Speed-Stacking: Fast Sublinear Zero-Knowledge Proofs for Disjunctions

Aarushi Goel[1], Mathias Hall-Andersen[2], Gabriel Kaptchuk[3], and Nicholas Spooner[4]

[1]NTT Research, aarushi.goel@ntt-research.com
[2]Aarhus University, ma@cs.au.dk
[3]Boston University, kaptchuk@bu.edu
[4]University of Warwick, Nicholas.Spooner@warwick.ac.uk

Abstract

Building on recent *disjunctive compilers* for zero-knowledge (*e.g.* Goel *et al.* [EUROCRYPT'22]) we propose a new compiler that, when applied to sublinear-sized proofs, can result in sublinear-size disjunctive zero-knowledge with *sublinear* proving times (without meaningfully increasing proof sizes). Our key observation is that simulation in sublinear-size *zero-knowledge* proof systems can be much faster (both concretely and asymptotically) than the honest prover. We study applying our compiler to two classes of $O(\log n)$-round protocols: interactive oracle proofs, specifically Aurora [EUROCRYPT'19] and Fractal [EUROCRYPT'20], and folding arguments, specifically Compressed Σ-protocols [CRYPTO'20, CRYPTO'21] and Bulletproofs [S&P'18]. This study validates that the compiler can lead to significant savings. For example, applying our compiler to Fractal enables us to prove a disjunction of ℓ clauses, each of size N, with only $O((N+\ell)\cdot \text{polylog}(N))$ computation, versus $O(\ell N \cdot \text{polylog}(N))$ when proving the disjunction directly. We also find that our compiler offers a new lens through which to understand zero-knowledge proofs, evidenced by multiple examples of protocols with the same "standalone" complexity that each behave very differently when stacked.

Contents

1 Introduction

Zero-knowledge proofs and arguments [GMR85] allow a prover to convince the verifier of the validity of an NP statement without revealing anything beyond the validity itself. Early results established that such protocols exist for all NP languages [GMW86], and recent work has proposed zero-knowledge proofs that are more practically efficient *e.g.* [JKO13, BCTV14, Gro16, KKW18, BBB$^+$18, BCR$^+$19, HK20b]. Many of these efficient zero-knowledge proofs are now being used in practice [BCG$^+$14, Zav20, se19], and zero-knowledge proofs have become a critical component of constructing larger cryptographic systems.

Disjunctive Zero-Knowledge. A *disjunctive statement* is an NP statement consisting of a logical OR of a set of clauses. We refer to zero-knowledge proofs for disjunctive statements as "disjunctive zero-knowledge." Disjunctive zero-knowledge is central to privacy-preserving systems where revealing *which* clause a prover has a witness for might reveal their identity. Disjunctive zero-knowledge has received a great deal of attention [CDS94, AOS02, GMY03] and recently there has been renewed interest in optimizing cryptographic protocols for disjunctions, both in the context of zero-knowledge [GK15, CPS$^+$16, Kol18, HK20b, GGHAK22, ACF21] and secure multiparty computation [BMRS21, HK20a, HK21].

The simplest approach to disjunctive zero-knowledge is to appeal to NP-completeness: a disjunction of NP statements is itself an NP statement which can be proved using a proof system for NP. In practice, however, this has two key drawbacks: first, the individual clauses may be of a special form that admits efficient zero-knowledge proofs (e.g. a discrete-log relation) but that structure can not be preserved under disjunction. Second, even if the clauses are general circuits, if the clauses are distinct then the resulting circuit is as large as the sum of the size of individual clauses. As a result, the complexity of the proof system grows at least linearly in the number of clauses.

In light of this, one alternative approach that has been explored in the literature is to manually modify *specific* zero-knowledge protocols directly [GK15, HK20b, ACF21] such that they naturally support disjunctive statements. Excitingly, recent work has shown that manual modification can result in protocols with communication sub-linear in the number of clauses [HK20b, ACF21]. However, such approaches rely strongly on the structure of individual protocols and do not necessarily generalize.

A more robust approach is to build *disjunctive compilers* [CDS94, AOS02, BMRS21, GGHAK22], generic approaches that automatically transform large classes of zero-knowledge protocols into disjunctive zero-knowledge protocols. The seminal work in this area is [CDS94], which proposed an approach that compiled Σ-protocols for disjunctions by having the prover *simulate* the clauses for which it did not have a witness. More recently, Baum et al. [BMRS21] and Goel et al. [GGHAK22] built upon this idea to compile large classes of zero-knowledge protocols into disjunctive zero-knowledge protocols with communication complexity sub-linear in the number of clauses.

Succinct Proofs. A proof system is succinct if its communication cost is polylogarithmic in the size of the computation being proven. Succinct zero-knowledge proofs are the subject of a long and active line of research ([Kil94, GGPR13, BCTV14, Gro16, BCS16, BCC$^+$16] and many others) and in recent years have become efficient enough to use in practice. Many such proof systems support some expressive NP-complete problem, e.g. arithmetic circuit satisfiability. This raises a natural suggestion: to prove a disjunctive statement, one could simply construct a circuit for the disjunction and employ a succinct proof system. The size of the resulting proof would be only slightly larger than a proof for a single clause.

The main caveat is that, while the proof size is essentially unaffected, the time and space complexities of the prover increase by at least a *multiplicative* factor of the number of distinct clauses, compared the cost of proving a single clause. Since succinct proof systems typically have quite high prover complexity, avoiding this increase would result in significant savings.

Stacking Succinct Proofs. In our work, we explore how we can apply the frameworks developed in recent research on minimizing the communication complexity of disjunctive zero-knowledge (specifically [GGHAK22]) to achieve succinct proofs for disjunctions which avoid this multiplicative blowup in the prover computation time.

At the heart of our approach is the observation that succinct proof systems often have faster simulators than provers. Intuitively, this is because the cost of "cheating" the verifier in a zero-knowledge protocol generally scales with the verifier's running time, rather than the prover's. Thus, following the approach of Cramer, Damgård and Schoenmakers, [CDS94], the prover in a succinct proof system can run the (more efficient) simulator for the inactive clauses instead of (less efficient) prover algorithm.

Taken together, we obtain succinct proof systems that can prove disjunctions without incurring a multiplicative increase in prover complexity in the number of clauses. We also show that in some cases, we can also avoid a similar increase in the *verifier's* complexity using batching techniques.

Set Membership vs. True Disjunctions. There is an important case in which appealing to NP completeness *is* concretely efficient: specifically, if (1) the zero-knowledge protocol supports an expressive NP-complete language, and (2) there is a high degree of homogeneity between the clauses. If a prover wants to prove e.g. $\mathbb{x}_1 \in \mathcal{L} \vee \ldots \vee \mathbb{x}_\ell \in \mathcal{L}$, it can do so efficiently by proving the statement "$\exists (i, \mathbb{x}), \text{st.}\, \mathbb{x} \in \mathcal{L} \wedge \mathbb{x} = \mathbb{x}_i$ (so the choice of branch is part of the NP witness). The size of this circuit is only slightly larger than the circuit for $\mathcal{L}$ itself. We refer to such statements as *set membership statements*. Our results are most significant in the case of what we call *true disjunctions*, i.e., where the prover wants to prove e.g. $\mathbb{x}_1 \in \mathcal{L}_1 \vee \ldots \vee \mathbb{x}_\ell \in \mathcal{L}_\ell$ making the above transformation more expensive.

1.1 Our Contributions

Framework for Prover-Efficient Succinct Disjunctive Zero-Knowledge. We present a framework, which we refer to as *speed stacking*, for composing succinct proofs for disjunctions that often yeild significant improvements in prover time. We do this by extending the notion of a "stackable" Σ-protocol, introduced by Goel et al. [GGHAK22], to a more general notion of a "stackable" interactive protocol. We then show how to compile a stackable zero-knowledge interactive protocol (ZK-IP) into a disjunctive zero-knowledge interactive protocol. Specifically, we prove the following theorem:

Theorem 1.1 (Informal)**.** *Let Π be a "stackable" zero-knowledge interactive protocol for a NP relation $\mathcal{R}$ with associated simulator $\mathcal{S}$. Then, there exists a zero-knowledge interactive protocol Π' for the NP relation $\mathcal{R}'((\mathbb{x}_1, \ldots, \mathbb{x}_\ell), \mathbb{w}) := \exists i, \mathcal{R}_i(\mathbb{x}_i, \mathbb{w}) = 1$ with communication complexity proportional to $\Pi + O(\log(\ell))$ and prover computational complexity* $\mathsf{Time}(\Pi) + (\ell - 1) \cdot \mathsf{Time}(\mathcal{S})$.

This theorem covers *true disjunctions* when $\mathcal{R}$ is sufficiently expressive, e.g.

$$\mathcal{R} = \textsf{circuit-SAT} : \mathcal{R}'(((C_1, \mathbb{x}_1), \ldots, (C_\ell, \mathbb{x}_\ell)), \mathbb{w}) = \exists i, C_i(\mathbb{x}_i, \mathbb{w}) = 1.$$

Note that while the above is a "universal" relation, our approach does not make use of universal circuits. As we discuss in the technical overview, while universal circuits are conceptually elegant (and sometimes achieve good asymptotic efficiency), the associated overhead makes them impractical.

Next, we study the speed-stackability of two protocols from each of two families of sublinear-sized zero-knowledge proof systems: interactive oracle proofs and folding arguments. Interestingly, we find that the concrete savings offered by each of the four protocols we consider differ dramatically, offering anything from dramatic, asymptotic speed-ups to concrete savings without asymptotic gains to minimal speedups. In addition to the new protocols we design, these results offer a new lens through which to study zero-knowledge proofs.

Speed Stacking Interactive Oracle Proofs. We adapt our stackability framework to interactive oracle proofs (IOPs) [BCS16], a generalization of interactive proofs that underlies various efficient succinct argument constructions. We show how to adapt the [BCS16, COS20] transformations to convert stackable IOPs (resp. holographic IOPs) into stackable succinct arguments (resp. with preprocessing).

We then consider the stackability of two existing IOP protocols for the *rank-one constraint satisfaction* (R1CS) language. Let ℓ be the number of clauses and N be the maximum circuit size of a clause.

- **Aurora** [BCR$^+$19] can be easily seen to be efficiently stackable by carefully examining the zero-knowledge simulator. By applying our compiler, we obtain a stackable succinct argument where the prover runs in time $O_{\mathbb{F}}(N(\log N + \ell \log^2 \lambda \log\log \lambda))$. By comparison, the cost of directly proving a disjunction using Aurora is $O_{\mathbb{F}}(\ell N \log(\ell N))$.[1]

[1] $O_{\mathbb{F}}$ indicates that time complexity is measured in field operations.

- **Fractal** [COS20] is not itself efficiently stackable: the verifier runs in polylogarithmic time after preprocessing, whereas any simulator for the original Fractal protocol involves a linear-time statement-dependent computation. To address this, we modify Fractal into a protocol we call Stactal, a stackable IOP for R1CS with polylogarithmic simulation. By applying our compiler, we obtain a stackable succinct argument where the prover runs in time $O_{\mathbb{F}}(N \log N + \ell \cdot \mathrm{polylog}(N))$. In particular, proving a disjunction on ℓ clauses for $\ell \ll N$ is asymptotically as efficient as proving a single clause.

Speed Stacking Folding Arguments. Finally, we show how to apply our framework to "folding arguments" [BCC⁺16, BBB⁺18, AC20, ACK21, ACF21]. This class of protocols, best represented by Compressed Σ-protocols [ACF21] and Bulletproofs [BBB⁺18], in which the prover replaces a linear-sized protocol message in a zero-knowledge interactive proof with a multi-round, privacy-free, interactive protocol with logarithmic communication complexity.

- **Compressed Σ-Protocols** [AC20, ACF21] is a stackable ZK-IP for openings of linear forms (after very minor modifications). By applying our compiler, we obtain a ZK-IP for the disjunction of linear form openings in which simulating each additional clause only requires computing one exponentiation and one group multiplication, in addition to a linear number of field operations. We also show that our ideas extend to circuit-satisfiablity variant of compressed Σ-protocols. We note that our results are stronger than the *set membership* version of Compressed Σ-protocols presented by Attema et al. [ACF21] in that our approach supports *true disjunctions* as well.[2]

- **Bulletproofs** [BBB⁺18] We observe that Bulletproofs (both for range proofs and circuit satisfiability) are stackable. However, we note that the runtime of the simulator for bulletproofs is roughly the same as that of the prover. As such, speed-stacking bulletproofs provides only marginal benefits over more direct techniques. The only exception we note is proving *set-membership* range proofs; because the range proof version of bulletproofs is not sufficiently expressive to directly capture set-membership, speed-stacking is preferable to rephrasing the statement to circuit satisfiability. This presents an interesting contrast between Compressed Σ-protocols and Bulletproofs, which otherwise seem to rely on very similar techniques.

2 Technical Overview

2.1 Disjunctive Templates for Zero-Knowledge

Given a sequence of statements $(\mathbb{x}_1, \ldots, \mathbb{x}_\ell)$, we wish to prove in zero-knowledge that either $\mathbb{x}_1 \in \mathcal{L}_1$, $\mathbb{x}_2 \in \mathcal{L}_2$, ..., or $\mathbb{x}_\ell \in \mathcal{L}_\ell$. While we might have access to appropriate and efficient zero-knowledge proof systems for each individual language $\mathcal{L}_1, \ldots, \mathcal{L}_\ell$, it is not clear how to apply these to the disjunction, while ensuring zero-knowledge. Let $\mathfrak{a}$ denote the clause for which the prover has a witness (the *active* clause). We will refer to the other clauses as *inactive*.

There are two main templates for disjunctive zero-knowledge in the literature: (1) *Statement Combination:* Combine the statements to define a new $\mathcal{L}$ with the relation $\mathcal{R}((\mathbb{x}_1, \ldots, \mathbb{x}_\ell), \mathbb{w}) := \mathcal{R}_1(\mathbb{x}_1, \mathbb{w}) \vee \ldots \vee \mathcal{R}_\ell(\mathbb{x}_\ell, \mathbb{w})$. and use any existing zero-knowledge proof protocol Π that supports general NP statements. (2) *Simulation of Inactive Clauses:* Initially suggested by Cramer, Damgård, and Schoenmakers [CDS94], this approach has been explored primarily in the context of Σ-protocols. In this template, the prover uses the honest prover algorithm for the active clause, and "cheats" by using the *zero-knowledge simulator* for each of the inactive clauses. The protocol guarantees that the prover can cheat for *all but one* of the clauses.

The best choice of template depends heavily on the underlying zero-knowledge protocol and the structure of the clauses. If the protocol is not for an NP-complete language (e.g. Schnorr's protocol [Sch90]), it may be impossible to combine the statements without protocol modifications, making the *simulation* template more attractive. When statement combination is possible, the efficiency of the combination often depends on the homogeneity of the clauses, *i.e* if it is more like *set membership* or a *true disjunctions*.

[2] We expand on the distinction between set membership and true disjunctions in the next section.

Of course, this difference is qualitative, rather than quantitative. Notably, a proof system for *set membership* can be used to construct a *true disjunction* by using universal circuits and a set membership over the programming of the circuit. However, transformations with universal circuits are notoriously expensive: for example, an implementation [LMS16] of Valiant's UC [Val76] shows that for a circuit implementating AES in $33{,}616$ gates the universal circuit capable of simulating it has $11{,}794{,}323$ gates (with $3{,}135{,}833$ multiplications)—an increase of $\approx 300\times$. Although there have been recent improvements on Valiant's initial constants [LYZ$^+$21], boolean UCs remain orders of magnitude larger than the circuits they can simulate, and arithmetic UCs would incur even higher constants [LMS16]

Disjunctive Templates for Succinct Proofs. When composing sublinear-sized proofs, succinctness directly implies communication-efficient disjunctive composition via statement combination; when a relation circuit's size is increased by a multiplicative factor of ℓ, a logarithmic-sized proof will only increase in size by an additive factor of $\log(\ell)$—only marginally larger.

This approach, however, increases the running time of the prover by (at least) a multiplicative factor ℓ. This is of special concern for succinct proof systems where the running time of the prover is often a bottleneck. In addition, many succinct proof systems have *space* complexity which grows linearly in the size of the circuit; in this case, the space requirements also increase by a factor ℓ.

The use of the *simulation* template in the sublinear setting has not yet been explored. We make the following initial observations:

- *Faster Simulators Means Faster Prover Time:* The key feature of the *simulation* template is the use of the simulator in each of inactive clauses. While the runtime of a simulator is typically proportional to the runtime of the prover in linear-sized zero-knowledge protocols, in sublinear-sized proofs it is common to have simulators that are more efficient—either asymptotically or concretely—than the prover. This observation means that applying the *simulation* template to sublinear-sized zero-knowledge proofs could produce disjunctive composition techniques that do not require the prover to pay—from a computational perspective—for the inactive clauses, resulting in significantly faster (and more space-efficient) provers than those produced by applying the *statement combination* template.

- *Communication Overhead Can Be Avoided:* The seminal construction of [CDS94] yields a protocol whose communication complexity is linear in ℓ. In a recent work, Goel et al. [GGHAK22] proposed a new instantiation of the *simulation* template for Σ-protocols that can achieve the same results while only introducing an additive term in $\log(\ell)$ to the proof size. At a high level, they observe that is is possible to simulate the inactive clauses such that they share a third round message with the active clause. When simulation is carried out in this way, there is no need for the prover to send transcripts for each clause, removing the communication overhead of [CDS94].

In this work, we use these two observations to motivate the study of applying the *simulation* template to the the disjunctive composition of sublinear-sized proofs. When taken together, these observations should facilitate the "the best of both worlds:" concrete computational savings for the prover without incurring any meaningful communication overhead. However, it is not immediately clear how to mobilize these observations into a concrete protocol proposal. In the paragraphs that follow, we start by summarizing the approach of Goel et al. [GGHAK22] and then proceed to discuss sublinear-sized proofs.

2.2 Stacking Sigmas for Sublinear-sized Proofs

The Approach of Stacking Sigmas [GGHAK22]. Goel et al. [GGHAK22] propose a new instantiation of the *simulation* template. Their compiler applies to Σ-protocols (three-round public coin zero-knowledge protocols) that have the following two properties (such Σ-protocols are called *stackable* Σ-protocols in their work):

1. *Recyclable Third Round Messages:* The distribution of the third round message across all instances must be the same. That is, there exists an efficient randomized algorithm that can produce a third round message from the correct distribution. Critically, this algorithm must be independent of the statement.

2. *Deterministic Transcript Completion:* The protocol supports a *deterministic simulator* $\mathcal{S}_{\text{DET}}$ that can produce an accepting first round message when supplied with a challenge and an arbitrary third round message (from the third round message distribution). Importantly this simulator must be deterministic, as it will be run locally by both the prover and the verifier.

Their compiler is based on a *1-out-of-ℓ partially binding commitment scheme*, a vector commitment scheme that is only binding in a single (pre-selected) index. First the prover generates the first round message for the active clause $a_\mathfrak{a}$ honestly. Instead of directly sending this message, the prover instead commits to a vector containing $a_\mathfrak{a}$ in the $\mathfrak{a}^{\text{th}}$ position and zeros in all other positions such that the binding position is $\mathfrak{a}$. The verifier then sends a challenge c to the prover as normal. Next, the prover generates the third round message for the active clause $z_\mathfrak{a}$. Rather than generate a separate third round message for the inactive clauses, the prover instead *reuses $z_\mathfrak{a}$ as the third round message for all clauses.* To do this, the prover uses the special deterministic simulator $\mathcal{S}_{\text{DET}}$ to produce a_i such that $a_i, c, z_\mathfrak{a}$ is an accepting transcript for the statement $\mathbb{x}_i$. The prover's final message consists of $z_\mathfrak{a}$ along with the randomness used to open the 1-out-of-ℓ partially binding commitment scheme to the vector $(a_1, \ldots, a_\ell)$. The verifier is then able to recompute the values a_i independently, checks that each transcript is accepting, and makes sure that the commitment matches.

Stackable Zero-Knowledge Interactive Protocols. In order to apply the *simulation* template to multi-round protocols, we must first extend Goel et al.'s notion of stackability to the muti-round setting (i.e., more than three-round setting). We extend the notion of recyclable messages so that it naturally applies to multi-round protocols. Goel et al. consider the distribution of third round messages with respect to the statement, we define a more fine-grained notion that consider the *joint distribution* of *parts* of multiple prover messages (i.e., messages sent across different rounds) with respect to the statement. That is, we let a part of each prover message be considered *recyclable*, in that it can be *re-used* across multiple statements. In order to be considered recyclable, it must be possible to design a randomized simulator $\mathcal{S}_{\text{RAND}}$ that can produce these messages independently of the statement. We note that identifying the recyclable component of each prover message is up to the protocol designer and it may be possible to produce multiple recyclable message sets for any given protocol.

