ_q of q (one per selector)
   - ℓ_w of $\mathbf{S}_\sigma$ from the product check (there is no need to query s_{id} as V can efficiently evaluate it).
7. V uses the claimed evaluations to verify all previous protocols.
8. P and V run the univariate batch-opening algorithm from [Bon+21] to reduce all the round polynomial queries to one.
9. P and V run $\mathcal{R}_{\text{BATCH}}$ on all evaluations using a degree 2, $\mu + \lceil \log_2(8 + 2 \cdot \ell_w + \ell_q) \rceil$ round sum-check. In the protocol, the prover directly transmits the round polynomial using 2 field elements. The verifier can compute the third from the claimed sum.

Figure 3.10: Optimized PIOP for $\mathcal{R}_{\text{PLONK}}$.

Chapter 4

Verifiable Delay Functions for Ecological Consensus

4.1 Introduction

Consider the problem of running a verifiable lottery using a *randomness beacon*, a concept first described by Rabin [Rab83] as an ideal service that regularly publishes random values which no party can predict or manipulate. A classic approach is to apply an extractor function to a public entropy source, such as stock prices [CH10]. Stock prices are believed to be difficult to predict for a passive observer, but an active adversary could manipulate prices to bias the lottery. For example, a high-frequency trader might slightly alter the closing price of a stock by executing (or not executing) a few transactions immediately before the market closes.

Suppose the extractor takes only a single bit per asset (e.g. whether the stock finished up or down for the day) and suppose the adversary is capable of changing this bit for k different assets using last-second trades. The attacker could read the prices of the assets it cannot control, quickly simulate 2^k potential lottery outcomes based on different combinations of the k outcomes it can control, and then manipulate the market to ensure its preferred lottery outcome occurs.

One solution is to add a delay function after extraction, making it slow to compute the beacon outcome from an input of raw stock prices. With a delay function of say, one hour, by the time the adversary simulates the outcome of any potential manipulation strategy, the market will be closed and prices finalized, making it too late to launch an attack. This suggests the key security property for a delay function: it should be infeasible for an adversary to distinguish the function's output from random in less than a specified amount of wall-clock time, even given a potentially large number of parallel processors.

A trivial delay function can be built by iterating a cryptographic hash function. For example, it

is reasonable to assume it is infeasible to compute 2^{40} iterations of SHA-256 in a matter of seconds, even using specialized hardware. However, a lottery participant wishing to verify the output of this delay function must repeat the computation in its entirety (which might take many hours on a personal computer). Ideally, we would like to design a delay function which any observer can quickly verify was computed correctly.

Defining delay functions. In this paper we formalize the requirements for a *verifiable delay function* (VDF) and provide the first constructions which meet these requirements. A VDF consists of a triple of algorithms: Setup, Eval, and Verify. $\mathsf{Setup}(\lambda, t)$ takes a security parameter λ and delay parameter t and outputs public parameters pp (which fix the domain and range of the VDF and may include other information necessary to compute or verify it). $\mathsf{Eval}(\mathsf{pp}, x)$ takes an input x from the domain and outputs a value y in the range and (optionally) a short proof π. Finally, $\mathsf{Verify}(\mathsf{pp}, x, y, \pi)$ efficiently verifies that y is the correct output on x. Crucially, for every input x there should be a *unique* output y that will verify. Informally, a VDF scheme should satisfy the following properties:

- *sequential*: honest parties can compute $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ in t sequential steps, while no parallel-machine adversary with a polynomial number of processors can distinguish the output y from random in significantly fewer steps.
- *efficiently verifiable*: We prefer Verify to be as fast as possible for honest parties to compute; we require it to take total time $O(\text{polylog}(t))$.
- *uniq ue*: for all inputs x, it is difficulty to find a y for which $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = \textit{Yes}$, but $y \neq \mathsf{Eval}(\mathsf{pp}, x)$.

A VDF should remain secure even in the face of an attacker able to perform polynomially bounded pre-computation.

Some VDFs may also offer additional useful properties:

- *decodable*: A VDF is decodable if there exists a decoding algorithm Dec such that $(\mathsf{Eval}, \mathsf{Dec})$ form a lossless encoding scheme. A lossless encoding scheme is a pair of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ where $\mathsf{Enc} : X \to Y$ and $\mathsf{Dec} : Y \to X$ such that $\mathsf{Dec}(\mathsf{Enc}(x)) = x$ for all $x \in X$. We say that a VDF is *efficiently decodable* if it is decodable and Dec is efficient. In this case, Eval need not include a proof as Dec itself can be used to verify the output. There are many different kinds of encoding schemes with different properties, including compression schemes, error correcting codes, ciphers, etc. Of course any VDF can be turned in a trivial encoding scheme by appending the input x to the output, however this would not have any interesting properties as an encoding scheme. In Section 4.2, we will describe one interesting application of an encoding scheme that is both an efficiently invertible ideal cipher and a VDF. The application is to *proofs-of-replication*.

– *incremental*: a single set of public parameters pp supports multiple hardness parameters t. The number of steps used to compute y is specified in the proof, instead of being fixed during Setup. The main benefit of incremental VDFs over a simple chaining of VDFs to increase the delay is a reduced aggregate proof size. This is particularly useful for applications of VDFs to computational time-stamping or blockchain consensus.

Classic slow functions Time-lock puzzles [RSW96] are similar to VDFs in that they involve computing an inherently sequential function. An elegant solution uses repeated squaring in an RivShaAdl78 group as a time-lock puzzle. However, time-lock puzzles are not required to be universally verifiable and in all known constructions the verifier uses its secret state to prepare each puzzle and verify the results. VDFs, by contrast, may require an initial trusted setup but then must be usable on any randomly chosen input.

Another construction for a slow function dating to Dwork and Naor [DN93] is extracting modular square roots. Given a challenge $x \in \mathbb{Z}_p$ (with $p \equiv 3 \pmod 4$), computing $y = \sqrt{x} = x^{\frac{p+1}{4}} \pmod p$ can be efficiently verified by checking that $y^2 = x \pmod p$. There is no known algorithm for computing modular exponentiation which is sublinear in the bit-length of the exponent. However, the difficulty of puzzles is limited to $t = O(\log p)$ as the exponent can be reduced modulo $p - 1$ before computation, requiring the use of a very large prime p to produce a difficult puzzle. While it was not originally proposed for its sequential nature, it has subsequently been considered as such several times [JM11; LW15]. In particular, Lenstra and Wesolowski [LW15] proposed chaining a series of such puzzles together in a construction called Sloth, with lotteries as a specific motivation. Sloth is best characterized as a *time-asymmetric encoding*, offering a trade-off in practice between computation and inversion (verification), and thus can be viewed as a pseudo-VDF. However, it does not meet our asymptotic definition of a VDF because it does not offer asymptotically efficient verification: the t-bit modular exponentiation can be computed in parallel time t, whereas the output (a t-bit number) requires $\Omega(t)$ time simply to read, and therefore verification cannot run in total time polylog$/(t)$. We give a more complete overview of related work in Section 4.8.

Our contributions: In addition to providing the first formal definitions of VDFs, we contribute the following candidate constructions and techniques:

1. A theoretical VDF can be constructed using *incrementally verifiable computation* [Val08] (IVC), in which a proof of correctness for a computation of length t can be computed in parallel to the computation with only polylog(t) processors. We prove security of this theoretical VDF using IVC as a black box. IVC can be constructed from succinct non-interactive arguments of knowledge (SNARKs) under a suitable extractor complexity assumption [Bit+13a]. We also present a simpler construction based only on verifiable computation (Section 4.5), which works if the delay parameter T is a-priori known.

2. We propose a construction based on injective polynomials over algebraic sets that cannot be inverted faster than computing polynomial GCDs. Computing polynomial GCD is sequential in the degree d of the polynomials on machines with fewer than $O(d^2)$ processors. We propose a candidate construction of time-asymmetric encodings from a particular family of *permutation polynomials* over finite fields [GM97]. This construction is asymptotically a strict improvement on Sloth, and to the best of our knowledge is the first encoding offering an exponential time gap between evaluation and inversion. We call this a *weak VDF* because it requires the honest Eval to use greater than $polylog(t)$ parallelism to run in parallel time t (the delay parameter).

3. In Section 4.7 we describe a practical efficiency boost to construSNARKss from IVC using time-asymmetric encodings as the underlying sequential computation, offering up to a 7,000 fold improvement (in the SNARK efficiency) over naive hash chains. In this construction the SNARK proof is only used to boost the efficiency of verification as the output (y, π) of Eval on an input x can also be verified directly without π by computing the inverse of y, which is still faster than computing y from x.

4.2 Applications

Before giving precise definitions and describing our constructions, we first informally sketch several important applications of VDFs.

Randomness beacons. VDFs are useful for constructing randomness beacons from sources such as stock prices [CH10] or proof-of-work blockchains (e.g. Bitcoin, Ethereum) [BCG15; PW16; BGZ16]. Proof-of-work blockchains include randomly sampled solutions to computational puzzles that network participants (called *miners*) continually find and publish for monetary rewards. Underpinning the security of proof-of-work blockchains is the strong belief that these solutions have high computational min-entropy. However, similar to potential manipulation of asset prices by high-frequency traders, powerful miners could potentially manipulate the beacon result by refusing to post blocks which produce an unfavorable beacon output.

Again, this attack is only feasible if the beacon can be computed quickly, as each block is fixed to a specific predecessor and will become "stale" if not published. If a VDF with a suitably long delay is used to compute the beacon, miners will not be able to determine the beacon output from a given block before it becomes stale. More specifically, given the desired delay parameter t, the public parameters $\mathsf{pp} = (\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, t)$ are posted on the blockchain, then given a block b the beacon value is determined to be r where $(r, \pi) = \mathsf{Eval}(\mathsf{ek}, b)$, and anyone can verify correctness by running $\mathsf{Verify}(\mathsf{vk}, b, r, \pi)$. The security of this construction, and in particular the length of delay parameter which would be sufficient to prevent attacks, remains an informal conjecture due to the lack of a complete game-theoretic model capturing miner incentives in Nakamoto-style consensus protocols.

We refer the reader to [BCG15; PW16; BGZ16] for proposed models for blockchain manipulation. Note that most formal models for Nakamoto-style consensus such as that of Garay et al. [GKL15] do not capture miners with external incentives such as profiting from lottery manipulation.

Another approach for constructing beacons derives randomness from a collection of participants, such as all participants in a lottery [GS98; LW15]. The simplest paradigm is "commit-and-reveal" paradigm where n parties submit commitments to random values $r_1, ..., r_n$ in an initial phase and subsequently reveal their commitments, at which point the beacon output is computed as $r = \bigoplus_i r_i$. The problem with this approach is that a malicious adversary (possibly controlling a number of parties) might manipulate the outcome by refusing to open its commitment after seeing the other revealed values, forcing a protocol restart. Lenstra and Wesolowski proposed a solution to this problem (called "Unicorn"[LW15]) using a delay function: instead of using commitments, each participant posts their r_i directly and $seed = H(r_1, \ldots, r_n)$ is passed through a VDF. The output of Eval is then posted and can be efficiently verified. The final beacon outcome is the hash of the output of Eval. With a sufficiently long delay parameter (longer than the time period during which values may be submitted), even the last party to publish their r_i cannot predict what its impact will be on the final beacon outcome. The beacon is unpredictable even to an adversary who controls $n-1$ of the participating parties. It has linear communication complexity and uses only two rounds. This stands in contrast to coin-tossing beacons which use verifiable secret sharing and are at best resistant to an adversary who controls a minority of the nodes [Ran; Syt+17; CD17]. These beacons also use super-linear communication and require multiple rounds of interaction. In the two party setting there are tight bounds that an r-round coin-flipping protocol can be biased with $O(1/r)$ bias [MNS16]. The "Unicorn" construction circumvents these bounds by assuming semi-synchronous communication, i.e. there exists a bound to how long an adversary can delay messages.

Resource-efficient blockchains. Amid growing concerns over the long-term sustainability of proof-of-work blockchains like Bitcoin, which consume a large (and growing) amount of energy, there has been concerted effort to develop resource-efficient blockchains in which miners invest an upfront capital expenditure which can then be re-used for mining. Examples include proof-of-stake [KN; Mic16; Kia+17; Dav+18; BPS16], proof-of-space [Par+18], and proof-of-storage [Mil+14; Fila]. However, resource-efficient mining suffers from *costless simulation* attacks. Intuitively, since mining is not computationally expensive, miners can attempt to produce many separate forks easily.

One method to counter simulation attacks is to use a randomness beacon to select new leaders at regular intervals, with the probability of becoming a leader biased by the quality of proofs (i.e. amount of stake, space, etc) submitted by miners. A number of existing blockchains already construct beacons from tools such as *verifiable random functions*, *verifiable secret sharing*, or *deterministic threshold signatures* [Kia+17; Dav+18; CD17; Dfi]. However, the security of these beacons requires a non-colluding honest majority; with a VDF-based lottery as described above this can potentially be improved to participation of any honest party.

A second approach, proposed by Cohen [Coh17], is to combine proofs-of-resources with incremental VDFs and use the product of resources proved and delay induced as a measure of blockchain quality. This requires a proof-of-resource which is costly to initialize (such as certain types of proof-of-space). This is important such that the resources are committed to the blockchain and cannot be used for other purposes. A miner controlling N units of total resources can initialize a proof π demonstrating control over these N units. Further assume that the proof is non-malleable and that in each epoch there is a common random challenge c, e.g. a block found in the previous epoch, and let H be a random oracle available to everyone. In each epoch, the miner finds $\tau = \min_{1 \leq i \leq N}\{H(c, \pi, i)\}$ and computes a VDF on input c with a delay proportional to τ. The first miner to successfully compute the VDF can broadcast their block successfully. Note that this process mimics the random delay to find a Bitcoin block (weighted by the amount of resources controlled by each miner), but without each miner running a large parallel computation.

Proof of replication. Another promising application of VDFs is *proofs of replication*, a special type of proof of data storage which requires the prover to dedicate unique storage even if the data is available from another source. For instance, this could be used to prove that a number of replicas of the same file are being stored. Classic *proofs of retrievability* [JK07] are typically defined in a private-key client/server setting, where the server proves to the client that it can retrieve the client's (private) data, which the client verifies using a private key.

Instead, the goal of a *proof of replication* [Arm+16; Fila; Filb] is to verify that a given server is storing a *unique replica* of some data which may be publicly available. An equivalent concept to proof-of-replication was first introduced by Sergio Demian Lerner in 2014 under the name "proof of unique blockchain storage" [Ler14]. Lerner proposed using time-asymmetric encodings to apply a slow transformation to a file using a unique identifier as a key. During a challenge-response protocol, a verifier periodically challenges the server for randomly sampled blocks of the uniquely encoded file. The basic idea is that the server should fail to respond quickly enough if it has deleted the encoding. Verifying that the received blocks of the encoding are correct is fast in comparison due to the time-asymmetry of the encoding. Lerner proposed using a Pohlig-Hellman cipher, using the permutation x^3 on $\mathbb{Z}_p^*$, which has asymmetry roughly equivalent to modular square roots. Armknecht et al. [Arm+16] proposed a similar protocol in the private verifier model using RivShaAdl78 time-lock puzzles. The verification of this protocol may be outsourced, but is still less transparent as it fundamentally requires a designated private-key verifier per file.

Efficiently decodable VDFs can be used to improve Lerner's publicly verifiable and transparent construction by using the VDF as the time-asymmetric encoding. As VDFs achieve an exponential gap between parallel-time computation and verification they would improve the challenge-response protocol by reducing the frequency of polling. The frequency needs to be set such that the server cannot delete and recompute the encoding between challenges. Technically, the security property we

need the VDF to satisfy is that a stateless adversary running in parallel time less than T cannot predict any component of the output on any given input, which is stronger than the plain sequentiality requirement of a VDF. A formal way to capture this requirement is to model the VDF as an *ideal delay cipher* [Fis18], namely as an oracle that implements an ideal cipher and takes T sequential steps to respond to queries on any point.

We review briefly the construction, now based on VDFs. The replicator is given an input file, a unique replicator identifier id, and public parameters $\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, t)$, and computes a *slow* encoding of the file using the VDF cipher. This takes sequential time T to derive. In more detail, the encoding is computed by breaking the file into b-bit blocks $B_1, \ldots, B_n$ and storing $y_1, ..., y_n$ where $(y_i, \perp) = \mathsf{Eval}(\mathsf{pp}, B_i \oplus H(id||i))$ where H is a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^b$. To verify that the replicator has stored this unique copy, a verifier can query an encoded block y_i (which must be returned in significantly less time than it is feasible to compute Eval). The verifier can quickly decode this response and check it for correctness, proving that the replicator has stored (or can quickly retrieve from somewhere) an encoding of this block which is unique to the identifier id. If the unique block encoding y_i has not been stored, the VDF ensures that it cannot be recomputed quickly enough to fool the verifier, even given access to B_i . The verifier can query for as many blocks as desired; each query has a $1 - \rho$ chance of exposing a cheating prover that is only storing a fraction ρ of the encoded blocks.

More generally, if the inputs $B_1, .., B_n$ are fixed and known to the verifier then this construction is also a *proof of space* [Dzi+15]. A proof of space is an interactive protocol in which the prover can provide a compact proof that it is persistently using $\Omega(N)$ space. This construction is in fact a *tight* PoS, meaning that it requires an honest prover to use N space and for any $\epsilon > 0$ it can be tuned so that an adversary who uses $(1 - \epsilon)N$ space will be caught with overwhelming probability. A proof-of-replication is a special kind of proof of space that is quite delicate to formally define and has been developed further in followup work [Fis18].

Computational timestamping. All known proof-of-stake systems are vulnerable to long-range forks due to post-hoc stakeholder misbehavior [KN; Mic16; Kia+17; BPS16]. In proof-of-stake protocols, at any given time the current stakeholders in the system are given voting power proportionate to their stake in the system. An honest majority (or supermajority) is assumed because the current stakeholders are CCS:KumBen14d to keep the system running correctly. However, after stakeholders have divested they no longer have this incentive. Once the majority (eq. supermajority) of stakeholders from a point in time in the past are divested, they can collude (or sell their key material to an attacker) in order to create a long alternate history of the system up until the present. Current protocols typically assume this is prevented through an external timestamping mechanism which can prove to users that the genuine history of the system is much older.

Incremental VDFs can provide computational evidence that a given version of the state's system is older (and therefore genuine) by proving that a long-running VDF computation has been performed

on the genuine history just after the point of divergence with the fraudulent history. This potentially enables detecting long-range forks without relying on external timestamping mechanisms.

We note however that this application of VDFs is fragile as it requires precise bounds on the attacker's computation speed. For other applications (such as randomness beacons) it may be acceptable if the adversary can speed up VDF evaluation by a factor of 10 using faster hardware; a higher t can be chosen until even the adversary cannot manipulate the beacon even with a hardware speedup. For computational timestamping, a 10-fold speedup would be a serious problem: once the fraudulent history is more than one-tenth as old as the genuine history, an attacker can fool participants into believing the fraudulent history is actually *older* than the genuine one.

4.3 Model and definitions

We now define VDFs more precisely. In what follows we say that an algorithm runs in *parallel time* t with p processors if it can be implemented on a PRAM machine with p parallel processors running in time t. We say *total time* (eq. sequential time) to refer to the time needed for computation on a single processor.

Definition 4.1. *A VDF* $V = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is a triple of algorithms as follows:*

- $\mathsf{Setup}(\lambda, t) \to \mathsf{pp} = (\mathsf{ek}, \mathsf{vk})$ *is a randomized algorithm that takes a security parameter* λ *and a desired puzzle difficulty* t *and produces public parameters* pp *that consists of an evaluation key* ek *and a verification key* vk*. We require* Setup *to be polynomial-time in* λ*. By convention, the public parameters specify an input space* $\mathcal{X}$ *and an output space* $\mathcal{Y}$*. We assume that* $\mathcal{X}$ *is efficiently sampleable.* Setup *might need secret randomness, leading to a scheme requiring a trusted setup. For meaningful security, the puzzle difficulty* t *is restricted to be sub-exponentially sized in* λ*.*

- $\mathsf{Eval}(\mathsf{ek}, x) \to (y, \pi)$ *takes an input* $x \in \mathcal{X}$ *and produces an output* $y \in \mathcal{Y}$ *and a (possibly empty) proof* π*.* Eval *may use random bits to generate the proof* π *but not to compute* y*. For all* pp *generated by* $\mathsf{Setup}(\lambda, t)$ *and all* $x \in \mathcal{X}$*, algorithm* $\mathsf{Eval}(\mathsf{ek}, x)$ *must run in parallel time* t *with* $\mathsf{poly}(\lambda)/(\log(t), \lambda)$ *processors.*

- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi) \to \{Yes, No\}$ *is a deterministic algorithm takes an input, output and proof and outputs Yes or No. Algorithm* Verify *must run in total time polynomial in* $\log t$ *and* λ*. Notice that* Verify *is much faster than* Eval*.*

Additionally V *must satisfy Correctness (Definition 4.2), Soundness (Definition 4.3), and Sequentiality (Definition 4.4).*

Correctness and Soundness Every output of Eval must be accepted by Verify. We guarantee that the output y for an input x is unique because Eval evaluates a deterministic function on $\mathcal{X}$. Note that we do not require the proof π to be unique, but we do require that the proof is sound and that a verifier cannot be convinced that some different output is the correct VDF outcome. More formally,

Definition 4.2 (Correctness)**.** *A VDF V is correct if for all λ, t, parameters $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(\lambda, t)$, and all $x \in \mathcal{X}$, if $(y, \pi) \leftarrow\!\!\$\ \mathsf{Eval}(\mathsf{ek}, x)$ then $\mathsf{Verify}(\mathsf{vk}, x, y, \pi) =$ Yes.*

We also require that for no input x can an adversary get a verifier to accept an incorrect VDF output.

Definition 4.3 (Soundness)**.** *A VDF is sound if for all algorithms $\mathcal{A}$ that run in time $O\left(\mathsf{poly}(\lambda)/(t, \lambda)\right)$*

$$Pr\left[\begin{array}{c|c} \mathsf{Verify}(\mathsf{vk}, x, y, \pi) = \mathit{Yes} & \mathsf{pp} = (\mathsf{ek}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Setup}(\lambda, t) \\ y \neq \mathsf{Eval}(\mathsf{ek}, x) & (x, y, \pi) \leftarrow\!\!\$\ \mathcal{A}(\lambda, \mathsf{pp}, t) \end{array}\right] \leq \mathsf{negl}(\lambda)(\lambda)$$

Size restriction on t Asymptotically t must be subexponential in λ. The reason for this is that the adversary needs to be able to run in time at least t (Eval requires this), and if t is exponential in λ then the adversary might be able to break the underlying computational security assumptions that underpin both the soundness as well as the sequentiality of the VDF, which we will formalize next.

Parallelism in Eval The practical implication of allowing more parallelism in Eval is that "honest" evaluators may be required to have this much parallelism in order to complete the challenge in time t. The sequentiality security argument will compare an adversary's advantage to this optimal implementation of Eval. Constructions of VDFs that do not require any parallelism to evaluate Eval in the optimal number of sequential steps are obviously superior. However, it is unlikely that such constructions exist (without trusted hardware). Even computing an iterated hash function or modular exponentiation (used for time-lock puzzles) could be computed faster by parallelizing the hash function or modular arithmetic. In fact, for an decodable VDF it is necessary that $|\mathcal{Y}| > \mathsf{poly}(\lambda)/(t)$, and thus the challenge inputs to Eval have size $\mathsf{poly}(\lambda)/\log(t)$. Therefore, in our definition we allow algorithm Eval up to $\mathsf{poly}(\lambda)/\log(t)$ parallelism.

4.3.1 VDF Security

We call the security property needed for a VDF scheme *σ-sequentiality*. Essentially, we require that no adversary is able to compute an output for Eval on a random challenge in parallel time $\sigma(t) < t$, even with up to "many" parallel processors and after a potentially large amount of pre-computation. It is critical to bound the adversary's allowed parallelism, and we incorporate this into the definition.

Note that for an efficiently decodable VDF, an adversary with $|\mathcal{Y}|$ processors can always compute outputs in $o(t)$ parallel time by simultaneously trying all possible outputs in $\mathcal{Y}$. This means that for efficiently decodable VDFs it is necessary that $|\mathcal{Y}| > \mathsf{poly}(\lambda)/(t)$, and cannot achieve σ-sequentiality against an adversary with greater than $|\mathcal{Y}|$ processors.

We define the following sequentiality game applied to an adversary $\mathcal{A} := (\mathcal{A}_0, \mathcal{A}_1)$:

$$
\begin{array}{rll}
\mathsf{pp} & \leftarrow\$\ \mathsf{Setup}(\lambda, t) & /\!/ \text{ choose a random } \mathsf{pp} \\
L & \leftarrow\$\ \mathcal{A}_0(\lambda, \mathsf{pp}, t) & /\!/ \text{ adversary preprocesses } \mathsf{pp} \\
x & \leftarrow\$\ \mathcal{X} & /\!/ \text{ choose a random input } x \\
y_A & \leftarrow\$\ \mathcal{A}_1(L, \mathsf{pp}, x) & /\!/ \text{ adversary computes an output } y_A
\end{array}
$$

We say that $(\mathcal{A}_0, \mathcal{A}_1)$ wins the game if $y_A = y$ where $(y, \pi) := \mathsf{Eval}(\mathsf{pp}, x)$.

Definition 4.4 (Sequentiality)**.** *For functions $\sigma(t)$ and $p(t)$, the VDF is (p, σ)-sequential if no pair of randomized algorithms $\mathcal{A}_0$, which runs in total time $O(\mathsf{poly}(\lambda)/(t, \lambda))$, and $\mathcal{A}_1$, which runs in parallel time $\sigma(t)$ on at most $p(t)$ processors, can win the sequentiality game with probability greater than $\mathsf{negl}(\lambda)$.*

The definition captures the fact that even after $\mathcal{A}_0$ computes on the parameters pp for a (polynomially) long time, the adversary $\mathcal{A}_1$ cannot compute an output from the input x in time $\sigma(t)$ on $p(t)$ parallel processors. If a VDF is (p, σ)-sequential for any polynomial p, then we simply say the VDF is σ-sequential. In the sequentiality game, we do not require the online attack algorithm $\mathcal{A}_1$ to output a proof π. The reason is that in many of our applications, for example, in a lottery, the adversary can profit simply by learning the output early, even without being able to prove correctness to a third party.

Values of $\sigma(t)$ Clearly any candidate construction trivially satisfies $\sigma(t)$-sequentiality for *some* σ (e.g. $\sigma(t) = 0$). Thus, security becomes more meaningful as $\sigma(t) \to t$. No construction can obtain $\sigma(t) = t$ because by design Eval runs in parallel time t. Ideal security is achieved when $\sigma(t) = t - 1$. This ideal security is in general unrealistic unless, for example, time steps are measured in rounds of queries to an ideal oracle (e.g. random oracle). In practice, if the oracle is instantiated with a concrete program (e.g. a hash function), then differences in hardware/implementation would in general yield small differences in the response time for each query. An almost-perfect VDF would achieve $\sigma(t) = t - o(t)$ sequentiality. Even $\sigma(t) = t - \epsilon t$ sequentiality for small ϵ is sufficient for most applications. Security degrades as $\epsilon \to 1$. The naive VDF construction combining a hash chain with succinct verifiable computation (i.e. producing a SNARG proof of correctness following the hash chain computation) cannot beat $\epsilon = 1/2$, unless it uses at least $\omega(t)$ parallelism to generate the proof in sublinear time (exceeding the allowable parallelism for VDFs, though see a relaxation to "weak" VDFs below).

Unpredictability and min-entropy Definition 4.4 captures an unpredictability property for the output of the VDF, similar to a one-way function. However, similar to random oracles, the output of the VDF on a given input is never indistinguishable from random. It is possible that no depth $\sigma(t)$ circuit can distinguish the output on a randomly sampled challenge from random, but only if the VDF proof is not given to the distinguisher. Efficiently decodable VDFs cannot achieve this stronger property.

For the application to random beacons (e.g. for lotteries), it is only necessary that on a random challenge the output is unpredictable and also has sufficient min-entropy[1] conditioned on previous outputs for different challenges. In fact, σ-sequentiality already implies that min-entropy is $\Omega(\log \lambda)$. Otherwise some fixed output y occurs with probability $1/\mathsf{poly}(\lambda)/(\lambda)$ for randomly sampled input x; the adversary $\mathcal{A}_0$ can computes $O(\mathsf{poly}(\lambda)/(\lambda))$ samples of this distribution in the preprocessing to find such a y' with high probability, and then $\mathcal{A}_1$ could output y' as its guess. Moreover, if σ-sequentiality is achieved for t superpolynomial (sub-exponential) in λ, then the preprocessing adversary is allowed $2^{o(\lambda)}$ samples, implying some $o(\lambda)$ min-entropy of the output must be preserved. By itself, σ-sequentiality does not imply $\Omega(\lambda)$ min-entropy. Stronger min-entropy preservation can be demonstrated in other ways given additional properties of the VDF, e.g. if it is a permutation or collision-resistant. Under suitable complexity theoretic assumptions (namely the existence of subexponential $2^{o(n)}$ circuit lower bounds) a combination of Nisan-Wigderson type PRGs and extractors can also be used to generate $\mathsf{poly}(\lambda)/(\lambda)$ pseudorandom bits from a string with min-entropy $\log \lambda$.

Random "Delay" Oracle In the random oracle model, *any* unpredictable string (regardless of its min-entropy) can be used to extract an unpredictable λ-bit uniform random string. For the beacon application, a random oracle H would simply be applied to the output of the VDF to generate the beacon value. We can even model this construction as an ideal object itself, a *Random Delay Oracle*, which implements a random function H' and on any given input x it waits for $\sigma(t)$ steps before returning the output $H'(x)$. Demonstrating a construction from a σ-sequential VDF and random oracle H that is provably *indifferentiable* [MRH04] from a Random Delay Oracle is an interesting research question.[2]

Remark: Removing any single property makes VDF construction easy. We note the existence of well-known outputs if any property is removed:

- If Verify is not required to be fast, then simply iterating a one-way function t times yields a trivial solution. Verification is done by re-computing the output, or a set of ℓ intermediate

[1] A randomness extractor can then be applied to the output to map it to a uniform distribution.

[2] The difficulty in proving indifferentiability arises because the distinguisher can query the VDF/RO construction and the RO itself separately, therefore the simulator must be able to simulate queries to the random oracle H given only access to the Random Delay Oracle. Indifferentiability doesn't require the simulator to respond in exactly the same time, but it is still required to be *efficient*. This becomes an issue if the delay t is superpolynomial.

points can be supplied as a proof which can be verified in parallel time $\Theta(t/\ell)$ using ℓ processors, with total verification time remaining $\Theta(t)$.

- If we do not require *uniqueness*, then the construction of Mahmoody et al. [MMV13] using hash functions and depth-robust graphs suffices. This construction was later improved by Cohen and Pietrzak [CP18]. This construction fails to ensure uniqueness because once an output y is computed it can be easily mangled into many other valid outputs $y' \neq y$, as discussed in Section 4.8.1.
- If we do not require *σ-sequentiality*, many solutions are possible, such as finding the discrete log of a challenge group element with respect to a fixed generator. Note that computing an elliptic curve discrete log can be done in parallel time $o(t)$ using a parallel version of the Pollard rho algorithm [vW94].

Weaker VDFs For certain applications it is still interesting to consider a VDF that requires even more than polylog/(t) parallelism in Eval to compute the output in parallel time t. For example, in the randomness beacon application only one party is required to compute the VDF and all other parties can simply verify the output. It would not be unreasonable to give this one party a significant amount of parallel computing power and optimized hardware. This would yield a secure beacon as long as no adversary could compute the outputs of Eval in faster that t steps given even more parallelism than this party. Moreover, for small values of t it may be practical for anyone to use up to $O(t)$ parallelism (or more). With this in mind, we define a weaker variant of a VDF that allows additional parallelism in Eval.

Definition 4.5. *We call a system $V = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ a weak-VDF if it satisfies Definition 4.1 with the exception that Eval is allowed up to poly(t, λ) parallelism.*

Note that (p, σ)-sequentiality can only be meaningful for a weak-VDF if Eval is allowed strictly less that $p(t)$ parallelism, otherwise the honest computation of Eval would require more parallelism than even the adversary is allowed.

4.4 VDFs from Incrementally Verifiable Computation

VDFs are by definition sequential functions. We, therefore, require the existence of sequential functions in order to construct any VDF. We begin by defining a sequential function.

Definition 4.6 ((t, ϵ)-Sequential function)**.** *$f : X \to Y$ is a (t, ϵ)-sequential function if for $\lambda = O(\log(|X|))$, if the following conditions hold.*

1. *There exists an algorithm that for all $x \in X$ evaluates f in parallel time t using $\mathsf{poly}(\lambda)/(\log(t), \lambda)$ processors.*

2. For all $\mathcal{A}$ that run in parallel time strictly less than $(1-\epsilon)\cdot t$ with $\mathsf{poly}(\lambda)/(t,\lambda)$ processors:

$$Pr\Big[y_A = f(x) \;\Big|\; y_A \leftarrow\!\$\; \mathcal{A}(\lambda, x),\; x \leftarrow\!\$\; X\Big] \leq \mathsf{negl}(\lambda)(\lambda)$$

.

In addition we consider *iterated sequential functions* that are defined as the iteration of some round function. The key property of an iterated sequential function is that iteration of the round function is the fastest way to evaluate the function.

Definition 4.7 (Iterated Sequential Function)**.** *Let $g : X \to X$ be a (t,ϵ)-sequential function. A function $f : \mathbb{N} \times X \to X$ defined as $f(k,x) = g^{(k)}(x) = \underbrace{g \circ g \circ \cdots \circ g}_{k \text{ times}}$ is said to be an* iterated sequential function *(with round function g) if for all $k = 2^{o(\lambda)}$ the function $h : X \to X$ defined by $h(x) = f(k,x)$ is $(k\cdot t, \epsilon)$-sequential as in Definition 4.6.*

It is widely believed that the function obtained from iterating a hash function like SHA-256 is an iterated sequential function with $t = O(\lambda)$ and ϵ negligible in λ. The sequentiality of such functions can be proved in the random oracle model [KMB17; MMV13]. We will use the functions g explicitly and require it to have an explicit arithmetic circuit representation. Modeling g as an oracle, therefore, does not suffice for our construction.

Another candidate for an iterated sequential function is exponentiation in a finite group of unknown order, where the round function is squaring in the group. The fastest known way to compute this is by repeated squaring which is an iterative sequential computation.

Based on these candidates, we can make the following assumption about the existence of iterated sequential functions:

Assumption 4.1. *For all $\lambda \in \mathbb{N}$ there exist (1) an ϵ, t with $t = \mathsf{poly}(\lambda)/(\lambda)$, and (2) a function $g_\lambda : X \to X$, where $\log_2 |X| = \lambda$ and X can be sampled in time $\mathsf{poly}(\lambda)/(\lambda)$. This g_λ satisfies: (i) g_λ is a (t,ϵ)-sequential function, and (ii) the function $f : \mathbb{N} \times X \to X$ with round function g_λ is an iterated sequential function.*

An iterated sequential function by itself gives us many of the properties needed for a secure VDF. It is sequential by definition and the trivial algorithm (iteratively computing g) uses only $\mathsf{poly}(\lambda)/(\lambda)$ parallelism. Such a function by itself, however, does not suffice to construct a VDF. The fastest generic verification algorithm simply recomputes the function. While this ensures soundness it does not satisfy the efficient verification requirement of a VDF. The verifier of a VDF needs to be exponentially faster than the prover.

SNARGs and SNARKs A natural idea to improve the verification time is to use verifiable computation. In verifiable computation the prover computes a succinct argument (SNARG) that a

computation was done correctly. The argument can be efficiently verified using resources that are independent of the size of the computation. A SNARG is a weaker form of a succinct non-interactive argument of knowledge (SNARK) [Gen+13] for membership in an NP language $\mathcal{L}$ with relation R (Definition 4.8). The additional requirement of a SNARK is that for any algorithm that outputs a valid proof of membership of an instance $x \in \mathcal{L}$ there is also an extractor that "watches" the algorithm and outputs a witness w such that $(x, w) \in R$. In the special case of providing a succinct proof that a (polynomial size) computation F was done correctly, i.e. y is the output of F on x, the NP witness is empty and the NP relation simply consists of pairs $((x, y), \perp)$ such that $F(x) = y$.

Definition 4.8 (Verifiable Computation / SNARK)**.** *Let $\mathcal{L}$ denote an NP language with relation $R_{\mathcal{L}}$, where $x \in \mathcal{L}$ iff $\exists w \; R_{\mathcal{L}}(x, w) = 1$. A SNARK system for $R_{\mathcal{L}}$ is a triple of polynomial time algorithms* $(\mathsf{Setup}, \mathit{SNKProve}, \mathit{SNKVerify})$ *that satisfy the following properties:*

Completeness:

$$\forall (x, w) \in R_{\mathcal{L}} : \; Pr\left[\mathit{SNKVerify}(\mathsf{vk}, x, \pi) = 1 \;\middle|\; \begin{array}{c} (\mathsf{vk}, \mathsf{ek}) \leftarrow\!\$\; \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow\!\$\; \mathit{SNKProve}(\mathsf{ek}, x, w) \end{array} \right] = 1$$

Succinctness: *The length of a proof and complexity of $\mathit{SNKVerify}$ is bounded by $poly\big(\lambda, log(|y| + |w|)\big)$.*

Knowledge extraction:*[sub-exponential adversary knowledge extractor] For all adversaries $\mathcal{A}$ running in time $2^{o(\lambda)}$ there exists an extractor $\mathsf{Ext}/_{\mathcal{A}}$ running in time $2^{o(\lambda)}$ such that for all auxiliary inputs z of size $poly(\lambda)$:*

$$Pr\left[\begin{array}{c} \mathit{SNKVerify}(\mathsf{vk}, x, \pi) = 1 \\ R_{\mathcal{L}}(x, w) \neq 1 \end{array} \;\middle|\; \begin{array}{l} (\mathsf{vk}, \mathsf{ek}) \leftarrow\!\$\; \mathsf{Setup}(1^\lambda) \\ (x, \pi) \leftarrow\!\$\; \mathcal{A}(z, \mathsf{ek}) \\ w \leftarrow\!\$\; \mathsf{Ext}/_{\mathcal{A}}(z, \mathsf{ek}) \end{array} \right] \leq \mathsf{negl}(\lambda)(\lambda)$$

Impractical VDF from SNARGs. Consider the following construction for a VDF from a (t, ϵ)-sequential function f. Let $\mathsf{pp} = (\mathsf{ek}, \mathsf{vk}) \leftarrow\!\$\; \mathsf{Setup}(\lambda)$ be the public parameter of a SNARG scheme for proving membership in the language of pairs (x, y) such that $f(x) = y$. On input $x \in X$ the Eval computes $y = f(x)$ and a succinct argument $\pi \leftarrow\!\$\; \mathsf{SNKProve}(\mathsf{ek}, (x, y), \perp)$. The prover outputs $((x, y), \pi)$. On input $((x, y), \pi)$ the verifier checks $y = f(x)$ by checking $\mathsf{SNKVerify}(\mathsf{vk}, (x, y), \pi) = 1$.

This construction clearly satisfies fast verification. All known SNARK constructions are quasi-linear in the length of the underlying computation f [Ben+14b]. Assuming the cost for computing a SNARG for a computation of length t is $k \cdot t \log(t)$ then the SNARG VDF construction achieves $\sigma(t) = \frac{(1-\epsilon) \cdot t}{(k+1) \cdot \log(t)}$ sequentiality. This does not even achieve the notion of $(1 - \epsilon')t$ sequentiality for any adversary. This means that the adversary can compute the output of the VDF in a small fraction of the time that it takes the honest prover to convince an honest verifier. If, however, $\mathsf{SNKProve}$ is sufficiently parallelizable then it is possible to partially close the gap between the sequentiality of

f and the sequentiality of the VDF. The Eval simply executes SNKProve on a parallel machine to reduce the relative total running time compared to the computation of f. SNARK constructions can run in parallel time polylog/(t) on $O(t \cdot \text{polylog}/(t))$ processors. This shows that a VDF can theoretically be built from verifiable computation.

The construction has, however, two significant downsides: First, in practice computing a SNARG is more than 100,000 times more expensive than evaluating the underlying computation [Wah+15]. This means that to achieve meaningful sequentiality, the SNARG computation would require massive parallelism using hundreds thousands of cores. The required parallelism additionally depends on the time t. Second, the construction does not achieve $(1-\epsilon)t$ sequentiality, which is the optimal sequentiality that can be achieved by a construction which involves the evaluation of f.

We therefore, now give a VDF construction[3] with required parallelism independent of t and σ-sequentiality asymptotically close to $(1-\epsilon)t$ where ϵ will be defined by the underlying sequential computation.

Incremental Verifiable Computation (IVC). IVC provides a direction for circumventing the problem mentioned above. IVC was first studied by Valiant [Val08] in the context of computationally sound proofs [Mic94]. Bitansky et al. [Bit+13a] generalized IVC to distributed computations and to other proof systems such as SNARKs. IVC requires that the underlying computation can be expressed as an iterative sequence of evaluations of the same Turing machine. An iterated sequential function satisfies this requirement.

The basic idea of IVC is that at every incremental step of the computation, a prover can produce a proof that a certain state is indeed the current state of the computation. This proof is updated after every step of the computation to produce a new proof. Importantly, the complexity of each proof in proof size and verification cost is bounded by $\mathsf{poly}(\lambda)/(\lambda)$ for any sub-exponential length computation. Additionally the complexity of updating the proof is independent of the total length of the computation.

Towards VDFs from IVC. Consider a VDF construction that runs a sequential computation and after each step uses IVC to update a proof that both this step and the previous proof were correct. Unfortunately, for IVC that requires knowledge extraction we cannot prove soundness of this construction for $t > O(\lambda)$. The problem is that a recursive extraction yields an extractor that is exponential in the recursion depth [Bit+13a].

The trick around this is to construct a binary tree of proofs of limited depth [Val08; Bit+13a]. The leaf proofs verify computation steps whereas the internal node proofs prove that their children are valid proofs. The verifier only needs to check the root proof against the statement that all computation steps and internal proofs are correct.

[3]The construction is largely subsumed by the subsequently added simpler construction in Section 4.5. This simpler construction is built directly from verifiable computation.

We focus on the special case that the function f is an iterated sequential function. The regularity of the iterated function ensures that the statement that the verifier checks is succinct. We impose a strict requirement on our IVC scheme to output both the output of f and a final proof with only an additive constant number of additional steps over evaluating f alone.

We define *tight* IVC for an iterated sequential functions, which captures the required primitive needed for our theoretical VDF. We require that incremental proving is almost overhead free in that the prover can output the proof almost immediately after the computation has finished. The definition is a special case of Valiant's definition [Val08].

Definition 4.9 (Tight IVC for iterated sequential functions)**.** *Let $f_\lambda : \mathbb{N} \times \mathcal{X} \to \mathcal{X}$ be an iterated sequential function with (t, ϵ)-sequential round function g_λ iterated $k = 2^{o(\lambda)}$ times. An IVC system for f_λ is a triple of polynomial time algorithms ($\mathsf{IVCGen}, \mathsf{IVCProve}, \mathsf{IVCVerify}$) that satisfy the following properties:*

Completeness:

$$\forall x \in \mathcal{X} : \; Pr\left[\mathsf{IVCVerify}(\mathsf{vk}, x, y, k, \pi) = \mathit{Yes} \;\middle|\; \begin{array}{c}(\mathsf{vk}, \mathsf{ek}) \leftarrow\!\$\; \mathsf{IVCGen}(\lambda, f) \\ (y, \pi) \leftarrow\!\$\; \mathsf{IVCProve}(\mathsf{ek}, k, x)\end{array}\right] = 1$$

Succinctness: *The length of a proof and the complexity of $\mathsf{SNKVerify}$ is bounded by $\mathsf{poly}(\lambda)/\big(\lambda, \log(k \cdot t)\big)$.*

Soundness:*[sub-exponential soundness] For all algorithms $\mathcal{A}$ running in time $2^{o(\lambda)}$:*

$$Pr\left[\begin{array}{c}\mathsf{IVCVerify}(\mathsf{vk}, x, y, k, \pi) = \mathit{Yes} \\ f(k, x) \neq y\end{array}\;\middle|\; \begin{array}{l}(\mathsf{vk}, \mathsf{ek}) \leftarrow\!\$\; \mathsf{IVCGen}(\lambda, f) \\ (x, y, k, \pi) \leftarrow\!\$\; \mathcal{A}(\lambda, \mathsf{vk}, \mathsf{ek})\end{array}\right] \leq \mathsf{negl}(\lambda)(\lambda)$$

Tight Incremental Proving*: There exists a k' such that for all $k \geq k'$ and $k = 2^{o(\lambda)}$, $\mathsf{IVCProve}(\mathsf{ek}, k, x)$ runs in parallel time $k \cdot t + O(1)$ using $\mathsf{poly}(\lambda)/(\lambda, t)$-processors.*

Existence of tight IVC. Bitansky et al. [Bit+13a] showed that any SNARK system such as [Par+13] can be used to construct IVC. Under strong knowledge of exponent assumptions there exists an IVC scheme using a SNARK tree of depth less than λ (Theorem 1 of [Bit+13a]). In every computation step the prover updates the proof by computing λ new SNARKs each of complexity $\mathsf{poly}(\lambda)/(\lambda)$, each verifying another SNARK and one of complexity t which verifies one evaluation of g_λ, the round function of f_λ. Ben Sasson et al. [Ben+13] discuss the parallel complexity of the Pinocchio SNARK [Par+13] and show that for a circuit of size m there exists a parallel prover using $O(m \cdot \log(m))$ processors that computes a SNARK in time $O(\log(m))$. Therefore, using these SNARKs we can construct an IVC proof system ($\mathsf{IVCGen}, \mathsf{IVCProve}, \mathsf{IVCVerify}$) where, for sufficiently large t, $\mathsf{IVCProve}$ uses $\tilde{O}(\lambda + t)$ parallelism to produce each incremental IVC output in

time $\lambda \cdot \log(t+\lambda) \leq t$. If t is not sufficiently large, i.e. $t > \lambda \cdot \log(t+\lambda)$ then we can construct an IVC proof system that creates proofs for k' evaluations of g_λ. The IVC proof system chooses k' such that $t \leq \lambda \cdot \log(k' \cdot t + \lambda)$. Given this the total parallel runtime of $\mathsf{IVCProve}$ on k iterations of an (t, ϵ)-sequential function would thus be $k \cdot t + \lambda \cdot \log(k' \cdot t + \lambda) = k \cdot t + O(1)$. This shows that we can construct tight IVC from existing SNARK constructions.

$VDF_{\mathbf{IVC}}$ construction. We now construct a VDF from a tight IVC. By Assumption 4.1 we are given a family $\{f_\lambda\}$, where each $f_\lambda : \mathbb{N} \times X_\lambda \to X_\lambda$ is defined by $f_\lambda(k, x) = g_\lambda^{(k)}(x)$. Here g_λ is a (s, ϵ)-sequential function on an efficiently sampleable domain of size $O(2^\lambda)$.
Given a tight IVC proof system $(\mathsf{IVCGen}, \mathsf{IVCProve}, \mathsf{IVCVerify})$ for f we can construct a VDF that satisfies $\sigma(t)$-sequentiality for $\sigma(t) = (1-\epsilon) \cdot t - O(1)$:

- $\mathsf{Setup}(\lambda, t)$: Let g_λ be a (t, ϵ)-sequential function and f_λ the corresponding iterated sequential function as described in Assumption 4.1. Run $(\mathsf{ek}, \mathsf{vk}) \leftarrow\!\$\ \mathsf{IVCGen}(\lambda, f_\lambda)$. Set k to be the largest integer such that $\mathsf{IVCProve}(\mathsf{ek}, k, x)$ takes time less than t. Output $\mathsf{pp} = \big((\mathsf{ek}, k),\ (\mathsf{vk})\big)$.
- $\mathsf{Eval}((\mathsf{ek}, k), x)$: Run $(y, \pi) \leftarrow\!\$\ \mathsf{IVCProve}(\mathsf{ek}, k, x)$, output (y, π).
- $\mathsf{Verify}(\mathsf{vk}, x, (y, \pi))$: Run and output $\mathsf{IVCVerify}(\mathsf{vk}, x, y, k, \pi)$.

Note that t is fixed in the public parameters. It is, however, also possible to give t directly to Eval. VDF_{IVC} is, therefore, *incremental*.

Lemma 4.1. *VDF_{IVC} satisfies soundness (Definition 4.3)*

Proof. Assume that an $\mathsf{poly}(\lambda)/(t, \lambda)$ algorithm $\mathcal{A}$ outputs (with non-negligible probability in λ) a tuple (x, y, π) on input λ, t, and $\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(\lambda, t)$ such that $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = \text{Yes}$ but $f_\lambda(k, x) \neq y$. We can then construct an adversary $\mathcal{A}'$ that violates IVC soundness. Given $(\mathsf{vk}, \mathsf{ek}) \leftarrow\!\$\ \mathsf{IVCGen}(\lambda, f_\lambda)$ the adversary $\mathcal{A}'$ runs $\mathcal{A}$ on λ, t, and $(\mathsf{vk}, \mathsf{ek})$. Since $(\mathsf{vk}, \mathsf{ek})$ is sampled from the same distribution as $\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(\lambda, t)$ it follows that, with non-negligible probability in λ, $\mathcal{A}'$ outputs (x, y, π) such that $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = \mathsf{IVCVerify}(\mathsf{vk}, x, y, k, \pi) = \text{Yes}$ and $f_\lambda(k, x) \neq y$, which directly violates the soundness of IVC. □

Theorem 4.1 (VDF_{IVC})**.** *VDF_{IVC} is a VDF scheme with $\sigma(t) = (1-\epsilon)t - O(1)$ sequentiality.*

Proof. First note that the VDF_{IVC} algorithms satisfy the definition of the VDF algorithms. $\mathsf{IVCProve}$ runs in time $(\frac{t}{s} - 1) \cdot s + s = t$ using $\mathsf{poly}(\lambda)/(\lambda, s) = \mathsf{poly}(\lambda)/(\lambda)$ processors. $\mathsf{IVCVerify}$ runs in total time $\mathsf{poly}(\lambda)/(\lambda, \log(t))$. Correctness follows from the correctness of the IVC scheme. Soundness was proved in Lemma 4.1. The scheme is $\sigma(t)$-sequential because $\mathsf{IVCProve}$ runs in time $k \cdot s + O(1) < t$. If any algorithm that uses $\mathsf{poly}(\lambda)/(t, \lambda)$ processors can produce the VDF output in time less than $(1-\epsilon)t - O(1)$ he can directly break the t, ϵ-sequentiality of f_λ. Since s is independent of t we can conclude that VDF_{IVC} has $\sigma(t) = (1-\epsilon)t - O(1)$ sequentiality.

□

4.5 VDFs from Verifiable Computation

We next present a VDF construction from verifiable computation. This construction is simpler than the one in the previous section in that it does not require tight incremental verifiable computation. Verifiable computation is sufficient.

The core idea is that we can run a sequential computation and, in parallel, compute multiple SNARGs that prove the correctness of different parts of the computation. The protocol we propose computes $\log N$ SNARGs in parallel for segments of geometrically decreasing size.

Let f be an iterated sequential function obtained from iterating a round function k times. As a warmup, assume that constructing a SNARG for an evaluation of f takes exactly the same time as computing the function f. The VDF prover first iterates the round function $k/2$ times, to complete half the computation of f on a given input. It then iterates the round function $k/2$ more times, and *in parallel*, computes a SNARG for the first $k/2$ iterations. This way the SNARG computation and the function evaluation will complete at the same time. The prover then continues by constructing a SNARG for the next $k/4$ iterations of the round function, then the next $k/8$ iterations, and so on. All these SNARGs are constructed in parallel to computing the function. The final iteration requires no SNARG as the verifier can check it directly. All these SNARG computations are done in parallel. Using $\log_2(k)$ processors, the evaluation of f and all the SNARG computations will complete at exactly the same time. Hence, constructing the SNARGs adds no time to the evaluation of f, but requires $\log_2 k$ parallelism at the evaluator.

We will now describe a more general construction that allows for arbitrary gaps between the SNARG prover time and the function evaluation time. The construction uses an iterated sequential function and a SNARG proof system. However, an analogous construction can be built from any underlying VDF scheme. The construction can amplify the VDF scheme to a "tight" VDF scheme in which the prover outputs the proof concurrently with the output. The amplification has a logarithmic overhead in proof size, verifier time, and prover parallelism. This is described in more detail in concurrent work by Döttling, Garg, Malavolta and Vasudevan[Döt+20].

$VDF_{\mathbf{VC}}$ construction As in the VDF_{IVC} we use a family of sequential iterated functions $\{f_\lambda\}$ such that each $f_\lambda : \mathbb{Z} \times X_\lambda \to X_\lambda$ is defined as $f_\lambda(k, x) = g_\lambda^{(k)}(x)$ for an (s, ϵ) sequential function g_λ. We are also given a SNARK systems $(\mathsf{Setup}, \mathsf{SNKProve}, \mathsf{SNKVerify})$ for the family of relations $R_{f_\lambda,k} := \{\big((x, y), \perp\big) : f_\lambda(k, x) = y\}$. The SNARK only needs to satisfy the soundness definition and not the knowledge extraction so a SNARG suffices. We slightly modify the system compared to Definition 4.8 by letting $\mathsf{Setup}_{f_\lambda,k}$ be the setup for a SNARG corresponding to $R_{f_\lambda,k}$. We also let $\alpha \in \mathbb{R}^+$ denote how much slower the SNARG prover runs compared to the evaluation of f_λ. That is, if $f_\lambda(k, x)$ can be evaluated in time t then $\mathsf{SNKProve}$ runs in time at most $\alpha \cdot t$ on the same machine. Note that we implicitly require that $\mathsf{SNKProve}$ is a linear algorithm but the construction can easily modified for quasilinear algorithms. VDF_{VC} works by running the computation of f_λ until $t \cdot (\frac{1}{\alpha+1})$

time has passed. That is, compute $\frac{k}{\alpha+1}$ iterations of g_λ on the input. Then the prover continues the computation and in parallel computes a SNARG for that first $\frac{1}{\alpha+1}$ fraction of the computation. The same process is repeated using geometrically decreasing parts of the computation. Namely, the prover produce a SNARG for the next $\frac{1}{\alpha+1}$ fraction of the remaining computation and in parallel continue the computation of f_λ as well as all other executing SNARG computations. After ℓ steps a $\left(\frac{\alpha}{\alpha+1}\right)^\ell$ fraction of the computation remains and ℓ SNARGs are being computed in parallel. Thus after $n = \log_{\frac{\alpha}{\alpha+1}}(k)$ steps only 1 invocation of g remains. The verifier can check this invocation directly. For simplicity we assume that k is a power of $\frac{\alpha}{\alpha+1}$. Note that all SNARG computations will finish at the same time, precisely when the computation of f_λ is completed.

- $\mathsf{Setup}(\lambda, t)$: Let g_λ be a (s, ϵ) sequential function. Let $k = \frac{t}{s}$. For $i = 1$ to $n = \log_{\frac{\alpha}{\alpha+1}}(k)$ the setup generates $(\mathsf{vk}_i, \mathsf{ek}_i) \leftarrow \mathsf{Setup}_{f_\lambda, k_i}(\lambda)$, where k_i is defined as

$$k_i = \left((\frac{\alpha}{\alpha+1})^{i-1} - (\frac{\alpha}{\alpha+1})^{i} \right) \cdot \frac{t}{s}.$$

  Output $\mathsf{pp} = \left\{ \left((\mathsf{ek}_i, k_i),\ \mathsf{vk}_i \right) \right\}_{i=1,\ldots,n}$.

- $\mathsf{Eval}(\mathsf{pp}, x)$: Let $x_0 = x$. For $i = 1$ to n compute $x_i = g^{(k_i)}(x_{i-1})$ and in parallel start the computation of $\pi_i = \mathsf{SNKProve}(\mathsf{ek}_i, (x_{i-1}, x_i), \perp)$. Finally let $y = g(x_n)$. Output $\{y, (x_1, \pi_1 \ldots, x_n, \pi_n)\}$

- $\mathsf{Verify}(\{\mathsf{vk}_1, \ldots, \mathsf{vk}_n\}, x, (y, (x_1, \pi_1 \ldots, x_n, \pi_n)))$: Verify the proofs by running $\mathsf{SNKVerify}(\mathsf{vk}_i, (x_{i-1}, x_i), \pi_i)$. If any verification fails, reject. Else output 'yes' if $g(x_n) = y$ and reject otherwise.

Lemma 4.2 (soundness)**.** *Given a sound SNARG system as defined by Definition 4.8, VDF_{VC} satisfies soundness (Definition 4.3)*

Proof. Assume that an $\mathsf{poly}(\lambda)/(t, \lambda)$ algorithm $\mathcal{A}$ outputs (with non-negligible probability in λ) a tuple (x, y, π) on input λ, t, and pp such that $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = \text{Yes}$ but $f_\lambda(k, x) \neq y$. We can then construct an adversary $\mathcal{A}'$ that violates the SNARG soundness. Note that the proof contains the intermediate computation steps $x_0, \ldots, x_n$ with $x_0 = x$. The verification guarantees that $g(x_n) = y$. However, if $f(x) \neq y$ then there must be an $i \in [0, n-1]$ such that $g^{(k_i)}(x_i) \neq x_{i+1}$ for $k_i = \frac{\alpha^{i-1}}{(\alpha+1)^i} \cdot \frac{t}{s}$. Note that this directly contradicts the soundness of the underlying SNARG. $\mathcal{A}'$ therefore simply runs $\mathcal{A}$ using honestly generated $(\mathsf{vk}_i, \mathsf{ek}_i)$ for all n SNARG proof systems. $\mathcal{A}'$ is able to break at the soundness of at least one of the proof systems simply using the output of $\mathcal{A}$, i.e. π_i and x_i, x_{i+1}. Since by assumption $\mathcal{A}'$ can only succeed with negligible probability $\mathcal{A}'$ also only succeeds with negligible probability. This shows that VDF_{VC} is sound. □

Theorem 4.2 (VDF_{VC} is a VDF)**.** *Given a (s, ϵ) sequential function f_λ and a SNARG proof system with perfect completeness, VDF_{VC} is a VDF scheme with $\sigma(t) = (1 - \epsilon)t$ sequentiality.*

Proof. The algorithms of VDF_{VC} satisfy the definition of the VDF algorithms. Treating α and s as constants, the verifier checks only a logarithmic (in t) number of succinct proofs and one evaluation

of g_λ. The prover requires $\log(t)$ parallelism for the computation of the proofs. Correctness is immediate from the construction and the completeness of the SNARG. Soundness was proved in Lemma 4.2. It remains to prove sequentiality. The Eval algorithm runs exactly in the time that it takes to compute $f_\lambda(\frac{t}{s}, x)$ as all the proof computation runs in parallel and by assumption completes the same moment the computation of f_λ completes. Any adversary that can output the VDF value in time less than $(1-\epsilon)t$ will therefore directly break the (t, ϵ) sequentiality of f_λ. □

4.5.1 Discussion

We note that both the proof size, the verifier time, and the required parallelism of VDF_{VC} are highly dependent on α. If $\alpha > 1$ the number of iterations, i.e. the number of proofs and required parallelism is close to $\log(t/s) \cdot \alpha$, i.e. linear in α. If $\alpha \approx 100,000$, as is the case for modern SNARGs on arbitrary computations, then this may become prohibitively large. In Section 4.6 and Section 4.7 we show how we can boost the construction to significantly reduce the prover overhead. In particular we construct specific instantiations of g and f that a) are more efficient to verify than to evaluate, allowing a SNARG proof to assert a simpler statement and b) that are specifically optimized for modern SNARG systems. With these optimizations and certain parameter selection it is feasible to bring α closer to 1 or possibly even below 1.

We also note that the same technique of computing several proofs in parallel can be used to boost subsequent VDF constructions such as the RivShaAdl78 based constructions by Pietrzak [Pie19b] and Wesolowski [Wes20]. These VDFs, in particular Wesolowski's construction, have a significant prover overhead. This leads to a suboptimal $\sigma(t)$-sequentiality. Using the same technique of computing proofs in parallel we can create "tight" VDFs with only a logarithmic overhead in terms of required parallelism, proof size, and verifier overhead.

4.6 A weak VDF based on injective rational maps

In this section we explore a framework for constructing a weak VDF satisfying $(t^2, o(t))$-sequentiality based on the existence of degree t injective rational maps that cannot be inverted faster than computing polynomial greatest common denominators (GCDs) of degree t polynomials, which we conjecture cannot be solved in parallel time less than $t - o(t)$ on fewer than t^2 parallel processors. Our candidate map will be a permutation polynomial over a finite field of degree t. The construction built from it is a weak VDF because the Eval will require $O(t)$ parallelism to run in parallel time t.

4.6.1 Injective rational maps

Rational maps on algebraic sets. An *algebraic rational function* on finite vector spaces is a function $F : \mathbb{F}_q^n \to \mathbb{F}_q^m$ such that $F = (f_1, \ldots, f_m)$ where each $f_i : \mathbb{F}_q^n \to \mathbb{F}_q$ is a rational function in

$\mathbb{F}_q(X_1, \ldots, X_n)$, for $i = 1, \ldots, m$. An *algebraic set* $\mathcal{Y} \subseteq \mathbb{F}_q^n$ is the complete set of points on which some set S of polynomials simultaneously vanish, i.e. $\mathcal{Y} = \{x \in \mathbb{F}_q^n | f(x) = 0 \text{ for all } f \in S\}$ for some $S \subset \mathbb{F}_q[X_1, \ldots, X_n]$. An *injective rational map* of algebraic sets $\mathcal{Y} \subseteq \mathbb{F}_q^n$ to $\mathcal{X} \subseteq \mathbb{F}_q^m$ is an algebraic rational function F that is injective on $\mathcal{Y}$, i.e. if $\mathcal{X} := F(\mathcal{Y})$, then for every $\bar{x} \in \mathcal{X}$ there exists a unique $\bar{y} \in \mathcal{Y}$ such that $F(\bar{y}) = \bar{x}$.

Inverting rational maps. Consider the problem of inverting an injective rational map $F = (f_1, ..., f_m)$ on algebraic sets $\mathcal{Y} \subseteq \mathbb{F}_q^n$ to $\mathcal{X} \subseteq \mathbb{F}_q^m$. Here $\mathcal{Y} \subseteq \mathbb{F}_q^n$ is the set of vanishing points of some set of polynomials S. For $x \in \mathbb{F}_q^m$, a solution to $F(\bar{y}) = \bar{x}$ is a point $\bar{y} \in \mathbb{F}_q^n$ such that all polynomials in S vanish at $\bar{y}$ and $f_i(\bar{y}) = x_i$ for $i = 1, ..., m$. Furthermore, each $f_i(\bar{y}) = g(\bar{y})/h(\bar{y}) = x_i$ for some polynomials g, h, and hence yields a polynomial constraint $z_i(\bar{y}) := g(\bar{y}) - x_i h(\bar{y}) = 0$. In total we are looking for solutions to $|S| + m$ polynomial constraints on $\bar{y}$.

We illustrate two special cases of injective rational maps that can be inverted by a univariate polynomial GCD computation. In general, inverting injective rational maps on $\mathbb{F}_q^d$ for constant d can be reduced to a univariate polynomial GCD computation using resultants.

- *Rational functions on finite fields.* Consider any injective rational function $F(X) = g(X)/h(X)$, for univariate polynomials h, g, on a finite field $\mathbb{F}_q$. A finite field is actually a special case of an algebraic set over itself; it is the set of roots of the polynomial $X^q - X$. Inverting F on a point $c \in \mathbb{F}_q$ can be done by calculating $GCD(X^q - X, g(X) - c \cdot h(X))$, which outputs $X - s$ for the unique s such that $F(s) = c$.

- *Rational maps on elliptic curves.* An elliptic curve $E(\mathbb{F}_q)$ over $\mathbb{F}_q$ is a 2-dimensional algebraic set of vanishing points in $\mathbb{F}_q^2$ of a bivariate polynomial $E(y, x) = y^2 - x^3 - ax - b$. Inverting an injective rational function F on a point in the image of $F(E(\mathbb{F}_q))$ involves computing the GCD of three bivariate polynomials: E, z_1, z_2, where z_1 and z_2 come from the two rational function components of F. The resultant $R = Res_y(z_1, z_2)$ is a univariate polynomial in x of degree $deg(z_1) \cdot deg(z_2)$ such that $R(x) = 0$ iff there exists y such that (x, y) is a root of both z_1 and z_2. Finally, taking the resultant again $R' = Res_y(R, E)$ yields a univariate polynomial such that any root x of R' has a corresponding coordinate y such that (x, y) is a point on E and satisfies constraints z_1 and z_2. Solving for the unique root of R' reduces to a Euclidean GCD computation as above. Then given x, there are two possible points $(x, y) \in E$, so we can try them both and output the unique point that satisfies all the constraints.

Euclidean algorithm for univariate polynomial GCD. Univariate polynomials over a finite field form a Euclidean domain, and therefore the GCD of two polynomials can be found using the Euclidean algorithm. For two polynomials f and g such that $deg(f) > deg(g) = d$, one first reduces $f \bmod g$ and then computes $GCD(f, g) = GCD(f \bmod g, g)$. In the example $f = X^q - X$, the first step of reducing $X^q \bmod g$ requires $O(\log(q))$ multiplications of degree $O(deg(g))$ polynomials.

Starting with X, we run the sequence of repeated squaring operations to get X^q, reducing the intermediate results mod g after each squaring operation. Then running the Euclidean algorithm to find $GCD(f \bmod g, g)$ involves $O(d)$ sequential steps where in each step we subtract two $O(d)$ degree polynomials. On a sequential machine this computation takes $O(d^2)$ time, but on $O(d)$ parallel processors this can be computed in parallel time $O(d)$.

NC algorithm for univariate polynomial GCD. There is an algorithm for computing the GCD of two univariate polynomials of degree d in $O(\log^2(d))$ parallel time, but requires $O(d^{3.8})$ parallel processors. This algorithm runs d parallel determinant calculations on submatrices of the Sylvester matrix associated with the two polynomials, each of size $O(d^2)$. Each determinant can be computed in parallel time $O(\log^2(d))$ on $M(d) \in O(d^{2.85})$ parallel processors [Cod+97]. The parallel advantage of this method over the euclidean GCD method kicks in after $O(d^{2.85})$ processors. For any $c \leq d/\log^2(d)$, it is possible to compute the GCD in $O(d/c)$ steps on $c\log^2(d)M(d)$ processors.

Sequentiality of univariate polynomial GCD. The GCD can be calculated in parallel time d using d parallel processors via the Euclidean algorithm. The NC algorithm only beats this bound on strictly greater than $d^{2.85}$ processors, but a hybrid of the two methods can gain an $o(d)$ speedup on only d^2 processors. Specifically, we can run the Euclidean method for $d - d^{2/3}$ steps until we are left with two polynomials of degree $d^{2/3}$, then we can run the NC algorithm using $\log^3(d)M(d^{2/3}) < (d^{2/3})^3 = d^2$ processors to compute the GCD of these polynomials in $O(d^{2/3}/\log(d))$ steps, for a total of $d - \epsilon d^{2/3}$ steps. This improvement can be tightened further, but generally results in $d - o(d)$ steps as long as $M(d) \in \omega(d^2)$.

We pose the following assumption on the parallel complexity of calculating polynomials GCDs on fewer that $O(d^2)$ processors. This assumption would be broken if there is an NC algorithm for computing the determinant of a $n \times n$ matrix on $o(n^2)$ processors, but this would require a significant advance in mathematics on a problem that has been studied for a long time.

Assumption 4.2. *There is no general algorithm for computing the GCD of two univariate polynomials of degree d over a finite field $\mathbb{F}_q$ (where $q > d^3$) in less than parallel time $d - o(d)$ on $O(d^2)$ parallel processors.*

On the other hand, evaluating a polynomial of degree d can be logarithmic in its degree, provided the polynomial can be expressed as a small arithmetic circuit, e.g. $(ax + b)^d$ can be computed with $O(\log(d))$ field operations.

Abstract weak VDF from an injective rational map. Let $F : \mathbb{F}_q^n \to \mathbb{F}_q^m$ be a rational function that is an injective map from $\mathcal{Y}$ to $\mathcal{X} := F(\mathcal{Y})$. We further require that $\mathcal{X}$ is efficiently sampleable and that F can be evaluated efficiently for all $\bar{y} \in \mathcal{Y}$. When using F in a VDF we will require that $|\mathcal{X}| > \lambda t^3$ to prevent brute force attacks, where t and λ are given as input to the Setup algorithm.

We will need a family $\mathcal{F} := \{(q, F, \mathcal{X}, \mathcal{Y})\}_{\lambda,t}$ parameterized by λ and t. Given such a family we can construct a weak VDF as follows:

- $\mathsf{Setup}(\lambda, t)$: choose a $(q, F, \mathcal{X}, \mathcal{Y}) \in \mathcal{F}$ specified by λ and t, and output $\mathsf{pp} := ((q, F), (q, F))$.
- $\mathsf{Eval}((q, F), \bar{x})$: for an output $\bar{x} \in \mathcal{X} \subseteq \mathbb{F}_q^m$ compute $\bar{y} \in \mathcal{Y}$ such that $F(\bar{y}) = \bar{x}$; The proof π is empty.
- $\mathsf{Verify}((q, F), \bar{x}, \bar{y}, \pi)$ outputs Yes if $F(\bar{y}) = \bar{x}$.

The reason we require that F be injective on $\mathcal{Y}$ is so that the solution $\bar{y}$ be unique.
The construction is a weak $(p(t), \sigma(t))$-VDF for $p(t) = t^2$ and $\sigma(t) = t - o(t)$ assuming that there is no algorithm that can invert of $F \in \mathcal{F}$ on a random value in less than parallel time $d - o(d)$ on $O(d^2)$ processors. Note that this is a stronger assumption than 4.2 as the inversion reduces to a specific GCD computation rather than a general one.

Candidate rational maps. The question, of course, is how to instantiate the function family $\mathcal{F}$ so that the resulting weak VDF system is secure. There are many examples of rational maps on low dimensional algebraic sets among which we can search for candidates. Here we will focus on the special case of efficiently computable permutation polynomials over $\mathbb{F}_q$, and one particular family of permutation polynomials that may be suitable.

4.6.2 Univariate permutation polynomials

The simplest instantiation of the VDF system above is when $n = m = 1$ and $\mathcal{Y} = \mathbb{F}_q$. In this case, the function F is a univariate polynomial $f : \mathbb{F}_q \to \mathbb{F}_q$. If f implements an injective map on $\mathbb{F}_q$, then it must be a permutation of $\mathbb{F}_q$, which brings us to the study of *univariate permutation polynomials* as VDFs.

The simplest permutation polynomials are the monomials x^e for $e \geq 1$, where $gcd(e, q-1) = 1$. These polynomials however, can be easily inverted and do not give a secure VDF. Dickson polynomials [LMT93] $D_{n,\alpha} \in \mathbb{F}_p[x]$ are another well known family of polynomials over $\mathbb{F}_p$ that permute $\mathbb{F}_p$. Dickson polynomials are defined by a recurrence relation and can be evaluated efficiently. Dickson polynomials satisfy $D_{t,\alpha^n}(D_{n,\alpha}(x)) = x$ for all n, t, α where $n \cdot t = 1 \bmod p - 1$, hence they are easy to invert over $\mathbb{F}_p$ and again do not give a secure VDF.

A number of other classes of permutation polynomials have been discovered over the last several decades [Hou15]. We need a class of permutation polynomials over a suitably large field that have a tunable degree, are fast to evaluate (i.e. have polylog/(d) circuit complexity), and cannot be inverted faster than running the parallelized Euclidean algorithm on O(d) processors.

Candidate permutation polynomial. We consider the following polynomial of Guralnick and Muller [GM97] over $\mathbb{F}_{p^m}$:

$$\frac{(x^s - ax - a) \cdot (x^s - ax + a)^s + ((x^s - ax + a)^2 + 4a^2x)^{(s+1)/2}}{2x^s} \tag{4.1}$$

where $s = p^r$ for odd prime p and a is not a $(s-1)st$ power in $\mathbb{F}_{p^m}$. This polynomial is a degree s^3 permutation on the field $\mathbb{F}_{p^m}$ for all s, m chosen independently.

Below we discuss why instantiating a VDF with nearly all other examples of permutation polynomials would not be secure and why attacks on these other polynomials do not work against this candidate.

Attacks on other families of permutation polynomials. We list here several other families of permutation polynomials that can be evaluated in $O(\text{polylog}/(d))$ time, yet would not yield a secure VDF. We explain why each of these attacks do not work against the candidate polynomial.

1. *Sparse permutation polynomials.* Sparse polynomials have a constant number of terms and therefore can be evaluated in time $O(\log(d))$. There exist families of non-monomial sparse permutation polynomials, e.g. $X^{2^{t+1}+1} + X^3 + X \in \mathbb{F}_{2^{2t+1}}[X]$ [Hou15, Thm 4.12]. The problem is that the degree of this polynomial is larger than the square root of the field size, which allows for brute force parallel attacks. Unfortunately, all known sparse permutation polynomials have this problem. In our candidate the field size can be made arbitrarily large relative to the degree of the polynomial.

2. *Linear algebraic attacks.* A classic example of a sparse permutation polynomial of tunable degree over an arbitrarily large field, due to Mathieu [Mat61], is the family $x^{p^i} - ax$ over $\mathbb{F}_{p^m}$ where a is not a $p-1$st power. Unfortunately, this polynomial is easy to invert because $x \mapsto x^{p^i}$ is a linear operator in characteristic p so the polynomial can be written as a linear equation over an m-dimensional vector space. To prevent linear algebraic attacks the degree of at least one non-linear term in the polynomial cannot be divisible by the field characteristic p. In our candidate there are many such non-linear terms, e.g. of degree $s+1$ where $s = p^r$.

3. *Exceptional polynomials co-prime to characteristic.* An *exceptional polynomial* is a polynomial $f \in \mathbb{F}_q[X]$ which is a permutation on $\mathbb{F}_{q^m}$ for infinitely many m, which allows us to choose sufficiently large m to avoid brute force attacks. Any permutation polynomial of degree at most $q^{1/4}$ over F_q is exceptional [Zie]. Since we want q to be exponential in the security parameter and the degree to be sub-exponential we can restrict the search for candidate polynomials to exceptional polynomials. However, all exceptional polynomials over $\mathbb{F}_q$. of degree co-prime to q can be written as the composition of Dickson polynomials and linear polynomials, which are easy to invert [Mül97]. In our candidate, the degree s^3 of the polynomial and field size are both powers of p, and are therefore not co-prime.

Additional application: a new family of one-way permutations. We note that a sparse permutation polynomial of sufficiently high degree over a sufficiently large finite field may be a good candidate for a one-way permutation. This may give a secure one-way permutation over a domain of smaller size than what is possible by other methods.