With this multi-round notion of message recycablility, we are ready to understand how to deterministically complete the protocol transcript. Goel et al.'s deterministic simulator for Σ-protocols does a statement-dependent mapping of the third round message (*i.e.* the set of recyclable prover messages for a Σ-protocol) to the first round message (*i.e.* the remaining prover messages). We define the notion of deterministic simulation in the exact same way in the multi-round setting: the deterministic simulator $\mathcal{S}_{\text{DET}}$ takes as input (1) the statement $\mathbb{x}$, (2) the verifier challenges, and (3) the recyclable components of each message, and produces the completion of each prover message such that they form an accepting transcript for the statement $\mathbb{x}$.

Stacking Multi-round Protocols. To stack multi-round zero-knowledge interactive protocols, we begin by partitioning each prover message of the protocol into two parts: a recyclable part $m_{\text{RAND},i}$ and a deterministic completion $m_{\text{DET},i}$. The prover then runs a modified version of the original prover for the active clause. When the prover would send a recyclable part of a message, it simply sends the message directly. When the prover would send a non-recyclable message, it instead uses a 1-out-of-ℓ binding commitment scheme to commit to a vector containing the message in the active clause's index. In the final round of the protocol, the prover uses the deterministic simulator to compute the "missing" non-recyclable messages for the inactive clauses and opens all of the commitments.

Finding the Recyclable Messages. We note that there may not be a unique partitioning of the messages into recyclable and non-recyclable components. For instance, consider the following toy example: imagine a protocol to prove that $\mathbb{x} \in \mathcal{L}$ contains two messages a and b, such that a and b are random elements of $\{0,1\}^{|\mathbb{x}|}$ subject to the constraint that $a \oplus b = \mathbb{x}$. Clearly it is not possible to partition the set of messages such that both a and b are recyclable. However, a can be consider recyclable while b is not, or vice versa. As such, choosing the most optimal partition (either with respect to prover runtime or proof size) may require a small amount of manual optimization. We discuss an informal process for finding the recyclable message set in Section 3.5.

2.3 Speed-Stacking Interactive Oracle Proofs

A key technique for obtaining sublinear-sized interactive arguments is the cryptographic "compilation" of interactive oracle proofs (IOPs) [Kil92, Mic94, BCS16, RRR16]. An interactive oracle proof is an interactive proof system where the verifier, rather than reading the messages it receives in their entirety, has oracle access to each message and can query the messages at any index. IOPs can be viewed as a natural multi-round generalization of the notion of *probabilistically checkable proof* (PCP) [BFLS91]. All IOPs discussed in this paper will be public-coin. A zero-knowledge IOP additionally has an efficient simulator: given the verifier's random tape, the simulator computes oracle responses to the verifier's queries which have the same distribution as in the real interaction. Given a succinct vector commitment scheme (e.g. a Merkle tree), an IOP can be transformed into a succinct interactive argument as follows [BCS16]: in each round, the prover simply computes a commitment to the message and sends the commitment to the verifier; the verifier then responds with the set of query points and the prover provides opening proofs for the responses.

In this section we give an overview of our results on the stackability of IOP-based succinct arguments. We provide a two-part framework: we first define a notion of stackability for IOPs, and then show how a stackable IOP can be "compiled" into a stackable interactive argument — with some minor tweaks the existing compiler outlined above preserves "stackability". We show that several interactive oracle proofs (IOPs) are stackable, specifically Aurora [BCR+19] and a variant of Fractal [COS20] that we call Stactal. Finally, we outline why it is possible to achieve prover computational savings when compiling these protocols. What follows is an informal description of the definitions and techniques described formally in Section 4. The central definition is the notion of a "stackable IOP":

Stackable IOPs. A stackable IOP is a zero-knowledge IOP with a particular simulation strategy: there exists a partition of the k oracles (rounds) into R_{rec} and $[k] \setminus R_{\mathsf{rec}}$, such (1) responses to queries for oracles in R_{rec} can be sampled *independently* from the relation/statement. (2) while responses to queries for oracles in $[k] \setminus R_{\mathsf{rec}}$ can be computed *deterministically* from the relation/statement and other query answers.

Inutively a stackable IOP enables reusing the same oracles in R_{rec} to simulate multiple IOPs for distinct relations/statement, while communicating the responses for the remaining (distinct) oracles in $[k] \setminus R_{\mathsf{rec}}$ requires no additional communication – since the expected responses can be deterministically computed by the verifier (by running the simulator).

Stackable IOPs to Stackable IPs. Analogously to the way that IOPs can be compiled into arguments in the plain model, stackable IOPs can be compiled into stackable arguments in the plain model. We show that the existing IOP to IP compiler (outlined above) from vector commitments, can be adapted to preserve the efficient "stackability" of the underlying IOP. In order to preserve efficient simulation for the inactive clauses we need the vector commitment scheme to allow committing to and opening a subvector in time that depends only on the size of the subvector. We show that specific instantiations of Merkle trees satisfy this requirement.

Efficiency. One of the advantages of IOPs over other sublinear-sized proofs is that the running time of the IOP verifier can be *polylogarithmic* in the size of the statement. To maintain this property when applying our stacking compiler, we require also that the (instance-dependent component of the) simulator be similarly efficient. This is typically not a design goal for simulators, since polynomial (rather than polylogarithmic) efficiency suffices for zero-knowledge. As such, the security proofs of many existing protocols construct simulators which are not efficient enough for us. In some cases, all that is required is a more careful simulator construction. In others, to achieve efficient simulation we must substantially modify the protocol itself.

Showing Stackability. Many IOP constructions share a similar basic structure, consisting of two main parts: an encoded protocol, where soundness holds assuming that the prover's messages are close to words in an error-correcting code, and a proximity test, which guarantees that this condition holds. The code of choice for most constructions is the Reed–Solomon code, the code of evaluations of low-degree univariate polynomials over finite field $\mathbb{F}$ on some domain $L \subseteq \mathbb{F}$. Achieving zero-knowledge for protocols constructed in this way typically involves only two techniques:

(1) **Bounded independence:** when the prover sends an encoding of a secret vector $v \in \mathbb{F}^k$, rather than directly encoding v, it chooses a random vector $r \in \mathbb{F}^b$ and encodes $v \| r \in \mathbb{F}^{k+b}$. The properties of the Reed–Solomon code guarantee that, under a mild condition on the evaluation domain L, the answers to any set of b queries to a codeword are distributed uniformly at random in $\mathbb{F}$ (that is, the code is b-wise independent). To simulate, the simulator simply answers any verifier query uniformly at random.

(2) **Masking:** often the verifier needs to check some linear property P with respect to the prover's messages (a property $P \subseteq \mathbb{F}^\ell$ is linear if it is an $\mathbb{F}$-linear subspace of $\mathbb{F}^\ell$). Examples of such properties include the Reed–Solomon code itself (low-degree testing), or the subcode of the Reed–Solomon code consisting of polynomials whose evaluations over a set $S \subseteq \mathbb{F}$ sum to zero (univariate sumcheck).

Linear properties allow for zero-knowledge via random self-reduction: to show that $f \in P$, the prover sends a uniformly random word $r \in P$ (the "mask"), the verifier chooses a challenge $\alpha \in \mathbb{F}$ uniformly at random, and the prover and verifier then engage in a protocol to show that $\alpha f + r \in P$. To simulate, the simulator first generates a transcript showing that $q \in P$ for uniformly random $q \in P$; it then answers queries to r by "querying" $q - \alpha \cdot f$. Note that this simulation strategy requires that the simulator can simulate some number of queries to f, which is typically achieved through bounded independence as described above.

These two techniques lend themselves to the [GGHAK22] stacking approach, as follows. Simulation for (1) is trivially instance-independent (recyclable), since the simulator simply answers queries uniformly at random. For (2), observe that provided P is an instance-independent property, the process of sampling a protocol transcript showing that $q \in P$ is also instance-independent. Given q, queries to the mask r can then be answered deterministically. Hence for essentially all zero-knowledge IOP constructions, every message is fully recyclable except for those in which the prover sends a random mask.

To demonstrate the above approach, we consider two key IOP constructions from the literature: Aurora [BCR+19] and Fractal [COS20]. We start with the more complicated case of Fractal:

Fractal/Stactal. Fractal is a Holographic IOP, which means it can be compiled to a preprocessing zkSNARK in which the verifier's running time is polylogarithmic. Unfortunately Fractal is *not* an efficiently stackable IOP. The challenge originates in the Fractal "holographic lincheck" which proves, for encodings of (secret) vectors x, y, that $Mx = y$ for a public matrix M that is "holographically" encoded. The central problem with this lincheck is that it reduces to opening a bivariate polynomial $u_M(\beta, \alpha)$ at a random $\beta, \alpha \in \mathbb{F}$. Since this evaluation depends (deterministically) on every nonzero entry of M, simulation requires reading all of M to compute the correct $u_M(\alpha, \beta)$. As a result, the stacked verifier becomes inefficient. To alleviate this we introduce "Stactal", a variant of "Fractal" which *does* admit very efficient stacking. "Stactal" modifies the lincheck protocol to allow the prover to extend the matrix M to a larger matrix M' that is "padded" with random values. This introduces sufficient bounded independence that $u_{M'}(\beta, \alpha)$ is uniformly random (and so independent of M).

Aurora. Aurora is naturally a stackable IOP: since the verifier in Aurora is quasi-linear the stacking simulator has enough "computational budget" to read all of M. Hence, the (simpler, non-holographic) lincheck of Aurora can easily be simulated with the same time complexity as the verifier.

2.4 Speed-Stacking Folding Arguments

The next class of sublinear-sized zero-knoweldge proofs are ones based on "folding arguments". These are interactive zero-knowledge protocols with logarithmic round complexity. The two most prominent examples of such protocols are Compressed Σ-protocols [AC20, ACK21, ACF21] and Bulletproofs [BCC+16, BBB+18].

Folding Technique. The central object in all folding argument based *zero-knowledge* protocols is a sub-linear, interactive, logarithmic-round *non zero-knowledge* protocol to demonstrate that the prover has knowledge of a witness. The key idea used in the design of these logarithmic-round *non zero-knowledge* protocols is to enable the prover (using randomness from the verifier) to "fold" the witness in on itself, thereby reduce the size of the witness by half in each round. This step is repeated for logarithmic number of rounds, until the witness is reduced to a constant size.

In order to build a sub-linear *zero-knowledge* protocol using the above *non zero-knowledge* protocol, most existing constructions rely on the same rough template—these constructions begin with a constant round "base" protocol containing a large final round message (*i.e.,* linear in the size of the original witness) that achieves zero-knowledge. Finally, instead of actually sending this large final round message, the prover uses the above (non zero-knowledge) recursive folding approach to prove knowledge of this large message over logarithmic rounds. The key observation used here is that since the "base" protocol achieves zero-knowledge even if the large final round message is sent to the verifier in the clear, it suffices for the prover to use the above *non zero-knowledge* sublinear protocol to prove knowledge of this message.

Folding Argument Based ZK-IP are Stackable. Most folding argument based zero-knowledge protocols including Compressed Σ-protocols and Bulletproofs fall into the category of sublinear-sized proofs, with verifier runtime roughly equivelent to the prover runtime. We observe that the folding arguments we study are *stackable* such that the prover's entire last round message is recyclable.[3] To see this, note that if the last round message could instead be computed deterministically using the rest of the transcript, without access to the witness (which is the case for non-recyclable messages), then this last round message could also be computed by the verifier independently and there would be no need to send this message.[4] Specifically, this holds for the final round message in the "base" protocol in both Bulletproofs [BCC+16, BBB+18] and Compressed Σ-protocols [AC20, ACK21, ACF21]. Because this last round message is recyclable, we observe that the entire folding argument—a proof of knowledge of a recyclable message—is itself recyclable and can be reused across clauses. We note, however, that the mere fact that these protocols are stackable doesn't immediately imply that there are vast computational savings available when stacking folding arguments. Interestingly, we find that stacking Compressed Σ-protocols offers significant computational savings, while stacking Bulletproofs does not.

Computational Savings via Stacking. As discussed earlier, our hope to get computational savings when stacking sublinear zero-knowledge proof systems for disjunctions, stems from the observation that the simulator in such proofs is typically much faster than the prover algorithm. This is because, the verifier in most such protocols runs in sublinear time and since the job of the simulator is to essentially "fool" the verifier into accepting a simulated proof, the work required from a simulator is somewhat proportional to the work done by the verifier. As a result, being able to replace the prover algorithm with the simulator for all inactive clauses in the disjunction, can yield significant computational savings.

Folding argument based sublinear proof systems we consider, however, do not have a sublinear-time verrifier. In fact, the work done by the verifier in these protocols is asymptotically equivalent to the work done by the prover. Hence, the overall simulator is not asymptotically more efficient than the prover algorithm. For computational savings, here we rely on our second observation about simulators: the simulator can often be split into two parts $\mathcal{S}_{\text{RAND}}$ and $\mathcal{S}_{\text{DET}}$, where $\mathcal{S}_{\text{RAND}}$ is responsible to simulating the statement independent part of the transcript, while $\mathcal{S}_{\text{DET}}$ simulates messages that are dependent on the statement/relation. Since the messages simulated by $\mathcal{S}_{\text{RAND}}$ are statement independent, the resulting messages can be re-used/recycled in all the inactive clauses, while we must compute the messages simulated by $\mathcal{S}_{\text{DET}}$ separately for each clause. If $\mathcal{S}_{\text{DET}}$ is significantly faster than $\mathcal{S}_{\text{RAND}}$, we can still hope to get significant concrete computational savings for the prover when stacking such protocols (even if there are no asymptotic savings). This is where the crucial difference between Bulletproofs and Compressed Σ-protocols appears.

[3] We formalize this claim in Section 3.5.

[4] We do note, however, that some protocols include deterministic messages in the final round in order to minimize verifier computation.

In Compressed Σ-protocols, the statement/relation-dependent verifier computation only consists of simple field operations, while the statement independent verification consists of expensive group multi-exponentiations. As a result $\mathcal{S}_{\text{DET}}$ is significantly more efficient than $\mathcal{S}_{\text{RAND}}$, yielding concrete computational savings for the prover upon stacking. Unfortunately, Bulletproofs lie on the other end of the spectrum, where the runtimes of $\mathcal{S}_{\text{DET}}$ and $\mathcal{S}_{\text{RAND}}$ are approximately the same (ie. up to a small constant factor). This is alludes to an interesting distinction between these two folding argument based protocols and motivates the design of sublinear-sized zero-knowledge protocols in which the verifier's *statement-dependent* computation is much faster than the verifier's *statement-independent* verifier computation—in other words, protocols that are more amenable to speed-stacking. We now give a brief technical overview of Compressed Σ-protocols and Bulletproofs to further highlight this distinction and demonstrate stackablilty.

Compressed Σ-Protocol. Compressed Σ-protocols [AC20, ACK21, ACF21] provide zero-knowledge interactive protocols for proving knowledge of openings of linear forms, i.e., proving that the output of a linear function f applied to a vector $\mathbf{x}$ contained in a commitment P equals some publicly known value y. The "base" protocol in Compressed Σ-protocols, performs a randomized self-reduction, in which the problem is reduced to the task of proving a different (related) statement for the same relation in a *privacy-free* way. To prove this related statment, they provide a log-sized privacy-free argument.

We observe that the entire folding argument transcript can be reused during stacking (after making very minor modifications to the protocol), but not all of the computation can be reused. That is, the randomized simulator $\mathcal{S}_{\text{RAND}}$ creates the folding argument transcript and then deterministic simulator $\mathcal{S}_{\text{DET}}$ completes the transcript, but runs in time linear in the size of the vector $\mathbf{x}$ ($\mathcal{S}_{\text{DET}}$ recursively folds the linear form to facilitate the final check). However, we observe that the linear number of operations in $\mathcal{S}_{\text{DET}}$ are all *field arithmetic*, and $\mathcal{S}_{\text{DET}}$ contains only a single group exponentiation and a single group multiplication, with no multi-exponentations. As a result, simulating each additional inactive clause remains significantly faster than the prover algorithm.

We note that were able to handle disjunctions where each clause $i \in [\ell]$ could have a different homomorphic linear function f_i and a different commitment P_i. This is a stronger notion of disjunctions than the ones considered in [ACF21], which give proofs where either the homomorphism or commitment is fixed across a disjunction of multiple clauses.

Bulletproofs. The main task in the initial "base" protocol in Bulletproofs [BBB$^+$18] is reduced to transforming any given relation into a *privacy-free* inner-product relation. This is followed by an efficient folding argument for $\mathcal{R}_{\text{innerprod}}$. This approach is used to acheive efficient zero-knowledge for range proofs and circuit satisfiability. Because the last message of the "base" protocol is recyclable, the folding argument transcript can be reused, and we find that only two of the messages in the "base" protocol are non-recyclable. However, simulating these two non-recyclable messages requires performing multi-exponentiations dependent on the relation function. As a result, any savings obtained from being able to recycle the entire *non zero-knowledge* sublinear-sized folding argument at the end across all inactive clauses are more-or-less eclipsed by the computation involved in individually simulating the above two non-recyclable messages for each inactive clause.

As such, stacking Bulletproofs for "true" disjunctions does not seem to offer considerable savings. We do note, however, one might consider set-membership for range proofs (ie. $\exists x_i \in \{x_1, \ldots, x_\ell\}$ st. $x_i \in \mathsf{Range}$), where appealing to NP completeness is expensive. Because the the statement dependent computation (that must be run separately for each clause) is remarkably inexpensive (involving only one group exponentiations and a constant number of group multiplications), applying the compiler in this case may be valuable. While set membership for range proofs is not particularly valuable, studying Bulletproofs illuminates fundamental differences between Compressed Σ-protocols and Bulletproofs, despite their superficial similarities. Moreover, this highlights the key parameters to keep in mind when stacking a protocol and points to new considerations when designing new—potentially stackable—zero-knowledge proof systems.

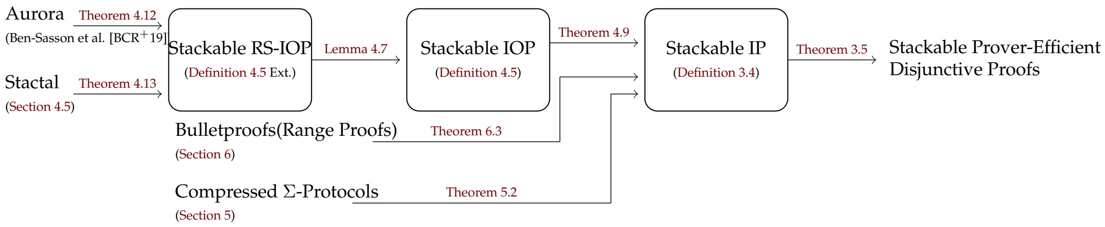

Figure 1: A roadmap for the results in our paper.

2.5 Roadmap to Our Results

In order to achieve our results, we follow the roadmap depicted in Figure 1. Our formalization of a Stackable Interactive Protocol appears in Section 3. Our speed-stacking compiler for ZK-IP is presented in Section 3.4. In Section 4, we show how to speed stack IOPs, including the top pathway in Figure 1. In Section 5 and Section 6 we show how to speed stack folding arguments (bottom pathway in Figure 1).

2.6 Notation

When discussing interactive protocols in this work, we will use both interactive turing machine notation, *ie.* $\langle \mathsf{P}, \mathsf{V} \rangle (\mathbb{x})$, and algorithmic notation, *ie.* the i^{th} message is computed with algorithm P_i. More formally, we assume that for zero-knowledge interactive proofs, the interaction $\langle \mathsf{P}, \mathsf{V} \rangle (\mathbb{x})$ contains an ordered list of algorithms P_i, such that the prover computes their i^{th} message using P_i. We use $\mathtt{CC}(\Pi)$ to denote the computational complexity of Π, and let $\mathsf{Time}(\Pi)$ denote the runtime of the algorithm Π. Finally, we note that our work spans different lines of research that commonly leverage different notation for the same concepts. Wherever possible we have made notation internally consistent, at the cost of being inconsistent with prior work.

For an NP relation $\mathcal{R}$, we denote the instance as $\mathbb{x}$ and the witness as $\mathbb{w}$. Let the number of clauses in the disjunction ℓ and the index of the active clause as $\mathfrak{a}$. Where applicable, we use N to denote the relevant size of $\mathbb{x}$. We use multiplicative notation for groups and group operations.

In lieu of a single preliminaries section for the entire paper, we have moved preliminaries content into localized preliminary sections in the most relevant technical section. We make this choice in recognition that understanding our technical results requires familiarity with a large number of existing work. As such, we aim to defer complexity until it is most needed, where possible. We include a standard definition of Public-Coin Zero-Knowledge Interactive Proofs in Section 3.1. For completeness we include the definition of Partially-Binding Vector Commitments, originally introduced in [GGHAK22], in Section 3.2. We give definitions for Holographic Interactive Oracle Proofs and Reed-Solomon encoded Holographic Interactive Oracle Proofs in Section 4.1 . We also provide a definition of hiding key-value commitments in Definition 4.6.

3 Stacking Zero-Knowledge Interactive-Proofs

In this section, we extend the notion of "stackability" introduced in Goel et al. [GGHAK22] to the multi-round setting proceed to give a generic compiler that can transform a stackable ZK-IP into a ZK-IP for disjunctive statements. We start by recalling the relevant preliminaries before proceeding to our technical results: definiting Stackable ZK-IP (Section 3.3), presenting our stacking compiler (Section 3.4), and providing a heuristic mechanism for preparing ZK-IP protocols for stacking (Section 3.5).

3.1 Public-Coin Zero-Knowledge Interactive Proofs

To establish notation, we start by recalling the definition of a public-coin zero-knowledge interactive proof.[5]

Definition 3.1 (Public-Coin ZK-IP). *Let $\mathcal{R}$ be an NP relation. A public coin zero-knowledge interactive-proof (ZK-IP) Π for $\mathcal{R}$ is an interactive protocol between a prover P and a verifier V consisting of k prover messages and $k-1$ verifier messages. The prover is defined by an interactive algorithm $\mathsf{P} = \{\mathsf{P}_i\}_{i\in[k]}$ and the verifier is defined by a predicate ϕ, with the following interfaces:*

- $m_i \leftarrow \mathsf{P}_i(\mathbb{x}, \mathbb{w}, (c_j)_{j\in[i-1]}; p_{\mathsf{rand}})$ $(\forall i \in [k])$*: On input $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, challenge messages $\{c_j\}_{j\in[i-1]}$ received from the verifier in previous rounds and prover randomness p_{rand}, this algorithm outputs message m_i that P sends to V in the i^{th} round.*
- $c_i \xleftarrow{\$} \{0,1\}^\kappa$, $\forall i \in [k-1]$*: The verifier V samples a random challenge c_i to send to P in the i^{th} round.*
- $b \leftarrow \phi(\mathbb{x}, \{m_i\}_{i\in[k]}, (c_i)_{i\in[k-1]})$*: On input the statement $\mathbb{x}$, prover's messages $(m_i)_{i\in[k]}$ and the challenge messages $(c_i)_{i\in[k-1]}$, this algorithm run by V, outputs a bit $b \in \{0,1\}$.*

We denote by $\langle \mathsf{P}(\mathbb{x}, \mathbb{w}), \mathsf{V}(\mathbb{x})\rangle$ the random variable corresponding to the bit b output by V in the above interaction.

For completeness, we include the standard security notions for ZK-IP.

- ***Completeness:*** *A public-coin ZK-IP Π is* complete *if for any $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$,*

$$\Pr[\langle \mathsf{P}(\mathbb{x}, \mathbb{w}), \mathsf{V}(\mathbb{x})\rangle = 1] = 1 \ .$$

- ***Computational Soundness.*** *Π has computational soundness if for all $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$ and PPT provers $\tilde{\mathsf{P}}$,*

$$\Pr[\langle \tilde{\mathsf{P}}, \mathsf{V}(\mathbb{x})\rangle = 1] \le \mathsf{negl}(\lambda).$$

- ***Special Honest Verifier Zero-Knowledge.*** *A public-coin ZK-IP $\Pi = (\{\mathsf{P}_i\}_{i\in[k]}, \phi)$ is said to be special honest verifier zero-knowledge, if there exists a PPT simulator $\mathcal{S}$, such that for any $\mathbb{x}, \mathbb{w}$ such that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, it holds that*

$$\{\{m_i\}_{i\in[k]} \mid \{c_i \xleftarrow{\$} \{0,1\}^\kappa\}_{i\in[k-1]}; \{m_i\}_{i\in[k]} \leftarrow \mathcal{S}(1^\lambda, \mathbb{x}, \{c_i \xleftarrow{\$} \{0,1\}^\kappa\}_{i\in[k-1]})\} \approx_c$$
$$\{\{m_i\}_{i\in[k]} \mid p_{rand} \xleftarrow{\$} \{0,1\}^*; \{m_i \leftarrow \mathsf{P}_i(\mathbb{x}, \mathbb{w}, \{c_j\}_{j\in[i-1]}; p_{rand})\}_{i\in[k]}; \{c_i \xleftarrow{\$} \{0,1\}^\kappa\}_{i\in[k-1]}\}$$

Preprocessing. We additionally consider ZK-IPs with a preprocessing step; we refer to these as preprocessing ZK-IPs. Preprocessing ZK-IPs operate on *indexed relations $\mathcal{R}$*, consisting of index-instance-witness triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$. We split the preprocessing into a statement-independent randomized setup algorithm Setup and an index-dependent (instance-independent) deterministic indexer I.

- The setup Setup takes in 1^λ and outputs public parameters pp.
- The indexer I takes in public parameters pp and the index $\mathbb{i}$ and outputs proving key ipk and verification key ivk.

The completeness definition now provides the honest prover with ipk and the honest verifier with ivk. The soundness definition now provides the adversary with pp and the honest verifier with ivk. The zero knowledge definition now allows the simulator to generate pp and provides the honest verifier with ivk.

Well-Behaved Simulator. We recall the notion of a well-behaved simulator, as defined by Goel et al. [GGHAK22]. We give the natural generalization to ZK-IPs here.

[5] Formally, the constructions in this paper are *arguments* rather than proofs because they are only computationally sound.

Definition 3.2 (Well-Behaved Simulator)**.** *We say that a ZK-IP for a* NP *language* $\mathcal{L}$ *and associated relation* $\mathcal{R}(\mathbb{x}, w)$ *has a* well-behaved simulator *if the simulator* $\mathcal{S}$ *defined for Special Honest Zero-Knowledge has the following property: For any statement* $\mathbb{x}$ *(for both* $\mathbb{x} \in \mathcal{L}$ *and* $\mathbb{x} \notin \mathcal{L}$*),*

$$\Pr\left[\phi(\mathbb{x}, \{c_i\}_{i\in[k-1]}, \{m_i\}_{i\in[k]}) = 1 \;\middle|\; \{c_i \xleftarrow{\$} \{0,1\}^{\kappa}\}_{i\in[k-1]}; \{m_i\}_{i\in[k]} \leftarrow \mathcal{S}(1^{\lambda}, \mathbb{x}, \{c_i \xleftarrow{\$} \{0,1\}^{\kappa}\}_{i\in[k-1]})\}\right] = 1$$

Recall that Goel et al. [GGHAK22] prove that simulators are well-behaved for Σ protocol, without loss of generality. It is simple to see that this result can extend to ZK-IP.

3.2 Partially-Binding Vector Commitments

In this section, we recall the partially-binding vector commitments introduced by [GGHAK22]. For a construction of these commitment schemes, we refer the reader to the work of Goel et al.

Definition 3.3 (t-out-of-ℓ Binding Vector Commitment)**.** *A t-out-of-ℓ binding non-interactive vector commitment scheme with message space $\mathcal{M}$, is defined by a tuple of the PPT algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{EquivCom}, \mathsf{Equiv}, \mathsf{BindCom})$ *defined as follows:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda})$ *On input the security parameter λ, the setup algorithm outputs public parameters* pp.
- $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B)$*: Takes public parameters* pp *and a t-subset of indices $B \in \binom{[\ell]}{t}$. Returns a commitment key* ck *and equivocation key* ek.
- $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}; r)$*: Takes public parameter* pp*, equivocation key* ek*, ℓ-tuple $\mathbf{v}$ and randomness $r \in \mathcal{R}$. Returns a partially-binding commitment* com *as well as some auxiliary equivocation information* aux.
- $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}, \mathbf{v}', \mathsf{aux})$*: Takes public parameters* pp*, equivocation key* ek*, original commitment value $\mathbf{v}$ and updated commitment values $\mathbf{v}'$ with $\forall i \in B : \mathbf{v}_i = \mathbf{v}'_i$, and auxiliary equivocation information* aux*. Returns equivocation randomness r.*
- $\mathsf{com} \leftarrow \mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v}; r)$*: Takes public parameters* pp*, commitment key* ck*, ℓ-tuple $\mathbf{v}$ and randomness r and outputs a commitment* com*. Note that this algorithm does not use the equivocation key* ek*. This algorithm plays a similar role to that of* Open *in a typical commitment scheme.*

The properties satisfied by the above algorithms are as follows:

(Perfect) Hiding: *The commitment key* ck *and commitment* com *(perfectly) hides the binding positions B and the equivocated values, even when opening the commitment. Formally, for all $\mathbf{v}^{(1)}, \mathbf{v}^{(2)} \in \mathcal{M}^{\ell}$, $B^{(1)}, B^{(2)} \in \binom{[\ell]}{t}$ and a 'valid equivocation' for both vectors $\mathbf{v}' \in \mathcal{M}^{\ell}$ i.e. $\forall i \in B^{(1)} : \mathbf{v}_i^{(1)} = \mathbf{v}'_i$ and $\forall i \in B^{(2)} : \mathbf{v}_i^{(2)} = \mathbf{v}'_i$) and* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda})$*, the two distributions are equal:*

$$\left[(\mathsf{ck}, \mathsf{com}, r') \;\middle|\; \begin{array}{c} (\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B^{(1)}); r \xleftarrow{\$} \{0,1\}^{\lambda}; \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}^{(1)}; r); \\ r' \leftarrow \mathsf{Equiv}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}^{(1)}, \mathbf{v}', \mathsf{aux}) \end{array}\right]$$

$$\underline{\underline{p}}$$

$$\left[(\mathsf{ck}, \mathsf{com}, r') \;\middle|\; \begin{array}{c} (\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B^{(2)}); r \xleftarrow{\$} \{0,1\}^{\lambda}; \\ (\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}^{(2)}; r); \\ r' \leftarrow \mathsf{Equiv}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}^{(2)}, \mathbf{v}', \mathsf{aux}) \end{array}\right]$$

The definition essentially states that any two sets of binding positions and original vectors, which could 'explain' the provided opening of $\mathbf{v}'$, are indistinguishable.

(Computational) Partial Binding: *An adversary (that generates* ck *itself) cannot equivocate on more than* $\ell - t$ *positions, even across multiple different commitments. Define the function* $\Delta : \mathcal{M}^\ell \times \mathcal{M}^\ell \mapsto \mathbb{P}([\ell])$ *taking two vectors and returning the set of indexes on which the vectors differ:*

$$\Delta(\mathbf{v}, \mathbf{v}') = \{j \in [\ell] : v_j \neq v'_j\} .$$

Consider an adversary $\mathcal{A}$ *that outputs* ck *and a set* S *of pairs of openings* $S \subseteq \mathcal{M}^\ell \times \mathcal{M}^\ell \times \mathcal{R} \times \mathcal{R}$ *such that each pair of openings share the same commitment under* ck*, then the set of index on which the openings differ across all pairs has cardinality at most* $t - \ell$*, formally, we require that the following probability is negligible in* λ *for any PPT* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{r} \left|\bigcup_{(\mathbf{v},\mathbf{v}',r,r')\in S} \ \Delta(\mathbf{v},\mathbf{v}')\right| > \ell - t \wedge \\ \forall (\mathbf{v},\mathbf{v}',r,r') \in S. \quad \begin{array}{c}\mathsf{BindCom}(\mathsf{pp},\mathsf{ck},\mathbf{v};r) \\ = \mathsf{BindCom}(\mathsf{pp},\mathsf{ck},\mathbf{v}';r')\end{array} \end{array} \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{ck}, S) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}) \end{array}\right] .$$

Partial Equivocation: *Given a commitment to* $\mathbf{v}$ *under a commitment key* $\mathsf{ck} \leftarrow \mathsf{Gen}(\mathsf{pp}, B)$*, it is possible to equivocate to any* $\mathbf{v}'$ *as long as* $\forall i \in B : v_i = v'_i$*. More formally, for all* $B \in \binom{[\ell]}{t}$*, and all* $\mathbf{v}, \mathbf{v}' \in \mathcal{M}^\ell$ *st.* $\forall i \in B : v_i = v'_i$ *then:*

$$\Pr\left[\mathsf{BindCom}(\mathsf{pp},\mathsf{ck},\mathbf{v}';r') = \mathsf{com} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); r \xleftarrow{\$} \{0,1\}^\lambda; \\ (\mathsf{ck},\mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B); \\ (\mathsf{com},\mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp},\mathsf{ek},\mathbf{v};r); \\ r' \leftarrow \mathsf{Equiv}(\mathsf{ek},\mathbf{v},\mathbf{v}',\mathsf{aux}) \end{array}\right] = 1 .$$

We note that Goel et al. [GGHAK22, Corollary 1] include a concretely efficient construction of 1-of-ℓ partially binding vector commitments with $O(\lambda \log(\ell))$ communication in their paper.

3.3 Defining Stackable ZK-IP

Recently, Goel et al. [GGHAK22] introduced the notion of "stackability" for Σ-protocols (i.e., three-round ZK-IPs), and showed most natural Σ-protocols are stackable. At the heart of their approach is the observation that the simulators for common Σ-protocols can be divided into two components: a randomized, statement independent part, which we will denote $\mathcal{S}_{\text{RAND}}$,[6] and a deterministic, statement dependent part, which we denote $\mathcal{S}_{\text{DET}}$.

We extend their intuition to the multi-round setting. Intuitively, we require that each message in the protocol can be subdivided into two (potentially empty) parts: a *recyclable part* that can be reused across multiple statements, and a deterministically computable part. More formally, we assume that each prover message i of a ZK-IP is a concatenation of two parts—$m_{\text{RAND},i}$ and $m_{\text{DET},i}$. To satisfy stackability, we require that it is possible to generate the messages $\{m_{\text{RAND},i}\}_{i\in[k]}$ using a randomized, statement independent algorithm $\mathcal{S}_{\text{RAND}}$. Additionally, we require that there exists a deterministic simulator $\mathcal{S}_{\text{DET}}$ that can simulate the remaining parts of the messages $\{m_{\text{DET},i}\}_{i\in[k]}$ such that the resulting transcript matches an "honest" execution of the protocol.

Definition 3.4 (Stackable ZK-IP)**.** *Let* Π *be a ZK-IP consisting of* k *prover messages and* $k-1$ *verifier messages (see Definition 3.1) for a relation* $\mathcal{R}$*. For each* $i \in [k]$*, let* $m_i = (m_{\text{RAND},i}, m_{\text{DET},i})$ *and let* $M_{\text{RAND}} = (m_{\text{RAND},i})_{i\in[k]}$ *and* $M_{\text{DET}} = (m_{\text{DET},i})_{i\in[k]}$*. We say that* Π *is* Stackable*, if there exists a PPT simulator* $\mathcal{S}_{\text{RAND}}$ *and a polynomial-time*

[6] We depart from the notation introduced by Goel et al. [GGHAK22], in which this first part is instead discussed as an efficiently sampleable distribution, rather than a simulator. We note that these notions are clearly equivalent: the output of $\mathcal{S}_{\text{RAND}}$ defines a distribution from which elements can be efficiently sampled (namely, by running $\mathcal{S}_{\text{RAND}}$)

Stacking Compiler

Statement: $\mathbb{x} = \mathbb{x}_1, \ldots, \mathbb{x}_\ell$
Witness: $\mathbb{w} = (\mathfrak{a}, \mathbb{w}_\mathfrak{a})$
For each $i \in [k-1]$, the Prover and Verifier take turns sending messages:

- **Prover in Round i:** Prover computes $\mathsf{P}'_i(\mathbb{x}, \mathbb{w}; r^p) \to m_i$ as follows:
  - Parse $r^p = (r^p_\mathfrak{a} \| \{r_j\}_{j\in[k]})$.
  - Compute $(m_{\text{RAND},i,\mathfrak{a}}, m_{\text{DET},i,\mathfrak{a}}) \leftarrow \mathsf{P}_i(\mathbb{x}_\mathfrak{a}, \mathbb{w}_\mathfrak{a}; r^p_\mathfrak{a})$.
  - Set $\mathbf{v}_i = (v_{i,1}, \ldots, v_{i,\ell})$, where $v_{i,\mathfrak{a}} = m_{\text{DET},i,\mathfrak{a}}$ and $\forall j \in [\ell] \setminus \mathfrak{a}$, $v_{i,j} = 0$.
  - If $i = 1$, compute $(\mathsf{ck}, \mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B = \{\mathfrak{a}\})$.
  - Compute $(\mathsf{com}_i, \mathsf{aux}_i) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}_i; r_i)$.
  - If $i = 1$, send $m_i = (\mathsf{ck}, \mathsf{com}_i, m_{\text{RAND},i,\mathfrak{a}})$ to the verifier, otherwise send $m_i = (\mathsf{com}_i, m_{\text{RAND},i,\mathfrak{a}})$ to the verifier.
- **Verifier in Round i:** Verifier samples $c_i \xleftarrow{\$} \{0,1\}^\lambda$ and sends it to the prover.

Round k: Prover computes $\mathsf{P}'_k(\mathbb{x}, \mathbb{w}, \{c_j\}_{j\in[k-1]}; r^p) \to z$ as follows:

- Parse $r^p = (r^p_\mathfrak{a} \| \{r_j\}_{j\in[k]})$.
- Compute $(m_{\text{RAND},k,\mathfrak{a}}, m_{\text{DET},k,\mathfrak{a}}) \leftarrow \mathsf{P}_k(\mathbb{x}_\mathfrak{a}, \mathbb{w}_\mathfrak{a}, \{c_j\}_{j\in[k-1]}; r^p_\mathfrak{a})$.
- For $j \in [\ell] \setminus \mathfrak{a}$, compute $\{m_{\text{DET},i,j}\}_{i\in[k]} := \mathcal{S}_{\text{DET}}(\mathbb{x}_j, \{c_j\}_{j\in[k-1]}, \{m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]})$.
- For each $i \in [k]$, set $\mathbf{v}'_i = (m_{\text{DET},i,1}, \ldots, m_{\text{DET},i,\ell})$ and compute $r'_i \leftarrow \mathsf{Equiv}(\mathsf{pp}, \mathsf{ek}, \mathbf{v}_i, \mathbf{v}'_i, \mathsf{aux}_i)$ (where aux_i can be regenerated with r_i).
- Send $m_k = (m_{\text{RAND},k,\mathfrak{a}}, \{r'_i\}_{i\in[k]})$ to the verifier.

Verification: Verifier computes $\phi'(\mathbb{x}, \{m_i, c_i\}_{i\in[k-1]}, m_k) \to b$ as follows:

- For each $i \in [k]$, if $i = 1$, parse $m_i = (\mathsf{ck}', \mathsf{com}_i, m_{\text{RAND},i,\mathfrak{a}})$, else parse $m_i = (\mathsf{com}_i, m_{\text{RAND},i,\mathfrak{a}})$. Parse $m_k = (m_{\text{RAND},k,\mathfrak{a}}, \mathsf{ck}_i, m_{k,\mathfrak{a}}, \{r'_i\}_{i\in[k]})$
- For $j \in [\ell]$, compute $\{m_{\text{DET},i,j}\}_{i\in[k]} := \mathcal{S}_{\text{DET}}(\mathbb{x}_i, \{c_i\}_{i\in[k-1]}, \{m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]})$.
- For each $i \in [k]$, set $\mathbf{v}'_i = (m_{\text{DET},i,1}, \ldots, m_{\text{DET},i,\ell})$.
- Compute and return:

$$b = \bigwedge_{i\in[k]} \left(\mathsf{com}_i \overset{?}{=} \mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v}'_i; r'_i)\right) \bigwedge_{j\in[\ell]} \left(\phi(\mathbb{x}_j, \{(m_{\text{RAND},i,\mathfrak{a}}, m_{\text{DET},i,j})\}_{i\in[k]}, \{c_i\}_{i\in[k-1]})\right)$$

Figure 2: A compiler for stacking multiple instances of a stackable ZK-IP

computable, well-behaved (see Definition 3.2), deterministic simulator $\mathcal{S}_{\text{DET}}$, such that for each $C = (c_i)_{i\in[k-1]} \in (\{0,1\}^\kappa)^{k-1}$ and for all instance-witness pairs $(\mathbb{x}, \mathbb{w})$ st. $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, it holds that:

$$\Big\{(M, C) \mid r^p \xleftarrow{\$} \{0,1\}^\lambda; \forall i \in [k], (m_{\text{RAND},i} \| m_{\text{DET},i}) \leftarrow \mathsf{P}_i(\mathbb{x}, \mathbb{w}, (c_j)_{j\in[i-1]}; r^p)\Big\}$$
$$\approx \big\{((m_{\text{RAND},i} \| m_{\text{DET},i})_{i\in[k]}, C) \mid M_{\text{RAND}} \leftarrow \mathcal{S}_{\text{RAND}}(1^\lambda, C); M_{\text{DET}} := \mathcal{S}_{\text{DET}}(\mathbb{x}, C, M_{\text{RAND}})\big\}$$

The natural variants (perfect/statistical/computational) are defined depending on the class of distinguishers with respect to which indistinguishability holds.

3.4 Compiler for Stacking ZK-IPs

We now present our compiler that can transform any stackable ZK-IP into a ZK-IP for disjunctions. As discussed earlier, similar to Goel et al. [GGHAK22], the main idea behind this construction is to honeslty compute the transcript of the active clause and reuse its recyclable messages for all the inactive clauses.