4.6.3 Comparison to square roots mod p

A classic approach to designing a sequentially slow verifiable function, dating back to Dwork and Naor [DN93], is computing modular square roots. Given a challenge $x \in \mathbb{Z}_p$, computing $y = x^{\frac{p+1}{4}} \pmod{p}$ can be efficiently verified by checking that $y^2 = x \pmod{p}$ (for $p \equiv 3 \pmod{4}$). There is no known way to compute this exponentiation in faster than $\log(p)$ sequential field multiplications.

This is a special case of inverting a rational function over a finite field, namely the polynomial $f(y) = y^2$, although this function is not injective and therefore cannot be calculated with GCDs. An injective rational function with nearly the same characteristics is the permutation $f(y) = y^3$. Since the inverse of 3 mod $p-1$ will be $O(\log p)$ bits, this requires $O(\log p)$ squaring operations to invert. Viewed another way, this degree 3 polynomial can be inverted on a point c by computing the $\text{GCD}(y^p - y, y^2 - c)$, where the first step requires reducing $y^p - y \bmod y^3 - c$, involving $O(\log p))$ repeated squarings and reductions mod $y^3 - c$.

While this approach appears to offer a delay parameter of $t = \log(p)$, as t grows asymptotically the evaluator can use $O(t)$ parallel processors to gain a factor t parallel speedup in field multiplications, thus completing the challenge in parallel time equivalent to one squaring operation on a sequential machine. Therefore, there is asymptotically no difference in the parallel time complexity of the evaluation and the total time complexity of the verification, which is why this does not even meet our definition of a weak VDF. Our approach of using higher degree injective rational maps gives a strict (asymptotic) improvement on the modular square/cubes approach, and to the best of our knowledge is the first concrete algebraic candidate to achieve an exponential gap between parallel evaluation complexity and total verification complexity.

4.7 Practical improvements on VDFs from IVC

In this section we propose a practical boost to constructing VDFs from IVC (Section 4.4). In an IVC construction the prover constructs a SNARK which verifies a SNARK. Ben-Sasson et al. [Ben+14b] showed an efficient construction for IVC using "cycles of Elliptic curves". This construction builds on the pairing-based SNARK [Par+13]. This SNARK system operates on arithmetic circuits defined over a finite field $\mathbb{F}_p$. The proof output consists of elements of an elliptic curve group $E/\mathbb{F}_q$ of prime order p (defined over a field $\mathbb{F}_q$). The SNARK verification circuit, which computes a pairing, is therefore an arithmetic circuit over $\mathbb{F}_q$. Since $q \neq p$, the prover cannot construct a new SNARK that directly operates on the verification circuit, as the SNARK operates on circuits defined over

$\mathbb{F}_p$. Ben-Sasson et. al. propose using two SNARK systems where the curve order of one is equal to the base field of the other, and vice versa. This requires finding a pair of pairing-friendly elliptic curves E_1, E_2 (defined over two different base fields $\mathbb{F}_1$ and $\mathbb{F}_2$) with the property that the order of each curve is equal to the size of the base field of the other.

The main practical consideration in VDF_{IVC} is that the evaluator needs to be able to update the incremental SNARK proofs at the same rate as computing the underlying sequential function, and without requiring a ridiculous amount of parallelism to do so. Our proposed improvements are based on two ideas:

1. In current SNARK/IVC constructions (including [Par+13], [Ben+14b]) the prover complexity is proportional to the multiplicative arithmetic complexity of the underlying statement over the field $\mathbb{F}_p$ used in the SNARK ($p \approx 2^{128}$). Therefore, as an optimization, we can use a "SNARK friendly" hash function (or permutation) as the iterated sequential function such that the verification of each iteration has a lower multiplicative arithmetic complexity over $\mathbb{F}_p$.

2. We can use the Eval of a weak VDF as the iterated sequential function, and compute a SNARK over the Verify circuit applied to each incremental output instead of the Eval circuit. This should increase the number of sequential steps required to evaluate the iterated sequential function relative to the number of multiplication gates over which the SNARK is computed.

An improvement of type (1) alone could be achieved by simply using a cipher or hash function that has better multiplicative complexity over the SNARK field $\mathbb{F}_q$ than AES or SHA256 (e.g., see MiMC [Alb+16], which has 1.6% complexity of AES). We will explain how using square roots in $\mathbb{F}_q$ or a suitable permutation polynomial over $\mathbb{F}_q$ (from Section 4.6) as the iterated function achieve improvements of both types (1) and (2).

4.7.1 Iterated square roots in $\mathbb{F}_q$

Sloth A recent construction called Sloth [LW15] proposed a secure way to chain a series of square root computations in $\mathbb{Z}_p$ interleaved with a simple permutation [4] such that the chain must be evaluated sequentially, i.e. is an iterated sequential function (Definition 4.7). More specifically, Sloth defines two permutations on $\mathbb{F}_p$: a permutation ρ such that $\rho(x)^2 = \pm x$, and a permutation σ such that $\sigma(x) = x \pm 1$ depending on the parity of x. The parity of x is defined as the integer parity of the unique $\hat{x} \in \{0, ..., p-1\}$ such that $\hat{x} = x \mod p$. Then Sloth iterates the permutation $\tau = \rho \circ \sigma$.

The verification of each step in the chain requires a single multiplication over $\mathbb{Z}_p$ compared to the $O(\log(p))$ multiplications required for evaluation. Increasing the size of p amplifies this gap,

[4] If square roots are iterated on a value x without an interleaved permutation then there is a shortcut to the iterated computation that first computes $v = (\frac{p+1}{4})^\ell \mod p$ and then the single exponentiation x^v.

however it also introduces an opportunity for parallelizing multiplication in $\mathbb{Z}_p$ for up to $O(\log(p))$ speedup.

Using Sloth inside VDF_{IVC} would only achieve a practical benefit if $p = q$ for the SNARK field $\mathbb{F}_q$, as otherwise implementing multiplication in $\mathbb{Z}_p$ in an arithmetic circuit over $\mathbb{F}_q$ would have $O(\log^2(p))$ complexity. On modern architectures, multiplication of integers modulo a 256-bit prime is near optimal on a single core, whereas multi-core parallelized algorithms only offer speed-ups for larger primes [BS12]. Computing a single modular square root for a 256-bit prime takes approximately 45,000 cycles[5] on an Intel Core i7 [LW15], while computing SHA256 for 256-bit outputs takes approximately 864 cycles[6].

The best known arithmetic circuit implementation of SHA256 has 27,904 multiplication gates[Ben+14a]. In stark contrast, the arithmetic circuit over $\mathbb{F}_p$ for verifying a modular square root is a single multiplication gate. Verifying the permutation σ is more complex as it requires a parity check, but this requires at most $O(\log(p))$ complexity.

Sloth++ extension Replacing SHA256 with Sloth as the iterated function in VDF_{IVC} already gives a significant improvement, as detailed above. Here we suggest yet a further optimization, which we call Sloth++. The main arithmetic complexity of verifying a step of Sloth comes from the fact that the permutation σ is not naturally arithmetic over $\mathbb{F}_p$, which was important for preventing attacks that factor $\tau^\ell(x)$ as a polynomial over $\mathbb{F}_p$. Our idea here is to compute square roots over a degree 2 extension field $\mathbb{F}_{p^2}$ interleaved with a permutation that is arithmetic over $\mathbb{F}_p$ but not over $\mathbb{F}_{p^2}$.

In any degree r extension field $\mathbb{F}_{p^r}$ of $\mathbb{F}_p$ for a prime $p = 3 \mod 4$ a square root of an element $x \in \mathbb{F}_{p^r}$ can be found by computing $x^{(p^r+1)/4}$. This is computed in $O(r \log(p))$ repeated squaring operations in $\mathbb{F}_p^r$. Verifying a square root requires a single multiplication over $\mathbb{F}_{p^r}$. Elements of $\mathbb{F}_{p^r}$ can be represented as length r vectors over $\mathbb{F}_p$, and each multiplication reduces to $O(r^2)$ arithmetic operations over $\mathbb{F}_p$. For $r = 2$ the verification *multiplicative* complexity over $\mathbb{F}_p$ is exactly 4 gates.

In Sloth++ we define the permutation ρ exactly as in Sloth, yet over $\mathbb{F}_{p^2}$. Then we define a simple non-arithmetic permutation σ on $\mathbb{F}_{p^2}$ that swaps the coordinates of elements in their vector representation over $\mathbb{F}_p$ and adds a constant, i.e. maps the element (x, y) to $(y + c_1, x + c_2)$. The arithmetic circuit over $\mathbb{F}_p$ representing the swap is trivial: it simply swaps the values on the input wires. The overall multiplicative complexity of verifying an iteration of Sloth++ is only 4 gates over $\mathbb{F}_p$. Multiplication can be parallelized for a factor 2 speedup, so 4 gates must be verified roughly every 89,000 parallel-time evaluation cycles. Thus, even if an attacker could manage to speedup the modular square root computation by a factor 100 using an ASIC designed for 256-bit multiplication, for parameters that achieve the same wall-clock delay, the SNARK verification

[5] This is extrapolated from [LW15], which reported that 30 million iterations of a modular square root computation for a 256-bit prime took 10 minutes on a single 2.3 GHz Intel Core i7.

[6] http://www.ouah.org/ogay/sha2/

complexity of Sloth++ is over a 7,000 fold improvement over that of a SHA256 chain.

Cube roots The underlying permutation in both Sloth and Sloth++ can be replaced by cube roots over $\mathbb{F}_q$ when $gcd(3, q-1) = 1$. In this case the slow function is computing $\rho(x) = x^v$ where $3v = 1 \bmod q-1$. The output can be verified as $\rho(x)^3 = x$.

4.7.2 Iterated permutation polynomials

Similar to Sloth+, we can use our candidate permutation polynomial (Equation 4.1) over $\mathbb{F}_q$ as the iterated function in VDF_{IVC}. Recall that $\mathbb{F}_q$ is an extension field chosen independently from the degree of the polynomial. We would choose $q \approx 2^{256}$ and use the same $\mathbb{F}_q$ as the field used for the SNARK system. For each $O(d)$ sequential provers steps required to invert the polynomial on a point, the SNARK only needs to verify the evaluation of the polynomial on the inverse, which has multiplicative complexity $O(\log(d))$ over $\mathbb{F}_q$. Concretely, for each 10^5 parallel-time evaluation cycles a SNARK needs to verify approximately 16 gates. This is yet another factor 15 improvement over Sloth+. The catch is that the evaluator must use 10^5 parallelism[7] to optimize the polynomial GCD computation. We must also assume that an adversary cannot feasibly amass more than 10^{14} parallel processors to implement the NC parallelized algorithm for polynomial GCD.

From a theory standpoint, using permutation polynomials inside VDF_{IVC} reduces it to a weak VDF because the degree of the polynomial must be super-polynomial in λ to prevent an adversary from implementing the NC algorithm on $poly(\lambda)$ processors, and therefore the honest evaluator is also required to use super-polynomial parallelism. However, the combination does yield a better weak VDF, and from a practical standpoint appears quite promising for many applications.

4.8 Related work

Taking a broad perspective, VDFs can be viewed as an example of *moderately hard* cryptographic functions. Moderately hard functions are those whose difficulty to compute is somewhere in between 'easy' (designed to be as efficient as possible) and 'hard' (designed to be so difficult as to be intractable). The use of moderately hard cryptographic functions dates back at least to the use of a deliberately slow DES variant for password hashing in early UNIX systems [MT79]. Dwork and Naor [DN93] coined the term *moderately hard* in a classic paper proposing client puzzles or "pricing functions" for the purpose of preventing spam. Juels and Brainard proposed the related notion of a *client puzzle*, in which a TCP server creates a puzzle which must be solved before a client can open a connection [JB99]. Both concepts have been studied for a variety of applications, including TLS handshake requests [ANL00; DS01], node creation in peer-to-peer networks [Dou02], creation

[7]This is reasonable if the evaluator has an NVIDIA Titan V GPU, which can compute up to 10^{14} pipelined arithmetic operations per second (https://www.nvidia.com/en-us/titan/titan-v/).

of digital currency [RS96; Dai98; Nak08] or censorship /resistance [BX11]. For interactive client puzzles, the most common construction is as follows: the server chooses a random ℓ-bit value x and sends to the client $\mathrm{H}(x)$ and $x[\ell - \log_2 t - 1]$. The client must send back the complete value of x. That is, the server sends the client $\mathrm{H}(x)$ plus all of the bits of x except the final $\log_2 t + 1$ bits, which the client must recover via brute force.

4.8.1 Inherently sequential puzzles

The simple interactive client puzzle described above is embarrassingly parallel and can be solved in constant time given t processors. In contrast, the very first construction of a client puzzle proposed by Dwork and Naor involved computing modular square roots and is believed to be inherently sequential (although they did not discuss this as a potential advantage).

The first interest in designing puzzles that require an inherently sequential solving algorithm appears to come for the application of hardware benchmarking. Cai et al. [Cai+93; CNW97] proposed the use of inherently sequential puzzles to verify claimed hardware performance as follows: a customer creates an inherently-sequential puzzle and sends it to a hardware vendor, who then solves it and returns the solution (which the customer can easily verify) as quickly as possible. Note that this work predated the definition of client puzzles. Their original construction was based on exponentiation modulo an RivShaAdl78 number N, for which the customer has created N and therefore knows $\varphi(N)$. They later proposed solutions based on a number of other computational problems not typically used in cryptography, including Gaussian elimination, fast Fourier transforms, and matrix multiplication.

Time-lock puzzles Rivest, Shamir, and Wagner [RSW96] constructed a time-lock encryption scheme, also based on the hardness of RivShaAdl78 factoring and the conjectured sequentiality of repeated exponentiation in a group of unknown order. The encryption key K is derived as $K = x^{2^t} \in \mathbb{Z}_N$ for an RivShaAdl78 modulus N and a published starting value x. The encrypting party, knowing $\varphi(N)$, can reduce the exponent $e = 2^t \bmod \varphi(N)$ to quickly derive $K = x^e \bmod N$. The key K can be publicly recovered slowly by 2^t iterated squarings. Boneh and Naor [BN00] showed that the puzzle creator can publish additional information enabling an efficient and sound proof that K is correct. In the only alternate construction we are aware of, Bitansky et al. [Bit+16] show how to construct time-lock puzzles from randomized encodings assuming any inherently-sequential functions exist.

Time-lock puzzles are similar to VDFs in that they involve computing an inherently sequential function. However, time-lock puzzles are defined in a private-key setting where the verifier uses its private key to prepare each puzzle (and possibly a verification proof for the eventual answer). In contrast to VDFs, this trusted setup must be performed per-puzzle and each puzzle takes no unpredictable input.

Proofs of sequential work Mahmoody et al.[MMV13] proposed publicly verifiable proofs of sequential work (PoSW) which enable proving to any challenger that a given amount of sequential work was performed on a specific challenge. As noted, time-lock puzzles are a type of PoSW, but they are not publicly verifiable. VDFs can be seen as a special case of publicly verifiable proofs of sequential work with the additional guarantee of a unique output (hence the use of the term "function" versus "proof").

Mahmoody et al.'s construction uses a sequential hash function H (modeled as a random oracle) and depth robust directed-acyclic graph G. Their puzzle involves computing a *labeling* of G using H salted by the challenge c. The label on each node is derived as a hash of all the labels on its parent nodes. The labels are committed to in a Merkle tree and the proof involves opening a randomly sampled fraction. Very briefly, the security of this construction is related to graph pebbling games (where a pebble can be placed on a node only if all its parents already have pebbles) and the fact that depth robust graphs remain sequentially hard to pebble even if a constant fraction of the nodes are removed (in this case corresponding to places where the adversary cheats). Mahmoody et. al. proved security unconditionally in the random oracle model. Depth robust graphs and parallel pebbling hardness are use similarly to construct memory hard functions [ABP17] and proofs of space [Dzi+15]. Cohen and Pietrzak [CP18] constructed a similar PoSW using a simpler non-depth-robust graph based on a Merkle tree.

PoSWs based on graph labeling don't naturally provide a VDF because removing any single edge in the graph will change the output of the proof, yet is unlikely to be detected by random challenges.

Sequentially hard functions The most popular solution for a slow function which can be viewed as a proto-VDF, dating to Dwork and Naor [DN93], is computing modular square roots. Given a challenge $x \in \mathbb{Z}_p$, computing $y = x^{\frac{p+1}{4}} \pmod p$ can be efficiently verified by checking that $y^2 = x \pmod p$ (for $p \equiv 3 \pmod 4$). There is no known algorithm for computing modular exponentiation which is sublinear in the exponent. However, the difficulty of puzzles is fixed to $t = \log p$ as the exponent can be reduced modulo $p - 1$ before computation, requiring the use of a very large prime p to produce a difficult puzzle.

This puzzle has been considered before for similar applications as our VDFs, in particular randomness beacons [JM11; LW15]. Lenstra and Wesolowski [LW15] proposed creating a more difficult puzzle for a small p by chaining a series of such puzzles together (interleaved with a simple permutation) in a construction called Sloth. We proposed a simple improvement of this puzzle in Section 4.7. Recall that this does not meet our asymptotic definition of a VDF because it does not offer (asymptotically) efficient verification, however we used it as an important building block to construct a more practical VDF based on IVC. Asymptotically, Sloth is comparable to a hash chain of length t with t checkpoints provided as a proof, which also provides $O(\text{polylog}/(t))$-time verification (with t processors) and a solution of size $\Theta(t \cdot \lambda)$.

Chapter 5

ProtoStar: Efficient IVC for VDFs and succinct Blockchains

5.1 Introduction

Incrementally Verifiable Computation[Val08] is a powerful primitive that enables a possibly infinite computation to be run, such that the correctness of the state of the computation can be verified at any point. IVC, and it's generalization to DAGs, PCD[CT10], have many applications, including distributed computation[Bit+13a; CTV15], blockchains[Bon+20a; KB20], verifiable delay functions [Bon+18], verifiable photo editing [NT16], and SNARKs for machine-computations[Ben+14b]. An IVC-based VDF construction is the current candidate VDF for Ethereum[KMT22]. One of the most exciting applications of IVC and PCD is the ZK-EVM. This is an effort to build a proof system that can prove that Ethereum blocks, as they exist today, are valid[Fou22].

Accumulation and folding. Historically, IVC was built from recursive SNARKs, proving that the previous computation step had a valid SNARK that proves correctness up to that point. Recently, an exciting new approach was initiated by Halo[BGH19] and has led to a series of significant advances[Bün+20; Bün+21a; KST22]. The idea is related to batch verification. Instead of verifying a SNARK at every step of the computation, we can instead *accumulate* the SNARK verification check with previous checks. We define an *accumulator*[1] such that we can combine a new SNARK and an old accumulator into a new accumulator. Checking or *deciding* the new accumulator implies that all previously accumulated SNARKs were valid. Now the recursive statement just needs to ensure the accumulation was performed correctly. Amazingly, this accumulation step can be significantly cheaper than SNARK verification[BGH19; Bün+20]. Even more surprising, this process does

[1] Unrelated to set accumulators.

not even require a SNARK but instead can be instantiated with a non-succinct NARK[Bün+21a], as long as there exists an efficient accumulation scheme for that NARK. The most efficient accumulation (aka folding) scheme constructions yield IVC constructions, where the recursive circuit is dominated by as few as 2 elliptic curve scalar multiplications[Bün+21a; KST22]. These constructions only require the discrete logarithm assumption in the random oracle model and, unlike many efficient SNARK-based IVCs, do not require a trusted setup, pairings, or FFTs. These constructions build an accumulation scheme for one fixed (but universal) R1CS language by taking a random linear combination between the accumulator and a new proof. R1CS is a minimal expression of NP, defined by three matrices A, B, C, that close resembles arithmetic circuits with addition and multiplication gates. However, it has limited flexibility, especially as the current constructions require fixing R1CS matrices that are used for all computation steps. These limitations are especially problematic for ZK-EVMs. In a ZK-EVM, each VM instruction (OP-CODE) is encoded in a different circuit. Each circuit uses high-degree gates, instead of just multiplication, and so-called lookup gates [GWC19]. These lookup gates enable looking up that a circuit value is in some table, simplifying range proofs and bit-operations. These R1CS-based accumulation schemes contrast non-IVC SNARK developments, with an increased focus on high-degree gate[GWC19; Che+22] and lookup support[GW20a]. For lookups, a recent line of work has shown that if the table can be pre-computed, we can perform n lookups in a table of size T in time $O(n \log n)$, independent of T[Zap+22b; PK22; Zap+22c; EFG22].

More expressive accumulation. There have been efforts to build accumulation schemes that overcome the limitations of fixed R1CS. SuperNova[KS22] enables selecting the appropriate R1CS instance at runtime without a recursive circuit that is linear in all R1CS instances. The approach, however, still has limitations. The recursive circuit still requires many (though a constant number of) hashes and a hash-to-group gadget, and additionally, the accumulator, and thus the final proof, is still linear in the total size of all instances. Sangria[Moh23] describes an accumulation scheme for a Plonk-like[GWC19] constraint system with degree-2 gates. It also proposes a solution for higher-degree gates in the future work section but without security proof. These accumulation schemes are built from simple underlying protocols performing a linear combination between an accumulator and a proof. However, the constructions seem ad hoc and need individual security proof. This leads us to our main research questions:

Recipe for accumulation Is there a general recipe for building accumulation schemes? Can we formalize this recipe, simplifying the task of constructing secure and efficient accumulation schemes?

Efficient accumulation for ZK-EVM Can we build an accumulation/folding scheme for a language that combines the benefits of the most advanced proof systems today? Can we support multiple circuits, high-degree, and lookup gates?

We answer both questions positively. Firstly we show a general compiler that takes any $(2k-1)$-move special-sound interactive argument for an NP-complete relation $\mathcal{R}_{\mathsf{NP}}$ with an algebraic degree d verifier and construct an efficient IVC-scheme from it. This is done in 4 simple steps.

1. We compress the prover message by committing to them in a homomorphic commitment scheme.
2. Then we apply the Fiat-Shamir transform to yield a secure NARK. [AFK22; Wik21]
3. We build a simple and efficient accumulation scheme that samples a random challenge α and takes a linear combination between the current accumulator and the new NARK.
4. We apply the compiler by [Bün+21a] to yield a secure IVC scheme.

The recursive circuit of this transformation is dominated by only $d+k-1$ scalar multiplications in the additive group of the commitment scheme[2] for a protocol with k prover messages and a degree d verifier. For R1CS, where $k=1$ and $d=2$, this yields the same protocol and efficiency as Nova[KST22]. We can further reduce the size of the recursive circuit to only $k+2$ group scalar multiplication, by compressing all verification equations using a random linear combination.

Efficient simple protocols for $\mathcal{R}_{\text{mplkup}}$. Equipped with this compiler, we design ProtoStar, a simple and efficient IVC scheme for a highly expressive language $\mathcal{R}_{\text{mplkup}}$ that supports multiple non-uniform circuits and enables high degree and lookup gates. The schemes can be instantiated from any linearly homomorphic vector commitment, e.g., the discrete logarithm-based Pedersen commitment[Ped92], and do not require a trusted setup or the computation of large FFTs. The protocol has several advantages over prior schemes:

Non-uniform IVC without overhead. Each iteration has a program counter **pc** that selects one out of I circuits. Part of the circuit constrains **pc**; e.g., **pc** could depend on the iteration or indicate which instruction within a VM is executed. The IVC-prover, including the recursive statement, only requires one exponentiation per non-zero bit in the witness. The prover's computation is independent of I.

Flexible high degree gates. Our protocol supports Plonk-like constraint systems with degree d gates instead of just addition and multiplication. The recursive statement consists of 3 group scalar multiplications and $d+O(1)$ hash and field operations. Unlike in traditional Plonk, there is no additional cost for additional gate types (of degree less than d) and additional selectors. This enables a high level of non-uniformity, even within a circuit.

[2]When instantiated with elliptic curve Pedersen commitments, this translates to $d+k-1$ elliptic curve multiplications. This is usually the largest component of the recursive statement.

	ProtoStar	HyperNova	SuperNova
Language	Degree d Plonk/CCS	Degree d CCS	R1CS (degree 2)
Non-uniform	yes	no	yes
P native	$\lvert\mathbf{w}\rvert\,\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d\log^2 d)\,\mathbb{F}$	$\lvert\mathbf{w}\rvert\,\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d\log^2 d)\,\mathbb{F}$	$\lvert\mathbf{w}\rvert\,\mathbb{G}$
extra P native w/ lookup	$O(\lvert\ell_{\mathsf{lk}}\rvert)\,\mathbb{G}$	$O(T)\,\mathbb{F}$	N/A
P recursive	$3\mathbb{G}$ $(d+O(1))\mathsf{H}+\mathsf{H}_{\mathsf{in}}$ $(d+O(1))\,\mathbb{F}$	$1\mathbb{G}$ $d\log n\mathsf{H}+\mathsf{H}_{\mathsf{in}}$ $O(d\log n)\mathbb{F}$	$2\mathbb{G}$ $\mathsf{H}_{\mathsf{in}}+O_\lambda(1)\mathsf{H}+1\mathsf{H}_{\mathbb{G}}$
extra P recursive w/ lookup	$1\mathsf{H}$	$O(\log T)\,\mathsf{H}$ $O(\ell_{\mathsf{lk}}\log T)\,\mathbb{F}$	N/A

Table 5.1: The comparison between IVCs.

Lookups, linear and independent of table size. ProtoStar supports lookup gates that ensure a value is in some precomputed table T. In each computation step, the prover commits to 2 vectors of length ℓ_{lk}, where ℓ_{lk} is the number of lookups. The prover, in each step, is independent of the table size (assuming free indexing in T). We also support tables that store tuples of size v using 1 additional challenge computations within the recursive circuit.

Our protocols are built of multiple small building blocks. In the protocol for high-degree gates, the prover simply sends the witness, and the degree d verifier checks the circuit with degree d gates. For lookup, we leverage an insight by Haböck [Hab22b] on logarithmic derivates. This yields a protocol where a prover performing ℓ_{lk} in a table of size T only needs to commit to two vectors of length ℓ_{lk}, independent of T. This is the most efficient lookup protocol today. While the verification is linear time, it is low degree (2) and thus compatible with our generic compiler. Combining all these yields ProtoStar, a new IVC-scheme for $\mathcal{R}_{\text{mplkup}}$. We compare ProtoStar, with SuperNova [KS22] and HyperNova [KS23], in Table 5.1 (for more detail see Corollary 5.4): In the table, $|\mathbf{w}|$ is the number of non-zero entries of the witness for circuit i, and ℓ_{lk} is the number of lookups in a table of size T. $\mathbb{G}$ is the cost of a group scalar multiplication. $\mathbb{F}$ is the cost of a field multiplication. $d\,\mathsf{H}$ denotes the cost of hashing d λ-bit numbers. We assume that the cost scales linearly with the size of the input and output. In ProtoStar d field elements are hashed once and in HyperNova d field elements are hashed $\log(n)$ times. $\mathsf{H}_{\mathbb{G}}$ is the cost of a hash-to-group function. H_{in} is the cost of hashing the public input and the accumulator instance. Note that the $O_\lambda(1)\,\mathsf{H}$ in SuperNova's recursive circuit involves constant number of hashes to the input of two accumulator instances and one circuit verification key, by using multiset-based offline memory checking in a circuit [Set+18].

Additional related work

Traditionally, IVC has been built by using recursive SNARKs[Bit+13a; Ben+14b; COS20]. SNARKs are succinct arguments that allow a prover to convince an efficient verifier that a single computation was performed correctly. There have been significant advances in SNARKs recently,

lowering the proof size time[Gro16], reducing the trust assumptions[Mal+19; Chi+20; Set20], and making the computational model more flexible [GWC19; Che+22].

In the IVC from SNARK construction, the prover constructs a SNARK that proves a step of the computation and that the previous SNARK was valid, i.e., a verifier would have accepted it. This requires an expensive NP-reduction of the SNARK verifier into the computational model of the SNARK, e.g. a circuit. This necessitates many compromises when implementing these schemes in practice, such as significant prover overhead, use of cycles of elliptic curves, strong cryptographic assumptions, and significant implementation overheads.

Concurrent work. In a paper concurrent with this work, Kothapalli and Setty [KS23] introduce an IVC for high degree relations. They use a generalization of R1CS called customizable constraint systems (CCS) [STW23] that covers the Plonkish relations. It also enables gates with a high additive fan-in. ProtoStar also has no restriction to the fan-in an individual gate has, but we subsequently showed that our compiler can also be directly applied to CCS (Section 5.8). HyperNova is based on so-called multi-folding schemes. They also provide a lookup argument suitable for recursive arguments. However, they do not explicitly explain how to integrate lookup to Plonk/CCS in their IVC scheme or provide any explicit constructions for non-uniform computations. Their scheme is built using sumchecks [Lun+92] and the resulting IVC recursive circuit is dominated by 1 group scalar multiplication, $d \log n + \ell_{\mathsf{in}}$ hash operations and $O(d \log n + \ell_{\mathsf{in}})$ field multiplications where d is the custom gate degree, n is the number of gates and ℓ_{in} is the public input length. In comparison, our IVC recursive circuit, even with lookup and non-uniformity support, is only dominated by 3 group scalar multiplications and $O(\ell_{\mathsf{in}} + d)$ field/hash operations, entirely independent of n. The 2 additional group operations compared to HyperNova are likely offset by the additional lookup support [Xio+22] and the significantly fewer hashes and non-native field operations (d vs. $d \log(n)$). A detailed comparison is given in Table 5.1.

For a lookup relation with table size T and ℓ_{lk} lookup gates, their accumulation/folding scheme leads to an accumulation prover whose work is dominated by $O(T)$ field operations and an accumulation verifier whose work is dominated by $O(\ell_{\mathsf{lk}} \log T)$ field operations and $O(\log T)$ hashes. This is undesirable when the table size $T \gg \ell_{\mathsf{lk}}$. In comparison, our scheme has prover complexity $O(\ell_{\mathsf{lk}})$ and the verifier is only dominated by 3 group scalar multiplications, 2 hashes and 2 field multiplications. Moreover, the lookup support adds almost no overhead to the IVC scheme for high-degree Plonk relations. In particular, it adds no group scalar multiplications. Lastly, their lookup scheme does not support vector-valued lookups, which is essential for applications like ZK-EVM and encoding bit-wise operations in circuits.

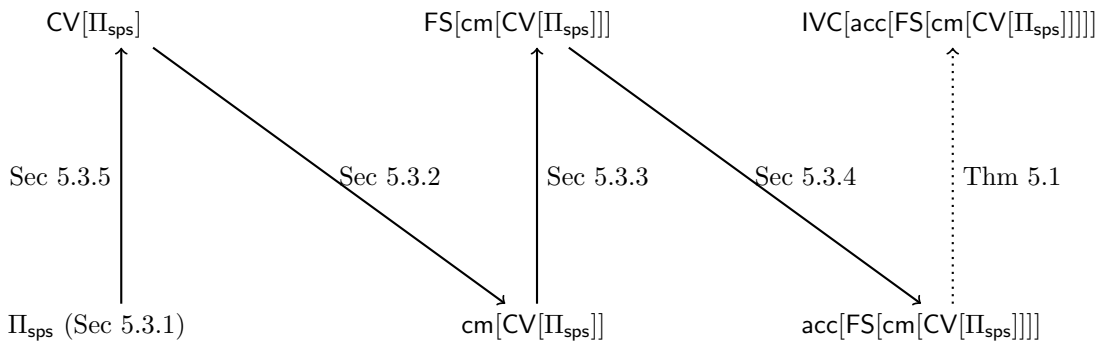


Figure 5.1: The workflow for building an IVC from a special sound protocol. We start from a special-sound protocol Π_{sps} for an NP-complete relation $\mathcal{R}_{\mathsf{NP}}$, and transform it to $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ with a compressed verifier check. $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ is converted to a NARK $\mathsf{FS}[\mathsf{cm}[\mathsf{CV}[\Pi_{\mathsf{sps}}]]]$ via commit-and-open and the Fiat-Shamir transform. We then build a generic accumulation scheme for the NARK and apply Theorem 5.1 from [Bün+21a] to obtain the IVC scheme. This last connection is dotted as it requires heuristically replacing random oracles with cryptographic hash functions.

5.1.1 Technical overview

Given an NP-complete relation $\mathcal{R}$, we introduce a generic framework for constructing efficient incremental verifiable computation (IVC) schemes with predicates expressed in $\mathcal{R}$. For $\mathcal{R}$ being the non-uniform Plonkup circuit satisfiability relation, we obtain an efficient (non-uniform) IVC scheme for proving correct program executions on stateful machines (e.g., EVM). The framework starts by designing a simple special-sound protocol Π_{sps} for relation $\mathcal{R}$, which is easy to analyze. Next, we use a generic compiler to transform Π_{sps} into a Non-interactive Argument of Knowledge Scheme (NARK) whose verification predicate is easy to accumulate/fold. Finally, we build an efficient accumulation/folding scheme for the NARK verifier, and apply the generic compiler from [Bün+21a] to obtain the IVC/PCD scheme for relation $\mathcal{R}$. We describe the workflow in Figure 5.1.

The paper begins by describing the compiler from special-sound protocols to NARKs in Section 5.3, and presents an efficient accumulation scheme for the compiled NARK verifier in Section 5.3.4. Next, we describe simple and efficient special-sound protocols for Plonkup circuit-satisfiability relationsin Section 5.5 and extend it to support non-uniform computation in Section 5.6. Similarly, we extend the CCS relation [STW23] to support non-uniform computation and lookup in Section 5.8. We give an overview of our approach below.

Efficient IVCs from special-sound protocols. Let Π_{sps} be any *multi-round* special-sound protocol for some relation $\mathcal{R}$, in which the verifier is *algebraic*, that is, the verifier algorithm only checks algebraic equations over the input and the prover messages. E.g., the following naive protocol for the Hadamard product relation over vectors $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{F}^n$ is special-sound and has a degree-2 algebraic verifier: The prover simply sends the vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ to the verifier, and the verifier checks that

$a_i \cdot b_i = c_i$ for all $i \in [n]$. However, as shown in the example, the prover message can be large in Π_{sps} and the folding scheme can be expensive if we directly accumulate the verifier predicate. Inspired by the splitting accumulation scheme [Bün+21a], to enable efficient accumulation/folding, we split each prover message into a short instance and a large opening, where the short instance is built from the homomorphic commitment to the prover message. Next, we use the Fiat-Shamir transform to compile the protocol into a NARK where the verifier challenges are generated from a random oracle.

Now we can view the NARK transcript as an accumulator (or a relaxed NP instance-witness pair in the language of folding schemes), where the accumulator instance consists of the prover message commitments and the verifier challenges; while the accumulator witness consists of the prover messages (i.e., the opening to the commitments). Note we also need to introduce an error vector/commitment into the accumulator witness/instance to absorb the "noise" that arises after each accumulation/folding step.

In the accumulation scheme, given two accumulators (or NARK proofs), the prover folds the witnesses and the instances of both accumulators via a random linear combination and generates a list of d "error-correcting terms" as accumulation proof (d is the degree of the NARK verifier); the verifier only needs to check that the folded accumulator instance is consistent with the accumulation proof and the original instances being folded, both of which are small. After finishing all the accumulation steps, a decider applies a final check to the accumulator, scrutinizing that (i) the accumulator witness is consistent with the commitments in the accumulator instance, and (ii) the "relaxed" NARK verifier check still passes. Here by "relaxed" we mean that the algebraic equation also involves the error vector in the accumulator. If the decider accepts, this implies that all accumulated NARKs were valid and thus that all accumulated statements are in $\mathcal{R}$ (and the prover knows witnesses for these statements).

Finally, given the accumulation scheme, if the relation $\mathcal{R}$ is NP-complete, we can apply the compiler in [Bün+21a] to obtain an efficient IVC scheme with predicates expressed in $\mathcal{R}$.

In Theorem 5.3, we show that for any $(2k-1)$-move[3] special-sound protocols with degree-d verifiers, the resulting IVC recursive circuit only involves $k+d+O(1)$ hashes, $k+1$ non-native field operations and $k+d-1$ commitment group scalar multiplications. We also introduce a generic approach for further reducing the number of group operations to $k+2$ in Section 5.3.5. This is favorable for $d \geq 3$. The idea is to compress all ℓ degree d verification checks into a single verification check using a random linear combination with powers of a challenge β. This means that error-correcting terms are field elements and, thus, can be sent directly without committing to them. The prover also sends a single commitment to powers of β and powers of $\beta^{\sqrt{\ell}}$. The verification equation uses one power of β and one power of $\beta^{\sqrt{\ell}}$, which increases the degree of the verification check to $d+2$. The verifier also checks the correctness of the powers of β using $2\sqrt{\ell}$ degree 2 checks.

[3] k prover messages, $k-1$ challenges

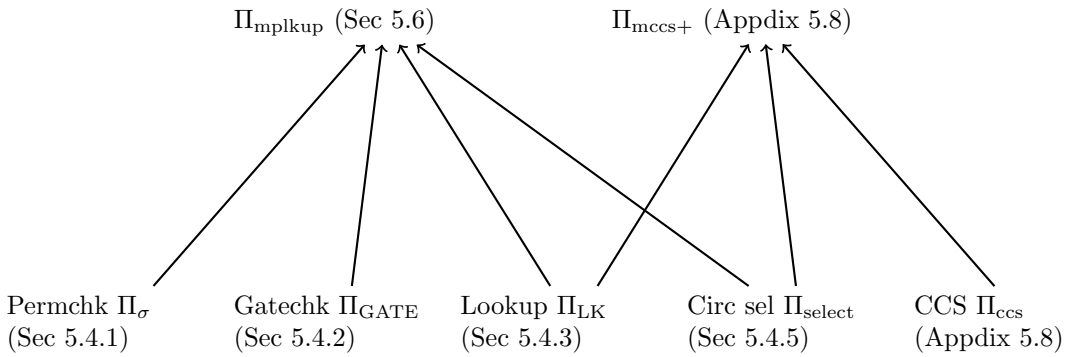


Figure 5.2: The special-sound protocols for ProtoStar and ProtoStar$_{\text{ccs}}$. The special-sound protocol Π_{mplkup} for the multi-circuit Plonkup relation $\mathcal{R}_{\text{mplkup}}$ consists of the sub-protocols for permutation, high-degree custom gate, lookup, and circuit selection relations. The special-sound protocol $\Pi_{\text{mccs+}}$ for the extended CCS relation $\mathcal{R}_{\text{mccs+}}$ consists of the sub-protocols for lookup, circuit selection, as well as the CCS relation [STW23]. From Π_{mplkup} or $\Pi_{\text{mccs+}}$, we can apply the workflow described in Fig 5.1 to obtain the IVC schemes ProtoStar or ProtoStar$_{\text{ccs}}$.

Special-sound protocols for (non-uniform) Plonkup relations. Given the generic compiler above, our ultimate goal of constructing a (non-uniform) IVC scheme for zkEVM becomes much easier. It is now sufficient to design a multi-round special-sound protocol for the (non-uniform) Plonkup relation. We describe the components of the special-sound protocol in Figure 5.2. Note we also extend CCS relation [STW23] to support lookup and non-uniform computation and build a special-sound protocol for it (See Figure 5.2). Recall that a Plonkup circuit-satisfiability relation consists of three modular relations, namely, (i) a high-degree gate relation checking that each custom gate is satisfied; (ii) a permutation (wiring-identity) relation checking that different gate values are consistent if the same wire connects them, and (iii) a lookup relation checking that a subset of gate values belongs to a preprocessed table. The special-sound protocols for the permutation and high-degree gate relations are trivial, where the prover directly sends the witness to the verifier, and the verifier checks that the permutation/high-degree gate relation holds. The degree of the permutation check is only 1, and the degree of the gate-check is the highest degree in the custom gate formula.

The special-sound protocol for the lookup relation $\mathcal{R}_{\text{LOOKUP}}$ is more interesting as the statement of the lookup relation is not algebraic. Inspired by the log-derivative lookup scheme [Hab22b], in Section 5.4.3, we design a simple 3-move special-sound protocol Π_{LK} for $\mathcal{R}_{\text{LOOKUP}}$, in which the verifier degree is only 2. A great feature of Π_{LK} is that the number of non-zero elements in the prover messages is only proportional to the number of lookups, but independent of the table size. Thus the IVC prover complexity for computing the prover message commitments is independent of the table size, which is advantageous when the table size is much larger than the witness size. However, the prover work for computing the error terms is not independent of the table size because the accumulator is not sparse. Fortunately, we observe that the prover can efficiently update the

error term commitments without recomputing the error term vectors from scratch, thus preserving the efficiency of the accumulation prover. Moreover, we extend Π_{LK} in Section 5.4.4 to further support vector-valued lookup, where each table entry is a vector of elements. This feature is useful in applications like zkEVM and for simulating bit operations in circuits.

Given the special-sound protocols for permutation/high-degree gate/lookup relations, the special-sound protocol Π_{plonkup} for Plonkup is just a parallel composition of the three protocols. Furthermore, in Section 5.6, we apply a simple trick to support *non-uniform* IVC. More precisely, let $\{\mathcal{C}_i\}_{i=1}^{I}$ be I different branch circuits (e.g., the set of supported instructions in EVM), let $\mathsf{pi} := (pc, \mathsf{pi}')$ be the public input where $pc \in [I]$ is a program counter indicating which instruction/branch circuit is going to be executed in the next IVC step. Our goal is to prove that $(\mathsf{pi}, \mathbf{w})$ is in the relation $\mathcal{R}_{\mathrm{mplkup}}$ in the sense that $\mathcal{C}_{pc}(\mathsf{pi}, \mathbf{w}) = 0$ for witness $\mathbf{w}$. The relation statement can also add additional constraints on pc depending on the applications. The special-sound protocol for $\mathcal{R}_{\mathrm{mplkup}}$ is almost identical to Π_{plonkup} for the Plonkup relation, except that the prover further sends a bool vector $\mathbf{b} \in \mathbb{F}^I$, and the verifier uses $2I$ degree 2 equations to check that $b_{pc} = 1$ and $b_i = 0 \forall i \neq pc$. Additionally, each algebraic equation $\mathcal{G}$ checked in Π_{plonkup} is replaced with $\sum_{i=1}^{I} \mathcal{G}_i \cdot b_i$ where $\mathcal{G}_i$ $(1 \leq i \leq I)$ is the corresponding gate in the i-th branch circuit. The resulting special-sound protocol has 3 moves, and the verifier degree is $d+1$, where d is the highest degree of the custom gates. This means that the IVC scheme for the non-uniform Plonkup relation adds negligible overhead to that for the Plonkup relation.

5.2 Preliminaries

5.2.1 Special-sound Protocols and Fiat-Shamir Transform

We define special-soundness and non-interactive arguments according to the definitions by [AFK22].

Definition 5.1 (Public-coin interactive proof)**.** *An interactive proof* $\Pi = (\mathsf{P}, \mathsf{V})$ *for relation* $\mathcal{R}$ *is an interactive protocol between two probabilistic machines, a prover* P*, and a polynomial time verifier* V*. Both* P *and* V *take as public input a statement* pi *and, additionally,* P *takes as private input a witness* $\mathbf{w} \in \mathcal{R}(\mathsf{pi})$ *. The verifier* V *outputs* 0 *if it accepts and a non-zero value otherwise. It's output is denoted by* $(\mathsf{P}(\mathbf{w}), \mathsf{V})(\mathsf{pi})$*. Accordingly, we say the corresponding transcript (i.e., the set of all messages exchanged in the protocol execution) is accepting or rejecting. The protocol is* public coin *if the verifier randomness is public. The verifier messages are referred to as challenges.* Π *is a* $(2k-1)$*-move protocol if there are* k *prover messages and* $k-1$ *verifier messages.*

Definition 5.2 (Tree of transcript)**.** *Let* $\mu \in \mathbb{N}$ *and* $(a_1, \ldots, a_\mu) \in \mathbb{N}^\mu$*. An* $(a_1, \ldots, a_\mu)$*-tree of transcript for a* $(2\mu+1)$*-move public-coin interactive proof* Π *is a set of* $a_1 \cdot a_2 \cdots a_\mu$ *accepting transcripts arranged in a tree of depth* μ *and arity* $a_1, \ldots, a_\mu$ *respectively. The nodes in the tree correspond to the prover messages and the edges to the verifier's challenges. Every internal node at*

depth $i-1$ ($1 \leq i \leq \mu$) has a_i children with distinct challenges. Every transcript corresponds to one path from the root to a leaf node. We simply write the transcripts as an (a^μ)-tree of transcript when $a = a_1 = a_2 = \cdots = a_\mu$.

Definition 5.3 (Special-sound Interactive Protocol)**.** *Let $\mu, N \in \mathbb{N}$ and $(a_1, \ldots, a_\mu) \in \mathbb{N}^\mu$. A $(2\mu+1)$-move public-coin interactive proof Π for relation $\mathcal{R}$ where the verifier samples its challenges from a set of size N is $(a_1, \ldots, a_\mu)$-out-of-N special-sound if there exists a polynomial time algorithm that, on input pi and any $(a_1, \ldots, a_\mu)$-tree of transcript for Π outputs $\mathbf{w} \in \mathcal{R}(\mathsf{pi})$. We simply denote the protocol as an a^μ-out-of-N (or a^μ) special-sound protocol if $a = a_1 = a_2 = \cdots = a_\mu$.*

Definition 5.4 (Random-Oracle Non-Interactive Argument of Knowledge (RO-NARK))**.** *A non-interactive random oracle proof for relation $\mathcal{R}$ is a pair (P, V) of probabilistic random-oracle algorithms, such that: Given $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}$ and access to a random oracle ρ_{NARK}, the prover $\mathsf{P}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \mathbf{w})$ outputs a proof π. Given pi, a proof π, and access to the same random oracle ρ_{NARK}, the verifier $\mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi)$ outputs 0 to accept or any other value to reject.*

Perfect Completeness: *The NARK has perfect completeness if for all $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}$*

$$P[\mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \mathsf{P}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \mathbf{w})) = 0] = 1$$

Knowledge Soundness: *The NARK has adaptive knowledge-soundness with knowledge error $\kappa : \mathbb{N} \times \mathbb{N} \to [0, 1]$ if there exists a knowledge extractor Ext, with the following properties: The extractor, given input n, and oracle-access to any polynomial-time Q-query random oracle prover P^* that outputs statement of size n, runs in an expected polynomial time in $|\mathsf{pi}| + Q$, and outputs $\{(\mathsf{pi}, \pi, \mathsf{aux}, v; \mathbf{w})\}$ such that a) $(\mathsf{pi}, \pi, \mathsf{aux}, v)$ is identically distributed to $\{(\mathsf{pi}, \pi, \mathsf{aux}, v)\} : (\mathsf{pi}, \pi, \mathsf{aux}) \leftarrow \mathsf{P}^{*, \rho_{\mathsf{NARK}}}, v \leftarrow \mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi)$ and b)*

$$\Pr\left[\begin{array}{c}(\mathsf{pi}; \mathbf{w}) \in \mathcal{R} \\ \mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi) = 0\end{array} : \{(\mathsf{pi}, \pi, \mathsf{aux}, v; \mathbf{w})\} \leftarrow \mathsf{Ext}^{\mathsf{P}^*}\right] \geq \frac{\epsilon(\mathsf{P}^*) - \kappa(n, Q)}{\mathsf{poly}(n)},$$

where $\epsilon(\mathsf{P}^*)$ is P^*'s success probability, i.e. $\epsilon(\mathsf{P}^*) = P[\mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi) = 0 : (\mathsf{pi}, \pi) \leftarrow \mathsf{P}^{*, \rho_{\mathsf{NARK}}}]$. Here, Ext implements ρ_{NARK} for P^*; in particular, it can arbitrarily program the random oracle.

Definition 5.5 (Fiat-Shamir Transform (adaptive))**.** *The Fiat-Shamir transform $\mathsf{FS}[\Pi] = (\mathsf{P}_{\mathsf{fs}}, \mathsf{V}_{\mathsf{fs}})$ is a RO-NARK, where $\mathsf{P}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}; \mathbf{w})$ runs $\mathsf{P}(\mathsf{pi}; \mathbf{w})$ but instead of receiving challenge c_i, on message m_i, from the verifier, it computes it as follows:*

$$c_i = \rho_{\mathsf{NARK}}(c_{i-1}, m_i) \tag{5.1}$$

and $c_0 = \rho_{\mathsf{NARK}}(\mathsf{pi})$. $\mathsf{P}_{\mathsf{fs}}^{\rho_{\mathsf{NARK}}}$ outputs $\pi = (m_1, \ldots, m_\mu)$. The verifier $\mathsf{V}_{\mathsf{fs}}^{\rho_{\mathsf{NARK}}}$ accepts, if V accepts the transcript $(m_1, c_1, \ldots, m_\mu, c_\mu, m_{\mu+1})$ for input pi and the challenges are computed as per equation

(5.1).

5.2.2 Adaptive Fiat-Shamir transform

Lemma 5.1 (Fiat-Shamir transform of Special-sound Protocols [AFK22])**.** *The Fiat-Shamir transform of a $(\alpha_1, \ldots, \alpha_\mu)$-out-of-$N$ special-sound interactive proof Π is knowledge sound with knowledge error*

$$\kappa_{\mathsf{fs}}(Q) = (Q+1)\kappa$$

where $\kappa = 1 - \prod(1 - \frac{\alpha_i}{N})$ is the knowledge error of the interactive proof Π.

5.2.3 Commitment Scheme

Definition 5.6 (Commitment Scheme)**.** $\mathsf{cm} = (\mathsf{Setup}, \mathsf{Commit})$ *is a binding commitment scheme, consisting of two algorithms:*
$\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ *takes as input the security parameter and outputs a commitment key* ck*.*
$\mathsf{Commit}(\mathsf{ck}, \mathbf{m} \in \mathcal{M}) \to C \in \mathcal{C}$*, takes as input the commitment key* ck *and a message* $\mathbf{m}$ *in* $\mathcal{M}$ *and outputs a commitment* $C \in \mathcal{C}$*.*
The scheme is binding if for all polynomial-time randomized algorithms P^**:*

$$\Pr\left[\begin{array}{c|c} \mathsf{Commit}(\mathsf{ck}, \mathbf{m}) = \mathsf{Commit}(\mathsf{ck}, \mathbf{m}') & \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge & \mathbf{m}, \mathbf{m}' \leftarrow \mathsf{P}^*(\mathsf{ck}) \\ \mathbf{m} \neq \mathbf{m}' & \end{array}\right] = \mathsf{negl}(\lambda)$$

Homomorphic commitment. *We say the commitment is homomorphic if $(\mathcal{C}, +)$ is an additive group of prime order p.*

5.2.4 Incremental Verifiable Computation (IVC)

We adapt and simplify the definition from [Bün+21a; KST22].

Definition 5.7 (IVC)**.** *An incremental verifiable computation (IVC) scheme for function predicates expressed in a circuit-satisfiability relation $\mathcal{R}_{\mathsf{NP}}$ is a tuple of algorithms $\mathsf{IVC} = (\mathsf{P}_{\mathsf{IVC}}, \mathsf{V}_{\mathsf{IVC}})$ with the following syntax and properties:*

- $\mathsf{P}_{\mathsf{IVC}}(m, z_0, z_m, z_{m-1}, \mathbf{w}_{\mathsf{loc}}, \pi_{m-1}]) \to \pi_m$*. The IVC prover* $\mathsf{P}_{\mathsf{IVC}}$ *takes as input a program output z_m at step m, local data $\mathbf{w}_{\mathsf{loc}}$, initial input z_0, previous program output z_{m-1} and proof π_{m-1} and outputs a new IVC proof π_m.*
- $\mathsf{V}_{\mathsf{IVC}}(m, z_0, z_m, \pi_m) \to b$*. The IVC verifier* $\mathsf{V}_{\mathsf{IVC}}$ *takes the initial input z_0, the output z_m at step m, and an IVC proof π_m, 'accepts' by outputting $b = 0$ and 'rejects' otherwise.*

The scheme IVC *has perfect adversarial completeness if for any function predicate* ϕ *expressible in* $\mathcal{R}_{\mathsf{NP}}$*, and any, possibly adversarially created,* $(m, z_0, z_m, , z_{m-1}, \mathbf{w}_{\mathsf{loc}}, \pi_{m-1})$ *such that*

$$\phi(z_0, z_m, z_{m-1}, \mathbf{w}_{\mathsf{loc}}) \wedge (\mathsf{V}_{\mathsf{IVC}}(m-1, z_0, z_{m-1}, \pi_{m-1}) = 0)$$

it holds that $\mathsf{V}_{\mathsf{IVC}}(m, z_0, z_m, \pi_m)$ *accepts for proof* $\pi_m \leftarrow \mathsf{P}_{\mathsf{IVC}}(m, z_0, z_{m-1}, z_m, \mathbf{w}_{\mathsf{loc}}, \pi_{m-1})$.

The scheme IVC *has knowledge soundness if for every expected polynomial-time adversary* P^**, there exists an expected polynomial-time extractor* $\mathsf{Ext}_{\mathsf{P}^*}$ *such that*

$$\Pr\left[\begin{array}{c} \mathsf{V}_{\mathsf{IVC}}(m, z_0, z, \pi_m) = 0 \wedge \\ ([\exists i \in [m], \neg\phi(z_0, z_i, z_{i-1}, \mathbf{w}_i)] \\ \vee z \neq z_m) \end{array} \middle| \begin{array}{c} [\phi, (m, z_0, z, \pi_m)] \leftarrow \mathsf{P}^* \\ [z_i, \mathbf{w}_i]_{i=1}^m \leftarrow \mathsf{Ext}_{\mathsf{P}^*} \end{array}\right] \leq \mathsf{negl}(\lambda)\,.$$

Here m *is a constant.*

5.2.5 Simple Accumulation

We take definitions and proofs from [Bün+21a].

Definition 5.8 (Accumulation Scheme)**.** *An accumulation scheme for a NARK* $(\mathsf{P}_{NARK}, \mathsf{V}_{NARK})$ *is a triple of algorithms* $\mathsf{acc} = (\mathsf{P}_{\mathsf{acc}}, \mathsf{V}_{\mathsf{acc}}, D)$*, all of which have access to the same random oracle* ρ_{acc} *as well as* ρ_{NARK}*, the oracle for the NARK. The algorithms have the following syntax and properties:*

- $\mathsf{P}_{\mathsf{acc}}(\mathsf{pi}, \pi = (\pi.x, \pi.\mathbf{w}), \mathsf{acc} = (\mathsf{acc}.x, \mathsf{acc}.\mathbf{w})) \rightarrow \{\mathsf{acc}' = (\mathsf{acc}'.x, \mathsf{acc}'.\mathbf{w}), \mathsf{pf}\}$*. The accumulation prover* $\mathsf{P}_{\mathsf{acc}}$ *takes as input a statement* pi*, NARK proof* π*, and an accumulator* acc *and outputs a new accumulator* acc' *and correction terms* pf*.*
- $\mathsf{V}_{\mathsf{acc}}(\mathsf{pi}, \pi.x, \mathsf{acc}.x, \mathsf{acc}'.x, \mathsf{pf}) \rightarrow v$*. The accumulation verifier takes as input the statement* pi*, the instances of the NARK proof, the old and new accumulator, the correction terms, and 'accepts' by outputting* 0 *and 'rejects' otherwise.*
- $D(\mathsf{acc}) \rightarrow v$*. The decider on input* acc *'accepts' by outputting* 0 *and 'rejects' otherwise.*

An accumulation scheme has knowledge-soundness with knowledge error κ *if the RO-NARK* $(\mathsf{P}', \mathsf{V}')$ *has knowledge error* κ *for the relation*

$$\mathcal{R}_{\mathsf{acc}}((\mathsf{pi}, \pi.x, \mathsf{acc}.x); (\pi.\mathbf{w}, \mathsf{acc}.\mathbf{w})) : (\mathsf{V}_{NARK}(\mathsf{pi}, \pi) = 0 \wedge D(\mathsf{acc}) = 0)\ ,$$

where P' *outputs* $\mathsf{acc}', \mathsf{pf}$ *and* V' *on input* $((\mathsf{pi}, \pi.x, \mathsf{acc}.x), (\mathsf{acc}', \mathsf{pf}))$ *accepts if* $D(\mathsf{acc}')$ *and* $\mathsf{V}_{\mathsf{acc}}(\mathsf{pi}, \pi.x, \mathsf{acc}.x, \mathsf{acc}'.x, \mathsf{pf})$ 0.

The scheme has perfect completeness if the RO-NARK $(\mathsf{P}', \mathsf{V}')$ *has perfect completeness for* $\mathcal{R}_{\mathsf{acc}}$*.*

Theorem 5.1 (IVC from accumulation[Bün+21a])**.** *Given a standard-model NARK for circuit-satisfiability and a standard-model accumulation scheme (Definition 5.8) for that NARK, both with negligible knowledge error, there exists an efficient transformation that outputs an IVC scheme (see Section 3.2 of [Bün+21a]) for constant-depth compliance predicates, assuming that the circuit complexity of the accumulation verifier* $\mathsf{V}_{\mathsf{acc}}$ *is sub-linear in its input.*

Random Oracle. Note that both the NARK and accumulation scheme we construct are in the random oracle model. However, Theorem 5.1 requires a NARK and an accumulation scheme in the standard model. It remains an open problem to construct such schemes. However, we can heuristically instantiate the random oracle with a cryptographic hash function and assume that the resulting schemes still have knowledge soundness.

Definition 5.9 (Fiat-Shamir Heuristic)**.** *The Fiat-Shamir Heuristic, relative to a secure cryptographic hash function* H*, states that a random oracle NARK with negligible knowledge error yields a NARK that has negligible knowledge error in the standard (CRS) model if the random oracle is replaced with* H*.*

Complexity. The IVC transformation from [Bün+21a] recursively proves that the accumulation was performed correctly. To do that, it implements $\mathsf{V}_{\mathsf{acc}}$ as a circuit and proves that the previous accumulation step was done correctly. Note that this recursive circuit is independent of the size of $\pi.\mathbf{w}, \mathsf{acc}.\mathbf{w}$ and the runtime of D. The IVC prover is linear in the size of the recursive circuit plus the size of the IVC computation step expressed as a circuit. The final IVC verifier and the IVC proof size are linear in these components. This can be reduced using an additional SNARK as in [KST22].

PCD. IVC can be generalized to arbitrary DAGs instead of just path graphs in a primitive called proof-carrying data[Bit+13a]. Accumulation schemes can be compiled into full PCD if they support accumulating an arbitrary number of accumulators and proofs[Bün+20; Bün+21a]. For simplicity, we only build accumulation for one proof and one accumulator, as well as for two accumulators. This enables PCD for DAGs of degree two. By transforming higher degree graphs into degree two graphs (by converting each degree d node into a $\log_2(d)$ depth tree), we can achieve PCD for these graphs.

Outsourcing the decider. In the accumulation to IVC transformation, the IVC proof is linear in the accumulator, and the IVC verifier runs the decider. The accumulation schemes we construct are linear in the witness of a single computation step. However, we can outsource the decider by providing a SNARK that, given $\mathsf{acc}.x$, proves knowledge of $\mathsf{acc}.\mathbf{w}$, such that $D(\mathsf{acc}) = 0$. Nova[KST22] constructs a custom, concretely efficient SNARK for their accumulation/folding scheme. However,

when outsourcing the decider, the IVC cannot continue. This breaks the strict completeness requirement of IVC, which says that any prover can continue from any valid IVC proof. Nevertheless, this may be fine for some applications of IVC.

5.3 Protocols

5.3.1 Special-sound Protocols

In this section, we describe a class of special-sound protocols whose verifier is algebraic. The protocol Π_{sps} has 3 essential parameters $k, d, \ell \in \mathbb{N}$, meaning that Π_{sps} is a $(2k-1)$-move protocol with verifier degree d and output length ℓ (i.e. the verifier checks ℓ degree d algebraic equations). In each round i $(1 \leq i \leq k)$, the prover $\mathsf{P}_{\mathsf{sps}}(\mathsf{pi}, \mathbf{w}, [\mathbf{m}_j, r_j]_{j=1}^{i-1})$ generates the next message $\mathbf{m}_i$ on input the public input pi, the witness $\mathbf{w}$, and the current transcript $[\mathbf{m}_j, r_j]_{j=1}^{i-1}$, and sends $\mathbf{m}_i$ to the verifier; the verifier replies with a random challenge $r_i \in \mathbb{F}$. After the final message $\mathbf{m}_k$, the verifier computes the algebraic map $\mathsf{V}_{\mathsf{sps}}$ and checks that the output is a zero vector of length ℓ. More precisely, $\deg(\mathsf{V}_{\mathsf{sps}}) = d$, s.t.

$$\mathsf{V}_{\mathsf{sps}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) := \sum_{j=0}^{d} f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}),$$

where $f_j^{\mathsf{V}_{\mathsf{sps}}}$ is a homogeneous degree-j algebraic map that outputs a vector of ℓ field elements.

We describe the special-sound protocol Π_{sps} below.

Special-sound Protocol $\Pi_{\mathsf{sps}} = (\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}})$ for relation $\mathcal{R}$ with algebraic verifier

Prover $\mathsf{P}_{\mathsf{sps}}(\mathsf{pi}, \mathbf{w})$		**Verifier** $\mathsf{V}_{\mathsf{sps}}(\mathsf{pi})$
$\mathbf{m}_i \leftarrow \mathsf{P}_{\mathsf{sps}}(\mathsf{pi}, \mathbf{w}, [\mathbf{m}_j, r_j]_{j=1}^{i-1})$	$\xrightarrow{\mathbf{m}_i}$	
	$\xleftarrow{r_i}$	$r_i \leftarrow\!\!\$\ \mathbb{F}$
Repeat $k-1$ time		Repeat $k-1$ time
	$\xrightarrow{\text{Final message } \mathbf{m}_k}$	
		$\mathsf{V}_{\mathsf{sps}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) \stackrel{?}{=} \mathbf{0}^{\ell}$

5.3.2 Commit and Open

For a commitment scheme $\mathsf{cm} = (\mathsf{Setup}, \mathsf{Commit})$, consider the following relation $\mathcal{R}_{\mathsf{cm}}^{\mathcal{R}} = (x; \mathbf{w}, \mathbf{m} \in \mathcal{M}, \mathbf{m}' \in \mathcal{M}) : \{(x, \mathbf{w}) \in \mathcal{R} \vee (\mathsf{Commit}(\mathbf{m}) = \mathsf{Commit}(\mathbf{m}') \wedge \mathbf{m} \neq \mathbf{m}')\}$. The relation's witness is

either a valid witness for $\mathcal{R}$ or a break of the commitment scheme cm. We now design a special-sound protocol $\Pi_{\mathsf{cm}} = (\mathsf{P}_{\mathsf{cm}}, \mathsf{V}_{\mathsf{cm}})$ for $\mathcal{R}_{\mathsf{cm}}^{\mathcal{R}}$ given $\Pi_{\mathsf{sps}} = (\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}})$, a special-sound protocol for $\mathcal{R}$. P_{cm} runs $\mathsf{P}_{\mathsf{sps}}$ to generate the ith message and then commits to the message. Along with the final message, P_{cm} sends the opening to the commitment. The verifier V_{cm} checks the correctness of the commitments and runs $\mathsf{V}_{\mathsf{sps}}$ on the commitment openings.

Special-sound Protocol Π_{cm} for $\mathcal{R}_{\mathsf{cm}}^{\mathcal{R}}$ as a cm transform of $\Pi_{\mathsf{sps}} = (\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}})$

Prover $\mathsf{P}_{\mathsf{cm}}(\mathsf{ck}, \mathsf{pi}, \mathbf{w})$		**Verifier** $\mathsf{V}_{\mathsf{cm}}(\mathsf{ck}, \mathsf{pi})$
$\mathbf{m}_i \leftarrow \mathsf{P}_{\mathsf{sps}}(\mathsf{pi}, \mathbf{w}, [\mathbf{m}_j, r_j]_{j=1}^{i-1})$		
$C_i \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$	$\xrightarrow{C_i}$	
	$\xleftarrow{r_i}$	$r_i \leftarrow\!\!\$\ \mathbb{F}$
Repeat $k-1$ time		Repeat $k-1$ time
$C_k \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_k)$	$\xrightarrow{C_k}$	
	$\xrightarrow{\text{Opening } [\mathbf{m}_i]_{i=1}^{k}}$	
		$\mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i) \stackrel{?}{=} C_i \forall i \in [k]$
		$\mathsf{V}_{\mathsf{sps}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) \stackrel{?}{=} \mathbf{0}^{\ell}$

Lemma 5.2 (Π_{cm} is $(a_1, \ldots, a_\mu)$-special-sound)**.** *Let Π_{sps} be an $(a_1, \ldots, a_\mu)$-out-of-N special-sound protocol for relation $\mathcal{R}$, where the prover messages are all in a set $\mathcal{M}$. Let $(\mathsf{Setup}, \mathsf{Commit})$ be a binding commitment scheme for messages in $\mathcal{M}$. For $\mathsf{ck} \leftarrow \mathsf{Setup}_{\mathsf{cm}}(1^\lambda)$ let $\mathcal{R}_{\mathsf{cm}} = (\mathsf{pi}; \mathbf{w}, m \in \mathcal{M}, m' \in \mathcal{M}) : (\mathsf{pi}; \mathbf{w}) \in \mathcal{R} \vee (\mathsf{Commit}(\mathsf{ck}, m) = \mathsf{Commit}(\mathsf{ck}, m') \wedge m \neq m')$. Then $\Pi_{\mathsf{cm}} = \mathsf{cm}[\Pi_{\mathsf{sps}}]$ is an $(a_1, \ldots, a_\mu)$-out-of-N special-sound protocol for $\mathcal{R}_{\mathsf{cm}}^{\mathcal{R}}$.*

Proof. Let $\mathsf{Ext}_{\mathsf{sps}}$ be the extractor for Π_{sps}. We will construct $\mathsf{Ext}_{\mathsf{cm}}$ for Π_{cm} that computes a witness for $\mathcal{R}_{\mathsf{cm}}$, i.e., a witness for $\mathcal{R}$ or a collision for cm given an $(a_1, \ldots, a_\mu)$-transcript tree for Π_{cm}. The extractor $\mathsf{Ext}_{\mathsf{cm}}$ first checks whether there exist two transcripts that have inconsistent final messages. That is, the final message opening is different for the nodes in the intersection of the root-to-leaf paths of these two transcripts. This means we have $\mathbf{m}_i$ and $\mathbf{m}'_i$, such that $\mathsf{Commit}(\mathbf{m}_i) = \mathsf{Commit}(\mathbf{m}'_i)$ and $\mathbf{m}_i \neq \mathbf{m}'_i$. This is a break for cm, i.e., a valid witness for $\mathcal{R}_{\mathsf{cm}}$. Otherwise $\mathsf{Ext}_{\mathsf{cm}}$ builds a transcript tree for Π_{sps} by replacing all commitments with the openings and use $\mathsf{Ext}_{\mathsf{sps}}$ to compute $\mathbf{w} \in \mathcal{R}(\mathsf{pi})$, such that $(\mathbf{w}, \perp, \perp) \in \mathcal{R}_{\mathsf{cm}}(\mathsf{pi})$. □

5.3.3 Fiat-Shamir transform

Let ρ_{NARK} be a random oracle. Let Π_{cm} be the commit-and-open protocol for the special-sound protocol $\Pi_{\mathsf{sps}} = (\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}})$. The Fiat-Shamir Transform $\mathsf{FS}[\Pi_{\mathsf{cm}}]$ of the protocol Π_{cm} is the following. By Lemma 5.1, $\mathsf{FS}[\Pi_{\mathsf{cm}}]$ is knowledge sound if Π_{sps} is special-sound.

Fiat-Shamir Transform FS of Special-sound Protocol Π for relation $\mathcal{R}_{\mathsf{cm}}^{\mathcal{R}}$: $\mathsf{FS}[\Pi_{\mathsf{cm}}]$

Prover $\mathsf{P}_{\mathbf{NARK}}^{\rho_{\mathbf{NARK}}}(\mathsf{ck}, \mathsf{pi}, \mathbf{w})$		**Verifier** $\mathsf{V}_{\mathbf{NARK}}^{\rho_{\mathbf{NARK}}}(\mathsf{ck}, \mathsf{pi})$
$r_0 \leftarrow \rho_{\mathsf{NARK}}(\mathsf{pi})$		
For $i \in [k-1]$:		
$\mathbf{m}_i \leftarrow \mathsf{P}_{\mathsf{sps}}(\mathsf{pi}, \mathbf{w}, [\mathbf{m}_j, r_j]_{j=1}^{i-1})$		
$C_i \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$		
$r_i \leftarrow \rho_{\mathsf{NARK}}(r_{i-1}, C_i)$		
$\mathbf{m}_k \leftarrow \mathsf{P}_{\mathsf{sps}}(\mathsf{pi}, \mathbf{w}, [\mathbf{m}_j, r_j]_{j=1}^{k-1})$		
$C_k \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$	$\pi.x = [C_i]_{i=1}^k \longrightarrow$	
	$\pi.\mathbf{w} = [\mathbf{m}_i]_{i=1}^k \longrightarrow$	
		$r_0 \leftarrow \rho_{\mathsf{NARK}}(x)$
		$r_i \leftarrow \rho_{\mathsf{NARK}}(r_{i-1}, C_i) \forall i \in [k-1]$
		$\mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i) \stackrel{?}{=} C_i \forall i \in [k]$
		$\mathsf{V}_{\mathsf{sps}}(\mathsf{pi}, \pi.x, \pi.\mathbf{w}, [r_i]_{i=1}^{k-1}) \stackrel{?}{=} \mathbf{0}^{\ell}$

5.3.4 Accumulation Scheme for $\mathsf{V}_{\mathsf{NARK}}$

Let ρ_{acc} and ρ_{NARK} be two random oracles, and let $\mathsf{V}_{\mathsf{NARK}}$ be the verifier of $\mathsf{FS}[\Pi_{\mathsf{cm}}]$ in Section 5.3.3, whose underlying special-sound protocol is $\Pi_{\mathsf{sps}} = (\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}})$ for a relation $\mathcal{R}$. We describe the accumulation scheme for $\mathsf{V}_{\mathsf{NARK}}$.

The accumulated predicate. The predicate to be accumulated is the "relaxed" verifier check of the NARK scheme $\mathsf{FS}[\Pi_{\mathsf{cm}}]$ for relation $\mathcal{R}$. Namely, given public input $\mathsf{pi} \in \mathcal{M}^{\ell_{\mathsf{in}}}$, random challenges $[r_i]_{i=1}^{k-1} \in \mathbb{F}^{k-1}$, a NARK proof

$$\pi.x = [C_i]_{i=1}^k,\ \pi.\mathbf{w} = [\mathbf{m}_i]_{i=1}^k$$

where $[C_i]_{i=1}^k \in \mathcal{C}^k$ are commitments and $[\mathbf{m}_i]_{i=1}^k$ are prover messages in the special-sound protocol Π_{sps}, and a slack variable μ, the predicate checks that (i) $r_i = \rho_{\mathsf{NARK}}(r_{i-1}, C_i)$ for all $i \in [k-1]$

(where $r_0 := \rho_{\mathsf{NARK}}(\mathsf{pi})$), (ii) $\mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i) = C_i$ for all $i \in [k]$, and (iii)

$$\mathsf{V}_{\mathsf{sps}}(\mathsf{pi}, \pi.x, \pi.\mathbf{w}, [r_i]_{i=1}^{k-1}, \mu) := \sum_{j=0}^{d} \mu^{d-j} \cdot f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{pi}, \pi.\mathbf{w}, [r_i]_{i=1}^{k-1}) = \mathbf{e}$$

where $\mathbf{e} = \mathbf{0}^\ell$ and $\mu = 1$ for the NARK verifier $\mathsf{V}_{\mathsf{NARK}}$. Here $f_j^{\mathsf{V}_{\mathsf{sps}}}$ is a degree-j homogeneous algebraic map that outputs ℓ field elements. Degree-j homogeneity says that each monomial term of $f_j^{\mathsf{V}_{\mathsf{sps}}}$ has degree exactly j.

Remark 5.3.1. *Without loss of generality, we assume that the public input* pi *is of constant size, as otherwise, we can set it as the hash of the original public input.*

Accumulator. The accumulator has the following format:

- *Accumulator instance* $\mathsf{acc}.x := \{\mathsf{pi}, [C_i]_{i=1}^k, [r_i]_{i=1}^{k-1}, E, \mu\}$, where $\mathsf{pi} \in \mathcal{M}^{\ell_{\mathsf{in}}}$ is the accumulated public input, $[C_i]_{i=1}^k \in \mathcal{C}^k$ are the accumulated commitments, $[r_i]_{i=1}^{k-1} \in \mathbb{F}^{k-1}$ are the accumulated challenges, $E \in \mathcal{C}$ is the accumulated commitment to the error terms, and $\mu \in \mathbb{F}$ is a slack variable.
- *Accumulator witness* $\mathsf{acc}.\mathbf{w} := \{[\mathbf{m}_i]_{i=1}^k\}$, where $[\mathbf{m}_i]_{i=1}^k$ are the accumulated prover messages.

Accumulation prover. On input commitment key ck (which can be hardwired in the prover's algorithm), accumulator acc, an instance-proof pair (pi, π) where

$$\begin{aligned} \mathsf{acc} &:= (\mathsf{acc}.x = \{\mathsf{pi}', [C_i']_{i=1}^k, [r_i']_{i=1}^{k-1}, E, \mu\}, \mathsf{acc}.\mathbf{w} = \{[\mathbf{m}_i']_{i=1}^k\}), \\ \pi &:= (\pi.x = [C_i]_{i=1}^k, \pi.\mathbf{w} = [\mathbf{m}_i]_{i=1}^k), \end{aligned}$$

the accumulation prover $\mathsf{P}_{\mathsf{acc}}$ works as in Figure 5.3.

Accumulation verifier. On input public input pi, NARK proof instance $\pi.x$, accumulator instance $\mathsf{acc}.x$, accumulation proof pf, and the updated accumulator instance $\mathsf{acc}'.x := \{\mathsf{pi}'', [C_i'']_{i=1}^k, [r_i'']_{i=1}^k, E', \mu'\}$, the accumulation verifier $\mathsf{V}_{\mathsf{acc}}$ works as in Figure 5.4.

Decider. On input the commitment key ck (which can be hardwired) and an accumulator

$$\mathsf{acc} = (\mathsf{acc}.x = \{\mathsf{pi}, [C_i]_{i=1}^k, [r_i]_{i=1}^{k-1}, E, \mu\}, \mathsf{acc}.\mathbf{w} = \{[\mathbf{m}_i]_{i=1}^k\}),$$

the decider does the checks described in Figure 5.5.