Concretely, the prover starts by generating a $(\mathsf{ck}, \mathsf{ek})$ pair for the index associated with the active clause. Subsequently, in each round it computes messages for the active clause honestly and commits to the non-recyclable messages along with a bunch of 0s for the inactive clause using the partially-binding vector commitment scheme and commitment key ck. It sends this commitment along with the honestly computed recyclable message to the verifier. In the last round, upon receving all the challenge messages from the verifier, it simulates to "complete" the transcript of the inactive clauses and equivocates all of the previously computed commitments to a commitment of these messages and sends the assiciated commitment opening/randomness to the verifier. Based on the recyclable messages, the verifier also simulates the non-recyclable messages for each clause, and checks if they were honestly committed inside the commitment. It also checks if the resulting transcript for each clause is accepting.

Theorem 3.5. *Let Π be a stackable ZK-IP (see Definition 3.4) consisting of k prover messages and $k-1$ verifier messages for the NP relation $\mathcal{R} : \mathcal{X} \times \mathcal{W} \to \{0,1\}$ and let $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{EquivCom}, \mathsf{Equiv}, \mathsf{BindCom})$ be a 1-out-of-ℓ binding vector commitment scheme (See Definition 3.3). For any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, the compiled protocol Π' described in Figure 2 is a stackable ZK-IP for the relation $\mathcal{R}' : \mathcal{X}^\ell \times ([\ell] \times \mathcal{W}) \to \{0,1\}$, where $\mathcal{R}'((\mathbb{x}_1, \ldots, \mathbb{x}_\ell), (\mathfrak{a}, \mathbb{w})) := \mathcal{R}(\mathbb{x}_\mathfrak{a}, \mathbb{w})$.*

Proof of Theorem 3.5. We now prove that the protocol $\Pi' = \{\mathsf{P}'_i\}_{i \in [k]}$ described in Figure 2 is a stackable ZK-IP for the relation $\mathcal{R}'((\mathbb{x}_1, \ldots, \mathbb{x}_\ell), (\mathfrak{a}, \mathbb{w})) := \mathcal{R}(\mathbb{x}_\mathfrak{a}, \mathbb{w})$.

Completeness. Completeness follows directly from the completeness of the underlying ZK-IP Π and the partially binding commitment scheme. Note that because the underlying ZK-IP has a well-behaved simulator, the prover will not produce non-accepting transcripts on any clauses embedding false instances.

Computational Soundness. We describe how an efficient prover $\tilde{\mathsf{P}}'$ for Π' that convinces with probability $\delta > 0$ can be used to derive a cheating prover $\tilde{\mathsf{P}}$ for the original ZK-IP Π that convinces or breaks partial binding with probability $\delta/(2\ell)$. Without loss of generality, $\tilde{\mathsf{P}}'$ is deterministic, and so we can view it as a function from $(k-1)$-tuples of challenges to protocol transcripts. We define $\tilde{\mathsf{P}}$ as follows, for each $\tau \in [k]$:

$\tilde{\mathsf{P}}_\tau(c_1, \ldots, c_{\tau-1}; \mathfrak{a})$:

1. Repeat the following at most $N = \lceil 2ke/\delta \rceil$ times:
   (a) Sample $c^\tau_\tau, \ldots, c^\tau_{k-1}$ uniformly at random.
   (b) Obtain transcript $\mathbf{T} := (\mathsf{ck}, (\mathsf{com}_i, m_{\text{RAND},i})_{i \in [k]}, (r_i)_{i \in [k]}) \leftarrow \tilde{\mathsf{P}}'(c_1, \ldots, c_{\tau-1}, c^\tau_\tau, \ldots, c^\tau_{k-1})$.
   (c) If $\mathsf{V}'(\mathbf{T}, (c_1, \ldots, c_{\tau-1}, c^\tau_\tau, \ldots, c^\tau_{k-1}))$ accepts, output $(m_{\text{RAND},\tau}, m_{\text{DET},\tau})$ where $(m_{\text{DET},i})_{i \in [k]} := \mathcal{S}_{\text{DET}}(x_\mathfrak{a}, (c_1, \ldots, c_{\tau-1}, c^\tau_\tau, \ldots, c^\tau_{k-1}), (m_{\text{RAND},i})_{i \in [k]})$.
2. Otherwise, abort.

We take $\mathfrak{a}$ above to be chosen uniformly at random.

Suppose first that $\tilde{\mathsf{P}}$ does not abort, and let

$$\mathbf{T}_\tau = (\mathsf{ck}^\tau, (\mathsf{com}^\tau_i, m^\tau_{\text{RAND},i})_{i \in [k]}, (r^\tau_i)_{i \in [k]})$$

be the accepting transcript obtained from $\tilde{\mathsf{P}}'$ by $\tilde{\mathsf{P}}_\tau$. Let $S := \{((m^i_{\text{DET},i,j})_{j \in [\ell]}, r^i_i), ((m^k_{\text{DET},i,j})_{j \in [\ell]}, r^k_i) : i \in [k]\}$, where $(m^\tau_{\text{DET},i,j})_{i \in [k]} := \mathcal{S}_{\text{DET}}(x_j, (c_1, \ldots, c_{\tau-1}, c^\tau_\tau, \ldots, c^\tau_{k-1}), (m_{\text{RAND},i})_{i \in [k]})$. Since for all i, $\mathsf{com}^k_i = \mathsf{com}^i_i$ by construction, each pair in S maps to the same commitment under ck. Hence by partial binding it holds (with overwhelming probability) that there exists $j \in [\ell]$ such that for all $i \in [k]$, $m^k_{\text{DET},i,j} = m^i_{\text{DET},i,j}$.

Recall that $(m_{\text{RAND},i}, m^i_{\text{DET},i,\mathfrak{a}})$ is the message sent by $\tilde{\mathsf{P}}_i$ to V in round i. Moreover it holds that V' accepts $(\mathbf{T}_k, \mathbf{c})$ where $\mathbf{c}$ are the verifier's challenges in the real interaction. Hence

$$1 = \phi(\mathbb{x}_j, \{(m_{\text{RAND},i}, m^k_{\text{DET},i,j})\}_{i \in [k]}, \mathbf{c}) = \phi(\mathbb{x}_j, \{(m_{\text{RAND},i}, m^i_{\text{DET},i,j})\}_{i \in [k]}, \mathbf{c}) \ .$$

It follows that V accepts in the real interaction when $\mathfrak{a} = j$, which occurs with probability $1/\ell$.

We conclude by bounding the probability that $\tilde{\mathsf{P}}$ aborts. To do so, we bound the following probability:

$$\gamma \coloneqq \Pr_{\mathbf{c}}[\exists \tau \in [k],\ \tilde{\mathsf{P}}_\tau(c_1,\ldots,c_{\tau-1}) \text{ aborts} \wedge \tilde{\mathsf{P}}'(\mathbf{c}) \text{ convinces } \mathsf{V}'] \ .$$

Then by a union bound, $\Pr[\tilde{\mathsf{P}} \text{ aborts}] \leq 1-\delta+\gamma$. For a vector of challenges $(c_1,\ldots,c_{\tau-1})$, define

$$\delta(c_1,\ldots,c_{\tau-1}) \coloneqq \Pr_{c_\tau,\ldots,c_k}[\tilde{\mathsf{P}}'(\mathbf{c}) \text{ convinces } \mathsf{V}'] \ .$$

We can then bound γ as follows:

$$\begin{aligned}\gamma &\leq \sum_{\tau=1}^{k} \Pr_{\mathbf{c}}[\tilde{\mathsf{P}}_\tau(c_1,\ldots,c_{\tau-1}) \text{ aborts} \wedge \tilde{\mathsf{P}}'(\mathbf{c}) \text{ convinces } \mathsf{V}'] \\ &= \sum_{\tau=1}^{k} \mathbb{E}_{c_1,\ldots,c_{\tau-1}}[(1-\delta(c_1,\ldots,c_{\tau-1}))^N \delta(c_1,\ldots,c_{\tau-1})] \leq \frac{ke}{N+1} \ ,\end{aligned}$$

where e is Euler's number. The final inequality follows because $x(1-x)^N \leq e/(N+1)$ for all $x \in [0,1]$. Hence the probability that $\tilde{\mathsf{P}}$ aborts is at most $1-\delta+\frac{ke}{N+1} \leq 1-\delta/2$.

Stackability. We construct the PPT simulator $\mathcal{S}'_{\text{RAND}}$ and the deterministic simulator $\mathcal{S}'_{\text{DET}}$ as follows, letting $\mathfrak{a}=1$:

```
$\mathsf{ck}, \{r_i, m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]} \leftarrow \mathcal{S}'_{\text{RAND}}(1^\lambda, \{c_i\}_{i\in[k]})$
Compute $(\mathsf{ck},\mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B=\{1\})$.
for $i \in [k]$, Sample $r_i \xleftarrow{\$} \{0,1\}^\lambda$
$\{m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]} \leftarrow \mathcal{S}_{\text{RAND}}(1^\lambda, \{c_i\}_{i\in[k]})$
```

```
$\{\mathsf{com}_i\}_{i\in[k]} \coloneqq \mathcal{S}'_{\text{DET}}((\mathbb{x}_1,\ldots,\mathbb{x}_\ell), \mathsf{ck}, \{r_i, m_{\text{RAND},i,\mathfrak{a}}, c_i\}_{i\in[k]})$ :
for $j \in [\ell] \setminus \mathfrak{a}$ : Compute $\{m_{\text{DET},i,j}\}_{i\in[k]} \coloneqq \mathcal{S}_{\text{DET}}(\mathbb{x}_j, \{c_i, m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]})$
for $i \in [k]$ : Compute $\mathsf{com}_i \leftarrow \mathsf{BindCom}(\mathsf{pp}, \mathsf{ck}, \mathbf{v} = (m_{\text{DET},i,1},\ldots,m_{\text{DET},i,\ell}); r_i)$
return $\{\mathsf{com}_i\}_{i\in[k]}$
```

We now proceed to show using a hybrid argument that an honest transcript resulting from an honest prover possessing witness $(\mathfrak{a}, \mathbb{w})$ running Π' with an honest verifier on the statement $(\mathbb{x}_1,\ldots,\mathbb{x}_\ell)$ is indistinguishable from the transcript simulated by $\mathcal{S}'_{\text{RAND}}$ and $\mathcal{S}'_{\text{DET}}$.

- $\mathcal{H}_0$: This is identical to the honest exceution.
- $\mathcal{H}_1$: Let this be the same as $\mathcal{H}_0$, except let the non-recyclable messages of clause $\mathfrak{a}$ be generated by simulation, *i.e.* $\{m_{\text{DET},i,\mathfrak{a}}\}_{i\in[k]} \coloneqq \mathcal{S}_{\text{DET}}(\mathbb{x}_\mathfrak{a}, \{c_i, m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]})$. Since $\mathcal{S}_{\text{DET}}$ is a deterministic simulator for the underlying protocol Π, indistinguishability between $\mathcal{H}_0$ and $\mathcal{H}_1$ follows from stackability of Π.
- $\mathcal{H}_2$: Let this be the same as $\mathcal{H}_1$, except let the commitment key ck be generated with the binding position as $B=\{1\}$, *i.e.* $(\mathsf{ck},\mathsf{ek}) \leftarrow \mathsf{Gen}(\mathsf{pp}, B=\{1\})$. Observe that $\mathcal{H}_1 \overset{p}{\equiv} \mathcal{H}_2$ by the (perfect) hiding of the partially-binding commitment scheme. Lastly, note that $\mathcal{H}_2$ matches the output distribution of $\mathcal{S}'_{\text{RAND}}$ and $\mathcal{S}'_{\text{DET}}$. □

Therefore Π' is a stackable Σ-protocol.

Complexity Discussion. Let $\mathtt{CC}(\Pi)$ be the communication complexity of Π. Then, the communication complexity of the Π' obtained from Theorem 3.5 is $(\mathtt{CC}(\Pi)+|\mathsf{ck}|+|\mathsf{com}|+|r'|)$, where the sizes of ck, com and r' depend on the choice of partially-binding vector commitment scheme and are independent of $\mathtt{CC}(\Pi)$. In the construction of partially-binding vector commitments from DLOG due to Goel et al. [GGHAK22, Corollary 1], $|\mathsf{ck}|, |r'| = O_\lambda(\log \ell)$, and $|\mathsf{com}| = O_\lambda(1)$. Hence the communication cost of proving a disjunction of ℓ clauses is $O(\log \ell)$.

3.5 Finding Recyclable Messages

It is natural to wonder, in light of our definition of stackability, how to decide which message parts are recyclable and which are not. In the case of a Σ-protocol, answering such a question was straight forward: the largest and most complex messages are usually in the third round, and so its desirable to make the entire third round message recyclable. In multi-round protocols, however, it may be more difficult to determine which message parts in each round are recyclable. Indeed, it may be possible to define many divisions of the simulator demonstrating that the protocol is stackable, but finding the "optimal" simulator and division of messages can be quite challenging.

In this subsection, we provide intuition which can be used to find *a* set of recyclable messages. We emphasize that this process is not a formal one and is not guaranteed to output the optimal set of messages (by any metric). Instead, we offer this intuition as a tool that can assist the reader in applying our techniques to other protocols. To that end, we offer an informal procedure and an insight that we found to be successful:

Start with the Last Round. We start with an observation about ZK-IPs that are designed to be minimal, *ie.* not contain unnecessary communication:

Lemma 3.6. *Let Π be a ZK-IP consisting of k prover messages and $k-1$ verifier messages. If Π is stackable, then either*

(1) there exists a simulator $\mathcal{S}_{\text{RAND}}$ which produces the entirety of the prover's k^{th} message, or

(2) there exists a ZK-IP Π' for the same language which is the same as Π, except it has a shorter k^{th} round prover message.

Proof. Let us assume for the sake of contradiction that no such $\mathcal{S}_{\text{RAND}}$ exists. From the definition of stackable ZK-IP, it follows that some part of the k^{th} prover message can be deterministically given the messages in preceding rounds without the knowledge of the witness. In that case, the verifier could simply compute this part of the message on their own during verification, given access to the previous messages in the transcript. As such, the messages added nothing to the verifier's view and could simply be omitted from the protocol without changing its properties. If this is not true, we get a contradiction and it follows that such a $\mathcal{S}_{\text{RAND}}$ exists. □

The ramifications of this observation is that if the protocol designers minimized the communication of the ZK-IP, then the entire last round message is recyclable. Note that not all ZK-IPs are designed to contain no "unnecessary" communication, as this communication can be used to improve the verifier's runtime. Indeed, this is the case in some IOPs. However, when such non-recyclable messages are included in a protocol, it will usually be quite obvious. As such, identifying the recyclable portion of the final message is usually a straight-forward task.

Verifier Checks as Constraints. It is helpful to use the *verifier equations* as a methodology for determining if a set of messages is recyclable, as the natural way to check is the joint distribution of messages is "correct" with respect to a statement is to simply check if those messages are part of an accepting transcript. These verifier equations create of a set of constraints over the messages of a protocol. When a set of messages is *over-constrained* with respect to the verifier equations, it is likely the set of messages is not recyclable. On the other hand, if the set of messages is under-constrained (*ie. retains additional degrees of freedom*) with respect to the verifier equations, the joint distribution of those messages is likely independent of the statement—making them recyclable. The "sweet spot" is when these constraints are perfectly, uniquely, satisfied.

For a simple example, consider the transcript of the Schnorr [Sch90] identification protocol for the statement $A = g^x$ with a three round transcript (T, c, z). To verify this transcript, a verifier will see if $A^c T = g^z$. Note that this means that the relationship between the two variables T and z is determined by the challenge c and the statement A. However, the distribution of each message T and z *alone* is not fixed, and for any choice of z, an appropriate T can be found. Thus, if z is a candidate recyclable message, then T cannot be recyclable, as it is uniquely defined by the existing recyclable messages and the statement A.

To make this more concrete, let us assume that the verifier equations to define e equations over v variables (*ie.* message parts) and f fixed values (*eg.* the statement). If assigning arbitrary values to each messages in a candidate set of recyclable messages makes there be no solution to the e equations, then the set is unlikely to be recyclable. On the other had, is an assignment leave multiple degrees of freedom for some of the e equations, the set of recyclable messages can likely grow. Finally, if there is a unique solution to all e equations, the messages are likely recyclable, but the set cannot grow (without first removing some candidate messages).

An Informal Procedure. We now present a simple, informal procedure for finding recyclable messages.

- Initialize the set of recyclable messages M_{RAND} to contain the recyclable part of the k^{th} prover message.
- For each $i \in [(k-1), \ldots, 1]$, do the following:
  - Divide m_i into its "natural" sub-components $(\mathsf{comp}_1, \mathsf{comp}_2, \ldots)$. (*e.g.*, divide so that each sub-component contains a single group element or field element).
  - For each component comp_j:
    - Assign uniform random values to each element in M_{RAND} and comp_j
    - If there exist any verifier equations that are unsolvable, discard comp_j and continue
    - If there exist unique values of all unassigned variables in all verifier equations that satify these equations, add comp_j to M_{RAND}.
    - If there any verifier equations have remaining degrees of freedom, add comp_j to M_{RAND} and continue
- Return M_{RAND} as the recyclable messages.

We note that this process is inherently informal. There may be statement-dependent relationships between messages that are not governed by the verifier equations. Moreover, the notion of a sub-component and a verifier equation are not formally defined. As such, this procedure should be understood as an intuition building exercise for future work and not a formal result. Finally, while we found success using this procedure, it is not guaranteed to find an optimal division of messages.

4 Speed-Stacking Interactive Oracle Proofs

Interactive oracle proofs, originally proposed by [BCS16, RRR16], form the basis of a widely-used framework for building succinct arguments. In this section we describe how to adapt this framework to build *stackable* succinct arguments.

We begin this section by recalling the preliminary definition of holographic IOPs (hIOPs), a generalization of IOPs introduced by [COS20] that allows for part of the input to be preprocessed, in Section 4.1. We then proceed to outline the technical machinery necessary to speed-stack two IOPs, Aurora IOP [BCR+19] Fractal hIOP [COS20]. Specifically, we use a series of compilers that speed-stacks these IOPs via several intermediary definitions. First, we define the notion of a *stackable* (holographic) IOP in Section 4.2. Next, we describe how to transform a stackable IOP into a stackable (succinct) interactive argument, which can in-turn be speed-stacked using the compiler in Section 3. Finally, in Section 4.4, we describe our two constructions of stackable hIOPs, based on the Aurora IOP [BCR+19] and Fractal hIOP [COS20] constructions respectively.

4.1 Holographic IOPs

In this section, we start by defining Holographic IOPs (hIOP) and then proceed to define a sub-class of hIOPs called Reed-Solomon encedoded hIOP (RS-hIOP).

Definition 4.1 (Holographic IOP)**.** *A holographic IOP [COS20] for an indexed relation $\mathcal{R}$ is specified by a tuple* $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$*, where* $\mathbf{I}$ *is the* indexer*,* $\mathbf{P}$ *the* prover*, and* $\mathbf{V}$ *the* verifier*. The indexer is a deterministic polynomial-time algorithm, while the prover and verifier are probabilistic polynomial-time interactive algorithms. In an offline phase, given an index $\mathbb{i}$, the indexer $\mathbf{I}$ computes an encoding of $\mathbb{i}$, denoted $\mathbf{I}(\mathbb{i})$. Subsequently, in an online phase, the prover $\mathbf{P}$ receives as input a triple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$, while the verifier $\mathbf{V}$ receives as input $\mathbb{x}$ and is granted oracle access to the encoded index $\mathbf{I}(\mathbb{i})$. The online phase consists of multiple rounds, and in each round the verifier $\mathbf{V}$ sends a message m_i and the prover $\mathbf{P}$ replies with a proof string $\Pi_i \colon L_i \to \Sigma$, which the verifier can query at any set of locations. At the end of the interaction, the verifier $\mathbf{V}$ accepts or rejects.*

We say that HOL *has perfect completeness and soundness error ϵ if the following holds.*

- ***Completeness.*** *For every index-instance-witness triple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the probability that $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept in the interactive oracle protocol is 1.*
- ***Soundness.*** *For every index-instance pair $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ and prover $\tilde{\mathbf{P}}$, the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept in the interactive oracle protocol is at most ϵ.*

The round complexity *k is the number of back-and-forth message exchanges between the verifier and the prover. The* proof length L *is the sum of the length of the encoded index plus the lengths $\mathsf{L}_i = |L_i|$ of all oracles sent by the prover. The* query complexity q *is the total number of queries made by the verifier; this includes queries to the encoded index and to the oracles sent by the prover.*

Public coins and oblivious queries. *In this work we will consider a certain subclass of IOPs:* public-coin *IOPs with* oblivious *queries. An IOP is public coin if each verifier message to the prover is a random string. This means that the verifier's randomness C consists of its messages $c_1, \ldots, c_{k-1} \in \{0,1\}^*$ and possibly additional randomness $c_k \in \{0,1\}^*$ used after the interaction (in particular, for choosing the query set). An IOP has* oblivious queries *if the verifier can be partitioned into a query algorithm V_Q and a decision algorithm V_D as follows. V_Q takes as input C (and nothing else) and outputs query sets $(Q_1, \ldots, Q_k)$. V_D takes as input $(\mathbb{x}, C, \Pi_1|_{Q_1}, \ldots, \Pi_k|_{Q_k})$ and outputs a bit b.*

Zero knowledge. *A public-coin holographic IOP* HOL *has (perfect) special honest verifier zero knowledge if there exists a probabilistic polynomial-time simulator $\mathbf{S}$ such that for every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ the random variables* $\mathrm{View}(\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; C))$ *and $(C, \mathbf{S}(\mathbb{i}, \mathbb{x}, C, V_Q(C)))$ are identical, where:*

- *$C = (c_1, \ldots, c_{k-1}, c_k)$ is the verifier's (public) randomness, chosen uniformly at random, and*
- $\mathrm{View}(\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; C))$ *is the* view *of* $\mathbf{V}$ *when interacting with* $\mathbf{P}$*, i.e., it is the random variable $(C, \Pi_1|_{Q_1}, \ldots, \Pi_k|_{Q_k})$.*

Reed–Solomon encoded holographic IOPs (RS-hIOPs). An RS-hIOP is a variant of hIOP where the prover's messages in both the honest and malicious case are required to be Reed–Solomon codewords of a specified rate. Before defining RS-hIOP, we define the notion of a *rational constraint*.