Remark 5.3.2. *The accumulation scheme for* $\mathsf{V}_{\mathsf{NARK}}$ *is also naturally a folding scheme as defined in Nova [KST22], where we can view an accumulator as a* relaxed *NP instance with error terms.*

$\mathsf{P}^{\rho_{\mathsf{acc}},\rho_{\mathsf{NARK}}}_{\mathsf{acc}}(\mathsf{ck},\mathsf{acc},(\mathsf{pi},\pi))$

1. $r_i \leftarrow \rho_{\mathsf{NARK}}(r_{i-1}, C_i) \forall i \in [k-1]$ where $r_0 := \rho_{\mathsf{NARK}}(\mathsf{pi})$.
2. Compute $[\mathbf{e}_j]_{j=1}^{d-1} \in (\mathbb{F}^\ell)^{d-1}$, such that

$$\begin{aligned}
&\sum_{j=0}^{d}(X+\mu)^{d-j}\cdot f_j^{\mathsf{V}_{\mathsf{sps}}}(X\cdot\mathsf{pi}+\mathsf{pi}', [X\cdot\mathbf{m}_i+\mathbf{m}'_i]_{i=1}^{k}, [X\cdot r_i + r'_i]_{i=1}^{k-1}) \\
=&\sum_{j=0}^{d}\mu^{d-j} f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{pi}', [\mathbf{m}'_i]_{i=1}^{k}, [r'_i]_{i=1}^{k-1}) + X^d\cdot\mathsf{V}_{\mathsf{NARK}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) + \sum_{j=1}^{d-1}\mathbf{e}_j X^j \\
=&\mathbf{e} + \sum_{j=1}^{d-1}\mathbf{e}_j X^j
\end{aligned}$$

3. $E_j \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{e}_j) \forall j \in [d-1]$
4. $\alpha \leftarrow \rho_{\mathsf{acc}}(\mathsf{acc}.x, \mathsf{pi}, \pi.x, [E_j]_{j=1}^{d-1}) \in \mathbb{F}$
5. Set vectors

$$\mathbf{v} := \left(1, \mathsf{pi}, [r_i]_{i=1}^{k-1}, [C_i]_{i=1}^{k}, [\mathbf{m}_i]_{i=1}^{k}\right),\ \mathbf{v}' := \left(\mu, \mathsf{pi}', [r'_i]_{i=1}^{k-1}, [C'_i]_{i=1}^{k}, [\mathbf{m}'_i]_{i=1}^{k}\right).$$

6. $\mathbf{v}'' := \left(\mu', \mathsf{pi}'', [r''_i]_{i=1}^{k-1}, [C''_i]_{i=1}^{k}, [\mathbf{m}''_i]_{i=1}^{k}\right) \leftarrow \alpha\cdot\mathbf{v} + \mathbf{v}'$.
7. $E' \leftarrow E + \sum_{j=1}^{d-1}\alpha^j \cdot E_j$.
8. Set $\mathsf{acc}'.x := \{\mathsf{pi}'', [C''_i]_{i=1}^{k}, [r''_i]_{i=1}^{k}, E', \mu'\}$, $\mathsf{acc}'.\mathbf{w} := \{[\mathbf{m}''_i]_{i=1}^{k}\}$.
9. Set accumulation proof $\mathsf{pf} := [E_j]_{j=1}^{d-1}$

Figure 5.3: Accumulation Prover for low-degree Fiat-Shamired NARKs

$\mathsf{V}^{\rho_{\mathsf{acc}},\rho_{\mathsf{NARK}}}_{\mathsf{acc}}(\mathsf{pi}, \pi.x = [C_i]_{i=1}^{k}, \mathsf{acc}.x = (\mathsf{pi}', [C'_i]_{i=1}^{k}, [r'_i]_{i=1}^{k-1}, E, \mu), \mathsf{pf} = [E_j]_{j=1}^{d-1}, \mathsf{acc}'.x)$

1. $r_i \leftarrow \rho_{\mathsf{NARK}}(r_{i-1}, C_i) \forall i \in [k-1]$ where $r_0 := \rho_{\mathsf{NARK}}(\mathsf{pi})$.
2. $\alpha \leftarrow \rho_{\mathsf{acc}}(\mathsf{acc}.x, \mathsf{pi}, \pi.x, \mathsf{pf})$
3. Set vectors

$$\mathbf{v} := \left(1, \mathsf{pi}, [r_i]_{i=1}^{k-1}, [C_i]_{i=1}^{k}\right),\ \mathbf{v}' := \mathsf{acc}.x.\left(\mu, \mathsf{pi}', [r'_i]_{i=1}^{k-1}, [C'_i]_{i=1}^{k}\right).$$

4. Check $\mathsf{acc}'.x.\left(\mu', \mathsf{pi}'', [r''_i]_{i=1}^{k-1}, [C''_i]_{i=1}^{k}\right) \stackrel{?}{=} \alpha\cdot\mathbf{v} + \mathbf{v}'$.
5. Check $\mathsf{acc}'.x.E' \stackrel{?}{=} \mathsf{acc}.x.E + \sum_{j=1}^{d-1}\alpha^j\cdot E_j$.

Figure 5.4: Accumulation Verifier for low-degree Fiat-Shamired NARKs

$D_{\mathsf{acc}}(\mathsf{acc} = (\mathsf{acc}.x = \{\mathsf{pi}, [C_i]_{i=1}^k, [r_i]_{i=1}^{k-1}, E, \mu\}, \mathsf{acc}.\mathbf{w} = \{[\mathbf{m}_i]_{i=1}^k\}))$

1. $C_i \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$ for all $i \in [k]$.
2. $\mathbf{e} \leftarrow \sum_{j=0}^{d} \mu^{d-j} f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^k, [r_i]_{i=1}^{k-1})$ where $f_j^{\mathsf{V}_{\mathsf{sps}}}$ is the degree-j homogeneous algebraic map described in the accumulated predicate.
3. $E \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$.

Figure 5.5: Accumulation Decider for low-degree Fiat-Shamired NARKs

A NARK proof π is an accumulator with $\mu = 1$ and $E = 0 \in \mathbb{G}$. We can use the same accumulation scheme to fold two accumulators $(\mathsf{acc}, \mathsf{acc}')$ into a new accumulator acc''. The scheme is identical to the one presented above but with non-trivial $\mu, \mathbf{e}, E$ terms for acc. The verifier performs one additional group scalar multiplication. In the language of folding schemes, we can fold two NARK instances into an accumulator; or fold a NARK instance and an accumulator into an updated accumulator; or fold two accumulators into an updated accumulator.

Complexity. Let Π_{sps} be a $(2k-1)$-move special-sound protocol with the verifier checking ℓ degree-d equations. Denote by $|M|$ the number of elements in prover messages and $|M^*|$ the number of non-zero elements in the prover messages. Assume that pi is a hash with length 1 (this saves the call $r_0 := \rho_{\mathsf{NARK}}(\mathsf{pi})$), and let $|R|$ be the number of elements in verifier's challenges. We analyze the computational complexity of the accumulation scheme:

- The *accumulation prover*
  - asks $k-1$ queries to ρ_{NARK} and 1 query to ρ_{acc};
  - computes $E_j = \mathsf{Commit}(\mathsf{ck}, \mathbf{e}_j)$ for all $j \in [d-1]$, where $\mathbf{e}_j \in \mathbb{F}^\ell$;
  - performs $|R| + |M^*| + 2$ $\mathbb{F}$-ops to combine $(\mu, \mathsf{pi}, [r_i]_{i=1}^{k-1}, [\mathbf{m}_i]_{i=1}^k)$;
  - performs k $\mathbb{G}$-ops to combine $[C_i]_{i=1}^k$;
  - computes the coefficients of ℓ degree-d polynomials for $[\mathbf{e}_j]_{j=1}^{d-1}$.
- The *accumulation verifier* performs
  - asks $k-1$ constant size queries to ρ_{NARK} and 1 d-sized query to ρ_{acc};
  - $|R| + 2$ $\mathbb{F}$-ops to combine $(\mu, \mathsf{pi}, [r_i]_{i=1}^{k-1})$;
  - k $\mathbb{G}$-ops to combine $[C_i]_{i=1}^k$;
  - $d-1$ $\mathbb{G}$-ops to add $[E_j]_{j=1}^{d-1}$ onto E.
- The *decider*
  - computes $C_i = \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$ for $i \in [k]$ and $E = \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$, with total complexity around $|M| + \ell$ $\mathbb{G}$-ops.

– evaluate ℓ degree-d multivariate polynomials to compute vector $\mathbf{e}$.

Theorem 5.2. *Let* $(\mathsf{P}_{NARK}, \mathsf{V}_{NARK})$ *be the RO-NARK defined in Section 5.3.3. Let* $\mathsf{cm} = (\mathsf{Setup}, \mathsf{Commit})$ *be a binding, homomorphic commitment scheme. Let* ρ_{acc} *be another random oracle. The accumulation scheme* $(\mathsf{P}_{\mathsf{acc}}, \mathsf{V}_{\mathsf{acc}}, D_{\mathsf{acc}})$ *for* V_{NARK} *satisfies perfect completeness and has knowledge error* $(Q+1)\frac{d+1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$ *as defined in Definition 5.8, against any randomized polynomial-time* Q*-query adversary.*

Proof.

Completeness. Consider any tuple $((\mathsf{pi}, \pi), \mathsf{acc}) \in \mathcal{R}_{\mathsf{acc}}$, that is, $\mathsf{V}_{\mathsf{NARK}}(\mathsf{pi}, \pi)$ and $D(\mathsf{acc})$ both accept. Let $(\mathsf{acc}', \mathsf{pf})$ denote the output of the accumulation prover $\mathsf{P}_{\mathsf{acc}}(\mathsf{ck}, \mathsf{acc}, (\mathsf{pi}, \pi))$. We argue that both the decider $D(\mathsf{acc}')$ and the accumulation verifier $\mathsf{V}_{\mathsf{acc}}(\mathsf{pi}, \pi.x, \mathsf{acc}.x, \mathsf{pf}, \mathsf{acc}'.x)$ will accept, which finishes the proof of perfect completeness by Definition 5.8.

$\mathsf{V}_{\mathsf{acc}}$ accepts as $\mathsf{P}_{\mathsf{acc}}$ and $\mathsf{V}_{\mathsf{acc}}$ go through the same process of computing challenges $[r_i]_{i=1}^{k-1}$ and α, thus the linear combinations of $\mathsf{acc}.x$ and $(\mathsf{pi}, \pi.x; \mathsf{pf}, [r_i]_{i=1}^{k-1})$ via α will be consistent.

We prove that $D(\mathsf{acc}')$ accepts by scrutinizing the following decider checks.

The check $\mathsf{acc}'.C_i \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, \mathsf{acc}'.\mathbf{m}_i)$ succeeds for all $i \in [k]$. This is because

$$\mathsf{acc}'.\{C_i, \mathbf{m}_i\} = \mathsf{acc}.\{C_i, \mathbf{m}_i\} + \alpha \cdot \pi.\{C_i, \mathbf{m}_i\}$$

for all $i \in [k]$, where $\pi.C_i = \mathsf{Commit}(\mathsf{ck}, \pi.\mathbf{m}_i)$ because $\mathsf{V}_{\mathsf{NARK}}(\mathsf{pi}, \pi)$ accepts, and $\mathsf{acc}.C_i = \mathsf{Commit}(\mathsf{ck}, \mathsf{acc}.\mathbf{m}_i)$ because $D(\mathsf{acc})$ accepts. Thus the check succeeds by the homomorphism of the commitment scheme.

The decider computes $\mathbf{e}' \leftarrow \sum_{j=0}^{d} (\mathsf{acc}'.\mu)^{d-j} f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{acc}'.\{\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}\})$ such that for $\mathbf{e} = \sum_{j=0}^{d} \mathsf{acc}.\mu^{(d-j)} \cdot f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{acc}.\{\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}\})$, it holds that

$$\begin{aligned}
\mathbf{e}' &= \mathbf{e} + \sum_{j=1}^{d-1} \alpha^j \cdot \mathsf{pf}.\mathbf{e}_j \\
&= \sum_{j=0}^{d} (\alpha + \mathsf{acc}.\mu)^{d-j} \cdot f_j^{\mathsf{V}_{\mathsf{sps}}}(\alpha \cdot \{\mathsf{pi}, \pi.[\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}\} + \mathsf{acc}.\{\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}\}) .
\end{aligned}$$

By the definition of $\mathsf{pf}.\mathbf{e}_j$ and the homomorphism of the commitment scheme, and because $D(\mathsf{acc})$ accepts and checks $E = \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$, we have that $E' = \mathsf{Commit}(\mathsf{ck}, \mathbf{e}')$.

Knowledge-Soundness. We show that the scheme has knowledge-soundness by showing that there exists an underlying $(d+1)$-special-sound protocol and then applying the Fiat-Shamir transform to show that the accumulation scheme is knowledge sound. Consider the public-coin interactive protocol $\Pi_I = (\mathsf{P}_I(\mathsf{pi}, \pi, \mathsf{acc}), \mathsf{V}_I(\mathsf{pi}, \pi.x, \mathsf{acc}.x))$ where P_I sends $\mathsf{pf} = [E_j]_{j=1}^{d-1} \in \mathbb{G}^{d-1}$ as computed by $\mathsf{P}_{\mathsf{acc}}$ to V_I. The verifier sends a random challenge $\alpha \in \mathbb{F}$, and the prover P_I responds with acc'

as computed by $\mathsf{P}_{\mathsf{acc}}$. V_I accepts if $D_{\mathsf{acc}}(\mathsf{acc}') = 0$ and $\mathsf{V}_{\mathsf{acc}}(\mathsf{pi}, \pi.x, \mathsf{acc}.x, \mathsf{pf}, \mathsf{acc}'.x) = 0$ using the random challenge α, instead of a Fiat-shamir challenge.

Claim 1: Π_I is $(d+1)$-special-sound Consider the relation $\mathcal{R}_{\mathsf{acc}}$ where $\mathcal{R}_{\mathsf{acc}}$ is defined in Definition 5.8. Consider $d+1$ accepting transcripts for Π_I :

$$\{\mathcal{T}_i := (\mathsf{pi}, \pi.x, \mathsf{acc}.x; \mathsf{acc}'_i, \mathsf{pf}_i)\}_{i=1}^{d+1} .$$

We construct an extractor $\mathsf{Ext}_{\mathsf{acc}}$ that extracts a witness for $\mathcal{R}_{\mathsf{acc}}(\mathsf{pi}.\pi.x, \mathsf{acc}.x)$ given $\mathcal{T}$.

For all $i \in [d+1]$,

$$(\mathsf{acc}'_i) = (\mu'_i, \mathsf{pi}'_i, [C'_{i,j}]_{j=1}^{k}, [r_{i,j}]_{j=1}^{k-1}, E'_i, [\mathbf{m}'_{i,j}]_{j=1}^{k})$$

and $\mathsf{pf}_i = \mathsf{pf} = [E_j]_{j=1}^{d-1}$.

Given that the transcripts are accepting, i.e. both $\mathsf{V}_{\mathsf{acc}}$ and D_{acc} accept, we have that $\mathsf{Commit}(\mathsf{ck}, \mathbf{e}'_i) = E'_i = \mathsf{acc}.E + \sum_{j=1}^{d-1} \alpha_i^j E_j$ for all $i \in [d+1]$, whereas

$$\mathbf{e}'_i := \sum_{j=0}^{d} {\mu'_i}^{d-j} f_j^{\mathcal{R}}(\pi'_i, [\mathbf{m}'_{i,j}]_{j=1}^{k}, [r_{i,j}]_{j=1}^{k-1}) .$$

Using a Vandermonde matrix of the challenges $\alpha_1, \ldots, \alpha_d$ we can compute $\mathbf{e}, [\mathbf{e}_j]_{j=1}^{d-1}$ such that $E_j = \mathsf{Commit}(\mathsf{ck}, \mathbf{e}_j)$ and $\mathsf{acc}.E = \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$ from the equations above. Therefore we have that $\mathbf{e}'_i = \mathbf{e} + \sum_{j=1}^{d-1} \alpha_i^j \mathbf{e}_j$ for all $i \in [d+1]$.

Additionally using two challenges (α_1, α_2), $\mathsf{Ext}_{\mathsf{acc}}$ can compute $\pi.\mathbf{w} = [\mathbf{m}_j]_{j=1}^{k} = [\frac{\mathsf{acc}'.\mathbf{m}_{1,j} - \mathsf{acc}'.\mathbf{m}_{2,j}}{\alpha_1 - \alpha_2}]_{j=1}^{k}$. It holds that $\mathsf{acc}.\mathbf{m}_j = \mathsf{acc}'.\mathbf{m}_{1,j} - \alpha_1 \cdot \pi.\mathbf{m}_j \forall j \in [k]$, such that $\pi.C_j = \mathsf{Commit}(\mathsf{ck}, \pi.\mathbf{m}_j)$ and $\mathsf{acc}.C_j = \mathsf{Commit}(\mathsf{ck}, \mathsf{acc}.\mathbf{m}_j)$. If for any other challenge and any j, $\mathsf{acc}'.\mathbf{m}_j \neq \alpha\pi.\mathbf{m}_j + \mathsf{acc}.\mathbf{m}_j$, then this can be used to compute a break of the commitment scheme cm. This happens with negligible probability by assumption.

Otherwise, we have that $\sum_{j=0} \mu_i^{d-j} f_j^{\mathcal{R}}(\pi_j, [\mathbf{m}_{i,j}]_{i=1}^{k}, [r_{i,j}]_{i=1}^{k-1}) - \mathbf{e}_i = 0$ for all $i \in [d+1]$. Together this implies that the degree d polynomial

$$\begin{aligned} p(X) = &\sum_{j=0}^{d} (X + \mathsf{acc}.\mu)^{d-j} \cdot f_j^{\mathsf{V}_{\mathsf{sps}}}(X \cdot \mathsf{pi} + \mathsf{acc}.\mathsf{pi}, [X \cdot \mathbf{m}_i + \mathsf{acc}.\mathbf{m}_i]_{i=1}^{k}, [X \cdot r_i + \mathsf{acc}.r_i]_{i=1}^{k-1}) \\ &- \mathbf{e} - \sum_{j=1}^{d-1} \mathbf{e}_j X^j , \end{aligned} \tag{5.2}$$

is zero on $d+1$ points $(\alpha_1, \ldots, \alpha_{d+1})$, i.e. is zero everywhere. The constant term of this polynomial

is

$$\sum_{j=0}^{d} \mathsf{acc}.\mu^{d-j} \cdot f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{acc}.\mathsf{pi}, [\mathsf{acc}.\mathbf{m}_i]_{i=1}^{k}, [\mathsf{acc}.r_i]_{i=1}^{k-1}) - \mathbf{e}\,.$$

It being 0 implies that $D(\mathsf{acc}) = 0$. Additionally, the degree d term of the polynomial is

$$\sum_{j=0}^{d} f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{pi}, [\pi.\mathbf{m}_i]_{i=1}^{k}, [\pi.r_i]_{i=1}^{k-1})\,.$$

Together with $\mathsf{V}_{\mathsf{acc}}$ checking that the challenges r_i are computed correctly this implies that $\mathsf{V}_{\mathsf{NARK}}(\mathsf{pi}, \pi) = 0$. Ext thus outputs a valid witness $(\pi.\mathbf{w}, \mathsf{acc}.\mathbf{w}) \in \mathcal{R}_{\mathsf{acc}}(\mathsf{pi}, \pi.x, \mathsf{acc}.x)$ and thus Π_I is $(d+1)$-special-sound. Using Lemma 5.1, we have that $\Pi_{AS} = \mathsf{FS}[\Pi_I]$ is a NARK for $\mathcal{R}_{\mathsf{acc}}$ with knowledge soundness $(Q+1) \cdot \frac{d+1}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$. This implies that acc is an accumulation scheme with $((Q+1) \cdot \frac{d+1}{|\mathbb{F}|} + \mathsf{negl}(\lambda))$-knowledge soundness. □

5.3.5 Compressing verification checks for high-degree verifiers

Observe that the accumulation prover needs to perform $\Omega(d\ell)$ group operations to commit to the $d-1$ error vectors $\mathbf{e}_j \in \mathbb{F}^\ell$ $(1 \leq j < d)$; and the accumulation verifier needs to check the combination of d error vector commitments. This can be a bottleneck when the verifier degree d is high. In this circumstance, we can optimize the accumulation complexity by transforming the underlying special-sound protocol Π_{sps} into a new special-sound protocol $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ for the same relation $\mathcal{R}$. This optimization compresses the ℓ degree-d equations checked by the verifier into a single degree-$(d+2)$ equation using a random linear combination, with the tradeoff of additionally checking $2\sqrt{\ell}$ degree-2 equations. We describe the generic transformation below.

Compressing verification checks. W.l.o.g. assume ℓ is a perfect square, then we can transform Π_{sps} into a special-sound protocol $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ where the $\mathsf{V}_{\mathsf{sps}}$ reduces from ℓ degree-d checks to 1 degree-$(d+2)$ check and additionally $2\sqrt{\ell}$ degree-2 checks. Instead of checking the output of $\mathsf{V}_{\mathsf{sps}}$ to be ℓ zeroes, we take a random linear combination of the ℓ verification equations using powers of a challenge β. For example, if the map is $\mathsf{V}_{\mathsf{sps}}(x_1, x_2) := (f_1(x_1, x_2), f_2(x_1, x_2)) = (x_1 + x_2, x_1 x_2)$ we can set the new algebraic map as $\mathsf{V}'_{\mathsf{sps}}(x_1, x_2, \beta) := f_1(x_1, x_2) + \beta \cdot f_2(x_1, x_2) = (x_1 + x_2) + \beta x_1 x_2$ for a random β. Doing this naively reduces the output length to 1 but also requires the verifier to compute the appropriate powers of β. This would increase the degree by ℓ, an undesirable tradeoff. To mitigate this, we can have the prover precompute powers of β, i.e. $\beta, \beta^2, \ldots, \beta^\ell$ and send them to the verifier. The verifier then only needs to check consistency between the powers of β, which can be done using a degree 2 check, e.g. $\beta^{i+1} = \beta^i \cdot \beta$ and the degree d verification equation increases in degree by 1. This mitigates the degree increase but requires the prover to send another message of length ℓ. To achieve a more optimal tradeoff, we write each $i = j + k \cdot \sqrt{\ell}$ for $j, k \in [1, \sqrt{\ell}]$. The

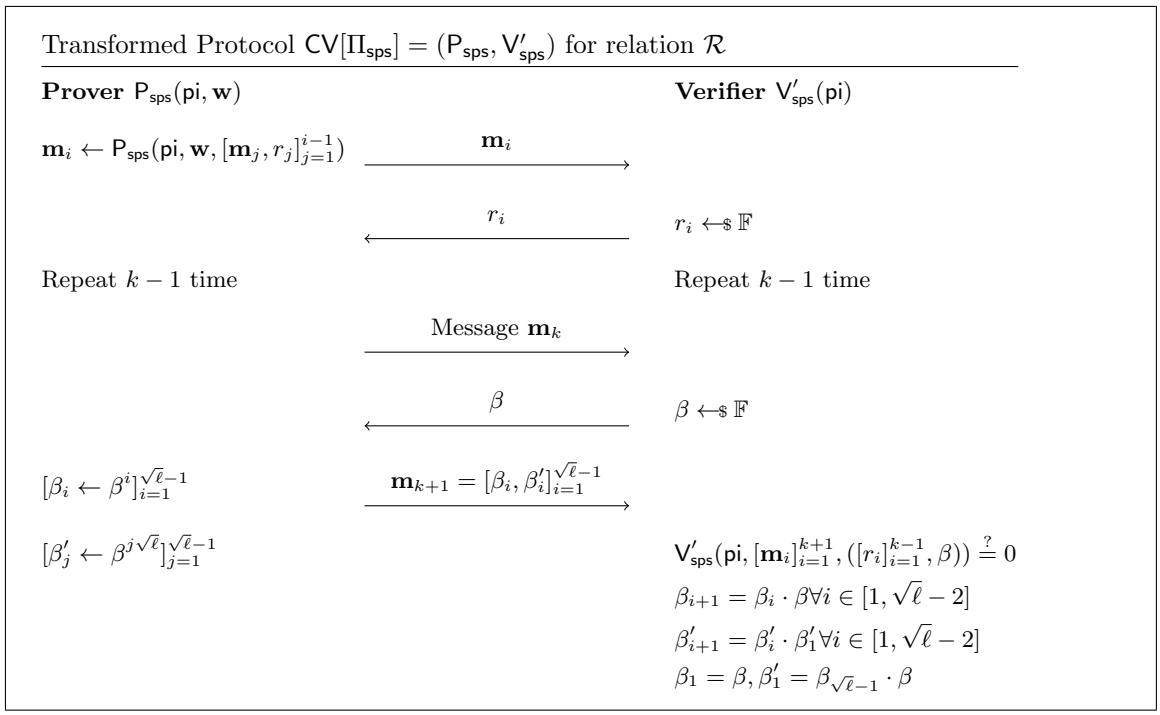


Figure 5.6: Compressed verification of Π_{sps}.

prover then sends $\sqrt{\ell}$ powers of β and $\sqrt{\ell}-1$ powers of $\beta^{\sqrt{\ell}}$. From these, each power of β from 1 to ℓ can be recomputed using just one multiplication. This results in the prover sending an additional message of length $2\sqrt{\ell}$, the original ℓ verification checks being transformed into a single degree $d+2$ check and additionally $2\sqrt{\ell}$ degree 2 checks for the consistency of the powers of β.

We describe the transformed protocol in Figure 5.6, where

$$\begin{aligned}\mathsf{V}'_{\mathsf{sps}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k+1}, ([r_i]_{i=1}^{k-1}, \beta)) &:= \sum_{i=0}^{\sqrt{\ell}-1} \sum_{j=0}^{\sqrt{\ell}-1} \beta_i \cdot \beta'_j \cdot \mathsf{V}_{\mathsf{sps},i+j\sqrt{\ell}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) \\ &= \sum_{j=0}^{\ell-1} \beta^j \cdot \mathsf{V}_{\mathsf{sps},j}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1})\end{aligned}$$

and $\mathsf{V}_{\mathsf{sps},j}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1})$ is the $(j+1)$-th $(0 \le j < \ell)$ equation checked by $\mathsf{V}_{\mathsf{sps}}$. The transformed protocol is a $(2k+1)$-move special-sound protocol for the same relation $\mathcal{R}$. The transformed verifier now checks 1 degree-$(d+2)$ equation and additionally $2\sqrt{\ell}$ degree-2 equations.

Lemma 5.3. *Let Π_{sps} be a $(2k-1)$-move protocol for relation $\mathcal{R}$ with $(a_1, \ldots, a_{k-1})$-special-soundness, in which the verifier outputs ℓ elements. The transformed protocol $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ of Π_{sps} is $(a_1, \ldots, , a_{k-1}, \ell)$-special-sound.*

Proof. Let $\mathsf{Ext}_{\mathsf{sps}}$ be the extractor for Π_{sps}. We construct a extractor $\mathsf{Ext}_{\mathsf{CV}}$ of $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ for the same relation $\mathcal{R}$. Given an $(a_1, \ldots, , a_{k-1}, \ell)$-tree $\mathcal{T}$ of accepting transcripts, $\mathsf{Ext}_{\mathsf{CV}}$ invokes $\mathsf{Ext}_{\mathsf{sps}}$ on input the depth-$(k-1)$ transcript subtree of $\mathcal{T}$, and return what $\mathsf{Ext}_{\mathsf{sps}}$ outputs.

We prove that the extractor succeeds. For each internal node u at depth $k-1$, it has ℓ children where each child maps to a distinct value of $\beta \in \mathbb{F}$. Fix the messages $\mathsf{msg} = (\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1})$ at node u and let $\mathsf{V}_{\mathsf{sps}} := (\mathsf{V}_{\mathsf{sps},1}, \ldots, \mathsf{V}_{\mathsf{sps},\ell})$ be the verifier of Π_{sps}. Define the degree $\ell - 1$ univariate polynomial

$$p(X) := \sum_{j=0}^{\ell-1} X^j \cdot c_j$$

where $c_j := \mathsf{V}_{\mathsf{sps},j}(\mathsf{msg}) \in \mathbb{F}$ is $\mathsf{V}_{\mathsf{sps},j}$'s output on message msg. Since the transcripts are accepting, it holds that p evaluates to zero on the ℓ different values of β that correspond to the ℓ children of node u. Thus the univariate polynomial p is a zero polynomial, which implies that $\mathsf{V}_{\mathsf{sps}}$ outputs zero vector on message msg. Therefore for every node u at depth $k-1$, the sub-transcript from root to node u is an accepting transcript to Π_{sps}. Therefore the input to $\mathsf{Ext}_{\mathsf{sps}}$ is a valid $(a_1, \ldots, , a_{k-1})$-tree of accepting transcripts, and $\mathsf{Ext}_{\mathsf{sps}}$ will output the correct witness. □

High-low degree accumulation. After the transformation, the error vectors $\mathbf{e}_j$ $(1 \le j \le d+1)$ become single field elements, and we can use the trivial commitment $E_j := \mathsf{Commit}(\mathsf{ck}, e_j) := e_j$ without group operations. Additionally, we can use a separate error vector $\mathbf{e}' \in \mathbb{F}^{2\sqrt{\ell}}$ to keep track of the error terms for the $2\sqrt{\ell}$ degree-2 checks, and set $E' := \mathsf{Commit}(\mathsf{ck}, \mathbf{e}') \in \mathbb{G}$ to be the corresponding error commitment. The accumulation prover only needs to perform $O(\sqrt{\ell})$ additional group operations to commit $\mathbf{m}_{k+1}$ and $\mathbf{e}'$, and compute the coefficients of a degree-$(d+2)$ univariate polynomial, which is described as the sum of $O(\ell)$ polynomials. The accumulator instance needs to include one more challenge β and two commitments (for $\mathbf{m}_{k+1}$ and $\mathbf{e}'$). The accumulator verifier needs to do only $k+2$ (rather than $k+d-1$) group scalar multiplications, with the tradeoff of 1 more hash and $O(d)$ more field operations. This high-low degree accumulation is described in detail in Appendix 5.7.

Theorem 5.3 (IVC for high-degree special-sound protocols)**.** *Let $\mathbb{F}$ be a finite field, such that $|\mathbb{F}| \ge 2^\lambda$ and $\mathsf{cm} = (\mathsf{Setup}, \mathsf{Commit})$ be a binding homomorphic commitment scheme for vectors in $\mathbb{F}$. Let $\Pi_{\mathsf{sps}} = (\mathsf{P}_{\mathsf{sps}}, \mathsf{V}_{\mathsf{sps}})$ be a special-sound protocol for an NP-complete relation $\mathcal{R}_{\mathsf{NP}}$ with the following properties:*

- *It's $(2k-1)$ move.*
- *It's $(a_1, \ldots, a_{k-1})$-out-of-$|\mathbb{F}|$ special-sound. Such that the knowledge error $\kappa = 1 - \prod_{i=1}^{k-1}(1 - \frac{a_i}{|\mathbb{F}|}) = \mathsf{negl}(\lambda)$*
- *The inputs are in $\mathbb{F}^{\ell_{\mathsf{in}}}$*

- *The verifier is degree* $d = \mathsf{poly}(\lambda)$ *with output in* $\mathbb{F}^\ell$

Then, under the Fiat-Shamir heuristic for a cryptographic hash function H*(Definition 5.9), there exist two IVC schemes* $\mathsf{IVC} = (\mathsf{P}_{\mathsf{IVC}}, \mathsf{V}_{\mathsf{IVC}})$ *and* $\mathsf{IVC}_{\mathsf{CV}} = (\mathsf{P}_{\mathsf{CV},\mathsf{IVC}}, \mathsf{V}_{\mathsf{CV},\mathsf{IVC}})$ *with predicates expressed in* $\mathcal{R}_{\mathsf{NP}}$ *with the following efficiencies:*

	No CV	CV
$\mathsf{P}_{\mathsf{IVC}}$ *native*	$\sum_{i=1}^{k} \lvert\mathbf{m}_i^*\rvert + (d-1)\ell\mathbb{G}$ $\mathsf{P}_{\mathsf{sps}} + L(\mathsf{V}_{\mathsf{sps}}, d)$	$\sum_{i=1}^{k} \lvert\mathbf{m}_i^*\rvert + O(\sqrt{\ell})\mathbb{G}$ $\mathsf{P}_{\mathsf{sps}} + L'(\mathsf{V}_{\mathsf{sps}}, d+2)$
$\mathsf{P}_{\mathsf{IVC}}$ *recursive*	$k + d - 1\mathbb{G}$ $k + \ell_{\mathsf{in}}\mathbb{F}$ $(k + d + O(1))H + 1\mathsf{H}_{\mathsf{in}}$	$k + 2\mathbb{G}$ $k + \ell_{\mathsf{in}} + d + 1\mathbb{F}$ $(k + d + O(1))H + 1\mathsf{H}_{\mathsf{in}}$
$\mathsf{V}_{\mathsf{IVC}}$*:*	$\ell + \sum_{i=1}^{k} \lvert\mathbf{m}_i\rvert\mathbb{G}$ $\mathsf{V}_{\mathsf{sps}}$	$O(\sqrt{\ell}) + \sum_{i=1}^{k} \lvert\mathbf{m}_i\rvert\mathbb{G}$ $O(\ell) + \mathsf{V}_{\mathsf{sps}}$
$\lvert\pi_{\mathsf{IVC}}\rvert$ *:*	$k + \ell_{\mathsf{in}}\mathbb{F}$ $k + 1\mathbb{G}$ $\sum_{i=1}^{k} \lvert\mathbf{m}_i\rvert$	$k + \ell_{\mathsf{in}} + 1\mathbb{F}$ $k + 2\mathbb{G}$ $\sum_{i=1}^{k} \lvert\mathbf{m}_i\rvert + O(\sqrt{\ell})$

The first row displays the native operations of the IVC *prover. The second row describes the size of the recursive statement expressed as an instance of* $\mathcal{R}_{\mathsf{NP}}$ *for which* $\mathsf{P}_{\mathsf{IVC}}$ *creates a proof. The third row is the computation of* $\mathsf{V}_{\mathsf{IVC}}$*, and the last row is the size of the proof.*

In the table, $\lvert\mathbf{m}_i\rvert$ *denotes the prover message length;* $\lvert\mathbf{m}_i^*\rvert$ *is the number of non-zero elements in* $\mathbf{m}_i$*;* $\mathbb{G}$ *for rows 1-3 is the total length of the messages committed using* Commit*.* $\mathbb{F}$ *are field operations.* H *are calls to the random oracle and* H_{in} *are the hash to the public input and accumulator instance.* $\mathsf{P}_{\mathsf{sps}}$ *(and* $\mathsf{V}_{\mathsf{sps}}$*) is the cost of running the prover (and the algebraic verifier) of the special-sound protocol, respectively.* $L(\mathsf{V}_{\mathsf{sps}}, d)$ *is the cost of computing the coefficients of the degree* d *polynomial*

$$\mathbf{e}(X) := \sum_{j=0}^{d} (\mu + X)^{d-j} \cdot f_j^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{acc} + X \cdot \pi)\,, \tag{5.3}$$

and $L'(\mathsf{V}_{\mathsf{sps}}, d+2)$ *is the cost of computing the coefficients of the degree* $d+2$ *polynomial*

$$e(X) := \sum_{a=0}^{\sqrt{\ell}-1} \sum_{b=0}^{\sqrt{\ell}-1} (X \cdot \pi.\beta_a + \mathsf{acc}.\beta_a)(X \cdot \pi.\beta'_b + \mathsf{acc}.\beta'_b) \sum_{j=0}^{d} (\mu + X)^{d-j} \cdot f_{j,a+b\sqrt{\ell}}^{\mathsf{V}_{\mathsf{sps}}}(\mathsf{acc} + X \cdot \pi)\,, \tag{5.4}$$

where all inputs are linear functions in a formal variable X[4]*, and* $f_{j,i}^{\mathsf{V}_{\mathsf{sps}}}$ *is the* i*th* $(0 \le i \le \ell - 1)$ *component of* $f_j^{\mathsf{V}_{\mathsf{sps}}}$*'s output. For the proof size,* $\mathbb{G}$ *and* $\mathbb{F}$ *are the number of commitments and field elements, respectively.*

[4]For example if $f_d = \prod_{i=1}^{d}(a_i + b_i \cdot X)$ then a naive algorithm takes $O(d^2)$ time but using FFTs it can be computed in time $O(d \log^2 d)$[Che+22].

Proof. The construction first defines the two NARKs

$$\Pi_{\mathsf{NARK}} = (\mathsf{P}_{\mathsf{NARK}}, \mathsf{V}_{\mathsf{NARK}}) = \mathsf{FS}[\mathsf{cm}[\Pi_{\mathsf{sps}}]]\,,$$

and

$$\Pi_{\mathsf{NARK},\mathsf{CV}} = (\mathsf{P}_{\mathsf{NARK}}, \mathsf{V}_{\mathsf{NARK}}) = \mathsf{FS}[\mathsf{cm}[\mathsf{CV}[\Pi_{\mathsf{sps}}]]]\,.$$

Then we construct the accumulation scheme $(\mathsf{P}_{\mathsf{acc}}, \mathsf{V}_{\mathsf{acc}}) = \mathsf{acc}[\Pi_{\mathsf{NARK}}]$ using the accumulation scheme from Section 5.3.4 and $(\mathsf{P}_{\mathsf{acc},\mathsf{HL}}\mathsf{V}_{\mathsf{acc},\mathsf{HL}}) = \mathsf{acc}_{\mathsf{HL}}[\Pi_{\mathsf{NARK},\mathsf{CV}}]$ using the accumulation scheme from Section 5.7. Then we apply the transformation from Theorem 5.1 to construct the IVC schemes IVC and $\mathsf{IVC}_{\mathsf{CV}}$.