Definition 4.2 (Rational Constraint)**.** *A* rational constraint *is a tuple $\mathbf{c} = (p, q, d)$ where $p \colon \mathbb{F}^{1+\ell} \to \mathbb{F}$ and $q \colon \mathbb{F} \to \mathbb{F}$ are arithmetic circuits, and $d \in \mathbb{N}$ is a degree bound. The arithmetic circuits (p, q) and a list of words $f_1, \ldots, f_\ell \colon L \to \mathbb{F}$ jointly define the word $(p, q)[f_1, \ldots, f_\ell] \colon L \to \mathbb{F}$ given by*

$$\forall\, a \in L\,,\ (p, q)[f_1, \ldots, f_\ell](a) := \frac{p(a, f_1(a), \ldots, f_\ell(a))}{q(a)}\ .$$

A rational constraint $\mathbf{c} = (p, q, d)$ is satisfied with respect to $(f_1, \ldots, f_\ell)$ *if*

$$(p, q)[f_1, \ldots, f_\ell] \in \mathrm{RS}[L, d]\ .$$[7]

When describing rational constraints, we will often use the shorthand notation "$\deg(\hat{f}) \leq d$", where $f \colon L \to \mathbb{F}$ is defined as a rational equation over some oracles. This should be taken to mean the rational constraint $\mathbf{c} = (p, q, d)$ that is naturally induced by the expression that defines f.

A special type of rational constraint is a *boundary constraint*, defined next.

[7] For $a \in L$, if $q(a) = 0$ then we define $(p, q)[f_1, \ldots, f_\ell](a) := \perp$. Note that if this holds for some $a \in L$ then, for *any* words $f_1, \ldots, f_\ell$ and degree bound d, the rational constraint (p, q, d) is not satisfied by $f_1, \ldots, f_\ell$.

Definition 4.3 (Boundary Constraint)**.** *A* boundary constraint *is a rational constraint that expresses a condition such as "$\hat{f}(\alpha) = \beta$" for some word $f\colon L \to \mathbb{F}$ and elements $\alpha, \beta \in \mathbb{F}$. Such a condition is represented via the rational constraint $\mathbf{c} = (p, q, \deg(\hat{f}) - 1)$ where $p(X,Y) \overset{\text{def}}{=} Y - \beta$ and $q(X) \overset{\text{def}}{=} X - \alpha$, which can be summarized as "$\deg(\hat{g}) \leq \deg(\hat{f}) - 1$" where $g(a) \overset{\text{def}}{=} (f(a) - \beta)/(a - \alpha)$. We denote this constraint simply by "$\hat{f}(\alpha) = \beta$".*

In the following we use $\mathrm{RS}[L, (d_1, \ldots, d_k)] \subseteq (\mathbb{F}^k)^L$ to denote the interleaved Reed–Solomon code over L with degree bounds $(d_1, \ldots, d_k)$, i.e., the set of $k \times |L|$ matrices where the i-th row is a codeword of $\mathrm{RS}[L, d_i]$ (which itself is all evaluations over L of univariate polynomials of degree at most d_i).

Definition 4.4 (Reed–Solomon encoded hIOP)**.** *A* Reed–Solomon encoded holographic IOP *(RS-hIOP) for an indexed relation $\mathcal{R}$ is a tuple*

$$(\mathbf{I}, \mathbf{P}, \mathbf{V}, \{\mathbf{d_I}, \mathbf{d}_{\mathbf{P},1}, \ldots, \mathbf{d}_{\mathbf{P},k}\})$$

where $\mathbf{I}$ is a deterministic algorithm, $\mathbf{P}$ and $\mathbf{V}$ are probabilistic interactive algorithms, and $\mathbf{d_I} \in \mathbb{N}^{\ell_0}, \mathbf{d}_{\mathbf{P},i} \in \mathbb{N}^{\ell_i}$ are vectors of degree bounds, that satisfies the following properties.

- ***Degree bounds.*** *On input any $\mathbb{i}$, the indexer $\mathbf{I}$ outputs a codeword of $\mathrm{RS}[L, \mathbf{d_I}]$. Moreover, on input any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and for every round i, the i-th message of $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ is a codeword of $\mathrm{RS}[L, \mathbf{d}_{\mathbf{P},i}]$.*
- ***Completeness.*** *For every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, all rational constraints output by $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ after interacting with $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ are satisfied with respect to $\mathbf{I}(\mathbb{i})$ and $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$'s messages with probability 1.*
- ***Soundness.*** *For every $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ and unbounded malicious prover $\tilde{\mathbf{P}}$ whose i-th message is a codeword of $\mathrm{RS}[L, \mathbf{d}_{\mathbf{P},i}]$, all rational constraints output by $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ after interacting with $\tilde{\mathbf{P}}$ are satisfied with respect to $\mathbf{I}(\mathbb{i})$ and the prover's messages with probability at most ϵ.*

Often we will write that $\mathbf{V}$ "accepts", which means that all of the rational constraints it outputs are satisfied, or that it "rejects", which means that at least one rational constraint is not satisfied.

Zero knowledge. *Honest-verifier zero knowledge for RS-IOPs is trivial, since the honest RS-IOP verifier makes no queries, and so learns nothing from the interaction. Instead, we introduce a notion of* special semi-honest verifier *zero knowledge (SSHVZK), which guarantees zero knowledge against verifiers that behave honestly during the interaction, and then make a bounded number b of arbitrary queries. Formally, an RS-IOP is SSHVZK with query bound b if there exists a PPT simulator $\mathbf{S}$ such that for every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, every large enough $\ell \in \mathbb{N}$ and every function $Q\colon \{0,1\}^\ell \to \binom{L}{\mathsf{b}}$, the random variables $\mathrm{View}_{Q(C)}(\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}))$ and $(C, \mathbf{S}(\mathbb{i}, \mathbb{x}, C, Q(C)))$ are identical, where*

- *$C = (c_1, \ldots, c_{k-1}, c^*)$, chosen uniformly at random, is the verifier's (public) randomness, (possibly) augmented to ℓ bits with additional randomness c^*, and*
- *$\mathrm{View}_{Q(C)}(\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})) = (C, \Pi_1|_{Q(C)}, \ldots, \Pi_k|_{Q(C)})$ is the view of the verifier in the protocol (which consists only of its own messages), augmented with the restriction of each prover message to the set $Q(C) \subseteq L$.*

Note that we count by b the number of distinct query locations across all *oracles; this only makes the class of adversaries larger as an adversary making b queries can query at most b distinct locations.*

4.2 Defining a Stackable IOP and Stackable RS-IOP

In this section we give definitions for a *stackable RS-IOP* and a *stackable IOP*, before showing how to compile from the former to the later in Section 4.3. Looking ahead, we will give modifications of Aurora IOP [BCR+19] and Fractal hIOP [COS20] that are stackable RS-IOPs. The two definitions are defined in largely the same way; the differences are analogous to the differences between an RS-IOP (Definition 4.4) and IOP (Definition 4.1). As such, we only explicitly give the definition of a stackable IOP, as the generalization is trivial.

Recall that the simulator for a ZKIOP is required to sample answers for exactly the points that the honest verifier queries in each round; these points are provided to the simulator as a vector $\mathbf{Q} = (Q_1, \ldots, Q_k)$, where Q_i is the set of points that the verifier queries in round i. Hence we can write the simulator's output as a sequence of functions $\Pi_i^*\colon Q_i \to \Sigma$, where Σ is the alphabet of the IOP. Given this template, stackability for IOPs is defined similarly to stackability for IPs (Definition 3.4), as follows.

Definition 4.5 (Stackable hIOP)**.** *We say that an k-round holographic IOP* $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ *is* stackable *if there exists a subset of "recyclable" rounds $R_{\mathsf{rec}} \subseteq [k]$ and a pair of algorithms $(\mathcal{S}_{\mathrm{RAND}}, \mathcal{S}_{\mathrm{DET}})$ where $\mathcal{S}_{\mathrm{DET}}$ is deterministic, such that for all $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the following algorithm is a special honest-verifier zero-knowledge simulator for* HOL*:*

$\mathbf{S}(\mathbb{i}, \mathbb{x}, C, \mathbf{Q})$*:*

1. *sample* $(\Pi^*_i : Q_i \to \Sigma)_{i \in R_{\mathsf{rec}}} \xleftarrow{\$} \mathcal{S}_{\mathrm{RAND}}(C, \mathbf{Q})$*;*
2. *compute* $(\Pi^*_i : Q_i \to \Sigma)_{i \in [k] \setminus R_{\mathsf{rec}}} := \mathcal{S}^{\mathbf{I}(\mathbb{i})}_{\mathrm{DET}}(\mathbb{x}, (\Pi^*_i)_{i \in R_{\mathsf{rec}}}, C, \mathbf{Q})$*;*
3. *output* $(\Pi^*_i)_{i \in [k]}$*;*

and for all $\lambda \in \mathbb{N}$ and $(\mathbb{i}', \mathbb{x}', \mathbb{w}')$ (whether in $\mathcal{R}$ or not), $\mathbf{S}(1^\lambda, \mathbb{i}', \mathbb{x}')$ outputs an accepting view with certainty.

The definition extends in the natural way to Reed–Solomon encoded IOPs (RS-IOPs), except that we require that **S** be an SSHVZK simulator (see Definition 4.4).

4.3 Compiling RS-IOP to Stackable IP via Stackable IOP

In this section we show how to "compile" a stackable RS-IOP into a stackable IOP, and a stackable IOP into a stackable IP using a key-value commitment schemes. In Definition 4.6 we give a formal definition for the key-value communitment schemes that we require. In this section we provide both compilers (in Lemma 4.7 and Theorem 4.9 respectively).

Hiding Key-Value Commitments. Key-value commitments, described by Boneh, Bünz and Fisch [BBF19] and Agrawal and Raghuraman [AR20] primarily in blockchain-related applications are a generalization of vector commitments: allowing the committer to efficiently commit to a (potentially) exponentially large but sparse vector in time that is polyomial in the security parameter and the number of entries in the sparse vector. Unlike the primary motivation for these works, we are not concerned with updateability of the map; however, we additionally require the commitments to hide the unopened entries. We formalize this notation below:

Definition 4.6 ((Insert-Only) Key-Value Commitments)**.** *Let $\mathsf{Map}(\mathcal{K}, \Sigma) \subseteq \mathbb{P}(\mathcal{K} \times \Sigma)$ be the set of dictionaries with keys in $\mathcal{K}$ and values in Σ, i.e. $\mathcal{M} \in \mathsf{Map}(\mathcal{K}, \Sigma) \iff \forall (\mathbb{k}_1, \mathbb{v}_1), (\mathbb{k}_2, \mathbb{v}_2) \in \mathcal{M} : \mathbb{k}_1 = \mathbb{k}_2 \implies \mathbb{v}_1 = \mathbb{v}_2$. A (insert-only) key-value commitment scheme consists of four polynomial time algorithms. For notational convince, we omit the explicit random tape as a parameter:*

$\mathcal{P} \leftarrow \mathsf{KV.Setup}(1^\lambda)$ *Takes a unary representation of the security parameter, a random tape and returns public parameters $\mathcal{P}$.*

$(\mathsf{C}, \mathsf{o}) \leftarrow \mathsf{KV.Com}(\mathcal{P}, \mathcal{M})$ *Takes public parameters $\mathsf{pp} \in \mathcal{P}$, a key-value map $\mathcal{M} = \{(\mathbb{k}_i, \mathbb{v}_i)\}_i$ and a random tape. Produces a commitment $\mathsf{C} \in \mathcal{C}$ and opening information $\mathsf{o} \in \mathcal{O}$.*

$\mathbb{o} \leftarrow \mathsf{KV.Open}(\mathcal{P}, \mathsf{o}, \mathcal{M}')$ *Takes public parameters $\mathsf{pp} \in \mathcal{P}$, opening information $\mathsf{o} \in \mathcal{O}$ and a subset $\mathcal{M}' \subseteq \mathcal{M}$ of the committed dictionary. Returns an inclusion proof $\mathbb{o} \in \mathcal{I}$.*

$\mathsf{C} \leftarrow \mathsf{KV.Verify}(\mathcal{P}, \mathcal{M}, \mathbb{o})$ *Takes public parameters $\mathsf{pp} \in \mathcal{P}$, a dictionary $\mathcal{M} \in \mathsf{Map}(\mathcal{K}, \Sigma)$ and an opening randomness $\mathbb{o}$. Returns the (recomputed) valid commitment C.*

For completeness we require that correctly computed inclusion proofs for subsets of the map successfully verify, i.e. $\forall \lambda : \forall \mathcal{M} \in \mathsf{Map}(\mathcal{K}, \Sigma) : \forall \mathcal{M}' \subseteq \mathcal{M} :$

$$1 = \Pr\left[\mathsf{KV.Verify}(\mathsf{pp}, \mathcal{M}', \mathbb{o}) = \mathsf{C} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{KV.Setup}(1^\lambda); \\ (\mathsf{C}, \mathsf{o}) \leftarrow \mathsf{KV.Com}(\mathsf{pp}, \mathcal{M}); \mathbb{o} \leftarrow \mathsf{KV.Open}(\mathsf{pp}, \mathsf{o}, \{\mathbb{k}\}_{(\mathbb{k},\mathbb{v}) \in \mathcal{M}'}) \end{array}\right]$$

Informally, for security we require that (1) every position $\mathbb{k}$ can be mapped to at-most one value $\mathbb{v}$ (2) unopened positions remain hidden: openings of any two maps that agree on the opened indices are indistinguishable.. Formally security is defined by the games in Figure 3 and the following:

```
$\mathsf{Game}_{\mathsf{Binding}}(\lambda, \mathcal{A})$
$\mathsf{pp} \leftarrow \mathsf{KV.Setup}(1^\lambda)$
$(\mathsf{C}, \mathbb{o}_1, \mathcal{M}_1, \mathbb{o}_2, \mathcal{M}_2) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp})$
if $\mathsf{KV.Verify}(\mathsf{pp}, \mathbb{o}_1, \mathcal{M}_1) \neq \mathsf{C}$ : return $0$
if $\mathsf{KV.Verify}(\mathsf{pp}, \mathbb{o}_2, \mathcal{M}_2) \neq \mathsf{C}$ : return $0$
// $\mathcal{A}$ wins if there exists inconsistent assignment of a key
if $\forall (\mathbb{k}, \mathbb{v}_1) \in \mathcal{M}_1, (\mathbb{k}, \mathbb{v}_2) \in \mathcal{M}_2 : \mathbb{v}_1 = \mathbb{v}_2$ : return $0$
return $1$
```

```
$\mathsf{Game}_{\mathsf{Hiding}}(\lambda, \mathcal{A})$
$\mathsf{pp} \leftarrow \mathsf{KV.Setup}(1^\lambda)$
// Commit to one of two maps chosen by $\mathcal{A}$.
$(\mathcal{M}_0, \mathcal{M}_1, K, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{find}, 1^\lambda, \mathsf{pp})$
$b \xleftarrow{\$} \{0,1\}; (\mathsf{C}, \mathsf{o}) \leftarrow \mathsf{KV.Com}(\mathsf{pp}, \mathcal{M}_b)$
// Open keys present in both maps.
if $K \not\subseteq \{\mathbb{k}\}_{(\mathbb{k},\mathbb{v}) \in \mathcal{M}_0} \cap \{\mathbb{k}\}_{(\mathbb{k},\mathbb{v}) \in \mathcal{M}_1}$ :
    return $0$
$\mathbb{o} \leftarrow \mathsf{KV.Open}(\mathsf{pp}, \mathsf{o}, K)$
// Guess which map the opening belongs to.
$b' \leftarrow \mathcal{A}(\mathsf{guess}, \mathbb{o}, \mathsf{st}, 1^\lambda, \mathsf{pp})$
return $b \overset{?}{=} b'$
```

Figure 3: Security games for key-value commitments

Position Binding. *For every PPT algorithm $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for any sufficiently large λ:* $\Pr\left[\mathsf{Game}_{\mathit{Binding}}(\lambda, \mathcal{A}) = 1\right] \leq \mathsf{negl}(\lambda)$

Value Hiding. *For every PPT algorithm $\mathcal{A}$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for sufficiently large λ:* $\Pr\left[\mathsf{Game}_{\mathit{Hiding}}(\lambda, \mathcal{A}) = 1\right] - 1/2 \leq \mathsf{negl}(\lambda)$

For our applications (compiling stackable interactive oracle proofs to stackable interactive arguments) a key-value commitment with polynomial size $\mathcal{K}$ (in the security parameter) suffices, i.e. $|\mathcal{K}| = O(\mathsf{poly}(\lambda))$. In Appendix B we give two simple instantiations of the above primitive based on Merkle trees: one in the random oracle model, and one in the standard model based on compressing commitments.

Compiling RS-IOP to Stackable IOP. We now show that, by slightly tweaking the RS-IOP to IOP transformation presented in [BCR+19, Section 8.1], we can preserve stackability. The compiler of [BCR+19, Section 8.1] converts an RS-IOP into an IOP using a (IOP) proximity test for Reed-Solomon codes [BBHR18] [BGKS20] (also called a Low-Degree Test (LDT)). Since the concrete cost of the proximity test is large, by exploiting the linearity of the code, all the oracles are combined using a random linear combination into a *single* claimed codeword; rather than repeating the proximity for every individual oracle. This incurs a soundness-error of $1/|\mathbb{F}|$[8]. This works for codewords in the *same code*, to account for multiple RS codes of different rate note that component-wise products of Reed-Solomon codes is a Reed-Solomon code, i.e. for a fixed $C_1 \in \mathrm{RS}[L, d_1]$: $C_1 \circ C_2 \in \mathrm{RS}[L, d_1 + d_2] \iff C_2 \in \mathrm{RS}[L, d_2]$. This allows homogenizing all the rates: for the verifier to query $(C_1 \circ C_2)(i)$ simply query $C_2(i)$ and compute $C_1(i) \cdot C_2(i)$, hence we can assume that the rate of all codewords is the same. Note that C_1 can be an arbitrary codeword, in particular it can be chosen such that computing $C_1(i)$ is very efficient. Lastly, since the proximity test is not zero-knowledge the prover samples a random codeword which is added to the linear combination: such that the distibution of the codeword on which the proximity test is run is uniform. In summary, the verifier samples $\mathbf{z} \in \mathbb{F}^k$ and the proximity test is run on the oracle:

$$q = \mathbf{z}^T \Pi + r$$

for codewords $\Pi \in \mathrm{RS}[L, d]^k$ and $r \in \mathrm{RS}[L, d]$. Note that $q(i)$ can be accessed by simply querying Π and r at i, hence in [BCR+19, Protocol 8.6] there is no need for the prover to send the oracle q explicitly[9], however we need this to efficiently stack.

[8] For fields where $1/|\mathbb{F}|$ is not negligible, parallel repetition is used: requiring repetitions of the proximity test as well.

[9] Which would also require an additional proximity test between q and $\mathbf{z}^T \Pi + r$

Lemma 4.7 (From Stackable RS-IOP to Stackable IOP)**.** *There is a transformation (an adaptation of [BCR+19, Protocol 8.6]) which composes a stackable RS-IOP and any IOPP for the Reed–Solomon code (i.e., a low-degree test) to produce a stackable IOP for the same relation. Moreover, the cost of $\mathcal{S}_{\mathrm{DET}}$ for the resulting IOP is the same as the cost of $\mathcal{S}_{\mathrm{DET}}$ for the RS-IOP. The construction follows easily from the discussion above, see* **??** *for details.*

Compiling Stackable IOP to Stackable IP. Next we show how to compile stackable IOPs into stackable interactive arguments using hiding key-value commitments (see Definition 4.6). The construction is an adaptation of the natural construction of a succinct argument from an IOP using vector commitments; the security and efficiency guarantees of hiding key-value commitments are necessary to preserve stackability and stacking efficiency of the underlying IOP.