Security: By Lemmas 5.1,5.2, we have that Π_{NARK} has $(Q+1)\cdot\left[1-\prod_{i=1}^{k-1}(1-\frac{a_i}{|\mathbb{F}|})\right]$ knowledge error for relation $\mathcal{R}_{\mathsf{cm}}^{\mathcal{R}_{\mathsf{NP}}}$ for a polynomial-time Q-query RO-adversary. Witnesses for $\mathcal{R}_{\mathsf{cm}}^{\mathcal{R}_{\mathsf{NP}}}$ are either a witness for $\mathcal{R}_{\mathsf{NP}}$ or a break of the binding property of cm. Assuming that cm is a binding commitment scheme, the probability that a polynomial time adversary and a polynomial time extractor can compute such a break is $\mathsf{negl}(\lambda)$. Thus Π_{NARK} has knowledge error $\kappa = (Q+1)\cdot\left[1-\prod_{i=1}^{k-1}(1-\frac{a_i}{|\mathbb{F}|})\right]+\mathsf{negl}(\lambda)$ for $\mathcal{R}_{\mathsf{NP}}$. Analogously and using Lemma 5.3, $\Pi_{\mathsf{NARK},\mathsf{CV}}$ has knowledge soundness with knowledge error $\kappa' = (Q+1)\cdot\left[1-(1-\frac{\ell}{|\mathbb{F}|})\prod_{i=1}^{k-1}(1-\frac{a_i}{|\mathbb{F}|})\right]+\mathsf{negl}(\lambda)$ for $\mathcal{R}_{\mathsf{NP}}$. By assumption, κ and κ' are negligible in λ. Using Theorem 5.2 and Corollary 5.5 we can construct accumulation schemes acc and $\mathsf{acc}_{\mathsf{CV}}$ for Π_{NARK} and $\Pi_{\mathsf{NARK},\mathsf{CV}}$, respectively. The accumulation schemes have negligible knowledge error as $d = \mathsf{poly}(\lambda)$. Under the Fiat-Shamir heuristic for H we can turn the NARKs and the accumulation schemes into secure schemes in the standard model.

By Theorem 5.1, this yields IVC and $\mathsf{IVC}_{\mathsf{CV}}$, secure IVC schemes with predicates expressed in $\mathcal{R}_{\mathsf{NP}}$.

Efficiency: We first analyze the efficiency for IVC. The IVC-prover runs $\mathsf{P}_{\mathsf{sps}}$ to compute all prover messages. It also commits to all the $\mathsf{P}_{\mathsf{sps}}$ messages using cm. Finally, it needs to compute all error terms $\mathbf{e}_1, \ldots, \mathbf{e}_{d-1}$ and commit to them. The error terms are computed by symbolically evaluating the polynomial $e(X)$ in Equation 5.4 with linear functions as inputs. The recursive circuit combines a new proof $\pi.x$ with an accumulator $\mathsf{acc}.x$. The size of the accumulator instance is ℓ_{in} field elements for the input, $k-1$ field elements for the interactive-proof challenges, 1 field element for the accumulator challenge, and k commitments for the $\mathsf{P}_{\mathsf{sps}}$ messages and $d-1$ commitments for the error terms. The IVC verifier checks the correctness of the commitments and runs $\mathsf{V}_{\mathsf{sps}}$.

For $\mathsf{IVC}_{\mathsf{CV}}$, the prover needs to additionally commit to a message $\mathbf{m}_{k+1}$ with length $O(\sqrt{\ell})$; the number of error terms also increases from $d-1$ to $d+1$. Fortunately, the error terms are only one element in $\mathbb{F}$, so we can use the identity function as the trivial commitment scheme. Thus, there is no cost for committing to the $d+1$ error terms when using CV. However, there is another separate error term $\mathbf{e}' \in \mathbb{F}^{2\sqrt{\ell}}$ for the additional $O(\sqrt{\ell})$ degree-2 checks, thus the prover needs to commit to

$E' = \mathsf{Commit}(\mathbf{e}')$. The size of the accumulator instance is ℓ_{in} field elements for the input, k field elements for the interactive-proof challenges, 1 field element for the accumulator challenge, $k+1$ commitments for the prover messages, $d+1$ field elements for the error terms of the high-degree checks, and 1 commitment for the additional error term $\mathbf{e}'$. □

Remark 5.3.3. *For simplicity, we assume that the public input, the prover messages, and the verifier challenges are all in the same field $\mathbb{F}$. This isn't strictly necessary; for example, the challenges could be drawn from a subset of $\mathbb{F}$. More generally, we can also allow prover messages to be group elements in $\mathbb{G}$ given a homomorphic commitment scheme to group elements(e.g. [Abe+10]).*

5.4 Special-sound subprotocols for ProtoStar

In this section, we present special-sound protocols for permutation, high-degree gate, circuit selection and lookup relations, which are the building blocks for the (non-uniform) Plonkish circuit-satisfiability relations. We can build accumulation schemes for (and thus IVCs from) these special-sound protocols via the framework presented in Section 5.3.

5.4.1 Permutation relation

Definition 5.10. *Let $\sigma : [n] \to [n]$ be a permutation, the relation $\mathcal{R}_\sigma$ is the set of tuples $\mathbf{w} \in \mathbb{F}^n$ such that $\mathbf{w}_i = \mathbf{w}_{\sigma(i)}$ for all $i \in [n]$.*

Special-sound protocol Π_σ for permutation relation $\mathcal{R}_\sigma$

Prover $\mathsf{P}(\sigma, \mathbf{w} \in \mathbb{F}^n)$		**Verifier** $\mathsf{V}(\sigma)$
	$\mathbf{w} \longrightarrow$	
		Check $\mathbf{w}_i - \mathbf{w}_{\sigma(i)} = 0 \forall i \in [n]$

Complexity. Π_σ is a 1-move protocol (i.e. $k = 1$); the degree of the verifier is 1.

5.4.2 High-degree custom gate relation

Definition 5.11. *Given configuration $\mathcal{C}_{GATE} := (n, c, d, [\mathbf{s}_i \in \mathbb{F}^n, G_i]_{i=1}^m)$ where n is the number of gates, c is the arity per gate, d is the gate degree, $[\mathbf{s}_i]_{i=1}^m$ are the selector vectors, and $[G_i]_{i=1}^m$ are the gate formulas, the relation $\mathcal{R}_{GATE}$ is the set of tuples $\mathbf{w} \in \mathbb{F}^{cn}$ such that $\sum_{j=1}^m \mathbf{s}_{j,i} \cdot G_j(\mathbf{w}_i, \mathbf{w}_{i+n}, \ldots, \mathbf{w}_{i+(c-1)\cdot n}) = 0$ for all $i \in [n]$.*

Special-sound protocol Π_{GATE} for relation $\mathcal{R}_{\text{GATE}}$

Prover $\mathsf{P}(\mathcal{C}_{\mathbf{GATE}}, \mathbf{w} \in \mathbb{F}^{cn})$		**Verifier** $\mathsf{V}(\mathcal{C}_{\mathbf{GATE}})$
	$\xrightarrow{\mathbf{w}}$	
		$\sum_{j=1}^{m} \mathbf{s}_{j,i} \cdot G_j(\mathbf{w}_i, \mathbf{w}_{i+n}, \ldots, \mathbf{w}_{i+(c-1)\cdot n})$ $\stackrel{?}{=} 0 \forall i \in [n]$

Complexity. Π_{GATE} is a 1-move protocol (i.e. $k = 1$) with verifier degree d.

5.4.3 Lookup relation

Definition 5.12. *Given configuration $\mathcal{C}_{LK} := (T, \ell, \mathbf{t})$ where ℓ is the number of lookups and $\mathbf{t} \in \mathbb{F}^T$ is the lookup table, the relation $\mathcal{R}_{LOOKUP}$ is the set of tuples $\mathbf{w} \in \mathbb{F}^\ell$ such that $\mathbf{w}_i \in \mathbf{t}$ for all $i \in [\ell]$.*

We recall a useful lemma for lookup relation from [Hab22b], and present a special-sound protocol for the lookup relation.

Lemma 5.4 (Lemma 5 of [Hab22b])**.** *Let $\mathbb{F}$ be a field of characteristic $p > \max(\ell, T)$. Given two sequences of field elements $[\mathbf{w}_i]_{i=1}^{\ell}$ and $[\mathbf{t}_i]_{i=1}^{T}$, we have $\{\mathbf{w}_i\} \subseteq \{\mathbf{t}_i\}$ as sets (with multiples of values removed) if and only if there exists a sequence $[\mathbf{m}_i]_{i=1}^{T}$ of field elements such that*

$$\sum_{i=1}^{\ell} \frac{1}{X + \mathbf{w}_i} = \sum_{i=1}^{T} \frac{\mathbf{m}_i}{X + \mathbf{t}_i}\,. \tag{5.5}$$

Special-sound protocol Π_{LK} for $\mathcal{R}_{\text{LOOKUP}}$

Prover $\mathsf{P}(\mathcal{C}_{\mathbf{LK}}, \mathbf{w} \in \mathbb{F}^\ell)$		**Verifier** $\mathsf{V}(\mathcal{C}_{\mathbf{LK}})$
Compute $\mathbf{m} \in \mathbb{F}^T$ such that		
$\mathbf{m}_i := \sum_{j=1}^{\ell} \mathbb{1}(\mathbf{w}_j = \mathbf{t}_i) \forall i \in [T]$	$\xrightarrow{\mathbf{w}, \mathbf{m}}$	
	$\xleftarrow{r}$	$r \leftarrow\!\!\$\ \mathbb{F}$
Compute $\mathbf{h} \in \mathbb{F}^\ell$, $\mathbf{g} \in \mathbb{F}^T$		
$\mathbf{h}_i := \frac{1}{\mathbf{w}_i + r} \forall i \in [\ell]$		
$\mathbf{g}_i := \frac{\mathbf{m}_i}{\mathbf{t}_i + r} \forall i \in [T]$	$\xrightarrow{\mathbf{h}, \mathbf{g}}$	
		$\sum_{i=1}^{\ell} \mathbf{h}_i \stackrel{?}{=} \sum_{i=1}^{T} \mathbf{g}_i$
		$\mathbf{h}_i \cdot (\mathbf{w}_i + r) \stackrel{?}{=} 1 \forall i \in [\ell]$
		$\mathbf{g}_i \cdot (\mathbf{t}_i + r) \stackrel{?}{=} \mathbf{m}_i \forall i \in [T]$

Achieving perfect completeness. Note that the protocol does not have perfect completeness. If there exists an $\mathbf{w}_i$ or $\mathbf{t}_i$ such that $\mathbf{w}_i + r = 0$ $\mathbf{t}_i + r = 0$ then the prover message is undefined. We can achieve perfect completeness by having the verifier set $\mathbf{h}_i = 0$ or $\mathbf{g}_i = 0$ in this case and changing the verification equations to

$$(\mathbf{w}_i + r) \cdot (\mathbf{h}_i \cdot (\mathbf{w}_i + r) - 1) = 0$$

and

$$(\mathbf{t}_i + r) \cdot (\mathbf{g}_i \cdot (\mathbf{t}_i + r) - \mathbf{m}_i) = 0\,.$$

These checks ensure that either $\mathbf{h}_i = \frac{1}{\mathbf{w}_i + r}$ or $\mathbf{w}_i + r = 0$. The checks increase the verifier degree to 3. Without these checks, the protocol has a negligible completeness error of $\frac{\ell + T}{|\mathbb{F}|}$. This completeness error can likely be ignored in practice, and these checks do not need to be implemented. However, to achieve the full definition of PCD (which has perfect completeness) and use Theorem 5.1 by [Bün+21a], we require that all protocols have perfect completeness.

Complexity. Π_{LK} is a 3-move protocol (i.e. $k = 2$); the degree of the verifier is 2; the number of non-zero elements in the prover message is at most 4ℓ.

Accumulation with $O(\ell)$ prover complexity. The prover complexity of Π_{LK} is due to the sparseness of $\mathbf{g} \in \mathbb{F}^T$ and $\mathbf{m} \in \mathbb{F}^T$. However, there is no guarantee that when building an accumulation scheme for Π_{LK}, the accumulated $\mathsf{acc}.\mathbf{g}$ and $\mathsf{acc}.\mathbf{m}$ are sparse. This is an issue, as the prover needs to compute the error term $\mathbf{e}_1$. If we expand the accumulation procedures, we see that the three verification checks lead to three components of the error term $\mathbf{e}_1$:

$$\mathbf{e}_1^{(1)} = \left(\sum_{i=1}^{\ell} \mathsf{acc}.\mathbf{h}_i - \sum_{i=1}^{T} \mathsf{acc}.\mathbf{g}_i\right) + \mu\left(\sum_{i=1}^{\ell} \pi.\mathbf{h}_i - \sum_{i=1}^{T} \pi.\mathbf{g}_i\right) \in \mathbb{F}$$

$$\mathbf{e}_1^{(2)} = \mathsf{acc}.\mathbf{h} \circ (\pi.\mathbf{w} + \pi.r \cdot \mathbf{1}^{\ell}) + \pi.\mathbf{h} \circ (\mathsf{acc}.\mathbf{w} + \mathsf{acc}.r \cdot \mathbf{1}^{\ell}) - 2\mu \cdot \mathbf{1}^{\ell} \in \mathbb{F}^{\ell}$$

$$\mathbf{e}_1^{(3)} = \mathsf{acc}.\mathbf{g} \circ (\mathbf{t} + \pi.r \cdot \mathbf{1}^{T}) + \pi.\mathbf{g} \circ (\mu \cdot \mathbf{t} + \mathsf{acc}.r \cdot \mathbf{1}^{T}) - \mu \cdot \pi.\mathbf{m} - \mathsf{acc}.\mathbf{m} \in \mathbb{F}^{T}\,.$$

We examine all three components below.

For $\mathbf{e}_1^{(1)}$, we see that $(\sum_{i=1}^{\ell} \pi.\mathbf{h}_i - \sum_{i=1}^{T} \pi.\mathbf{g}_i) = 0$ by the assumption that π is valid, and $(\sum_{i=1}^{\ell} \mathsf{acc}.\mathbf{h}_i - \sum_{i=1}^{T} \mathsf{acc}.\mathbf{g}_i) = \mathsf{acc}.\mathbf{e}^{(1)}/\mathsf{acc}.\mu$ (where $\mathsf{acc}.\mathbf{e}^{(1)}$ is the first component of the error vector for acc). Thus $\mathbf{e}_1^{(1)} = \mathsf{acc}.\mathbf{e}^{(1)}/\mathsf{acc}.\mu$. We observe that since in IVC the accumulator $\mathsf{acc}.\mathbf{e}^{(1)}$ is initiated with 0, this implies that for all iterations $\mathbf{e}_1^{(1)} = 0$.

For $\mathbf{e}_1^{(2)}$, it is computed from terms of size ℓ, so can be computed in time $O(\ell)$.

For $\mathbf{e}_1^{(3)}$, note that $\mathsf{acc}.\mu$, $\mathsf{acc}.r$ and $\pi.r$ are all scalars. Also note that the accumulation prover only needs to compute the commitment $E_1 = \mathsf{Commit}(\mathsf{ck}, \mathbf{e}_1) = \mathsf{Commit}(\mathsf{ck}, \mathbf{e}_1^{(1)}) + \mathsf{Commit}(\mathsf{ck}, 0||\mathbf{e}_1^{(2)}) + \mathsf{Commit}(\mathsf{ck}, \mathbf{0}^{\ell+1}||\mathbf{e}_1^{(3)})$, not the actual vector $\mathbf{e}_1$. We will compute $E_1^{(3)} = \mathsf{Commit}(\mathsf{ck}, \mathbf{e}_1^{(3)})$ homomorphically from the commitments below (dropping the zero padding for readability):

1. $G = \mathsf{Commit}(\mathsf{ck}, \pi.\mathbf{g})$,
2. $G' = \mathsf{Commit}(\mathsf{ck}, \mathsf{acc}.\mathbf{g})$,
3. $M = \mathsf{Commit}(\mathsf{ck}, \pi.\mathbf{m})$,
4. $M' = \mathsf{Commit}(\mathsf{ck}, \mathsf{acc}.\mathbf{m})$,
5. $GT = \mathsf{Commit}(\mathsf{ck}, \pi.\mathbf{g} \circ \mathbf{t})$,
6. $GT' = \mathsf{Commit}(\mathsf{ck}, \mathsf{acc}.\mathbf{g} \circ \mathbf{t})$.

Given these commitments, we can compute

$$E_1^{(3)} = GT' + \pi.r \cdot G' + \mathsf{acc}.\mu \cdot GT + \mathsf{acc}.r \cdot G - \mathsf{acc}.\mu \cdot M - M'\,.$$

This reduces the problem to the problem of efficiently computing and updating the commitments. G, M and GT are all commitments to ℓ-sparse vectors, thus can be efficiently computed. The prover can cache the commitments G', M', and GT' and efficiently update them during accumulation. That is $G'' \leftarrow G' + \alpha G$, $M'' \leftarrow M' + \alpha M$ and $GT'' \leftarrow GT' + \alpha GT$. Additionally, we need to update the accumulation witnesses: $\mathsf{acc}'.\mathbf{m} \leftarrow \mathsf{acc}.\mathbf{m} + \alpha\pi.\mathbf{m}$ and $\mathsf{acc}'.\mathbf{g} \leftarrow \mathsf{acc}.\mathbf{g} + \alpha\pi.\mathbf{g}$. Again because $\pi.\mathbf{g}, \pi.\mathbf{m}$ are sparse this can be done in time $O(\ell)$ independent of $T = |\mathbf{t}|$.

When Π_{LK} is used in composition with another special-sound protocol with a higher degree d, the accumulation is made homogeneous using a $(X+\mu)^{d-2}$ factor when computing the error terms. The contribution to the error terms $\mathbf{e}_i$ $(1 \leq i \leq d-1)$ is still a linear function in $\mathsf{acc}.\mathbf{g}$, $\mathsf{acc}.\mathbf{m}$ and $\mathsf{acc}.\mathbf{g} \circ \mathbf{t}$, and thus can be computed homomorphically from commitments to these values.

Special-soundness. We prove special-soundness for the perfect complete version of Π_{LK}, the proof for Π_{LK} is almost identical (but even simpler).

Lemma 5.5. *The perfect complete version of Π_{LK} is $2(\ell+T)$-special-sound.*

Proof. We construct an extractor Ext that outputs $\mathbf{w}$. To show that the witness is valid, we look at the $2(\ell+T)$ transcripts that all have $\mathbf{w}, \mathbf{m}$ as the first message but different $(r^{(j)}, \mathbf{h}^{(j)} \in \mathbb{F}^{\ell}, \mathbf{g}^{(j)} \in \mathbb{F}^{T})$ as the second message. Note that by the pigeonhole principle, there must exist a subset of $S \subseteq [2(\ell+T)]$ transcripts such that $|S| = \ell + T$ and $\mathbf{w}_i + r^{(j)} \neq 0$ for all $i \in [\ell]$ and $j \in S$, and $\mathbf{t}_i + r^{(j)} \neq 0$ for all $i \in [T]$ and $j \in S$. For these transcripts, we have that $\mathbf{h}_i = \frac{1}{\mathbf{w}_i + r^{(j)}}$ and $\mathbf{g}_i = \frac{\mathbf{m}_i}{\mathbf{t}_i + r^{(j)}}$. Define the degree $\ell + T - 1$ polynomial

$$p(X) = \prod_{k=1}^{\ell}(X + \mathbf{w}_k) \cdot \prod_{j=1}^{T}(X + \mathbf{t}_j) \cdot \left(\sum_{i=1}^{\ell} \frac{1}{X + \mathbf{w}_i} - \sum_{i=1}^{T} \frac{\mathbf{m}_i}{X + \mathbf{t}_i} \right).$$

If $p(X)$ is the zero polynomial then $\sum_{i=1}^{\ell} \frac{1}{X+\mathbf{w}_i} = \sum_{i=1}^{T} \frac{\mathbf{m}_i}{X+\mathbf{t}_i}$ and by Lemma 5.4 $(\mathcal{C}_{\mathrm{LK}}; \mathbf{w}) \in \mathcal{R}_{\mathrm{LOOKUP}}$. Since we have $\ell + T$ points $r^{(j)}$ at which $p(r_j) = 0$ we get that $p = 0$ and thus that the extracted witness $\mathbf{w}$ is valid. □

5.4.4 Vector-valued lookup

In some applications (e.g., simulating bit operations in circuits), we need to support lookup for a vector, i.e., each table value is a vector of field elements. In this section, we adapt the scheme in Section 5.4.3 to support vector lookups.

Definition 5.13. *Consider configuration $\mathcal{C}_{VLK} := (T, \ell, v \in \mathbb{N}, \mathbf{t})$ where ℓ is the number of lookups, and $\mathbf{t} \in (\mathbb{F}^v)^T$ is a lookup table in which the ith $(1 \leq i \leq T)$ entry is*

$$\mathbf{t}_i := (\mathbf{t}_{i,1}, \ldots, \mathbf{t}_{i,v}) \in \mathbb{F}^v .$$

A sequence of vectors $\mathbf{w} \in (\mathbb{F}^v)^{\ell}$ is in relation $\mathcal{R}_{VLK}$ if and only if for all $i \in [\ell]$,

$$\mathbf{w}_i := (\mathbf{w}_{i,1}, \ldots, \mathbf{w}_{i,v}) \in \mathbf{t} .$$

As noted in Section 3.4 of [Hab22b], we can extend Lemma 5.4 and replace Equation 5.5 with

$$\sum_{i=1}^{\ell} \frac{1}{X + w_i(Y)} = \sum_{i=1}^{T} \frac{\mathbf{m}_i}{X + t_i(Y)} \tag{5.6}$$

where the polynomials are defined as

$$w_i(Y) := \sum_{j=1}^{v} \mathbf{w}_{i,j} \cdot Y^{j-1}, \quad t_i(Y) := \sum_{j=1}^{v} \mathbf{t}_{i,j} \cdot Y^{j-1},$$

which represent the witness vector $\mathbf{w}_i \in \mathbb{F}^v$ and the table vector $\mathbf{t}_i \in \mathbb{F}^v$. We, therefore, can describe a special-sound protocol for the vector lookup relation as follows.

Special-sound protocol Π^v_{VLK} for $\mathcal{R}_{\text{VLK}}$

Prover $\mathsf{P}(\mathcal{C}_{\mathbf{VLK}}, \mathbf{w} \in (\mathbb{F}^v)^\ell)$		**Verifier** $\mathsf{V}(\mathcal{C}_{\mathbf{VLK}})$
Compute $\mathbf{m} \in \mathbb{F}^T$ such that		
$\mathbf{m}_i := \sum_{j=1}^{\ell} \mathbb{1}(\mathbf{w}_j = \mathbf{t}_i) \forall i \in [T]$	$\xrightarrow{\mathbf{w}, \mathbf{m}}$	
	$\xleftarrow{\beta}$	$\beta \leftarrow\!\!\$\ \mathbb{F}$
	$\xrightarrow{\perp}$	
	$\xleftarrow{r}$	$r \leftarrow\!\!\$\ \mathbb{F}$
Compute $[\beta_i = \beta^{i-1}]_{i=1}^v$		
and $\mathbf{h} \in \mathbb{F}^\ell$, $\mathbf{g} \in \mathbb{F}^T$		
$\mathbf{h}_i := \frac{1}{w_i(\beta) + r} \forall i \in [\ell]$		
$\mathbf{g}_i := \frac{\mathbf{m}_i}{t_i(\beta) + r} \forall i \in [T]$	$\xrightarrow{[\beta_i]_{i=1}^v, \mathbf{h}, \mathbf{g}}$	
		$\sum_{i=1}^{\ell} \mathbf{h}_i \stackrel{?}{=} \sum_{i=1}^{T} \mathbf{g}_i$
		$\mathbf{h}_i \cdot \left[\left(\sum_{j=1}^{v} \mathbf{w}_{i,j} \cdot \beta_j \right) + r \right] \stackrel{?}{=} 1 \forall i \in [\ell]$
		$\mathbf{g}_i \cdot \left[\left(\sum_{j=1}^{v} \mathbf{t}_{i,j} \cdot \beta_j \right) + r \right] \stackrel{?}{=} \mathbf{m}_i \forall i \in [T]$
		$\beta_{i+1} \stackrel{?}{=} \beta_i \cdot \beta \forall i \in [v-1], \beta_1 \stackrel{?}{=} 1$

Achieving perfect completeness. We can use the same trick in Section 5.4.3 to achieve perfect completeness for Π^v_{VLK}. Namely, the verifier sets $\mathbf{h}_i = 0$ or $\mathbf{g}_i = 0$ when $w_i(\beta)+r = 0$ or $t_i(\beta)+r = 0$ respectively. The verification equations become

$$(w_i(\beta_1, \ldots, \beta_v) + r) \cdot (\mathbf{h}_i \cdot (w_i(\beta_1, \ldots, \beta_v) + r) - 1) = 0$$

and

$$(t_i(\beta_1, \ldots, \beta_v) + r) \cdot (\mathbf{g}_i \cdot (t_i(\beta_1, \ldots, \beta_v) + r) - \mathbf{m}_i) = 0\,,$$

where $w_i(\beta_1, \ldots, \beta_v) := \left(\sum_{j=1}^{v} \mathbf{w}_{i,j} \cdot \beta_j\right)$ and $t_i(\beta_1, \ldots, \beta_v) := \left(\sum_{j=1}^{v} \mathbf{t}_{i,j} \cdot \beta_j\right)$. The degree of the verifier is 5. In practice, the negligible completeness error can likely be ignored without implementing these checks.

Accumulation complexity. Π_{VLK} is a 5-move protocol (i.e. $k = 3$) with the 2nd prover message being empty; the degree of the verifier is 3; the number of non-zero elements in the prover message is at most $(v + 3)\ell + v$. To ensure that the accumulation procedure only requires $O(v\ell)$ operations independent of T, we can apply the same trick as in Section 5.4.3 and compute all error terms from homomorphic commitments to $\mathbf{g}$ and $\mathbf{m}$. This works because the error terms are a linear function of $\mathbf{g}$ and $\mathbf{m}$ and scalars. This means the contributions of the not necessarily sparse $\mathsf{acc}.\mathbf{g}, \mathsf{acc}.\mathbf{m}$ to $\mathbf{e}_1, \mathbf{e}_2$ can be computed using the commitment homomorphism.

Special-soundness. We prove that the perfect complete version of Π^v_{VLK} is special-sound.

Lemma 5.6. *For any $v \in \mathbb{N}$, the perfect complete version of Π^v_{VLK} is $[1+(v-1)\cdot(\ell+T-1), 2(\ell+T)]$-special-sound.*

Proof. We construct an extractor Ext that outputs $\mathbf{w}$. To show that the witness is valid, we look at the $[1+(v-1)\cdot(\ell+T-1), 2(\ell+T)]$-tree of accepting transcripts. Note that for each depth-1 internal node u that fixes the message $(\mathbf{w}, \mathbf{m}, \beta)$, it has $2(\ell + T)$ different choices of challenge $r^{(j)}$. By the pigeonhole principle, there exists at least $\ell + T$ challenges r such that $t_i(\beta) + r \neq 0$ for all $i \in [T]$ and $w_i(\beta) + r \neq 0$ for all $i \in [\ell]$. Let $\mathbf{h}, \mathbf{g}$ be the last prover message in the corresponding leaf node. Since the transcript is accepting, we have that $\mathbf{h}_i = 1/(w_i(\beta) + r)$ for all $i \in [\ell]$, $\mathbf{g}_i = \mathbf{m}_i/(t_i(\beta) + r)$ for all $i \in [T]$, and $\sum_{i=1}^{\ell} \mathbf{h}_i = \sum_{i=1}^{T} \mathbf{g}_i$.

Define the bivariate polynomial where the degree of X is $\ell + T - 1$ and the degree of Y is at most $(v - 1) \cdot (\ell + T - 1)$,

$$p(X, Y) \quad = \quad \prod_{k=1}^{\ell}(X \ + \ w_k(Y)) \ \cdot \ \prod_{j=1}^{T}(X \ + \ t_j(Y)) \ \cdot \ \left(\sum_{i=1}^{\ell} \frac{1}{X + w_i(Y)} - \sum_{i=1}^{T} \frac{\mathbf{m}_i}{X + t_i(Y)}\right).$$

For every depth-1 internal node u, we denote by (r, β) the partial transcript for one of the u's

children whose challenge r satisfies $t_i(\beta) + r \neq 0$ for all $i \in [T]$ and $w_i(\beta) + r \neq 0$ for all $i \in [\ell]$. As argued in the previous paragraph, we observe that $\sum_{i=1}^{\ell} \frac{1}{r+w_i(\beta)} - \sum_{i=1}^{T} \frac{\mathbf{m}_i}{r+t_i(\beta)} = 0$, hence p evaluates to zero at point (r, β). Note that there are $(v-1)\cdot(\ell+T-1)+1$ depth-1 internal nodes (i.e. $(v-1)\cdot(\ell+T-1)+1$ different βs) and each node has $\ell+T$ children (i.e. $\ell+T$ different r) such that p evaluates to zero at point (r, β). Hence p is the zero polynomial and $\sum_{i=1}^{\ell} \frac{1}{X+w_i(Y)} = \sum_{i=1}^{T} \frac{\mathbf{m}_i}{X+t_i(Y)}$. Then by the extension of Lemma 5.4 described in Equation 5.6, we have $(\mathcal{C}_{\text{VLK}}, \mathbf{w}) \in \mathcal{R}_{\text{VLK}}$ and the extracted witness is valid. □

5.4.5 Circuit selection

We provide a sub-protocol for showing that a vector has a single one-bit (and zeros otherwise) at the location of a program counter *pc*. This is later used to select the appropriate circuit.

Definition 5.14. *For an integer n the relation $\mathcal{R}_{select}$ is the set of tuples $(\mathbf{b}, pc) \in \mathbb{F}^n \times \mathbb{F}$ such that $b_i = 0 \forall i \in [n] \setminus \{pc\}$ and if $pc \in [n]$ then $b_{pc} = 1$.*

Special-sound protocol Π_{select} for circuit selecting relation $\mathcal{R}_{\text{select}}$

Prover $\mathsf{P}(\mathbf{b} \in \mathbb{F}^n, pc \in \mathbb{F})$		**Verifier** V
	$\xrightarrow{\mathbf{b}, pc}$	
		$b_i \cdot (pc - i) \stackrel{?}{=} 0 \forall i \in [n]$
		$b_i \cdot (b_i - 1) \stackrel{?}{=} 0 \forall i \in [n]$

Complexity and security. Π_{select} is a 1-move protocol (i.e. $k = 1$); the degree of the verifier is 2.

5.5 Special-sound protocols for Plonkup relations

Definition 5.15. *Consider configuration $\mathcal{C}_{plonkup} := (n, T; \sigma; c, d, [\mathbf{s}_i, G_i]_{i=1}^m; L, \mathbf{t})$ where $\sigma : [cn] \to [cn]$ is a permutation, $(c, d, [\mathbf{s}_i, G_i]_{i=1}^m)$ are the parameters for the high-degree custom gates, $L \subseteq [cn]$ is the subset of indices for variables that have a lookup gate, $\mathbf{t} \in \mathbb{F}^T$ is the lookup table. The relation $\mathcal{R}_{PLONK+}$ is the set of tuples $(\mathsf{pi} \in \mathbb{F}^{\ell_{\mathsf{in}}}, \mathbf{w} \in \mathbb{F}^{cn})$ such that*

$$\mathbf{w} \in \mathcal{R}_\sigma \wedge \mathbf{w} \in \mathcal{R}_{GATE} \wedge \mathbf{w}_L \in \mathcal{R}_{LOOKUP} \wedge \mathbf{w}[1..\ell_{\mathsf{in}}] = \mathsf{pi}\,.$$

We present the special-sound protocol for the Plonkup relation $\mathcal{R}_{\text{PLONK+}}$ below.

Special-sound protocol Π_{plonkup} for relation $\mathcal{R}_{\text{PLONK+}}$

Prover $\mathsf{P}(\mathcal{C}_{\mathbf{plonkup}}, \mathsf{pi}, \mathbf{w})$		**Verifier** $\mathsf{V}(\mathcal{C}_{\mathbf{plonkup}}, \mathsf{pi})$
Compute $\mathbf{m} \in \mathbb{F}^T$ such that		
$\mathbf{m}_i := \sum_{j \in L} \mathbb{1}(\mathbf{w}_j = \mathbf{t}_i) \forall i \in [T]$	$\xrightarrow{\mathbf{w}, \mathbf{m}}$	$\mathbf{w}_i - \mathbf{w}_{\sigma(i)} \stackrel{?}{=} 0 \forall i \in [cn]$
		$\sum_{j=1}^{m} \mathbf{s}_{j,i} \cdot G_j(\mathbf{w}_i, \ldots, \mathbf{w}_{i+cn-n}) \stackrel{?}{=} 0 \forall i \in [n]$
		$\mathbf{w}_i \stackrel{?}{=} \mathsf{pi}_i \forall i \in [\ell_{\mathsf{in}}]$
	$\xleftarrow{r}$	$r \leftarrow\!\!\$\ \mathbb{F}$
Compute $\mathbf{h} \in \mathbb{F}^{\lvert L \rvert}$, $\mathbf{g} \in \mathbb{F}^T$		
$\mathbf{h}_i := \frac{1}{\mathbf{w}_{L_i} + r} \forall i \in [\lvert L \rvert]$		
$\mathbf{g}_i := \frac{\mathbf{m}_i}{\mathbf{t}_i + r} \forall i \in [T]$	$\xrightarrow{\mathbf{h}, \mathbf{g}}$	$\sum_{i=1}^{\lvert L \rvert} \mathbf{h}_i \stackrel{?}{=} \sum_{i=1}^{T} \mathbf{g}_i$
		$\mathbf{h}_i \cdot (\mathbf{w}_{L_i} + r) \stackrel{?}{=} 1 \forall i \in [\lvert L \rvert]$
		$\mathbf{g}_i \cdot (\mathbf{t}_i + r) \stackrel{?}{=} \mathbf{m}_i \forall i \in [T]$

Complexity. Π_{plonkup} is a 3-move protocol (i.e. $k = 2$); the degree of the verifier is d; the number of non-zero elements in the prover message is at most $cn + 3|L|$.

Completeness and security. We need to add the checks described in Section 5.4.3 to achieve perfect completeness. This changes the verification degree to $\max(d, 3)$. Without these checks, the protocol still has all but negligible completeness.

Lemma 5.7. *$\Pi_{plonkup}$ is $2(T + |L|)$-special-sound.*

Proof. The protocol is a parallel composition of $\Pi_\sigma, \Pi_{\text{GATE}}$ and Π_{LK} plus a public input check. In Π_σ and Π_{GATE}, the prover simply sends the witness, and the verifier checks it is in the relation. These protocols are thus trivially 1-special-sound, i.e. perfectly sound. The public input relation also trivially holds as the verifier checks $\mathbf{w}_i = \mathsf{pi}_i$ for all $i \in [\ell_{\mathsf{in}}]$. By Lemma 5.5 Π_{LK} is $2(T + |L|)$-special-sound. Thus Π_{plonkup} is $2(T + |L|)$-special-sound. □

5.6 Protostar

In this section, we describe ProtoStar. ProtoStar is built using a special-sound protocol for capturing non-uniform Plonkup circuit computations. In particular, the relation is checking that one of the I circuits is satisfied, where the index of the target circuit is determined by a part of the public input called program counter pc. The non-uniform Plonkup circuit can add arbitrary constraints on input pc. For example, the list of I circuits can be the opcodes supported by EVM, the program counter pc can be computed from the pc' and the register state in the previous step, and the circuit will further check that $\mathsf{opcode}[pc]$ is executed correctly in the current step. For another application, we can consider the I circuits as the predicates of I smart contracts (or transaction types), a user can call one of the smart contracts/transaction types by specifying the index pc, and the cost of proving correct execution is only proportional to the size of an individual smart contract/transaction type rather than the sum of the sizes of the supported smart contracts/transaction types.

For ease of exposition, we assume that the I circuits have the same

- number of gates n;
- gate arity c;
- maximum gate degree d;
- number of gate types m;
- number of public inputs ℓ_{in};
- number of lookup gates ℓ_{lk}.

The scheme naturally extends when different branch circuits have different parameters.

Definition 5.16. *Consider configuration* $\mathcal{C}_{mplkup} := \left(\mathsf{pp} = [n, T, c, d, m, \ell_{\mathsf{in}}, \ell_{\mathsf{lk}}]; [\mathcal{C}_i]_{i=1}^{I}; \mathbf{t}\right)$ *where the* i*th* $(1 \leq i \leq I)$ *branch circuit has configuration* $\mathcal{C}_i := (\mathsf{pp}, \sigma_i, [\mathbf{s}_{i,j}, G_{i,j}]_{j=1}^{m}, L_i)$*, and* $\mathbf{t} \in \mathbb{F}^T$ *is the global lookup table. For a public input* $\mathsf{pi} := (pc, \mathsf{pi}') \in \mathbb{F}^{\ell_{\mathsf{in}}}$ *where* $pc \in [I]$ *is a program counter, we say that a instance-witness pair* $(\mathsf{pi}, \mathbf{w} \in \mathbb{F}^{cn})$ *is in the relation* $\mathcal{R}_{mplkup}$ *if and only if* $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}_{PLONK+}$ *w.r.t. circuit configuration* $(\mathcal{C}_{pc}, \mathbf{t})$.

Protocol $\Pi_{\mathbf{mplkup}} = \langle \mathsf{P}(\mathcal{C}_{\mathbf{mplkup}}, \mathsf{pi}, \mathbf{w}), \mathsf{V}(\mathcal{C}_{\mathbf{mplkup}}, \mathsf{pi} = (pc \in [I], \mathsf{pi}')) \rangle$:

1. P sends V vector $\mathbf{b} = (0, \ldots, 0, b_{pc} = 1, 0, \ldots, 0) \in \mathbb{F}^I$.
2. V checks that $b_i \cdot (1 - b_i) \stackrel{?}{=} 0$ and $b_i \cdot (i - pc) \stackrel{?}{=} 0$ for all $i \in [I]$.
3. P computes vector $\mathbf{m} \in \mathbb{F}^T$ such that $\mathbf{m}_i := \sum_{j \in L_{pc}} \mathbb{1}(\mathbf{w}_j = \mathbf{t}_i) \forall i \in [T]$.
4. P sends V vectors $\mathbf{w}, \mathbf{m}$.
5. V checks that

   Permutation check: $\sum_{j=1}^{I} b_j (\mathbf{w}_i - \mathbf{w}_{\sigma_j(i)}) \stackrel{?}{=} 0$ for all $i \in [cn]$.

   Public input check: $\mathbf{w}[1..\ell_{\mathsf{in}}] \stackrel{?}{=} \mathsf{pi}$.

   Gate check: for all $i \in [n]$, it holds that

$$\sum_{j=1}^{I} b_j \cdot \mathsf{GT}_j(\mathbf{s}_{j,1}[i], \ldots, \mathbf{s}_{j,m}[i], \mathbf{w}_i, \ldots, \mathbf{w}_{i+cn-n}) = 0$$

   where $\mathsf{GT}_j(s_1, \ldots, s_m, x_1, \ldots, x_c) := \sum_{i=1}^{m} s_i \cdot G_{j,i}(x_1, \ldots, x_c)$.
6. V samples and sends P random challenge $r \leftarrow\!\!\$\, \mathbb{F}$.
7. P computes vectors $\mathbf{h} \in \mathbb{F}^{\ell_{\mathsf{lk}}}$, $\mathbf{g} \in \mathbb{F}^T$ such that

$$\mathbf{h}_i := \frac{1}{\mathbf{w}_{L_{pc}[i]} + r} \forall i \in [\ell_{\mathsf{lk}}], \qquad \mathbf{g}_i := \frac{\mathbf{m}_i}{\mathbf{t}_i + r} \forall i \in [T].$$

8. V checks that $\sum_{i=1}^{\ell_{\mathsf{lk}}} \mathbf{h}_i \stackrel{?}{=} \sum_{i=1}^{T} \mathbf{g}_i$ and

$$\sum_{j=1}^{I} b_j \cdot \left[\mathbf{h}_i \cdot (\mathbf{w}_{L_j[i]} + r)\right] \stackrel{?}{=} 1 \quad \forall i \in [\ell_{\mathsf{lk}}],$$

$$\mathbf{g}_i \cdot (\mathbf{t}_i + r) \stackrel{?}{=} \mathbf{m}_i \quad \forall i \in [T]$$

We present the special-sound protocol Π_{mplkup} for the multi-circuit Plonkup relation.

Remark 5.6.1. *By the public input check $\mathbf{w}[1..\ell_{\mathsf{in}}] \stackrel{?}{=} \mathsf{pi}$, we guarantee that $\mathbf{w}[1] = pc$, and the circuit relation can add arbitrary constraints on pc depending on the applications (like the ones described in Section 5.6).*

Complexity. Π_{mplkup} is a 3-move protocol (i.e. $k = 2$); the degree of the verifier is $d + 1$; the number of non-zero elements in the prover message is at most $cn + 3\ell_{\mathsf{lk}} + 1$; the prover message length

is $I + cn + 3T$. Hence in the resulting accumulation scheme, the accumulation prover complexity is only $O(n + \ell_{\mathsf{lk}})$ that is independent of the table size, and the accumulator size is $O(n + T + I)$ that is independent of the sum of the sizes of the branch circuits. The decider still runs in time $O(I \cdot c \cdot n)$ as it needs to evaluate all circuits at the accumulator.

Special-soundness. We prove the special-soundness property of Π_{mplkup} below.

Lemma 5.8. *Π_{mplkup} is $2(T + \ell_{\mathsf{lk}})$-special-sound.*

Proof. The extractor Ext outputs the witness $\mathbf{w}$ sent by the prover. Note that if the verifier checks in step 2 pass, it must be the case that $\mathbf{b}$ is a bool vector with a single non-zero element b_{pc}. Also, note that given $2(T + \ell_{\mathsf{lk}})$ accepting transcripts with distinct challenges r, the vector $\mathbf{b}$ won't change. Therefore the sub-transcript after step 2 is essentially a transcript for a Plonkup special-sound protocol Π_{plonkup} with configuration $\mathcal{C}_{\text{plonkup}} := (n, T, c, d, \mathcal{C}_{pc}, \mathbf{t})$. By Lemma 5.7, it holds that Π_{mplkup} is $2(T + \ell_{\mathsf{lk}})$-special-sound. □

We will now use Π_{mplkup} and our compiler described in Theorem 5.3 to design ProtoStar. Before that, we must address an efficiency issue when combining the high-degree gate and sparse lookup protocols with the generic transform CV in Section 5.3.5.

Error separation between high-degree checks and low-degree checks. In some scenarios, the set of equations that a verifier checks can be with heterogeneous degrees. For example, in Section 5.3.5, the transformed protocol $\mathsf{CV}[\Pi_{\mathsf{sps}}]$ has one degree-$(d+2)$ checks and $O(\sqrt{\ell})$ degree-2 checks. In this case, we can partition the checks into two sets whereas one with high-degree equations and the other with low-degree equations. We then assign separate sets of error terms for the two sets of equations. The first set of error terms can use the trivial identity commitment (as shown in Section 5.3.5) and thus reduces the number of group operations. The rest of the error term vectors can be committed using group elements. Since the degree of the second set is low, the number of error commitments will be a small constant (e.g. 2), thus the number of additional group operations performed in the recursive circuit will be small. We refer to Section 5.7 for more details.

Efficient accumulation of $\mathsf{CV}[\Pi_{\mathbf{mplkup}}]$. $\mathsf{CV}[\Pi_{\text{GATE}}]$ reduces the number of degree-d verification checks in Π_{GATE} from n to 1, with the tradeoff of $O(\sqrt{n})$ additional degree-2 checks. In the resulting accumulation scheme, the error terms for high-degree gates are, thus, only of length 1. This enables using the trivial identity commitment for these error terms and thus reduces the number of group operations by the accumulation verifier. Unfortunately, applying CV to mplkup seems to have a major tradeoff. The number of verification checks is $n + \ell_{\mathsf{lk}} + T + c \cdot n$. This requires using a) CV[mplkup] and b) is not composable with the sparseness optimizations for lookup described in Sections 5.4.3 and 5.4.4. These optimizations make the prover computation independent of T.

Fortunately, a closer look at the verification of mplkup reveals that only n of these verification checks are of high degree d, namely the checks in Π_{GATE}. The other checks are of degree 2 or lower. With a slight abuse of notation, we can define $\mathsf{CV}[\Pi_{\text{mplkup}}]$ as applying the generic transform CV only to the Π_{GATE} part of Π_{mplkup}. This means that there are $d+1$ cross error vectors (each of length 1) for the degree $d+2$ check in $\mathsf{CV}[\Pi_{\text{GATE}}]$; and 1 cross error vector of length $T + \ell_{\mathsf{lk}} + cn + O(\sqrt{n})$ for the rest checks—namely the low-degree checks in Π_{mplkup} and the $O(\sqrt{n})$ degree-2 checks in $\mathsf{CV}[\Pi_{\text{GATE}}]$. By leveraging the error separation technique described above, we can use the identity function to commit to the field elements and a vector commitment to commit to the long error term. We can again leverage homomorphism as described in Section 5.4.3 to make the prover independent of T.

Corollary 5.4 (ProtoStar protocol)**.** *Consider the configuration*

$$\mathcal{C}_{mplkup} := (n, T, c, d, m, \ell_{\mathsf{in}}, \ell_{\mathsf{lk}}; [\mathcal{C}_i]_{i=1}^{I}; \mathbf{t}).$$

Given a binding homomorphic commitment scheme $\mathsf{cm} = (\mathsf{Setup}, \mathsf{Commit})$*, and under the Fiat-Shamir Heuristic (Definition 5.9) for a hash function* H*, there exists an IVC scheme* ProtoStar *for* $\mathcal{R}_{mplkup}$ *relations with the following efficiencies for* $m = 1$ *(i.e. each circuit has a single degree-d gate type), public input length* $\ell_{\mathsf{in}} = 1$*: (we omit cost terms that are negligible compared to the dominant parts)*

$\mathsf{P}_{\text{ProtoStar}}$ *native*	$\mathsf{P}_{\text{ProtoStar}}$ *recursive*	$\mathsf{V}_{\text{ProtoStar}}$	$\lvert\pi_{\text{ProtoStar}}\rvert$
$O(\lvert\mathbf{w}\rvert + \ell_{\mathsf{lk}})\mathbb{G}$ $L'(\mathcal{C}_i, d+2) + 2\ell_{\mathsf{lk}}\mathbb{F}$	$3\mathbb{G}$ $d + 4\mathbb{F}$ $d + O(1)H + 1\mathsf{H}_{\mathsf{in}}$	$O(c \cdot n + T + \ell_{\mathsf{lk}})\mathbb{G}$ $n + \sum_{i=1}^{I} \mathcal{C}_i + T + \ell_{\mathsf{lk}}\mathbb{F}$	$O(c \cdot n + T + \ell_{\mathsf{lk}})$

Here $\lvert\mathbf{w}\rvert \leq cn$ *is the number of non-zero entries in the witness,* $\sum_{i=1}^{I} \mathcal{C}_i$ *is the cost of evaluating all circuits on some random input, and* $L'(\mathcal{C}_i, d)$ *is the cost of computing the coefficients of the polynomial* $e(X)$ *defined in Equation 5.4.*[5] H_{in} *is the cost of hashing the public input and the (constant-sized) accumulator instance.*

Proof. Let $\mathsf{SPS} - \mathsf{IVC}[\Pi] = \mathsf{IVC}[\mathsf{acc}[\mathsf{FS}[\mathsf{cm}[\mathsf{CV}[\Pi]]]]]$ be the transformation from a special-sound protocol to an IVC-scheme described by Theorem 5.3 (including CV). Then given a commitment scheme cm by that theorem ProtoStar $= \mathsf{SPS} - \mathsf{IVC}[\Pi_{\text{mplkup}}]$ is an IVC scheme for predicates expressed in $\mathcal{R}_{\text{mplkup}}$. We apply Theorem 5.3 to get the efficiencies in the table above.

[5] As noted in Theorem 5.3, $L'(\mathcal{C}_i, d+2)$ is bounded by $O(nd\log^2(d))$.

Security: Since $\mathsf{CV}[\Pi_{\mathrm{GATE}}]$ is only applied to Π_{GATE} which has perfect soundness, by Lemma 5.8 and Lemma 5.3, the NARK scheme $\mathsf{FS}[\mathsf{cm}[\mathsf{CV}[\Pi]]]$ for $\mathcal{R}_{\mathrm{mplkup}}$ has knowledge soundness with knowledge error $(Q+1)\cdot\frac{n+2(T+\ell_{\mathsf{lk}})}{|\mathbb{F}|}+\mathsf{negl}(\lambda)$, where Q is the number of RO queries by the adversary. Using Theorem 5.2 and Corollary 5.5 we can construct a accumulation scheme for the NARK scheme $\mathsf{FS}[\mathsf{cm}[\mathsf{CV}[\Pi]]]$. The accumulation scheme has negligible knowledge error as $d=\mathsf{poly}(\lambda)$. Therefore, under the Fiat-Shamir heuristic and by Theorem 5.1, $\mathsf{SPS}-\mathsf{IVC}[\Pi]$ is a secure IVC scheme. □

5.7 Accumulation Scheme for high/low degree verifier

We describe a modification for the accumulation scheme in Section 5.3.4 that can be useful if $\mathsf{V}_{\mathsf{sps}}$ has both a single high-degree verification check and multiple low-degree checks. E.g. assume that $\mathsf{V}_{\mathsf{sps}}=\mathsf{V}_{\mathsf{sps},1}||\mathsf{V}_{\mathsf{sps},2}$ where $\mathsf{V}_{\mathsf{sps},1}:(\mathsf{pi},[\mathbf{m}_i]_{i=1}^k,[r_i]_{i=1}^{k-1})\to\mathbb{F}$ is degree d and $\mathsf{V}_{\mathsf{sps},2}:(\mathsf{pi},[\mathbf{m}_i]_{i=1}^k,[r_i]_{i=1}^{k-1})\to\mathbb{F}^\ell$ is degree 2.

For simplicity, we assume that $\mathsf{V}_{\mathsf{sps},1}$ maps to a single field element and that $\mathsf{V}_{\mathsf{sps},2}$ is degree 2, but this naturally extends to more arbitrary degrees, sizes and more components.

The accumulation scheme $\mathsf{acc}_{\mathsf{HL}}=(\mathsf{P}_{\mathsf{acc},\mathsf{HL}},\mathsf{V}_{\mathsf{acc},\mathsf{HL}})$ for $\mathsf{FS}[\mathsf{V}_{\mathsf{sps},1}||\mathsf{V}_{\mathsf{sps},2}]$ is essentially a parallel composition of the accumulation presented Section 5.3.4 applied to $\mathsf{V}_{\mathsf{sps},1}$ and $\mathsf{V}_{\mathsf{sps},2}$. Concretely there are the following modifications to the scheme from Section 5.3.4:

- The prover computes error terms separately for $\mathsf{V}_{\mathsf{sps},1}$ and $\mathsf{V}_{\mathsf{sps},2}$. This means there are $d-1$ constant size error terms and 1 error term vector of length ℓ.
- The prover uses the identity function to commit to the d error terms and a homomorphic vector commitment $\mathsf{cm}=(\mathsf{Setup},\mathsf{Commit})$ to commit to the single length ℓ error term.
- The accumulator stores two error terms, one for each verifier. One is a field element e, and the other is a commitment E to a length ℓ vector.
- The accumulation verifier checks the correct accumulation for each error term separately, thus performing $d-1$ field operations and 1 homomorphic commitment scalar multiplication.

Complexity and security. The scheme has the following complexity:

- The *accumulation prover*
  - asks $k-1$ queries to ρ_{NARK} with constant-sized inputs and 1 query to ρ_{acc} with input size $d+O(1)$;
  - computes the coefficients of

$$e(X)=\sum_{j=0}^{d}(\mu+X)^{d-j}f_j^{\mathsf{V}_{\mathsf{sps},1}}(X\cdot\mathsf{pi}+\mathsf{acc.pi},[X\cdot\mathbf{m}_i+\mathsf{acc.}\mathbf{m}_i]_{i=1}^k,[X\cdot r_i+\mathsf{acc.}r_i]_{i=1}^{k-1})\in\mathbb{F}[X]$$

and computes $\mathbf{e} \in \mathbb{F}^\ell$, which are the coefficients of X in the polynomials

$$\sum_{j=0}^{2}(\mu+X)^{2-j} f_j^{\mathsf{V}_{\mathsf{sps},2}}(X \cdot \mathsf{pi} + \mathsf{acc.pi}, [X \cdot \mathbf{m}_i + \mathsf{acc}.\mathbf{m}_i]_{i=1}^{k}, [X \cdot r_i + \mathsf{acc}.r_i]_{i=1}^{k-1}) \in (\mathbb{F}[X])^\ell$$

  - commits to $\mathbf{e}$ using ℓ $\mathbb{G}$ ops.
  - performs $|R|+|M^*|+2$ $\mathbb{F}$-ops to combine $(\mu, \mathsf{pi}, [r_i]_{i=1}^{k-1}, [\mathbf{m}_i]_{i=1}^{k})$ (where $|R|$ is the number of challenges and $|M^*|$ is the number of non-zero elements in prover messages);
  - performs k $\mathbb{G}$-ops to combine $[C_i]_{i=1}^{k}$;
  - performs 1 $\mathbb{G}$-op to add $E = \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$ to $\mathsf{acc}.E$.
- The *accumulation verifier* performs
  - asks $k-1$ queries to ρ_{NARK} and 1 query to ρ_{acc};
  - $|R|+2$ $\mathbb{F}$-ops to combine $(\mu, \mathsf{pi}, [r_i]_{i=1}^{k-1})$;
  - k $\mathbb{G}$-ops to combine $[C_i]_{i=1}^{k}$;
  - $d-1$ $\mathbb{F}$-ops to add $[e_j]_{j=1}^{d-1}$ onto $\mathsf{acc}.e$.
  - 1 $\mathbb{G}$-op to add $E = \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$ to $\mathsf{acc}.E$
- The *decider*
  - computes $C_i = \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$ for $i \in [k]$ and $E = \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$, with total complexity around $|M| + \ell$ $\mathbb{G}$-ops.
  - evaluate $[f_i^{\mathsf{V}_{\mathsf{sps},1}}]_{i=0}^{d}$ and $[f_i^{\mathsf{V}_{\mathsf{sps},2}}]_{i=0}^{2}$ to verify e and $\mathbf{e}$.

Corollary 5.5 (Hi-LowAccumulation)**.** *Let* $(\mathsf{P}_{NARK,\mathsf{HL}}, \mathsf{V}_{NARK,\mathsf{HL}}) = \mathsf{FS}[\mathsf{V}_{\mathsf{sps},1}||\mathsf{V}_{\mathsf{sps},2}]$ *be an RO-NARK as defined above. Let* cm *be a binding, homomorphic commitment scheme and* ρ_{acc} *be a random oracle. The accumulation scheme* $\mathsf{acc}_{\mathsf{HL}}$ *for* V_{NARK} *satisfies perfect completeness and has knowledge-error* $(Q+1)\frac{d+4}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$*, as defined in Definition 5.8.*

Proof sketch: Perfect completeness follows immediately from Theorem 5.2. For knowledge-soundness, consider that $\mathsf{acc}_{\mathsf{HL}}$ is a parallel composition of two accumulation schemes applied to a high-degree and a low-degree verifier. Given an adversary that can break the knowledge soundness of $\mathsf{acc}_{\mathsf{HL}}$, we can construct an adversary that can break the knowledge soundness of either the high or the low-degree accumulation. By union bound, this leads to a knowledge error of $(Q+1)\frac{d+4}{|\mathbb{F}|} + \mathsf{negl}(\lambda)$.

5.8 Protostar for CCS

Recently, Setty, Thaler and Wahby introduced Customizable Constraint System (CCS), a new characterization of NP that is a generalization of R1CS[STW23]. It enables the use of high-degree gates

while not requiring permutation arguments. It is also powerful enough to capture both R1CS and Plonk constraints. As described in the introduction, HyperNova builds an accumulation scheme and, thus, IVC for CCS. However, HyperNova does not natively support non-uniform circuits and both the recursion cost, as well as the cost for lookups is more expensive than ProtoStar, which is built for mplkup. However, mplkup still requires a permutation argument and so-called copy-constraints. We show that our general compiler is powerful enough to port the benefits of ProtoStar directly to CCS. The starting point is the trivial special-sound protocol for CCS:

Definition 5.17 ([STW23])**.** *Given public parameters $m, n, N, \ell_{\mathsf{in}}, t, q, d \in \mathbb{N}$ where $n > \ell$, let $M_1, \ldots, M_t \in \mathbb{F}^{m \times n}$ be matrices with at most N total non-zero entries. Let $S_1, \ldots S_q$ be multi-sets over domain $[t]$ and each multiset has cardinality at most d. Let $c_1, \ldots, c_q \in \mathbb{F}$ be constants. A tuple $(\mathsf{pi}, \mathbf{w}) \in \mathbb{F}^{\ell_{\mathsf{in}}} \times \mathbb{F}^{n-\ell_{\mathsf{in}}-1}$ is in the relation $\mathcal{R}_{ccs}$ if and only if*

$$\sum_{i=0}^{q-1} c_i \cdot \bigcirc_{j \in S_j} M_j \cdot \mathbf{z} = \mathbf{0}^m \,,$$

where $z = (1, \mathsf{pi}, \mathbf{w})$. $\bigcirc$ denotes the Hadamard product between vectors.

Special-sound protocol Π_{ccs} for relation $\mathcal{R}_{\text{ccs}}$

Prover $\mathsf{P}(\mathcal{C}_{\mathbf{ccs}}, \mathsf{pi} \in \mathbb{F}^{\ell_{\mathsf{in}}}, \mathbf{w} \in \mathbb{F}^{n-\ell_{\mathsf{in}}-1})$		**Verifier** $\mathsf{V}(\mathcal{C}_{\mathbf{ccs}}, \mathsf{pi})$
	$\xrightarrow{\quad \mathbf{w} \quad}$	
		$\mathbf{z} = (1, \mathsf{pi}, \mathbf{w}) \in \mathbb{F}^n$
		$\sum_{i=0}^{q-1} c_i \cdot \bigcirc_{j \in S_j} M_j \cdot \mathbf{z} \stackrel{?}{=} \mathbf{0}^m$

Complexity. Π_{ccs} is a 1-move protocol (i.e. $k = 1$) with verifier degree d.

Next, we present the special-sound protocol $\Pi_{\text{mccs+}}$ for the extended CCS relation that has multi-circuits and lookup support. Compared to Π_{mplkup} in Section 5.6, the protocol $\Pi_{\text{mccs+}}$ removes the need of a permutation check.

Definition 5.18. *Consider configuration $\mathcal{C}_{mccs+} := \big(\mathsf{pp} = [m, n, N, \ell_{\mathsf{in}}, t, q, d, T, \ell_{\mathsf{lk}}]; [\mathcal{C}_i]_{i=1}^{I}; \mathbf{t}\big)$ where the ith $(1 \le i \le I)$ branch circuit has configuration $\mathcal{C}_i := (\mathsf{pp}, [M_{j,i}]_{j=1}^{t}, [S_{j,i}, c_{j,i}]_{j=1}^{q}, L_i)$, and $\mathbf{t} \in \mathbb{F}^T$ is the global lookup table. For a public input $\mathsf{pi} := (pc, \mathsf{pi}') \in \mathbb{F}^{\ell_{\mathsf{in}}}$ where $pc \in [I]$ is a program counter, we say that a instance-witness pair $(\mathsf{pi}, \mathbf{w}) \in \mathbb{F}^{n-1}$ is in the relation $\mathcal{R}_{mccs+}$ if and only if $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}_{ccs}$ w.r.t. circuit configuration $\mathcal{C}_{pc}$ and $\mathbf{w}_{L_{pc}} \subseteq \mathbf{t}$.*

Protocol $\Pi_{\mathbf{mccs+}} = \langle \mathsf{P}(\mathcal{C}_{\mathbf{mccs+}}, \mathsf{pi}, \mathbf{w}), \mathsf{V}(\mathcal{C}_{\mathbf{mccs+}}, \mathsf{pi} = (pc \in [I], \mathsf{pi}')) \rangle$**:**

1. P sends V vector $\mathbf{b} = (0, \ldots, 0, b_{pc} = 1, 0, \ldots, 0) \in \mathbb{F}^I$.
2. V checks that $b_i \cdot (1 - b_i) \stackrel{?}{=} 0$ and $b_i \cdot (i - pc) \stackrel{?}{=} 0$ for all $i \in [I]$.
3. P computes vector $\mathbf{m} \in \mathbb{F}^T$ such that $\mathbf{m}_i := \sum_{j \in L_{pc}} \mathbb{1}(\mathbf{w}_j = \mathbf{t}_i) \forall i \in [T]$.
4. P sends V vectors $\mathbf{w}, \mathbf{m}$.
5. V computes $\mathbf{z} = (1, \mathsf{pi}, \mathbf{w}) \in \mathbb{F}^n$ and checks that
$$\sum_{k=1}^{I} b_k \cdot \sum_{i=1}^{q} c_{i,k} \cdot \bigcirc_{j \in S_{i,k}} M_{j,k} \cdot \mathbf{z} = \mathbf{0}^m .$$
6. V samples and sends P random challenge $r \leftarrow\!\!\$\ \mathbb{F}$.
7. P computes vectors $\mathbf{h} \in \mathbb{F}^{\ell_{\mathsf{lk}}}$, $\mathbf{g} \in \mathbb{F}^T$ such that
$$\mathbf{h}_i := \frac{1}{\mathbf{w}_{L_{pc}[i]} + r} \forall i \in [\ell_{\mathsf{lk}}] , \qquad \mathbf{g}_i := \frac{\mathbf{m}_i}{\mathbf{t}_i + r} \forall i \in [T] .$$
8. V checks that $\sum_{i=1}^{\ell_{\mathsf{lk}}} \mathbf{h}_i \stackrel{?}{=} \sum_{i=1}^{T} \mathbf{g}_i$ and
$$\sum_{j=1}^{I} b_j \cdot \left[\mathbf{h}_i \cdot (\mathbf{w}_{L_j[i]} + r) \right] \stackrel{?}{=} 1 \quad \forall i \in [\ell_{\mathsf{lk}}] ,$$
$$\mathbf{g}_i \cdot (\mathbf{t}_i + r) \stackrel{?}{=} \mathbf{m}_i \quad \forall i \in [T] .$$

Complexity. $\Pi_{\text{mccs+}}$ is a 3-move protocol (i.e. $k = 2$); the degree of the verifier is $d + 1$; the number of non-zero elements in the prover message is at most $n + 3\ell_{\mathsf{lk}}$; the prover message length is at most $I + n + 3T$. Hence in the resulting accumulation scheme, the accumulation prover complexity is only $O(n + \ell_{\mathsf{lk}})$ that is independent of the table size, and the accumulator size is $O(n + T + I)$ that is independent of the sum of the sizes of the branch circuits. We detail the efficiency of the resulting IVC scheme in the table below. The efficiency is almost identical to the ProtoStar scheme for $\mathcal{R}_{\text{mplkup}}$. However, the cost of computing the error terms $L'(\text{ccs}_i, d + 2)$ now depends on the ith CCS instance.

$\mathsf{P}_{\text{ProtoStar,mccs+}}$ native	$\mathsf{P}_{\text{ProtoStar,mccs+}}$ recursive	$\mathsf{V}_{\text{ProtoStar,mccs+}}$	$\lvert\pi_{\text{ProtoStar,mccs+}}\rvert$
$O(\lvert\mathbf{w}\rvert + \ell_{\mathsf{lk}})\mathbb{G}$ $L'(\text{ccs}_i, d+2) + 2\ell_{\mathsf{lk}}\mathbb{F}$	$3\mathbb{G}$ $d + 4\mathbb{F}$ $d + O(1)H + 1\mathsf{H}_{\mathsf{in}}$	$O(n + T + \ell_{\mathsf{lk}})\mathbb{G}$ $n + I \cdot N + T + \ell_{\mathsf{lk}}\mathbb{F}$	$O(n + T + \ell_{\mathsf{lk}})$

Chapter 6

Conclusion

The hype around blockchains is built on big promises. Sending money like email around the internet, equitable and fair access to financial systems, strong censorship, and monopoly resistance. While, some applications are starting to realize these promises, the failures and limitations of blockchains are at least equally glaring. This might lead one to easily dismiss the space entirely, as many commentators have done. However, this is entirely premature. In this dissertation, we show that many of the most pressing issues can be directly addressed and solved through the use of technology. In particular, we show the power of proof systems that seem to be a perfect antidote for many of the issues blockchains face. Proof systems, such as the ones presented here, have, in the past decade, morphed from a theoretical idea to an entirely practical tool. This can not only be seen in academia but also in industry with many startups emerging and the rapid implementation of academic ideas. An exciting prospect, that is starting to emerge, is that those proof systems will find many more practical applications beyond blockchains. This includes trusted AI, verifiable outsourcing, and private auditing. This can enable a digital future in which privacy, scalability, and authenticity are not contradictory but symbiotic properties.

Bibliography

[Abe+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. "Structure-Preserving Signatures and Commitments to Group Elements". In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 209–236. DOI: 10.1007/978-3-642-14623-7_12.

[ABP17] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. "Depth-Robust Graphs and Their Cumulative Memory Complexity". In: *EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, Apr. 2017, pp. 3–32. DOI: 10.1007/978-3-319-56617-7_1.

[AC20] Thomas Attema and Ronald Cramer. "Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 513–543. DOI: 10.1007/978-3-030-56877-1_18.

[ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl. "A Compressed Σ-Protocol Theory for Lattices". In: *CRYPTO 2021, Part II*. Ed. by Tal Malkin and Chris Peikert. Vol. 12826. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 549–579. DOI: 10.1007/978-3-030-84245-1_19.

[AFK21] Thomas Attema, Serge Fehr, and Michael Klooß. *Fiat-Shamir Transformation of Multi-Round Interactive Proofs*. Cryptology ePrint Archive, Report 2021/1377. https://eprint.iacr.org/2021/1377. 2021.

[AFK22] Thomas Attema, Serge Fehr, and Michael Klooß. "Fiat-Shamir Transformation of Multi-round Interactive Proofs". In: *TCC 2022, Part I*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13747. LNCS. Springer, Heidelberg, Nov. 2022, pp. 113–142. DOI: 10.1007/978-3-031-22318-1_5.

[Alb+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and

Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 191–219. DOI: 10.1007/978-3-662-53887-6_7.

[Aly+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols". In: *IACR Trans. Symm. Cryptol.* 2020.3 (2020), pp. 1–45. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i3.1-45.

[Ame+17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2087–2104. DOI: 10.1145/3133956.3134104.

[And+13] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. "Evaluating User Privacy in Bitcoin". In: *FC 2013*. Ed. by Ahmad-Reza Sadeghi. Vol. 7859. LNCS. Springer, Heidelberg, Apr. 2013, pp. 34–51. DOI: 10.1007/978-3-642-39884-1_4.

[And17] Oleg Andreev. *Hidden in Plain Sight: Transacting Privately on a Blockchain*. blog.chain.com. 2017.

[ANL00] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. "DOS-resistant authentication with client puzzles". In: *International workshop on security protocols*. Springer. 2000, pp. 170–177.

[ano] anonymous. "Mimblewimble". In: ().

[Ara+21] Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. *ECLIPSE: Enhanced Compiling method for Pedersen-committed zkSNARK Engines*. Cryptology ePrint Archive, Report 2021/934. https://eprint.iacr.org/2021/934. 2021.

[Arm+16] Frederik Armknecht, Ludovic Barman, Jens-Matthias Bohli, and Ghassan O. Karame. "Mirror: Enabling Proofs of Data Replication and Retrievability in the Cloud". In: *USENIX Security 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, Aug. 2016, pp. 1051–1068.

[Bar+21] Mario Barbara, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lueftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. *Reinforced Concrete: Fast Hash Function for Zero Knowledge Proofs and Verifiable Computation*. Cryptology ePrint Archive, Report 2021/1038. https://eprint.iacr.org/2021/1038. 2021.

[BB04] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles". In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 56–73. DOI: 10.1007/978-3-540-24676-3_4.

[BC23] Benedikt Bünz and Binyi Chen. *ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols*. Cryptology ePrint Archive, Paper 2023/620. `https://eprint.iacr.org/2023/620`. 2023. URL: `https://eprint.iacr.org/2023/620`.

[BCG15] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. *On Bitcoin as a public randomness source*. Cryptology ePrint Archive, Report 2015/1015. `https://eprint.iacr.org/2015/1015`. 2015.

[BCG20] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. "Linear-Time Arguments with Sublinear Verification from Tensor Codes". In: *TCC 2020, Part II*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. LNCS. Springer, Heidelberg, Nov. 2020, pp. 19–46. DOI: `10.1007/978-3-030-64378-2_2`.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *TCC 2016-B, Part II*. Ed. by Martin Hirt and Adam D. Smith. Vol. 9986. LNCS. Springer, Heidelberg, Oct. 2016, pp. 31–60. DOI: `10.1007/978-3-662-53644-5_2`.

[BCS21] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. "Sumcheck Arguments and Their Applications". In: *CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 742–773. DOI: `10.1007/978-3-030-84242-0_26`.

[Bd94] Josh Cohen Benaloh and Michael de Mare. "One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract)". In: *EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. LNCS. Springer, Heidelberg, May 1994, pp. 274–285. DOI: `10.1007/3-540-48285-7_24`.

[Ben+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge". In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 90–108. DOI: `10.1007/978-3-642-40084-1_6`.

[Ben+14a] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 459–474. DOI: `10.1109/SP.2014.36`.

[Ben+14b] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *CRYPTO 2014, Part II*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8617. LNCS. Springer, Heidelberg, Aug. 2014, pp. 276–294. DOI: `10.1007/978-3-662-44381-1_16`.

[Ben+17a] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. "Computational Integrity with a Public Random String from Quasi-Linear PCPs". In: *EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, Apr. 2017, pp. 551–579. DOI: 10.1007/978-3-319-56617-7_19.

[Ben+17b] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. "Interactive Oracle Proofs with Constant Rate and Query Complexity". In: *ICALP 2017*. Ed. by Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl. Vol. 80. LIPIcs. Schloss Dagstuhl, July 2017, 40:1–40:15. DOI: 10.4230/LIPIcs.ICALP.2017.40.

[Ben+18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Fast Reed-Solomon Interactive Oracle Proofs of Proximity". In: *ICALP 2018*. Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. LIPIcs. Schloss Dagstuhl, July 2018, 14:1–14:17. DOI: 10.4230/LIPIcs.ICALP.2018.14.

[Ben+18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Report 2018/046. https://eprint.iacr.org/2018/046. 2018.

[Ben+19a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 701–732. DOI: 10.1007/978-3-030-26954-8_23.

[Ben+19b] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *EUROCRYPT 2019, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. LNCS. Springer, Heidelberg, May 2019, pp. 103–128. DOI: 10.1007/978-3-030-17653-2_4.

[Ber+12] Daniel J. Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. "Faster Batch Forgery Identification". In: *INDOCRYPT 2012*. Ed. by Steven D. Galbraith and Mridul Nandi. Vol. 7668. LNCS. Springer, Heidelberg, Dec. 2012, pp. 454–473. DOI: 10.1007/978-3-642-34931-7_26.

[BF22] Benedikt Bünz and Ben Fisch. *Schwartz-Zippel for multilinear polynomials mod N*. Cryptology ePrint Archive, Report 2022/458. https://eprint.iacr.org/2022/458. 2022.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK Compilers". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Heidelberg, May 2020, pp. 677–706. DOI: 10.1007/978-3-030-45721-1_24.

[BG12] Stephanie Bayer and Jens Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 263–280. DOI: 10.1007/978-3-642-29011-4_17.

[BGB17] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. "Proofs-of-delay and randomness beacons in Ethereum". In: *IEEE SECURITY and PRIVACY ON THE BLOCKCHAIN (IEEE S&B)* (2017).

[BGG19] Sean Bowe, Ariel Gabizon, and Matthew D. Green. "A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK". In: *FC 2018 Workshops*. Ed. by Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala. Vol. 10958. LNCS. Springer, Heidelberg, Mar. 2019, pp. 64–77. DOI: 10.1007/978-3-662-58820-8_5.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. https://eprint.iacr.org/2019/1021. 2019.

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. "Fast Batch Verification for Modular Exponentiation and Digital Signatures". In: *EUROCRYPT'98*. Ed. by Kaisa Nyberg. Vol. 1403. LNCS. Springer, Heidelberg, May 1998, pp. 236–250. DOI: 10.1007/BFb0054130.

[BGZ16] Iddo Bentov, Ariel Gabizon, and David Zuckerman. "Bitcoin Beacon". In: *arXiv preprint arXiv:1605.04559* (2016).

[Bit+12a] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again". In: *ITCS 2012*. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 326–349. DOI: 10.1145/2090236.2090263.

[Bit+12b] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. *Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data*. Cryptology ePrint Archive, Report 2012/095. https://eprint.iacr.org/2012/095. 2012.

[Bit+13a] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "Recursive composition and bootstrapping for SNARKS and proof-carrying data". In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 111–120. DOI: 10.1145/2488608.2488623.

[Bit+13b] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. "Succinct Non-interactive Arguments via Linear Interactive Proofs". In: *TCC 2013*. Ed. by Amit Sahai. Vol. 7785. LNCS. Springer, Heidelberg, Mar. 2013, pp. 315–333. DOI: 10.1007/978-3-642-36594-2_18.

[Bit+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. "Time-Lock Puzzles from Randomized Encodings". In: *ITCS 2016*. Ed. by Madhu Sudan. ACM, Jan. 2016, pp. 345–356. DOI: 10.1145/2840728.2840745.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 297–319. DOI: 10.1007/s00145-004-0314-9.

[BM88] László Babai and Shlomo Moran. "Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes". In: *J. Comput. Syst. Sci.* 36.2 (1988), pp. 254–276.

[BN00] Dan Boneh and Moni Naor. "Timed Commitments". In: *CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. LNCS. Springer, Heidelberg, Aug. 2000, pp. 236–254. DOI: 10.1007/3-540-44598-6_15.

[Bon+15] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies". In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 104–121. DOI: 10.1109/SP.2015.14.

[Bon+18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. "Verifiable Delay Functions". In: *CRYPTO 2018, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. LNCS. Springer, Heidelberg, Aug. 2018, pp. 757–788. DOI: 10.1007/978-3-319-96884-1_25.

[Bon+20a] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. *Coda: Decentralized Cryptocurrency at Scale*. Cryptology ePrint Archive, Report 2020/352. https://eprint.iacr.org/2020/352. 2020.

[Bon+20b] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. "Mina: Decentralized Cryptocurrency at Scale". In: *New York Univ. O (1) Labs, New York, NY, USA, Whitepaper* (2020), pp. 1–47.

[Bon+21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. "Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments". In: *CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 649–680. DOI: 10.1007/978-3-030-84242-0_23.