Construction 4.8 (Stackable IOP to Stackable IP Compiler)**.** *Assume wlog. that the (public coin) $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ only makes queries to the oracles after the k'th round and transform an k-round holographic IOP into a $k+1$ stackable IP follows: In round i, when $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ outputs Π_i, compute the commitment to the oracle:*

$$(\mathsf{C}_i, \mathsf{o}_i) \leftarrow \mathsf{KV.Com}(\mathsf{pp}, \{(j, \Pi_i(j))\}_{j \in [|\mathsf{L}_i|]})$$

And sends C_i to V. After round k, $\mathbf{V}$ outputs the set of queries $\mathbf{Q} = \{Q_i\}_{i \in [k]}$ to each oracle Π_i. The prover P responds by opening the key-value commitments at the requested positions: for all $i \in [k]$ defining $\mathcal{M}_i = \{(j, \Pi_i(j))\}_{j \in Q_i}$, followed by sending $\mathcal{M}_i$ and $\mathbb{o}_i \leftarrow \mathsf{KV.Open}(\mathsf{pp}, \mathsf{o}_i, \mathcal{M})$ to V. The transformation above is essentially the one by Ben-Sasson et al. [BCS16, Section 6] (from IOPs to IPs) but replacing Merkle trees with the related notion of a key-value commitment.

Theorem 4.9 (Correctness of Construction 4.8)**.** *Given a key-value commitment scheme (see Definition 4.6): a stackable holographic IOP $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ can be compiled into an efficient stackable interactive argument (P, V). Furthermore, the running time of the compiled $\mathcal{S}_{\mathrm{DET}}$ is that of $\mathcal{S}_{\mathrm{DET}}^{\mathbf{I}(\mathbb{i})}$ from the IOP, plus that of computing $(\mathsf{C}_1, \ldots, \mathsf{C}_k)$, which is $O(\sum_i |\Pi_i^*| \cdot \mathsf{poly}(\lambda, \log(|\Pi_i|)))$ (where $|\Pi_i|$ is the length of the i-th oracle in the real execution) See* **??** *for the proof.*

4.4 Stackable RS-IOPs

In this section we show that two key IOP protocols from the literature, Aurora [BCR+19] and Fractal [COS20] can be made stackable. These protocols are proof systems for the R1CS relation, defined formally below.

Definition 4.10. *Rank-one constraint satisfiability (R1CS) is an indexed NP relation consisting of all index-instance-witness tuples $((\mathbb{F}, A, B, C), x, w)$ for $A, B, C \in \mathbb{F}^{n \times n}$, $x \in \mathbb{F}^k$, $w \in \mathbb{F}^{n-k}$, such that for $z = (x \| w)$, $Az \circ Bz = Cz$, where $\circ$ is the element-wise product.*

Before proceeding to discuss how to make these protocols stackable, we provide a brief overview of the Aurora and Fractal RS-IOPs. These descriptions are not comprehensive, but rather aim to give context for the stackable variants presented later. Both protocols start from the same basic template:

1. On input $((\mathbb{F}, A, B, C), x, w)$, the prover sends to the verifier a (Reed–Solomon) encoding f_w of w, from which the verifier can deterministically compute an encoding f_z of $z = (x \| w)$. The prover also computes vectors Az, Bz, Cz and sends their corresponding encodings f_A, f_B, f_C to the verifier.

2. For each $M \in \{A, B, C\}$, the prover and verifier engage in the "lincheck" protocol to show that f_M is an encoding of Mz. This involves one or two rounds of interaction for Aurora and Fractal respectively, after which the verifier will output some rational constraints.

3. Lastly the verifier outputs the constraint "$f_A(i) \cdot f_B(i) - f_C(i) = 0$ for all $i \in [n]$".

To achieve zero-knowledge, the encodings f_w, f_A, f_B, f_C are *randomized* so that any "view" consisting of b locations in the encoding is distributed as a uniformly random vector in $\mathbb{F}^{\mathsf{b}}$; hence the messages sent in

Step 1 are recyclable. Because the prover does not send any information in Step 3, it is not relevant for zero-knowledge or stackability. As such, we need only focus on Step 2. Indeed, the difference between Aurora and Fractal lies in this step: Aurora's lincheck has verification time linear in the number of nonzero entries in A, B, C, whereas Fractal's lincheck is exponentially faster after preprocessing. As a result, they behave quite differently when stacked.

Aurora is Stackable. We show that a small modification to Aurora yields a stackable RS-IOP. We first outline the lincheck protocol used in Aurora. Both the prover and verifier take as input a matrix M, and have access to Reed–Solomon codewords f, f_M, which purportedly satisfy the relation $f_M|_H = Mf|_H$ for specified $H \subseteq \mathbb{F}$. For $\alpha \in \mathbb{F}$, denote by $\mathbf{u}_\alpha$ the vector $(1, \alpha, \alpha^2, \ldots, \alpha^{|H|-1}) \in \mathbb{F}^H$.

1. The verifier sends a challenge point $\alpha \in \mathbb{F}$.
2. The prover and verifier both compute the vector $\mathbf{u}_\alpha M \in \mathbb{F}^H$ along with its low-degree extension $\hat{g}$.
3. The prover and verifier then engage in the zero-knowledge sumcheck protocol to show that

$$\langle \mathbf{u}_\alpha M, f|_H \rangle - \langle \mathbf{u}_\alpha, f_M|_H \rangle = \sum_{a \in H} \hat{u}_{\alpha,M}(a) f(a) - \hat{g}(a) f_M(a) = 0 \,.$$

This protocol is complete because if $f_M|_H = Mf|_H$ then for all vectors u, $\langle u, f_M|_H \rangle = \langle u, Mf|_H \rangle = \langle uM, f|_H \rangle$. For soundness, observe that if $f_M|_H \neq Mf|_H$ then $\langle \mathbf{u}_\alpha M, f|_H \rangle - \langle \mathbf{u}_\alpha, f_M|_H \rangle$ is a nonzero low-degree polynomial in α; soundness follows by elementary algebra and the soundness of the zero-knowledge sumcheck protocol.

Observe that the only prover-to-verifier communication in this lincheck protocol is within Step 3; specifically, in the execution of zero-knowledge sumcheck. We now recall (and slightly modify) the zero-knowledge sumcheck protocol, which relies on the following lemma.

Lemma 4.11 (By Ben-Sasson et al. [BCR+19])**.** *Let H be a coset of an additive or multiplicative subgroup of $\mathbb{F}$. Then there is a polynomial $\Sigma_{H,Y}(X)$, which can be evaluated in time $\mathrm{polylog}(|H|)$, such that the following holds: let $\hat{f} \in \mathbb{F}[X]$ be such that $\deg(\hat{f}) < |H|$. Then $\sum_{\alpha \in H} \hat{f}(\alpha) = \sigma$ if and only if there exists $\hat{g}$ with $\deg(\hat{g}) < |H| - 1$ such that $\hat{f}(X) \equiv \Sigma_{H,\sigma}(\hat{g}(X))$.*

The protocol proceeds as follows: The prover and verifier have access to a summand codeword f of degree d, which purportedly satisfies $\sum_{a \in H} \hat{f}(a) = 0$.

1. The prover chooses a random polynomial $\hat{r}$ of degree d, computes $\zeta = \sum_{a \in H} \hat{r}(a)$, and sends r, ζ to the verifier.
2. The verifier sends a challenge β.
3. The prover divides $\hat{q} := \hat{r} + \beta \hat{f}$ by v_H to obtain $\hat{g}, \hat{h}$ satisfying the identity $\hat{q} \equiv \Sigma_{H,\zeta}(\hat{g}) + \hat{h} \cdot v_H$ with $\deg(\hat{g}) < |H| - 1$, and sends h to the verifier.
4. The verifier outputs the rational constraint "$\deg(\hat{e}) < |H| - 1$", where $\hat{e} := \Sigma^{-1}_{H,\zeta}(\hat{q} - \hat{h} \cdot v_H)$.

The zero-knowledge simulator given by [BCR+19] operates by first choosing a random polynomial $\hat{q}$, and sending $\zeta = \sum_{a \in H} \hat{q}(a)$ in the first round. $\hat{g}, \hat{h}$ are obtained from this $\hat{q}$ in the same way as the honest prover. Queries to r are answered using $q - \beta f$.

We are now ready to show that the above protocol is stackable, after a small modification.

Theorem 4.12. *The Aurora zero-knowledge RS-IOP for R1CS [BCR+19, Protocol 7.5] is stackable (after a small modification) with $\mathcal{S}_{\textsc{det}}$ running in time $O(\|A\| + \|B\| + \|C\| + n \log^2 \mathsf{b} \log\log \mathsf{b})$ (measured in field operations).*

Proof. The only modification necessary is to the zero-knowledge sumcheck protocol. Specifically, in Step 3, the prover will also send g; this is purely for the purposes of simulation and does not affect soundness.

Note that in a real execution, g, h are (marginally) uniformly random codewords, and so can be generated by $\mathcal{S}_{\textsc{rand}}$ (i.e., they are recyclable). Hence the only oracle in the protocol that is *not* recyclable is r. The inclusion of g in the protocol allows $\mathcal{S}_{\textsc{det}}$ to compute r as $\Sigma_{H,\zeta}(g) + h \cdot v_H - \beta f$.

As a result, the time complexity of $\mathcal{S}_{\textsc{det}}$ is dominated by the evaluation of f at b points. This requires computing $rA, rB, rC \in \mathbb{F}^n$ for some $r \in \mathbb{F}^m$, which takes $O(\|A\| + \|B\| + \|C\|)$ field operations, and evaluating the low-degree extensions of these vectors at b points, which takes $O(n \log^2 \mathsf{b} \log\log \mathsf{b})$ operations using the algorithm of [BM74]. □

4.5 Stactal

Next we describe "stackable Fractal", or Stactal, a variant of the Fractal protocol [COS20] which can be efficiently stacked. The verifier in Fractal runs in time quasilinear in the length of the input vector x and *polylogarithmic* in the dimensions of A, B, C. This is achieved via a sparse holographic encoding of A, B, C using the Reed–Solomon code.

First, we discuss why directly stacking Fractal leads to an inefficient protocol. Recall that in the stacked protocol, the prover and verifier run the instance-dependent part of the simulator $\mathcal{S}_{\text{DET}}$ on each clause $j \in [\ell]$. Therefore, to achieve the desired computational savings for the prover while maintaining the complexity of the verifier, we want $\mathcal{S}_{\text{DET}}$ to run in polylogarithmic time. Unfortunately, this is not possible for the original Fractal protocol (in the true disjunction setting), as we explain next.

The verifier's running time in the Aurora protocol is dominated by the lincheck subprotocol: specifically, the cost of evaluating, for each input matrix $M \in \{A, B, C\}$, the low-degree extension $\hat{u}_{\alpha,M}$ of the vector $\mathbf{u}_\alpha M$. To eliminate this cost, Fractal replaces Aurora's lincheck protocol with a *holographic* variant. In particular, [COS20] shows that, given an appropriate encoding of the input matrices, there is a protocol that allows the verifier to check an evaluation of this low-degree extension in time $\mathrm{polylog}(\|M\|)$.

Since the verifier cannot compute this evaluation itself, the prover sends $\hat{u}_{\alpha,M}(\beta)$ for the desired evaluation point β. In the standard setting of zero-knowledge, since the input matrices are *public*, this is not a problem: the simulator can simply compute this evaluation as the honest prover would, in time linear in $\|M\|$. In the stacking setting, however, this computation would be part of $\mathcal{S}_{\text{DET}}$, more than negating the computation savings obtained via holography.

Worse, it is not possible to simply design a better simulator: for most choices of α, β, $\hat{u}_{\alpha,M}(\beta)$ depends on every nonzero entry of M. Thus $\mathcal{S}_{\text{DET}}$ must run in at least linear time. To resolve this, we must instead significantly modify the Fractal protocol. In more detail, we allow the prover to "pad" the input matrices with randomness, in a way that does not affect the satisfiability of the statement, so that $\hat{u}_{\alpha,M}(\beta)$ becomes uniformly random. The simulator for this protocol runs in time $\mathrm{polylog}(\|M\|)$ and makes a small number of queries to the encoding of M.

Theorem 4.13 (Stactal)**.** *The protocol obtained from Fractal by replacing the holographic lincheck protocol with Construction A.3 is stackable, with $\mathcal{S}_{\text{DET}}$ running in time $O(\mathsf{b} \cdot (|x| + \mathrm{polylog}(\|A\| + \|B\| + \|C\|)))$ (measured in field operations).*

We defer the details of the construction and proofs to Appendix A.

5 Speed-Stacking Compressed Σ-Protocols

We now turn our attention to stacking sublinear proofs based on folding arguments. "Folding arguments" refers to a class of proof system that relies on algebraic structure and interaction to iteratively reduce the size of (or "fold") the statement of interest. The two most notable instantiations of this class are Bulletproofs [BBB+18], which give a folding argument for inner products, and Compressed Σ-Protocols [AC20, ACF21, ACK21], which give folding arguments for linear forms. In this section, we show how to stack Compressed Σ-Protocols and demonstrate the computational savings that our techniques offer when applied to them. In the next section, we demonstrate how to stack Bulletproofs, which as discussed earlier are less amenable to computational savings from our stacking approach.

5.1 Overview of Compressed Σ-Protocols

Compressed Σ-protocols were proposed in a series of recent works by Attema, Cramer, Fehr and Kohl [AC20, ACF21, ACK21]. In this section, we focus on the specific instantiation of this approach proposed by Attema,

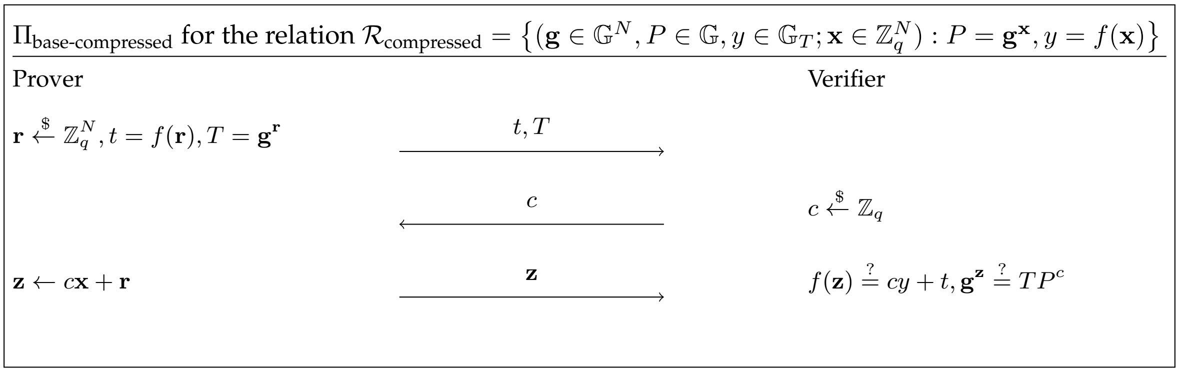
$\Pi_{\text{base-compressed}}$ for the relation $\mathcal{R}_{\text{compressed}} = \left\{(\mathbf{g} \in \mathbb{G}^N, P \in \mathbb{G}, y \in \mathbb{G}_T; \mathbf{x} \in \mathbb{Z}_q^N) : P = \mathbf{g}^{\mathbf{x}}, y = f(\mathbf{x})\right\}$
Prover
Verifier
$\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^N, t = f(\mathbf{r}), T = \mathbf{g}^{\mathbf{r}}$
$\xrightarrow{t, T}$
$\xleftarrow{c}$
$c \xleftarrow{\$} \mathbb{Z}_q$
$\mathbf{z} \leftarrow c\mathbf{x} + \mathbf{r}$
$\xrightarrow{\mathbf{z}}$
$f(\mathbf{z}) \overset{?}{=} cy + t, \mathbf{g}^{\mathbf{z}} \overset{?}{=} TP^c$

Figure 4: Base Σ-protocol protocol used in Compressed Σ-protocols [ACF21].

Cramer, and Fehr [ACF21], as it has a clean presentation.

Notation. We slightly modify some of the notation presented by Attema, Cramer, and Fehr in [ACF21] for clarity of presentation, but endeavor to make it sufficiently consistent that an interested reader can easily refer back to their work for additional details. Let $\mathbb{G}$ be a cyclic group of prime order p. Let f be a homomorphism from (additive) $\mathbb{Z}_q^N$ to some group $\mathbb{G}_T$.[10] We denote the set of such homomorphisms as $\mathcal{L}^N$.

Let $\mathbf{g_i} = (g_{i,1}, g_{i,2}, \ldots, g_{i,N})$ be vectors of generators in $\mathbb{G}$, where the size of the vector will either be stated explicitly or, when clear from context, left implicit. All other lower-case letters, *e.g.* c, a, refer to elements in $\mathbb{Z}_q$, and bold lower-case letters, *e.g.* $\mathbf{x_i}, \mathbf{z}$ refer to vector of elements in $\mathbb{Z}_q$. Let $\mathbf{x} = \{x_1, \ldots, x_M\} \in \mathbb{Z}_q^N$, and $f : \mathbb{Z}_q^N \to \mathbb{G}_T$. We denote $\mathbf{x}_L = \{x_1, \ldots, x_{N/2}\}$ and $\mathbf{x}_R = \{x_{N/2+1}, \ldots, x_N\}$. We denote $f_L : \mathbb{Z}_q^{N/2} \to \mathbb{G}_T$ as the function $f(\mathbf{x}_L, 0, \ldots, 0)$ and $f_R : \mathbb{Z}_q^{N/2} \to \mathbb{G}_T$ as the function $f(0, \ldots, 0, \mathbf{x}_R)$. Uppercase letters refer to elements of $\mathbb{G}$. For a vector $\mathbf{g_i}$ of length N, we denote the first $N/2$ elements of $\mathbf{g_i}$ as $\mathbf{g_i}_L$ and the remaining $N/2$ elements of $\mathbf{g_i}$ as $\mathbf{g_i}_R$. We denote the element-wise group operation of two vectors of group elements as $\mathbf{g} * \mathbf{g}' = (g_1 g'_1, \ldots, g_N g'_N)$, where N is an arbitrary size parameter. Finally we denote multiexponentation by $\mathbf{g}^{\mathbf{x}} = \prod_i g_i^{x_i}$.

Compressed Σ-Protocols. Attema et al. [ACF21] consider the relation

$$\mathcal{R}_{\text{compressed}} = \left\{(\mathbf{g} \in \mathbb{G}^N, P \in \mathbb{G}, y \in \mathbb{G}_T, f \in \mathcal{L}^N; \mathbf{x} \in \mathbb{Z}_q^N) : P = \mathbf{g}^{\mathbf{x}}, y = f(\mathbf{x})\right\},$$

where $\mathbf{x}$ is a vector of length N and f is a homomophism from $\mathbb{Z}_q^N$ to $\mathbb{G}_T$. Intuitively, their protocol is a "standard" (Schnorr-type) Σ-protocol (shown in Figure 4), where the prover computes $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^N, T = \mathbf{g}^{\mathbf{r}}$ and $t = f(\mathbf{r})$ and sends t, T to the verifier. Upon receiving a challenge c, it computes and sends $\mathbf{z} = c\mathbf{x} + \mathbf{r}$ to the verifier. The verifier then verifies if: $\mathbf{g}^{\mathbf{z}} \overset{?}{=} TP^c$ and $f(\mathbf{z}) \overset{?}{=} cy + t$ (later in this section, we will denote the value TP^c as Q). Note that, the third round message $\mathbf{z} \in \mathbb{Z}_q^N$ that the prover sends in this protocol contains $O(N)$ elements, which is undesirable.

To compress the communication complexity of this last round message, this line of works makes the observation that the message $\mathbf{z}$ is itself a trivial proof of knowledge for an instance of $\mathcal{R}_{\text{compressed}}$. Specifically,

$$\left\{(\mathbf{g} \in \mathbb{G}^N, TP^c \in \mathbb{G}, cy + t \in \mathbb{G}_T, f \in \mathcal{L}^N; \mathbf{z} \in \mathbb{Z}_q^N) : P = \mathbf{g}^{\mathbf{z}}, y = f(\mathbf{z})\right\}.$$

Importantly, however, sending $\mathbf{z}$ reveals nothing about $\mathbf{x}$. As such, for reducing the communication complexity of the base protocol, it suffices to design a proof of knowledge for $\mathcal{R}_{\text{compressed}}$ that need not be

[10] Although $\mathbb{G}_T$ is often used to indicate a target group in a pairing, in this context it simply refers to the target group of the homomophism; there are no pairings here. Additionally, we encourage the reader to think of $\mathbb{G}$ simply as $\mathbb{Z}_q$, as this is the clear motivation for the proof system.

zero-knowledge. The various versions of Compressed Σ-Protocols design slightly different variants of this compressive "folding" proof of knowledge. In this work, we focus on the one presented in [ACF21].

Folding Argument. They start by enabling the prover and the verifier to split the statement in half and fold it in on itself, resulting in a transcript that is half the size. This is done as follows: the verifier generates a random challenge $c \in \mathbb{Z}_q$, and the task of proving the original instance is reduced to the problem of proving another instance of $\mathcal{R}_{\text{compressed}}$ for a new linear form $f' = cf_L + f_R$ with bases $\mathbf{g}' = \mathbf{g}_L^c * \mathbf{g}_R$. Note the dimension of each of these is half the dimension of the original. All that remains now is to generate a new commitment P' and find a new target value y' for this reduced-dimension instance. The prover and verifier compute this as follows:

(1) Before c is sent by the verifier, the prover computes $A = \mathbf{g}_R^{\mathbf{x}_L}, a = f_R(\mathbf{x}_L), B = \mathbf{g}_L^{\mathbf{x}_R}, b = f_L(\mathbf{x}_R)$ and sends (A, a, B, b) to the verifier.