[Boo+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Heidelberg, May 2016, pp. 327–357. DOI: 10.1007/978-3-662-49896-5_12.

[Boo+17] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. "Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability". In: *ASIACRYPT 2017, Part III*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10626. LNCS. Springer, Heidelberg, Dec. 2017, pp. 336–365. DOI: 10.1007/978-3-319-70700-6_12.

[Boo+20] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. "A Non-PCP Approach to Succinct Quantum-Safe Zero-Knowledge". In: *CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. LNCS. Springer, Heidelberg, Aug. 2020, pp. 441–469. DOI: 10.1007/978-3-030-56880-1_16.

[Boo+22a] Jonathan Bootle, Alessandro Chiesa, Ziyi Guan, and Siqi Liu. *Linear-Time Probabilistic Proofs with Sublinear Verification for Algebraic Automata Over Every Field*. Cryptology ePrint Archive, Report 2022/1056. https://eprint.iacr.org/2022/1056. 2022.

[Boo+22b] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. "Gemini: Elastic SNARKs for Diverse Environments". In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Heidelberg, May 2022, pp. 427–457. DOI: 10.1007/978-3-031-07085-3_15.

[Bow+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. "ZEXE: Enabling Decentralized Private Computation". In: *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 947–964. DOI: 10.1109/SP40000.2020.00050.

[BPS16] Iddo Bentov, Rafael Pass, and Elaine Shi. *Snow White: Provably Secure Proofs of Stake*. Cryptology ePrint Archive, Report 2016/919. https://eprint.iacr.org/2016/919. 2016.

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *ACM CCS 93*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM Press, Nov. 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[BS12] Selcuk Baktir and Erkay Savas. *Highly-Parallel Montgomery Multiplication for Multi-core General-Purpose Microprocessors*. Cryptology ePrint Archive, Report 2012/140. https://eprint.iacr.org/2012/140. 2012.

[BS+22] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. "Elliptic Curve Fast Fourier Transform (ECFFT) Part II: Scalable and Transparent Proofs over All Large Fields". In: (2022).

[BSS08] Eli Ben-Sasson and Madhu Sudan. "Short PCPs with polylog query complexity". In: *SIAM Journal on Computing* 38.2 (2008), pp. 551–607.

[Bün+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[Bün+20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. "Recursive Proof Composition from Accumulation Schemes". In: *TCC 2020, Part II*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. LNCS. Springer, Heidelberg, Nov. 2020, pp. 1–18. DOI: 10.1007/978-3-030-64378-2_1.

[Bün+21a] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. "Proof-Carrying Data Without Succinct Arguments". In: *CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 681–710. DOI: 10.1007/978-3-030-84242-0_24.

[Bün+21b] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. "Proofs for Inner Pairing Products and Applications". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Heidelberg, Dec. 2021, pp. 65–97. DOI: 10.1007/978-3-030-92078-4_3.

[BX11] Joseph Bonneau and Rubin Xu. "Scrambling for lightweight censorship resistance". In: *International Workshop on Security Protocols*. Springer. 2011.

[Cai+93] Jin-yi Cai, Richard J. Lipton, Robert Sedgewick, and Andrew Chi-Chih Yao. "Towards Uncheatable benchmarks". In: *Structure in Complexity Theory*. 1993.

[Cam+17] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. "Zero-Knowledge Contingent Payments Revisited: Attacks and Payments for Services". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 229–243. DOI: 10.1145/3133956.3134060.

[Cam+21] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. "Lunar: A Toolbox for More Efficient Universal and Updatable zkSNARKs and Commit-and-Prove Extensions". In: *ASIACRYPT 2021, Part III*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13092. LNCS. Springer, Heidelberg, Dec. 2021, pp. 3–33. DOI: 10.1007/978-3-030-92078-4_1.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. "Efficient Protocols for Set Membership and Range Proofs". In: *ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. LNCS. Springer, Heidelberg, Dec. 2008, pp. 234–252. DOI: 10.1007/978-3-540-89255-7_15.

[CD17] Ignacio Cascudo and Bernardo David. "SCRAPE: Scalable Randomness Attested by Public Entities". In: *ACNS 17*. Ed. by Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi. Vol. 10355. LNCS. Springer, Heidelberg, July 2017, pp. 537–556. DOI: 10.1007/978-3-319-61204-1_27.

[CD98] Ronald Cramer and Ivan Damgård. "Zero-Knowledge Proofs for Finite Field Arithmetic; or: Can Zero-Knowledge Be for Free?" In: *CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. LNCS. Springer, Heidelberg, Aug. 1998, pp. 424–441. DOI: 10.1007/BFb0055745.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2075–2092. DOI: 10.1145/3319535.3339820.

[CFS17] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. *A Zero Knowledge Sumcheck and its Applications*. Cryptology ePrint Archive, Report 2017/305. https://eprint.iacr.org/2017/305. 2017.

[CH10] Jeremy Clark and Urs Hengartner. *On the Use of Financial Data as a Random Beacon*. Cryptology ePrint Archive, Report 2010/361. https://eprint.iacr.org/2010/361. 2010.

[Cha82] David Chaum. "Blind Signatures for Untraceable Payments". In: *CRYPTO'82*. Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, USA, 1982, pp. 199–203.

[Che+22] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. *HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates*. Cryptology ePrint Archive, Report 2022/1355. https://eprint.iacr.org/2022/1355. 2022.

[Che+23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: *EUROCRYPT 2023, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. LNCS. Springer, Heidelberg, Apr. 2023, pp. 499–530. DOI: 10.1007/978-3-031-30617-4_17.

[Chi+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Heidelberg, May 2020, pp. 738–768. DOI: 10.1007/978-3-030-45721-1_26.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. "Compact E-Cash". In: *EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 302–321. DOI: 10.1007/11426639_18.

[Cho+19] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. "Finding a Nash equilibrium is no easier than breaking Fiat-Shamir". In: *51st ACM STOC*. Ed. by Moses Charikar and Edith Cohen. ACM Press, June 2019, pp. 1103–1114. DOI: 10.1145/3313276.3316400.

[Chu+20] Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. *Bulletproofs+: Shorter Proofs for Privacy-Enhanced Distributed Ledger*. Cryptology ePrint Archive, Report 2020/735. https://eprint.iacr.org/2020/735. 2020.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. "Non-interactive Batch Arguments for NP from Standard Assumptions". In: *CRYPTO 2021, Part IV*. Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 394–423. DOI: 10.1007/978-3-030-84259-8_14.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. *SNARGs for $\mathcal{P}$ from LWE*. Cryptology ePrint Archive, Report 2021/808. https://eprint.iacr.org/2021/808. 2021.

[CLs10] Rafik Chaabouni, Helger Lipmaa, and abhi shelat. "Additive Combinatorics and Discrete Logarithm Based Range Protocols". In: *ACISP 10*. Ed. by Ron Steinfeld and Philip Hawkes. Vol. 6168. LNCS. Springer, Heidelberg, July 2010, pp. 336–351.

[CNW97] Jin-Yi Cai, Ajay Nerurkar, and Min-You Wu. *The design of uncheatable benchmarks using complexity theory*. 1997.

[Cod+97] Bruno Codenottia, Biswa N. Datta, Karabi Datta, and Mauro Leoncini. "Parallel algorithms for certain matrix computations". In: *Theoretical Computer Science*. 1997.

[Coh17] Bram Cohen. *Proofs of Space and Time*. Blockchain Protocol Analysis and Security Engineering. https://cyber.stanford.edu/sites/default/files/bramcohen.pdf. 2017.

[con22] arkworks contributors. ***arkworks*** *zkSNARK ecosystem*. 2022. URL: https://arkworks.rs.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Heidelberg, May 2020, pp. 769–793. DOI: 10.1007/978-3-030-45721-1_27.

[CP18] Bram Cohen and Krzysztof Pietrzak. "Simple Proofs of Sequential Work". In: *EUROCRYPT 2018, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. LNCS. Springer, Heidelberg, Apr. 2018, pp. 451–467. DOI: 10.1007/978-3-319-78375-8_15.

[CRR11] Ran Canetti, Ben Riva, and Guy N. Rothblum. "Practical delegation of computation using multiple servers". In: *ACM CCS 2011*. Ed. by Yan Chen, George Danezis, and Vitaly Shmatikov. ACM Press, Oct. 2011, pp. 445–454. DOI: 10.1145/2046707.2046759.

[CT10] Alessandro Chiesa and Eran Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: *ICS 2010*. Ed. by Andrew Chi-Chih Yao. Tsinghua University Press, Jan. 2010, pp. 310–331.

[CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. "Cluster Computing in Zero Knowledge". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 371–403. DOI: 10.1007/978-3-662-46803-6_13.

[CW77] J Lawrence Carter and Mark N Wegman. "Universal classes of hash functions". In: *Proceedings of the ninth annual ACM symposium on Theory of computing*. 1977, pp. 106–112.

[Dag+15] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. "Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges". In: *ACM CCS 2015*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM Press, Oct. 2015, pp. 720–731. DOI: 10.1145/2810103.2813674.

[Dai98] Wei Dai. "B-money". In: *Consulted* 1 (1998), p. 2012.

[Dao+23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. *Weak Fiat-Shamir Attacks on Modern Proof Systems*. Cryptology ePrint Archive, Report 2023/691. https://eprint.iacr.org/2023/691. 2023.

[Dav+18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain". In: *EUROCRYPT 2018, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. LNCS. Springer, Heidelberg, Apr. 2018, pp. 66–98. DOI: 10.1007/978-3-319-78375-8_3.

[De +19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. "Verifiable Delay Functions from Supersingular Isogenies and Pairings". In: *ASIACRYPT 2019, Part I*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11921. LNCS. Springer, Heidelberg, Dec. 2019, pp. 248–277. DOI: 10.1007/978-3-030-34578-5_10.

[Dfi] *Threshold Relay*. Dfinity. https://dfinity.org/pdfs/viewer.html?file=../library/threshold-relay-blockchain-stanford.pdf. 2017.

[DG23] Quang Dao and Paul Grubbs. "Spartan and Bulletproofs are Simulation-Extractable (for Free!)" In: *EUROCRYPT 2023, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. LNCS. Springer, Heidelberg, Apr. 2023, pp. 531–562. DOI: 10.1007/978-3-031-30617-4_18.

[DKT21] Soubhik Deb, Sreeram Kannan, and David Tse. "PoSAT: Proof-of-Work Availability and Unpredictability, Without the Work". In: *FC 2021, Part II*. Ed. by Nikita Borisov and Claudia Díaz. Vol. 12675. LNCS. Springer, Heidelberg, Mar. 2021, pp. 104–128. DOI: 10.1007/978-3-662-64331-0_6.

[DN93] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. LNCS. Springer, Heidelberg, Aug. 1993, pp. 139–147. DOI: 10.1007/3-540-48071-4_10.

[Döt+20] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. "Tight Verifiable Delay Functions". In: *SCN 20*. Ed. by Clemente Galdi and Vladimir Kolesnikov. Vol. 12238. LNCS. Springer, Heidelberg, Sept. 2020, pp. 65–84. DOI: 10.1007/978-3-030-57990-6_4.

[Dou02] John R Douceur. "The sybil attack". In: *International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 251–260.

[Dra19] Justing Drake. *PLONK-style SNARKs without FFTs*. link. 2019.

[DS01] Drew Dean and Adam Stubblefield. "Using Client Puzzles to Protect TLS". In: *USENIX Security 2001*. Ed. by Dan S. Wallach. USENIX Association, Aug. 2001.

[Dzi+15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. "Proofs of Space". In: *CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. LNCS. Springer, Heidelberg, Aug. 2015, pp. 585–605. DOI: 10.1007/978-3-662-48000-7_29.

[Eag22] Liam Eagen. *Bulletproofs++*. Cryptology ePrint Archive, Report 2022/510. https://eprint.iacr.org/2022/510. 2022.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. *cq: Cached quotients for fast lookups*. Cryptology ePrint Archive, Report 2022/1763. https://eprint.iacr.org/2022/1763. 2022.

[Eph+20a] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. "Continuous Verifiable Delay Functions". In: *EUROCRYPT 2020, Part III*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. LNCS. Springer, Heidelberg, May 2020, pp. 125–154. DOI: 10.1007/978-3-030-45727-3_5.

[Eph+20b] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. "SPARKs: Succinct Parallelizable Arguments of Knowledge". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Heidelberg, May 2020, pp. 707–737. DOI: 10.1007/978-3-030-45721-1_25.

[Esp22] EspressoSystems. *Specifications: Configurable Asset Privacy*. Github. 2022. URL: https://github.com/EspressoSystems/cap/blob/96336e8822a9672c107a05e39cf7ec9fe897f2ea/cap-specification.pdf.

[Fcw] In.

[Fila] *Filecoin: A Decentralized Storage Network*. Protocol Labs. https://filecoin.io/filecoin.pdf. 2017.

[Filb] *Proof of replication*. Protocol Labs. https://filecoin.io/proof-of-replication.pdf. 2017.

[Fis18] Ben Fisch. *PoReps: Proofs of Space on Useful Data*. Cryptology ePrint Archive, Report 2018/678. https://eprint.iacr.org/2018/678. 2018.

[Fou22] Ethereum Foundation. *Zkevm Specifications*. 2022. URL: https://github.com/privacy-scaling-explorations/zkevm-specs.

[Fur+03] Jun Furukawa, Hiroshi Miyauchi, Kengo Mori, Satoshi Obana, and Kazue Sako. "An Implementation of a Universally Verifiable Electronic Voting Scheme based on Shuffling". In: *FC 2002*. Ed. by Matt Blaze. Vol. 2357. LNCS. Springer, Heidelberg, Mar. 2003, pp. 16–30.

[Gab] Ariel Gabizon. *Multiset checks in PLONK and Plookup*. https://hackmd.io/@arielg/ByFgSDA7D.

[Gan+21] Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. *Fiat–Shamir Bulletproofs are Non-Malleable (in the Algebraic Group Model)*. Cryptology ePrint Archive, Report 2021/1393. https://eprint.iacr.org/2021/1393. 2021.

[Gen+13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 626–645. DOI: 10.1007/978-3-642-38348-9_37.

[GI08] Jens Groth and Yuval Ishai. "Sub-linear Zero-Knowledge Argument for Correctness of a Shuffle". In: *EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. LNCS. Springer, Heidelberg, Apr. 2008, pp. 379–396. DOI: 10.1007/978-3-540-78967-3_22.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *EUROCRYPT 2015, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for muggles". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 113–122. DOI: 10.1145/1374376.1374396.

[GM97] Robert M Guralnick and Peter Müller. "Exceptional polynomials of affine type". In: *Journal of Algebra* 194.2 (1997), pp. 429–454.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208.

[Gol+21] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. *Brakedown: Linear-time and post-quantum SNARKs for R1CS*. Cryptology ePrint Archive, Report 2021/1043. https://eprint.iacr.org/2021/1043. 2021.

[Gor98] Daniel M Gordon. "A survey of fast exponentiation methods". In: *Journal of algorithms* 27.1 (1998), pp. 129–146.

[Gra+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems". In: *USENIX Security 2021*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, Aug. 2021, pp. 519–535.

[Gro05] Jens Groth. "Non-interactive Zero-Knowledge Arguments for Voting". In: *ACNS 05*. Ed. by John Ioannidis, Angelos Keromytis, and Moti Yung. Vol. 3531. LNCS. Springer, Heidelberg, June 2005, pp. 467–482. DOI: 10.1007/11496137_32.

[Gro10a] Jens Groth. "A Verifiable Secret Shuffle of Homomorphic Encryptions". In: *Journal of Cryptology* 23.4 (Oct. 2010), pp. 546–579. DOI: 10.1007/s00145-010-9067-9.

[Gro10b] Jens Groth. "Short Pairing-Based Non-interactive Zero-Knowledge Arguments". In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 321–340. DOI: 10.1007/978-3-642-17373-8_19.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Heidelberg, May 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11.

[Gro+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: *CRYPTO 2018, Part III*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10993. LNCS. Springer, Heidelberg, Aug. 2018, pp. 698–728. DOI: 10.1007/978-3-319-96878-0_24.

[GS08] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. LNCS. Springer, Heidelberg, Apr. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24.

[GS98] David M. Goldschlag and Stuart G. Stubblebine. "Publicly Verifiable Lotteries: Applications of Delaying Functions". In: *FC'98*. Ed. by Rafael Hirschfeld. Vol. 1465. LNCS. Springer, Heidelberg, Feb. 1998, pp. 214–226.

[GSV98] Oded Goldreich, Amit Sahai, and Salil P. Vadhan. "Honest-Verifier Statistical Zero-Knowledge Equals General Statistical Zero-Knowledge". In: *30th ACM STOC*. ACM Press, May 1998, pp. 399–408. DOI: 10.1145/276698.276852.

[GT21] Ashrujit Ghoshal and Stefano Tessaro. "Tight State-Restoration Soundness in the Algebraic Group Model". In: *CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. LNCS. Virtual Event: Springer, Heidelberg, Aug. 2021, pp. 64–93. DOI: 10.1007/978-3-030-84252-9_3.

[GVW01] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. "On Interactive Proofs with a Laconic Prover". In: *ICALP 2001*. Ed. by Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen. Vol. 2076. LNCS. Springer, Heidelberg, July 2001, pp. 334–345. DOI: 10.1007/3-540-48224-5_28.

[GW11] Craig Gentry and Daniel Wichs. "Separating succinct non-interactive arguments from all falsifiable assumptions". In: *43rd ACM STOC*. Ed. by Lance Fortnow and Salil P. Vadhan. ACM Press, June 2011, pp. 99–108. DOI: 10.1145/1993636.1993651.

[GW20a] Ariel Gabizon and Zachary J. Williamson. *plookup: A simplified polynomial protocol for lookup tables*. Cryptology ePrint Archive, Report 2020/315. https://eprint.iacr.org/2020/315. 2020.

[GW20b] Ariel Gabizon and Zachary J Williamson. *Proposal: The Turbo-PLONK program syntax for specifying SNARK programs*. https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf. 2020.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. https://eprint.iacr.org/2019/953. 2019.

[Hab22a] Ulrich Haböck. "A summary on the FRI low degree test". In: *Cryptology ePrint Archive* (2022).

[Hab22b] Ulrich Haböck. *Multivariate lookups based on logarithmic derivatives*. Cryptology ePrint Archive, Report 2022/1530. `https://eprint.iacr.org/2022/1530`. 2022.

[Hop+22] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. *Zcash Protocol Specification. Version 2022.3.8*. Online. `https://zips.z.cash/protocol/protocol.pdf`. 2022. URL: `https://zips.z.cash/protocol/protocol.pdf`.

[Hou15] Xiang-dong Hou. "Permutation polynomials over finite fields—a survey of recent advances". In: *Finite Fields and Their Applications* 32 (2015), pp. 82–119.

[HVDH22] David Harvey and Joris Van Der Hoeven. "Polynomial Multiplication over Finite Fields in Time". In: *Journal of the ACM (JACM)* 69.2 (2022), pp. 1–40.

[JB99] Ari Juels and John G. Brainard. "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks". In: *NDSS'99*. The Internet Society, Feb. 1999.

[Jed16] TE Jedusor. *Mimblewimble*. 2016.

[JK07] Ari Juels and Burton S. Kaliski Jr. "Pors: proofs of retrievability for large files". In: *ACM CCS 2007*. Ed. by Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson. ACM Press, Oct. 2007, pp. 584–597. DOI: `10.1145/1315245.1315317`.

[JM11] Yves Igor Jerschow and Martin Mauve. "Non-parallelizable and non-interactive client puzzles from modular square roots". In: *Availability, Reliability and Security (ARES)*. 2011.

[JT20] Joseph Jaeger and Stefano Tessaro. "Expected-Time Cryptography: Generic Techniques and Applications to Concrete Soundness". In: *TCC 2020, Part III*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12552. LNCS. Springer, Heidelberg, Nov. 2020, pp. 414–443. DOI: `10.1007/978-3-030-64381-2_15`.

[KB20] Assimakis Kattis and Joseph Bonneau. *Proof of Necessary Work: Succinct State Verification with Fairness Guarantees*. Cryptology ePrint Archive, Report 2020/190. `https://eprint.iacr.org/2020/190`. 2020.

[Kia+17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". In: *CRYPTO 2017, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 357–388. DOI: `10.1007/978-3-319-63688-7_12`.

[Kil92] Joe Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *24th ACM STOC*. ACM Press, May 1992, pp. 723–732. DOI: `10.1145/129712.129782`.

[KMB17] Dmitry Kogan, Nathan Manohar, and Dan Boneh. "T/Key: Second-Factor Authentication From Secure Hash Chains". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 983–999. DOI: 10.1145/3133956.3133989.

[KMT22] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. *MinRoot: Candidate Sequential Function for Ethereum VDF*. Cryptology ePrint Archive, Report 2022/1626. https://eprint.iacr.org/2022/1626. 2022.

[KN] S King and S Nadal. "Peercoin–Secure & Sustainable Cryptocoin". In: *Aug-2012 [Online]. Available: https://peercoin. net/whitepaper* ().

[Kos+16] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 839–858. DOI: 10.1109/SP.2016.55.

[KP98] Joe Kilian and Erez Petrank. "An Efficient Noninteractive Zero-Knowledge Proof System for NP with General Assumptions". In: *Journal of Cryptology* 11.1 (Jan. 1998), pp. 1–27. DOI: 10.1007/s001459900032.

[KPV19] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. *RedShift: Transparent SNARKs from List Polynomial Commitment IOPs*. Cryptology ePrint Archive, Report 2019/1400. https://eprint.iacr.org/2019/1400. 2019.

[KS22] Abhiram Kothapalli and Srinath Setty. *SuperNova: Proving universal machine executions without universal circuits*. Cryptology ePrint Archive, Report 2022/1758. https://eprint.iacr.org/2022/1758. 2022.

[KS23] Abhiram Kothapalli and Srinath Setty. "HyperNova: Recursive arguments for customizable constraint systems". In: *Cryptology ePrint Archive* (2023).

[KST21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. *Nova: Recursive Zero-Knowledge Arguments from Folding Schemes*. Cryptology ePrint Archive, Report 2021/370. https://eprint.iacr.org/2021/370. 2021.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: *CRYPTO 2022, Part IV*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13510. LNCS. Springer, Heidelberg, Aug. 2022, pp. 359–388. DOI: 10.1007/978-3-031-15985-5_13.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[Lee21] Jonathan Lee. "Dory: Efficient, Transparent Arguments for Generalised Inner Products and Polynomial Commitments". In: *TCC 2021, Part II*. Ed. by Kobbi Nissim and Brent Waters. Vol. 13043. LNCS. Springer, Heidelberg, Nov. 2021, pp. 1–34. DOI: 10.1007/978-3-030-90453-1_1.

[Ler14] Sergio Demian Lerner. *Proof of unique blockchain storage*. https://bitslog.wordpress.com/2014/11/03/proof-of-local-blockchain-storage/. 2014.

[Lin03] Yehuda Lindell. "Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation". In: *Journal of Cryptology* 16.3 (June 2003), pp. 143–184. DOI: 10.1007/s00145-002-0143-7.

[Lip03] Helger Lipmaa. "On Diophantine Complexity and Statistical Zero-Knowledge Arguments". In: *ASIACRYPT 2003*. Ed. by Chi-Sung Laih. Vol. 2894. LNCS. Springer, Heidelberg, Nov. 2003, pp. 398–415. DOI: 10.1007/978-3-540-40061-5_26.

[LMT93] Rudolf Lidl, Gary L Mullen, and Gerhard Turnwald. *Dickson polynomials*. Vol. 65. Chapman & Hall/CRC, 1993.

[Lun+92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. "Algebraic methods for interactive proof systems". In: *Journal of the ACM (JACM)* 39.4 (1992), pp. 859–868.

[LV20] Alex Lombardi and Vinod Vaikuntanathan. "Fiat-Shamir for Repeated Squaring with Applications to PPAD-Hardness and VDFs". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 632–651. DOI: 10.1007/978-3-030-56877-1_22.

[LW15] Arjen K. Lenstra and Benjamin Wesolowski. *A random zoo: sloth, unicorn, and trx*. Cryptology ePrint Archive, Report 2015/366. https://eprint.iacr.org/2015/366. 2015.

[Mal+19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2111–2128. DOI: 10.1145/3319535.3339817.

[Mat61] Émile Mathieu. "Mémoire sur l'étude des fonctions de plusieurs quantités sur la manière de les former et sur les substitutions qui les laissent invariables". In: *J. Math. Pures Appl. (2) 6*. 1861.

[Max] G Maxwell. "Zero knowledge contingent payment. 2011". In: *URl: https://en. bitcoin. it/wiki/Zero_Knowledge_Contingent_Payment (visited on 05/01/2016)* ().

[Max13] Gregory Maxwell. *CoinJoin: Bitcoin privacy for the real world*. `bitcointalk.org`. 2013.

[Max16] Greg Maxwell. *Confidential Transactions*. `https://people.xiph.org/~greg/confidential_values.txt`. 2016.

[Mei+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. "A fistful of bitcoins: characterizing payments among men with no names". In: *IMC*. 2013.

[Mic16] Silvio Micali. "ALGORAND: the efficient and democratic ledger". In: *arXiv preprint arXiv:1607.01341* (2016).

[Mic94] Silvio Micali. "CS Proofs (Extended Abstracts)". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 436–453. DOI: `10.1109/SFCS.1994.365746`.

[Mil+14] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. "Permacoin: Repurposing Bitcoin Work for Data Preservation". In: *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 475–490. DOI: `10.1109/SP.2014.37`.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. "Publicly verifiable proofs of sequential work". In: *ITCS 2013*. Ed. by Robert D. Kleinberg. ACM, Jan. 2013, pp. 373–388. DOI: `10.1145/2422436.2422479`.

[MNS16] Tal Moran, Moni Naor, and Gil Segev. "An Optimally Fair Coin Toss". In: *Journal of Cryptology* 29.3 (July 2016), pp. 491–513. DOI: `10.1007/s00145-015-9199-z`.

[Moh23] Nicholas Mohnblatt. *Sangria: A Folding Scheme for PLONK*. `https://github.com/geometryresearch/technical_notes/blob/main/sangria_folding_plonk.pdf`. Accessed: 2023-04-27. 2023.

[Mon] *Monero - Private Digital Currency*. `https://getmonero.org/`.

[MP15] Gregory Maxwell and Andrew Poelstra. "Borromean ring signatures". In: *Accessed: Jun* 8 (2015), p. 2019.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. "Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology". In: *TCC 2004*. Ed. by Moni Naor. Vol. 2951. LNCS. Springer, Heidelberg, Feb. 2004, pp. 21–39. DOI: `10.1007/978-3-540-24638-1_2`.

[MSH17] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. "A Smart Contract for Boardroom Voting with Maximum Voter Privacy". In: *FC 2017*. Ed. by Aggelos Kiayias. Vol. 10322. LNCS. Springer, Heidelberg, Apr. 2017, pp. 357–375.

[MSZ21] Simon Masson, Antonio Sanso, and Zhenfei Zhang. *Bandersnatch: a fast elliptic curve built over the BLS12-381 scalar field*. Cryptology ePrint Archive, Report 2021/1152. `https://eprint.iacr.org/2021/1152`. 2021.

[MT79] Robert Morris and Ken Thompson. "Password security: A case history". In: *Communications of the ACM* 22.11 (1979), pp. 594–597.

[Mül97] Peter Müller. "A Weil-bound free proof of Schur's conjecture". In: *Finite Fields and Their Applications* 3.1 (1997), pp. 25–32.

[Nak08] S Nakamoto. *Bitcoin: A peer-to-peer electionic cash system*. Unpublished. 2008.

[Nef01] C. Andrew Neff. "A Verifiable Secret Shuffle and Its Application to e-Voting". In: *ACM CCS 2001*. Ed. by Michael K. Reiter and Pierangela Samarati. ACM Press, Nov. 2001, pp. 116–125. DOI: `10.1145/501983.502000`.

[NM+16] Shen Noether, Adam Mackenzie, et al. "Ring confidential transactions". In: *Ledger* 1 (2016), pp. 1–18.

[NT16] Assa Naveh and Eran Tromer. "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 255–271. DOI: `10.1109/SP.2016.23`.

[Par+13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. "Pinocchio: Nearly Practical Verifiable Computation". In: *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 238–252. DOI: `10.1109/SP.2013.47`.

[Par+18] Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gazi, Joël Alwen, and Krzysztof Pietrzak. "SpaceMint: A Cryptocurrency Based on Proofs of Space". In: *FC 2018*. Ed. by Sarah Meiklejohn and Kazue Sako. Vol. 10957. LNCS. Springer, Heidelberg, Feb. 2018, pp. 480–499. DOI: `10.1007/978-3-662-58387-6_26`.

[Pea+22] Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. *PlonKup: Reconciling PlonK with plookup*. Cryptology ePrint Archive, Report 2022/086. `https://eprint.iacr.org/2022/086`. 2022.

[Ped92] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 129–140. DOI: `10.1007/3-540-46766-1_9`.

[Pie19a] Krzysztof Pietrzak. "Proofs of Catalytic Space". In: *ITCS 2019*. Ed. by Avrim Blum. Vol. 124. LIPIcs, Jan. 2019, 59:1–59:25. DOI: `10.4230/LIPIcs.ITCS.2019.59`.

[Pie19b] Krzysztof Pietrzak. "Simple Verifiable Delay Functions". In: *ITCS 2019*. Ed. by Avrim Blum. Vol. 124. LIPIcs, Jan. 2019, 60:1–60:15. DOI: `10.4230/LIPIcs.ITCS.2019.60`.

[Pip80] Nicholas Pippenger. "On the evaluation of powers and monomials". In: *SIAM Journal on Computing* 9 (1980), pp. 230–250. ISSN: 0097–5397.

[PK22] Jim Posen and Assimakis A. Kattis. *Caulk+: Table-independent lookup arguments*. Cryptology ePrint Archive, Report 2022/957. https://eprint.iacr.org/2022/957. 2022.

[Poe+19] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. "Confidential Assets". In: *FC 2018 Workshops*. Ed. by Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala. Vol. 10958. LNCS. Springer, Heidelberg, Mar. 2019, pp. 43–63. DOI: 10.1007/978-3-662-58820-8_4.

[PS17] Rafael Pass and Elaine Shi. "The Sleepy Model of Consensus". In: *ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. LNCS. Springer, Heidelberg, Dec. 2017, pp. 380–409. DOI: 10.1007/978-3-319-70697-9_14.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. "Signatures of Correct Computation". In: *TCC 2013*. Ed. by Amit Sahai. Vol. 7785. LNCS. Springer, Heidelberg, Mar. 2013, pp. 222–242. DOI: 10.1007/978-3-642-36594-2_13.

[PW16] Cecile Pierrot and Benjamin Wesolowski. *Malleability of the blockchain's entropy*. Cryptology ePrint Archive, Report 2016/370. https://eprint.iacr.org/2016/370. 2016.

[Rab83] Michael O Rabin. "Transaction protection by beacons". In: *Journal of Computer and System Sciences* (1983).

[Ran] *RANDAO: A DAO working as RNG of Ethereum*. Tech. rep. 2016.

[RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin". In: *ESORICS 2014, Part II*. Ed. by Miroslaw Kutylowski and Jaideep Vaidya. Vol. 8713. LNCS. Springer, Heidelberg, Sept. 2014, pp. 345–364. DOI: 10.1007/978-3-319-11212-1_20.

[RS96] Ronald L Rivest and Adi Shamir. "PayWord and MicroMint: Two simple micropayment schemes". In: *International Workshop on Security Protocols*. Springer. 1996, pp. 69–87.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. "Time-lock puzzles and timed-release crypto". In: (1996).

[Sab13] Nicolas van Saberhagen. *Cryptonote v 2. 0*. 2013.

[San99] Tomas Sander. "Efficient Accumulators without Trapdoor Extended Abstracts". In: *ICICS 99*. Ed. by Vijay Varadharajan and Yi Mu. Vol. 1726. LNCS. Springer, Heidelberg, Nov. 1999, pp. 252–262.

[Set+18] Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. "Proving the correct execution of concurrent services in zero-knowledge". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 339–356.

[Set20] Srinath Setty. "Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Heidelberg, Aug. 2020, pp. 704–737. DOI: 10.1007/978-3-030-56877-1_25.

[SL20] Srinath Setty and Jonathan Lee. *Quarks: Quadruple-efficient transparent zkSNARKs*. Cryptology ePrint Archive, Report 2020/1275. https://eprint.iacr.org/2020/1275. 2020.

[Sti94] Douglas R. Stinson. "Universal hashing and authentication codes". In: *Designs, Codes and Cryptography* 4.3 (1994), pp. 369–380.

[STW23] Srinath Setty, Justin Thaler, and Riad Wahby. "Customizable constraint systems for succinct arguments". In: *Cryptology ePrint Archive* (2023).

[Sys22] Espresso System. *Jellyfish Jellyfish cryptographic library*. 2022. URL: https://github.com/EspressoSystems/jellyfish.

[Syt+17] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. "Scalable Bias-Resistant Distributed Randomness". In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 444–460. DOI: 10.1109/SP.2017.45.

[Tha13] Justin Thaler. "Time-Optimal Interactive Proofs for Circuit Evaluation". In: *CRYPTO 2013, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 71–89. DOI: 10.1007/978-3-642-40084-1_5.

[Tha20] Justin Thaler. *Proofs, arguments, and zero-knowledge*. 2020.

[TR] Jason Teutsch and Christian Reitwießner. "A scalable verification solution for blockchains". In: ().

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *TCC 2008*. Ed. by Ran Canetti. Vol. 4948. LNCS. Springer, Heidelberg, Mar. 2008, pp. 1–18. DOI: 10.1007/978-3-540-78524-8_1.

[Val22] Henry de Valence. *Merlin transcript*. 2022. URL: https://merlin.cool/.

[vW94] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Application to Hash Functions and Discrete Logarithms". In: *ACM CCS 94*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu. ACM Press, Nov. 1994, pp. 210–218. DOI: 10.1145/191177.191231.

[Wah+15] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. "Efficient RAM and control flow in verifiable outsourced computation". In: *NDSS 2015*. The Internet Society, Feb. 2015.

[Wah+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. "Doubly-Efficient zkSNARKs Without Trusted Setup". In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[WC81] Mark N Wegman and J Lawrence Carter. "New hash functions and their use in authentication and set equality". In: *Journal of computer and system sciences* 22.3 (1981), pp. 265–279.

[Wes19] Benjamin Wesolowski. "Efficient Verifiable Delay Functions". In: *EUROCRYPT 2019, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. LNCS. Springer, Heidelberg, May 2019, pp. 379–407. DOI: 10.1007/978-3-030-17659-4_13.

[Wes20] Benjamin Wesolowski. "Efficient Verifiable Delay Functions". In: *Journal of Cryptology* 33.4 (Oct. 2020), pp. 2113–2147. DOI: 10.1007/s00145-020-09364-x.

[Wik21] Douglas Wikström. *Special Soundness in the Random Oracle Model*. Cryptology ePrint Archive, Report 2021/1265. https://eprint.iacr.org/2021/1265. 2021.

[Woo14] Gavin Wood. *Ethereum: A secure decentralized transaction ledger*. http://gavwood.com/paper.pdf. 2014.

[WW22] Brent Waters and David J. Wu. *Batch Arguments for NP and More from Standard Bilinear Group Assumptions*. Cryptology ePrint Archive, Report 2022/336. https://eprint.iacr.org/2022/336. 2022.

[Xie+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. "Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 733–764. DOI: 10.1007/978-3-030-26954-8_24.

[Xio+22] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. *VERI-ZEXE: Decentralized Private Computation with Universal Setup*. Cryptology ePrint Archive, Report 2022/802. https://eprint.iacr.org/2022/802. 2022.

[XZS22a] Tiancheng Xie, Yupeng Zhang, and Dawn Song. *Orion: Zero Knowledge Proof with Linear Prover Time*. Cryptology ePrint Archive, Report 2022/1010. https://eprint.iacr.org/2022/1010. 2022.

[XZS22b] Tiancheng Xie, Yupeng Zhang, and Dawn Song. "Orion: Zero Knowledge Proof with Linear Prover Time". In: *CRYPTO 2022, Part IV*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13510. LNCS. Springer, Heidelberg, Aug. 2022, pp. 299–328. DOI: 10.1007/978-3-031-15985-5_11.

[Zam+21] Abel Zambrano, Alejandro Palacio Betancur, Luis Burbano, Andres Felipe Niño, Luis Felipe Giraldo, Mariantonieta Gutierrez Soto, Jairo Giraldo, and Alvaro A. Cárdenas. "You Make Me Tremble: A First Look at Attacks Against Structural Control Systems". In: *ACM CCS 2021*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 1320–1337. DOI: 10.1145/3460120.3485386.

[Zap+22a] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. *Caulk: Lookup Arguments in Sublinear Time*. Cryptology ePrint Archive, Report 2022/621. https://eprint.iacr.org/2022/621. 2022.

[Zap+22b] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. "Caulk: Lookup Arguments in Sublinear Time". In: *ACM CCS 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM Press, Nov. 2022, pp. 3121–3134. DOI: 10.1145/3548606.3560646.

[Zap+22c] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. *Baloo: Nearly Optimal Lookup Arguments*. Cryptology ePrint Archive, Report 2022/1565. https://eprint.iacr.org/2022/1565. 2022.

[Zca22] Zcash. *PLONKish Arithmetization*. link. 2022.

[Zha+20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. "Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof". In: *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 859–876. DOI: 10.1109/SP40000.2020.00052.

[Zie] Micheal Zieve. *Exceptional polynomials*. http://dept.math.lsa.umich.edu/~zieve/papers/epfacts.pdf.