(2) The verifier then samples and sends c.

(3) The prover and verifier compute $P' = AP^cB^{c^2}$ and $y' = a + cy + c^2b$.

The new instance is now of the form:

$$\left\{(\mathbf{g}' \in \mathbb{G}^{N/2}, AP^cB^{c^2} \in \mathbb{G}, y' \in \mathbb{G}_T, f' \in \mathcal{L}^{N/2}; \mathbf{x}' \in \mathbb{Z}_q^{N/2}) : AP^cB^{c^2} = \mathbf{g}'^{\mathbf{x}'}, y' = f'(\mathbf{x}')\right\}$$

Note that a trivial proof of knowledge for this new instance is just $\mathbf{x}' = \mathbf{x}_L + c\mathbf{x}_R$, which is already half the length of the initial $\mathbf{x}$. The same process can be repeated again for computing a proof knowledge of $\mathbf{x}'$, to further reduce the communication complexity. This process is recursively applied until the final trivial witness is of a constant size.

We note that Attema et al. have demonstrated how to use their protocol(s) to prove generic circuit satisfiability, by arithmetizing the circuit into a compatible format. We focus on the simpler base case where the prover only wishes to prove a linear form, and disucss the generalization in Section 5.3. We now state the main Theorem from [ACF21].

Theorem 5.1 ([ACF21])**.** *Let $N > 2$. There exists a $(2\mu + 3)$-move protocol $\Pi_{compressed}$ for relation $\mathcal{R}_{compressed}$, where $\mu = \lceil \log_2(N) \rceil - 2$. It is perfectly complete, special honest-verifier zero-knowledge and unconditionally (2,3,3, …, 3)-special sound.*

We give an unrolled description of protocol $\Pi_{\text{compressed}}$ from [ACF21] (with some modifications highlighted in red, which we discuss next) in Figure 5.

5.2 Compressed Σ-Protocols are Stackable

We consider statements of the form:

$$\mathcal{R}_{\text{dis-compressed}} = \{(\mathbf{g} \in \mathbb{G}^N, \{P_i \in \mathbb{G}, y_i \in \mathbb{G}_T^i, f_i\}_{i \in [\ell]}; \mathfrak{a} \in [\ell], \mathbf{x}_\mathfrak{a} \in \mathbb{Z}_q^N) : P_\mathfrak{a} = \mathbf{g}^{\mathbf{x}_\mathfrak{a}}, y_\mathfrak{a} = f_\mathfrak{a}(\mathbf{x}_\mathfrak{a})\}.$$

Notice that this statement allows for different homomorphisms and commitments for each clause $i \in [\ell]$. This is a stronger notion of disjunctions than considered in [ACF21], which give proofs where either the homomorphism or commitment is fixed across a disjunction of multiple clauses. Our goal in stacking will be concrete speed; specifically, we aim to minimize the number of expensive group operations and multi-exponentations the prover is required to do for each clause.

Intuition. A first order intuition for speed-stacking Compressed Σ-Protocols is as discussed in the technical overview: first stack the communication inefficient base protocol, and then apply the recursive folding "after" stacking the protocols together. The base Σ protocol in Compressed Σ-Protocols can trivially be stacked using the stacking compiler given from Goel et al. [GGHAK22], reusing the entirety of $\mathbf{z}$ as a recyclable message and allowing t, T to be deterministically recomputed. As such, it is natural to expect that this multiround protocol should contain all recyclable messages besides t, T, and indeed it does.

$\Pi_{\text{compressed}}$ for the relation $\mathcal{R}_{\text{compressed}} = \left\{(\mathbf{g_1} \in \mathbb{G}^N, P \in \mathbb{G}, y \in \mathbb{G}_T, f \in \mathcal{L}^N; \mathbf{x} \in \mathbb{Z}_q^N) : P = \mathbf{g_1}^{\mathbf{x}}, y = f(\mathbf{x})\right\}$

Prover		Verifier
$\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^N, t = f(\mathbf{r}), T = \mathbf{g_1}^{\mathbf{r}}$	$\xrightarrow{t, T}$	$c \xleftarrow{\$} \mathbb{Z}_q$
$\mathbf{x_1} \leftarrow c\mathbf{x} + \mathbf{r}$	$\xleftarrow{c}$	
$f_1 := f, y_1 = cy + t, Q_1 = TP^c$ $A_1 = \mathbf{g_1}_R^{\mathbf{x_1}_L}, a_1 = \mathsf{UniqueOrRand}(f, y, f_{1,R}(\mathbf{x_1}_L), c)$		$f_1 := f, y_1 = cy + t$ $Q_1 := TP^c$
$B_1 = \mathbf{g_1}_L^{\mathbf{x_1}_R}, b_1 = \mathsf{UniqueOrRand}(f, y, f_{1,L}(\mathbf{x_1}_R), c)$	$\xrightarrow{Q_1, A_1, B_1, a_1, b_1}$	$c_1 \xleftarrow{\$} \mathbb{Z}_q$
$\mathbf{x_2} \leftarrow c_1\mathbf{x_1}_L + \mathbf{x_1}_R$	$\xleftarrow{c_1}$	$Q_2 := A_1 {Q_1}^{c_1} {B_1^{c_1}}^2, \mathbf{g_2} := \mathbf{g_1}_L^{c_1} * \mathbf{g_1}_R$
$\mathbf{g_2} := \mathbf{g_1}_L^{c_1} * \mathbf{g_1}_R$ $f_2 := c_1 f_{1,L} + f_{1,R}, y_2 = a_1 + c_1 y_1 + c_1^2 b_1$		$a_1, b_1 \leftarrow \mathsf{UniqueOrRand}(f, y, (\{a_1, b_1\}), c)$ $f_2 := c_1 f_{1,L} + f_{1,R}, y_2 = a_1 + c_1 y_1 + c_1^2 b_1$
	$\vdots$	
$A_i = \mathbf{g_i}_R^{\mathbf{x_i}_L}, B_i = \mathbf{g_i}_L^{\mathbf{x_i}_R},$ $a_i = \mathsf{UniqueOrRand}(f, y, (\{a_j, b_j\}_{j \in [i-1]}, f_{i,R}(\mathbf{x_i}_L)), \{c_j\}_{j \in [i-1]})$		
$b_i = \mathsf{UniqueOrRand}(f, y, (\{a_j, b_j\}_{j \in [i-1]}, f_{i,L}(\mathbf{x_i}_R)), \{c_j\}_{j \in [i-1]})$	$\xrightarrow{A_i, B_i, a_i, b_i}$	$c_i \xleftarrow{\$} \mathbb{Z}_q$
$\mathbf{x_{i+1}} \leftarrow c_i \mathbf{x_i}_L + \mathbf{x_i}_R$	$\xleftarrow{c_i}$	$Q_{i+1} := A_i {Q_i}^{c_i} {B_i^{c_i}}^2 \mathbf{g_{i+1}} := \mathbf{g_i}_L^{c_i} * \mathbf{g_i}_R$
$\mathbf{g_{i+1}} := \mathbf{g_i}_L^{c_i} * \mathbf{g_i}_R$ $f_{i+1} := c_i f_{1,L} + f_{1,R}, y_{i+1} = a_i + c_i y_i + c_i^2 b_i$		$a_i, b_i \leftarrow \mathsf{UniqueOrRand}(f, y, (\{a_j, b_j\}_{j \in [i]}), \{c_j\}_{j \in [i-1]})$ $f_{i+1} := c_i f_{1,L} + f_{1,R}, y_{i+1} = a_i + c_i y_i + c_i^2 b_i$
	$\vdots$	
$\mathbf{z} \leftarrow c_{\mu-1} \mathbf{x_{\mu-1}}_L + \mathbf{x_{\mu-1}}_R$	$\xrightarrow{\mathbf{z}}$	$(\mathbf{g}_\mu)^{\mathbf{z}} \stackrel{?}{=} Q_\mu$
		$f_\mu(\mathbf{z}) \stackrel{?}{=} y_\mu$

Figure 5: Compressed protocol $\Pi_{\text{compressed}}$ for proving linear forms by Attema et. al. [ACF21]. We introduce two minor modifications intro the protocol, described in the text. First, we have the prover send Q_1 in the protocol. Second, when the value of a_i, b_i are already uniquely determined, we have the prover simply send random elements instead. We annotate both these changes in red (the origional protocol can be seen simply be ignoring the red text). We define the helper function UniqueOrRand, which implements the second protocol modification. Specifically, this function determines which messages a_i, b_i were already uniquely constrained and which were "new" information, and therefore randomly distributed. The output of the function is the "corrected" values. This function can be implemented by Gauss-Jordan elimination, with runtime $N \log^2(N)$. Note that we abuse notation slightly in the definition of this function as written here and in the main text in order not to obscure the key parts of the protocol or simulator.

We note, however, that Attema et al.'s choice of compression mechanism requires a more careful analysis of this stacking approach. Not all the messages in the tail are recyclable. Observe that the messages in the tail are of the form A_i, B_i, a_i, b_i. While A_i, B_i are clearly recyclable, a_i, b_i are outputs of the some combination of parts of the linear form f and hence depend on f. Moreover, the *computation* required to verify the tail is also not re-usable. Specifically, the linear form f is itself incorporated into the compression mechanism, and f is never blinded, *i.e.* the computations relying on f cannot be "recycled" (to slightly abuse our terminology). As such, directly stacking the protocol will run into both efficiency problems and difficulty in proving zero-knowledge (i.e., in ensuring that the index of the active branch remains hidden).

We propose two minor modifications to this protocol in order for stacking to be maximally valuable:

(1) **Sending Q_1:** In the original protocol described in [ACF21], the prover and verifier independently compute the value Q_1 (*i.e.,* TP^c from the base protocol). The first modification that we propose is to have the prover send Q_1 during the first folding. This modification is simply for *efficiency* reasons (and therefore does not impact soundness or zero-knowledge) as Q_1 can be deterministically computed by the verifier and the deterministic simulator. However, computing Q_1 directly from the transcript (and, looking ahead, the recyclable messages) for simulating other messages is *expensive* — involving many exponentiations — and therefore we would like to avoid computing it as part of our deterministic simulator $\mathcal{S}_{\text{DET}}$. This modification is similar to the one used to make Aurora efficiently stackable in the previous Section.

(2) **Randomizing a_i and b_i:** In each round i of the folding argument, the prover sends $a_i = f_{i,R}(\mathbf{x}_{i,L})$ and $b_i = f_{i,L}(\mathbf{x}_{i,R})$. As such, as discussed above, a_i and b_i are not recyclable. Note that there are cases when the verifier *already knows* the values of a_i and b_i that it should expect to receive based on the functions $f_{i,L}, f_{i,R}$; for example, if either is the zero function. More generally, the verifier might be able to predict the value of a_i, b_i given $f, a_{i-1}, b_{i-1}, c_{i-1}, a_{i-2}, b_{i-2}, c_{i-2} \ldots$. As such, a_i, b_i are not generally recyclable. However, since f is a linear form, we observe that the possible values of a_i, b_i correspond to the solutions of a linear system in the coefficients of f and the challenges so far. As such, they are either marginally uniform, or there is an efficient algorithm determining their unique assignment. Hence, we propose to modify the protocol to have the prover send *uniform* elements a_i or b_i when their "correct" value can already be determined by the verifier. The verifier can simply *ignore* these elements when the "correct" value is already determined. It is easy to see that this does not affect soundness or zero-knowledge of Compressed Σ-protocols.

We give a complete description of the protocol, including these modifications (highlighted in red), in Figure 5. To capture our second modification, we define a function UniqueOrRand that is used to determine values a_i and b_i in each folding. In particular, for each folding (to compute a_i, b_i), it takes the following inputs: the function f, evaluation $y = f(\mathbf{x})$, previously computed values and challenges $a_{i-1}, b_{i-1}, c_{i-1}, a_{i-2}, b_{i-2}, c_{i-2} \ldots$ and $f_{1,R}(\mathbf{x_i}_L)$ (when computing a_i) or $f_{1,L}(\mathbf{x_i}_R)$ (when computing b_i). UniqueOrRand checks if the values a_i and b_i are already determined based on previous computed values and challenges — in which case it outputs a random value —- else, it outputs $f_{1,R}(\mathbf{x_i}_L)$ for a_i and $f_{1,L}(\mathbf{x_i}_R)$ for b_i. We are now ready to describe how to speed-stack Compressed Σ-Protocols and prove the following theorem, setting $M^{\text{COMP}}_{\text{RAND}} = (Q_1, A_1, B_1, a_1, b_1, \ldots, A_{\mu-1}, B_{\mu-1}, a_{\mu-1}, b_{\mu-1}, \mathbf{z})$ for notational convenience.

Theorem 5.2. *Compressed Σ-protocols [ACF21] (Figure 5), denoted as $\Pi_{compressed}$, is stackable.*

Proof. **Random Simulation $\mathcal{S}_{\text{RAND}}$.** We start by showing that there exists a statement-independent simulator $\mathcal{S}^{\text{COMP}}_{\text{RAND}}$ that produces the set of messages $M^{\text{COMP}}_{\text{RAND}}$ such that those messages are indistinguishable from an honest execution of the protocol. The simulator can be round on the left side of Figure 6.

$\mathcal{S}^{\text{COMP}}_{\text{RAND}}$ samples random values for A_i, B_i, a_i, b_i, and $\mathbf{z}$. From these values, Q_1 is already uniquely determined. As such, $\mathcal{S}^{\text{COMP}}_{\text{RAND}}$ folds the generators using the challenges to find the final value of $Q_{\log(N)}$, and then solves for Q_1.

To see why these messages are all recyclable, simply observe that the value $\mathbf{x}_1$ is uniformly random in $\mathbb{Z}_q^N$, as it was masked with $\mathbf{r}$. First consider, the distribution of a_i, b_i. Because f are only linear forms, this means that the range of f is either (1) uniquely determined, or (2) uniformly random in $\mathbb{Z}_q$. In case (1),

```
$(M^{\text{COMP}}_{\text{RAND}}) \leftarrow \mathcal{S}^{\text{COMP}}_{\text{RAND}}(1^\lambda, C = (c, c_1, \dots c_\mu))$
for $i \in [\log(N) - 1]$ :
    $A_i, B_i \xleftarrow{\$} \mathbb{G}, a_i, b_i \xleftarrow{\$} \mathbb{G}_T$
    $\mathbf{g_{i+1}} := \mathbf{g_i}^{c}_{L} * \mathbf{g_i}_R$
$\mathbf{z} \xleftarrow{\$} \mathbb{G}_T$
$Q_{\log(N)} := (\mathbf{g}_{\log(N)})^{\mathbf{z}}$
for $i \in [\log(N) - 1, \dots, 1]$
    $Q_i := (Q_{i+1}(A_i)^{-1}(B_i^{c^2})^{-1})^{-c}$
return $M^{\text{COMP}}_{\text{RAND}} = (Q_1, \{A_i, B_i, a_i, b_i\}_{i \in [\log(N)-1]}, \mathbf{z})$
```

```
$(t, T) := \mathcal{S}^{\text{COMP}}_{\text{DET}}(\mathsf{st}, \mathbb{x} = (P, y, f), C, M^{\text{COMP}}_{\text{RAND}})$
for $i \in [2, \dots, \log(N)]$
    $f_i := cf_{i-1,L} + f_{i-1,R}$
$y_{\log(N)} := f_{\log(N)}(\mathbf{z})$
$\{\hat{a}_i, \hat{b}_i\}_{i \in [\log(N)-1]} = \mathsf{UniqueOrRand}(f, y, \{a_i, b_i\}_{i \in [\log(N)-1]}, C)$
for $i \in [\log(N) - 1, \dots, 1]$
    $y_i := \frac{y_{i+1} - a_i - c_i^2 b_i}{c_i}$
$T := Q_1 P^{-c}, t := y_1 - cy$
return $(t, T)$
```

Figure 6: Simulators for Compressed Σ-protocols. We define the helper function $\mathsf{UniqueOrRand}$, which implements the second protocol modification outlined in the text. Specifically, this function determines which messages a_i, b_i were already uniquely constrained and which were "new" information, and therefore randomly distributed. The output of the function is the "corrected" values $\hat{a}_i, \hat{b}_i$. This function can be implemented by Gauss-Jordan elimination, with runtime $O(N \log^2(N))$.

we have already modified the protocol such that the values a_i, b_i are uniform by definition. In case (2), we are evaluating f at a random point $\mathbf{x}_1$, and therefore the output is uniform. The uniformity of A_i, B_i and $\mathbf{z}$ follows trivially in the exact same way.

Deterministic Simulation $\mathcal{S}_{\text{DET}}$. Next, we construct a *deterministic*, simulator $\mathcal{S}^{\text{COMP}}_{\text{DET}}$ that takes in a set of messages $M^{\text{COMP}}_{\text{RAND}}$ and completes the transcript (see the right side of Figure 6). It is trivial to see that these messages are computed deterministically and correctly.

□

Combining Theorems 5.2 and 3.5, we get the following Corollary.

Corollary 5.3. *Let $\Pi_{\textit{speed-compressed}}$ be output of the compiler in Figure 2 recursively applied to $\Pi_{\textit{compressed}}$ using $\mathcal{S}^{\text{COMP}}_{\text{RAND}}$ and $\mathcal{S}^{\text{COMP}}_{\text{DET}}$ as defined in the proof of Theorem 5.2. Then $\Pi_{\textit{speed-compressed}}$ is a stackable ZK-IP for $\mathcal{R}_{\textit{dis-compressed}}$ with logarithmic communication complexity, and prover computational complexity $O(\mathsf{Time}(\Pi_{\textit{compressed}}) + \ell \cdot \mathsf{Time}(\mathcal{S}^{\text{COMP}}_{\text{DET}}))$.*

Efficiency of Speed-Stacked Compressed Σ-Protocols. Our goal in stacking Compressed Σ-Protocols is to minimize the number of group operations that the prover must perform when proving a disjunctive statement, as group operations are typically significantly more expensive than field operations. Based on our compiler, it is easy to see that we get the most savings when the linear form f is actually a homomorphism from one field to another field. In that case the vast majority of the group operations are only necessary in the active clause. Concretely, the prover's computational cost for running the compiled protocol is $\mathsf{Time}(\Pi_{\text{compressed}}) + \ell \cdot \mathsf{Time}(\mathcal{S}^{\text{COMP}}_{\text{DET}}) + \mathsf{Time}(\mathsf{Gen}) + \mathsf{Time}(\mathsf{EquivCom}) + \mathsf{Time}(\mathsf{Equiv})$. In this case, in $\mathcal{S}^{\text{COMP}}_{\text{DET}}$, the prover computes only 1 exponentiation and 1 group operation ($T := Q_1 P^{-c}$). If we consider the commitment scheme proposed by Goel et al. [GGHAK22], both key generation and committing require ℓ exponentations and group operations, while equivocation requires only field operations. Thus the overhead (when counting group operations) introduced from running a disjunction with ℓ clauses is only 2ℓ exponentiation and 2ℓ group operations. Importantly all the multi-exponentations resulting from folding $\mathbf{g}$ and computing the A_i, B_i can be completely avoided.

We note that our modifications to the protocol do introduce some overheads. Namely, the verifier (and thus the deterministic simulator) need to decide when a message is already uniquely determined. This computation requires attempting to solve the system of equations for the particular value a_i, b_i. The verifier can simply to this using Gauss-Jordan elimination, which will take $N \log^2(N)$ field operations.

$\Pi_{\text{circ-compressed}}$ for the relation $\mathcal{R}_{\text{circ-compressed}} = \left\{(\mathbf{g} \in \mathbb{G}^{(2m+I+3)}, h \in \mathbb{G}, C : \mathbb{Z}_q^I \to \mathbb{Z}_q; \mathbf{x} \in \mathbb{Z}_q^I) : C(\mathbf{x}) = 0\right\}$

Prover		Verifier
$u, v \xleftarrow{\$} \mathbb{Z}_q[X]$ s.t. $\mathsf{deg}(u), \mathsf{deg}(u) < m,$ and $\forall i \in [1, m], u(i) = \alpha_i, v(i) = \beta_i$ $w(X) := u(X)v(X)$ $\mathbf{y} = (\mathbf{x}, u(0), v(0), w(0), w(1), \ldots, w(2m))$		
$\rho \xleftarrow{\$} \mathbb{Z}_q^*$	$\xrightarrow{\mathsf{com} = h^{\rho}\mathbf{g}^{\mathbf{y}}}$	
	$\xleftarrow{\hat{c}}$	$\hat{c} \xleftarrow{\$} \mathbb{Z}_q^* \setminus \{0, 1, 2, \ldots, 2m+1\}$
$z_1 = u(\hat{c}), z_2 = v(\hat{c}), z_3 = w(\hat{c})$	$\xrightarrow{z_1, z_2, z_3}$	
	$\xleftarrow{\rho}$	$\rho \xleftarrow{\$} \mathbb{Z}_q^*$
$L =$ Linear combination of $C(\mathbf{x}) = 0,$ $u(\hat{c}) - z_1 = 0, v(\hat{c}) - z_2 = 0, w(\hat{c}) - z_3 = 0$ with coefficients $1, \rho, \rho^2, \ldots$		
	$\xrightarrow{\Pi_{\text{compressed}}\left((h, \mathbf{g}), \mathsf{com}, 0, L; \mathbf{y}\right)}$	$z_3 \overset{?}{=} z_1 z_2$, Verify $\left(\Pi_{\text{compressed}}\right)$

Figure 7: The airthmetic circuit satisfiability version of Compressed Σ-Protocols presented in [AC20]. We leverage several notation shortcuts introduced in [AC20] in our presentation: we denote the linear forms derived from the circuit topology as $C(\mathbf{x})$. We unroll Π_{nullity}, presented in [AC20], to explicitly expose the underlying protocol $\Pi_{\text{compressed}}$. This unrolled protocol verifies that the constraints $C((x)), u(\hat{c}) - z_1, v(\hat{c}) - z_2$ and $w(\hat{c}) - z_3$ are all zero with respect to the commitment $h^{\rho}\mathbf{g}^{\mathbf{y}}$, where in secret prover input in $\mathbf{y}$.

5.3 Extension to Circuit Satisfiability

In [AC20], Attema and Cramer present a reduction from general circuit satisfiability to the opening of linear forms, based on the techniques presented by Cramer, Damgård and Pastro [CDP12]. We briefly summarize this reduction and present the circuit satisfiability protocol presented in [AC20], before showing how to speed-stack this resulting protocol.

Overview of Circuit Satisfiability based on Compressed Σ-Protocols. Let $C : \mathbb{Z}_q^I \to \mathbb{Z}_q$ be an arithmetic circuit consisting of m multiplication gates, and let $\mathbf{x} \in \mathbb{Z}_q^I$ be a input such that $C(\mathbf{x}) = 0$. The prover wishes to prove, in zero-knowledge, that it knows such a satisfying input $\mathbf{x}$ for C, that is, the prover wants a zero-knowledge proof for the relation

$$\mathcal{R}_{\text{compressed-circ}} = \{(C : \mathbb{Z}_q^I \to \mathbb{Z}_q; \mathbf{x} \in \mathbb{Z}_q^I) : C(\mathbf{x}) = 0\}$$

Let $\{(\alpha_i, \beta_i, \gamma_i)\}_{i \in [m]}$ be the multiplication triples induced on the circuit C when evaluated on $\mathbf{x}$. That is, for the i^{th} multiplication gate, let α_i be the left input wire value, β_i be the right input wire value, and γ_i be the output wire value. To check that $C(\mathbf{x}) = 0$, the prover can demonstrate that for some fixed set of values $\{(\alpha_i, \beta_i, \gamma_i)\}_{i \in [m]}$, (1) $\forall i \in [m], \alpha_i \beta_i = \gamma_i$, (2) the output wire of the circuit is equal to 0 (which is a linear function of the γ_i's), and (3) that the proper linear relationships between $\{(\alpha_i, \beta_i, \gamma_i)\}_{i \in [m]}$ induced

```
$(M^{\text{COMP-CIRC}}_{\text{RAND}} \leftarrow \mathcal{S}^{\text{COMP-CIRC}}_{\text{RAND}}(1^\lambda, C = (\hat{c}, c, c_1, \dots c_\mu))$
    $\mathsf{com} \xleftarrow{\$} \mathbb{G}$
    $z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q, z_3 = z_1 z_2$
    $M^{\text{COMP}}_{\text{RAND}} \leftarrow \mathcal{S}_{\text{RAND}}(1^\lambda, C = (c, c_1, \dots c_\mu))$
    $\textbf{return } (\mathsf{com}, z_1, z_2, z_3, M^{\text{COMP}}_{\text{RAND}})$
```

```
$(t, T) \coloneqq \mathcal{S}^{\text{COMP-CIRC}}_{\text{DET}}(\mathbb{x} = (\mathbf{g}, \mathsf{com}, 0, L), C, M^{\text{COMP-CIRC}}_{\text{RAND}})$
    $\textbf{return } \mathcal{S}_{\text{DET}}(\mathbb{x} = (\mathbf{g}, \mathsf{com}, 0, L), C, M^{\text{COMP}}_{\text{RAND}})$
```

Figure 8: Simulators for Circuit Sat Compressed Σ-protocols.

by the circuit topology are preserved. For example, the circuit topology may dictate that $\alpha_3 = \gamma_1 + \gamma_2$ or that $\alpha_2 = \gamma_1$.

Attema and Cramer [AC20] adopt an idea from Cramer, Damgård and Pastro [CDP12] to effectively linearize these constraints and thereby allow checking them using their protocol for proving the opening of linear forms. The prover begins by sampling random degree m polynomials u and v such that $\forall i \in [m], u(i) = \alpha_i, v(i) = \beta_i$. The prover then computes the degree $2m$ polynomial $w(X) = u(X)v(X)$.[11] Finally, the prover commits to the polynomials by committing to the first m points of u and v and the first $2m + 1$ points of w. The prover and verifier then engage in an interactive protocol to checks all the constraints as follows:

- **Check all multiplication triples.** The verifier can check all of the multiplication triples in a batch be sampling a random $\hat{c} \in \mathbb{Z}_q$ such that $\hat{c} \geq 2m$ and checking that $u(\hat{c})v(\hat{c}) = w(\hat{c})$. This satisfies condition (1) and relies on the Shwartz-Zippel Lemma that has become a standard technique in efficient zero-knowledge. Moreover, note that $u(\hat{c}), v(\hat{c})$, and $w(\hat{c})$ can all be computed using a linear combination of the commited values via polynomial interpolation. Thus, the prover can convince the verifier that $u(\hat{c})v(\hat{c}) = w(\hat{c})$ using a proof of a linear opening.

- **Check linear relationships.** Attema and Cramer observe that each wire in the circuit can be computed as a linear combination of the input values $\mathbf{x}$ and values of γ_i, where the coefficients on the linear combination are a function of the circuit topology. This induces one linear form for each wire in the circuit, which jointly satisfy conditions (2) and (3).

Attema and Cramer slightly simplify the protocol of [CDP12] by only commiting to $u(0)$ and $v(0)$, rather than all of u and v, as the remaining points in u and v can be computed as a linear combination of $w(i)$ and $\mathbf{x}$. Additionally, in [AC20], Attema and Cramer present a version of the their proof of linear forms that allows a prover to batch together multiple linear forms that all open to 0 called Π_{nullity}; the protocol is a simple modification of underlying $\Pi_{\text{compressed}}$ that involves taking a random linear combination of the linear forms (for details of that protocol, we refer the reader to Section 5 of [AC20]). We have unrolled this protocol explicitly, such that it exposes the underlying instance of $\Pi_{\text{compressed}}$. They use this protocol to check that all the linear forms discussed above are satisfied.

The full protocol for circuit satisfiability presented in [AC20] is shown in Figure 7.

Speed-stacking $\Pi_{\textbf{circ-compressed}}$. We now discuss how to apply our speed stacking techniques to $\Pi_{\text{circ-compressed}}$. Specifically, when stacked, we get a prover-efficient protocol for the relation

$$\mathcal{R}_{\text{dis-compressed-circ}} = \{(\{C_i : \mathbb{Z}_q^I \to \mathbb{Z}_q\}_{i \in [\ell]}; \alpha \in [\ell], \mathbf{x}_\alpha \in \mathbb{Z}_q^I) : C_\alpha(\mathbf{x}_\alpha) = 0\}^{12}$$

The vast majority of the computation and communication for $\Pi_{\text{circ-compressed}}$ is contained within an instance of $\Pi_{\text{compressed}}$. As such, it is simple to see how to speed-stack the resulting protocol. Indeed, the

[11] In the description in [AC20], the authors use f, g, h instead of u, v, w.

[12] Note that the final statement, as show in Figure 7, also contaings genertors that are used for commitments. We ommit them in the text for readability.

simulator required for speed-stacking $\Pi_{\text{circ-compressed}}$ is virtually identical to that presented in Section 5.2, and is presented in Figure 8.

Notice that the additional messages at the beginning of the $\Pi_{\text{circ-compressed}}$ are all distributed independently of the statement: com is, definitionally, distributed independently of $\mathbf{y}$ and the only constraint on z_1, z_2 and z_3 is that $z_3 = z_1 z_2$, and because u and v and sampled at random, both z_1 and z_2 are distributed uniformly. This means that that the the randomized simulator need only sample these messages at random and then invoke the randomized simulator of the underlying protocol $\Pi_{\text{compressed}}$. The deterministic simulators need only invoke the deterministic simulators of $\Pi_{\text{compressed}}$ directly. The proof is a trivial extension of that in Section 5, so we leave it as an exercise to the reader. To see the efficient gains offered by speed-stacking $\Pi_{\text{circ-compressed}}$, observe that the length of $\mathbf{y}$ in $\Pi_{\text{circ-compressed}}$ is proportional to the size of the circuit $(2m + I + 3)$, meaning the dimensions of the resulting multiexponentiation will be a function of the circuit size. When speed stacking, the deterministic simulator need only computing field operations proporitonal

follows directly from the derivations in the previous section.

6 Speed-Stacking Bulletproofs

In this section, we show how to stack Bulletproofs and demonstrate why our techniques only offer computational savings in very limited scenarios, when applied to Bulletproofs.

6.1 Overview

Bünz et al. demonstrate that several relations including range proofs and circuit satisfiability can be reduced to a "privacy-free" inner product relation via an interactive protocol. The inner-product relation can be proved using a folding argument based sublinear argument system introduced in their work. Overall, the argument system for proving the inner product relation is the most computationally intensive component in Bulletproofs. As discussed in the introduction, our main observation is that in the case of disjunctions, all branches can be reduced to the same privacy-free inner product relation and the transcript for that part of the proof can be "recycled". However, this does not always yield significant computational savings. In this subsection, we give an overview of the Bulletproofs protocol and the main ideas underlying our approach for how to stack Bulletproofs for disjunctions. In subsequent subsections, we show the outcome of applying these techniques to the two types of relations considered in [BBB+18] – range proofs and circuit satisfiability.

Notation. In order to ensure readability and consistency with the original presentation of Bulletproofs, we briefly recall their notation here. Let $\mathbb{G}$ be a cyclic group of prime order p and $\mathbb{Z}_p^*$ denote $\mathbb{Z}_p \backslash \{0\}$. Let $g, h \in \mathbb{G}$ be generators of $\mathbb{G}$. Let $\mathbf{g} = (g_1, g_2, \dots, g_N), \mathbf{h} = (h_1, h_2, \dots, h_N) \in \mathbb{G}^N$ be vectors of generators, where N is some arbitrary size parameter (note that we use the size parameter N instead of n, used in [BBB+18] to avoid notational collisions with the rest of this work). Capital letters, *e.g.* V, A, S, T_1, T_2, denote Pedersen Commitments and Greek letters, *e.g.* $\alpha, \gamma, \rho, \tau \in \mathbb{Z}_p^*$, are blinding factors (in Bulletproofs, Greek letters denote elements of $\mathbb{Z}_p^*$ more generally). If $y \in \mathbb{Z}_p$, then $\mathbf{y}$ is a vector of elements in $\mathbb{Z}_p$, generally of size N. Additionally, let the vectors $\mathbf{y}^N = (1, y, y^2, \dots, y^{N-1}), \mathbf{1}^N = (1, 1, \dots, 1)$ and $\mathbf{2}^N = (1, 2, 2^2, \dots, 2^{N-1})$. Let $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^n a_i \cdot b_i$ be the inner product between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}^n$, and $\mathbf{a} \circ \mathbf{b} = (a_1 \cdot b_1, \dots, a_n \cdot b_n)$ be the Hadamard product between those two vectors (note that we used $*$ to denote Hadamard product when discussing Compressed Σ-protocols—we allow the sections to differ in their notation to allow for easier reference to the original works). Finally we denote multiexponentation by $\mathbf{g}^{\mathbf{x}} = \prod_i g_i^{x_i}$.

Manipulating a Set of Constraints. Bünz et al. [BBB+18] consider NP relations of the following form:

$$\mathcal{R} = \left\{ \begin{matrix} (g, h \in \mathbb{G}, \{V_i\}_{i \in [I]} \in \mathbb{G}^I, f; \{\gamma_i\}_{i \in [I]} \in \mathbb{Z}^I, \{v_i\}_{i \in [I]} \in \mathbb{Z}_p^I, \{w_j\}_{j \in [J]} \in \mathbb{Z}_p^J) : \\ V_i = h^{\gamma_i} g^{v_i}, \forall i \in [I] \wedge f(\{v_i\}_{i \in [I]}, \{w_j\}_{j \in [J]}) = 1 \end{matrix} \right\}$$

Here the function f can be re-imagined as a collection of constraints of the following form:

1. *Sum of inner products:* These are constaints of the form $\sum_q \langle \mathbf{a}_q, \mathbf{b}_q \rangle = c$, where each $\mathbf{a}_q, \mathbf{b}_q \in \mathbb{Z}_p^N, c \in \mathbb{Z}_p$ are some functions in v_i's and w_j's.
2. *Sum of hadamard products:* These are constaints of the form $\sum_q \mathbf{a}_q \circ \mathbf{b}_q = \mathbf{c}$, where each $\mathbf{a}_q, \mathbf{b}_q, \mathbf{c} \in \mathbb{Z}_p^N$ are some functions in v_i's and w_j's.

It is possible to reduce the second type of constraints to the first type. In particular, let $y \in \mathbb{Z}_p$ be a random challenge received from the verifier. Then for the second type of constraints it suffices for the prover to prove $\sum_q \langle \mathbf{a}_q, \mathbf{b}_q \circ \mathbf{y}^N \rangle = \langle \mathbf{c}, \mathbf{y}^N \rangle$.

Since all the constraints are now sums of inner products, we can combine them into a single sum of inner products by taking another random linear combination of all the constraints. Let $\sum_q \langle \mathbf{a}_{k,q}, \mathbf{b}_{k,q} \rangle = c_k$ be the set of k constraints and let $z \in \mathbb{Z}_p$ be a random challenge received from the verifier. Given this, it suffices for the verifier to simply prove $\sum_k z^{k-1} \cdot \sum_q \langle \mathbf{a}_{k,q}, \mathbf{b}_{k,q} \rangle = \sum_k z^{k-1} \cdot c_k$.

Next, Bünz et al. observe that this sum of inner products can often be re-written as a single inner product relation of the form $\langle \mathbf{a}, \mathbf{b} \rangle = c$, where c only depends on y, z and the v_i values. However, since the two vectors $\mathbf{a}$ and $\mathbf{b}$ invoved in this inner product reveal information about the witnesses $\{w_j\}_{j \in [J]}$, the prover cannot send them to the verifier in the clear.

Reducing to a "Privacy-Free" Inner Product Relation. We now recall the main ideas in Bulletproofs for reducing the above inner-product relation into a *privacy-free* inner product relation.

This problem is solved in Bulletproofs by introducing two blinding vectors $\mathbf{s}_L, \mathbf{s}_R \in \mathbb{Z}_p^N$ and defining two vector polynomials $l(X), r(X) \in \mathbb{Z}_p^N[X]$, where the coefficients in $l(X)$ depend on $\mathbf{a}$ and $\mathbf{s}_L$, while the coefficients in $r(X)$ depend on $\mathbf{b}$ and $\mathbf{s}_R$.

Let $l(X)$ and $r(X)$ be polynomials of degree d each. Let $t(X) = \langle l(X), r(X) \rangle = t_0 + t_1 \cdot X + \ldots + t_{2d} \cdot X^{2d}$ be a polynomial of degree $2d$ in $\mathbb{Z}_p[X]$. The polynomials $l(X)$ and $r(X)$ are chosen such that at least one of the coefficients T_e in polynomial $t(X)$ is independent of any of the w_j values and can be computed by the verifier. Let T_e be some deterministic function $p(v_1, \ldots, v_I, y, z)$.

The blinding terms $\mathbf{s}_L$ and $\mathbf{s}_R$ ensure that the prover can now publish $l(x)$ and $r(x)$ for one $x \in \mathbb{Z}_p^*$ (chosen randomly by the verifier) without revealing information about $\mathbf{a}$ and $\mathbf{b}$ (and in turn about any of the w_j witnesses). However, since the size of coefficients in $l(x)$ and $r(x)$ could potentially be dependent on the size of the relation function/witness, publishing $l(x), r(x)$ directly, will require communication that is linear in the size of the relation/witness. Therefore, instead of directly sending $l(x), r(x)$ to the verifier, Bünz et al. devise a logarithmic-sized argument system for proving correctness of inner-product relations of the following form

$$\mathcal{R}_{\text{innerprod}} = \left\{ \begin{matrix} (\mathbf{g}, \mathbf{h} \in \mathbb{G}^N, P \in \mathbb{G}, t(x) \in \mathbb{Z}_p; l(x), r(x) \in \mathbb{Z}_p^N) : \\ P = \mathbf{g}^{l(x)} \mathbf{h}^{r(x)} \wedge t(x) = \langle l(x), r(x) \rangle \end{matrix} \right\}$$

where the commitment P to $l(x)$ and $r(x)$ is fixed given all the previously sent messages to the verifier and can be derived by the verifier during the verification process. In particular, the following Theorem is proved in [BBB+18].

Theorem 6.1 (Inner-Product Argument). *There exists an argument $\Pi_{innerprod}$ for relation $\mathcal{R}_{innerprod}$ that has perfect completeness and statistical witness-extended-emulation and requires the prover to only send $2\log_2(N)$ group elements to the verifier.*

Although a significant portion of Bünz et al.'s contribution is in designing an efficient $\Pi_{\text{innerprod}}$, we note that understanding the details of this protocol is not important for understanding how to speed-stack Bulletproofs, as we will be able to reuse the transcript produced by $\Pi_{\text{innerprod}}$ in a blackbox way. Using this $\Pi_{\text{innerprod}}$, the prover can now prove that $t(x) = \langle l(x), r(x) \rangle$, without having to send $l(x), r(x)$.

Summarizing the Bulletproofs Protocol. To summarize the above discussion, we now present a high level sketch of the main steps involved in the Bulletproofs protocol.

- *Round 1:* The prover starts by committing to the w_j witnesses and the blinding terms $\mathbf{s}_L$ and $\mathbf{s}_R$ using pederson commitments. Let $S = h^{\rho}\mathbf{g}^{\mathbf{s}_L}\mathbf{h}^{\mathbf{s}_R}$ ans let $A_1, \dots, A_d$ be the remaining commitments to the witnesses computed using randomness $\alpha_1, \dots, \alpha_d$ (We note that each of the witnesses are not necessarily committed using a different commitment).
- *Round 2:* The verifier sends challenges y, z.
- *Round 3:* The prover computes pederson commitments $\{T_i = g^{t_i}h^{\tau_i}\}_{i\in\{0,\dots,2d\}\setminus e}$, where t_i is the i^{th} coefficient in $t(X)$ and τ_i is a random value in $\mathbb{Z}_p$. It sends these commitments to the verifier
- *Round 4:* The verifier responds with the challenge x.
- *Round 5:* The prover evaluates $l(X)$ and $r(X)$ on the challenge point x to obtain $\mathbf{l}$ and $\mathbf{r}$ respectively. Let $\hat{t} = \langle \mathbf{l}, \mathbf{r}\rangle$. Compute $\tau_x = \sum_{i\in[2d, i\neq e]} \tau_i x^i + x^e p(\gamma_1, \dots, \gamma_I, y, z)$. Let $\mu = \sum_{i\in[d-1]} \alpha_i x^i + \rho x^d$. The prover sends $\tau_x, \mu, \hat{t}$ to the verifier. Given this information, the verifier is able to deduce a commitment $P = \mathbf{g}^{\mathbf{l}}\mathbf{h}^{\mathbf{r}}$. Finally, the prover proves $\hat{t} = \langle \mathbf{l}, \mathbf{r}\rangle$ to the verifier using $\Pi_{\text{innerprod}}$.

Bottleneck in Speed-Stacking Bulletproofs. We start by showing that Bulletproofs are stackable. We proceed in two steps: (1) we begin by showing how to stack the communication *inefficient* version of Bulletproofs in which the prover simply sends $\mathbf{l}$ and $\mathbf{r}$ directly, then (2) observe that if all clauses share the same values of $\mathbf{l}$ and $\mathbf{r}$, then only a single instance of $\Pi_{\text{innerprod}}$ is neccerary to convince the verifier.

In order to stack the communication inefficient version of Bulletproofs, we observe that all but two messages – namely S and any one of $\{T_i\}_{i\neq e}$ commitments – sent by the prover in Bulletproofs is recyclable. To see this, we observe that for fixed values of $\{V_i\}_{i\in[I]}$, function f and transcripts of the full protocol excluding S and one of the $\{T_i\}_{i\neq e}$ values (say T_j), it is possible to deterministically find the values of S and T_j that will satisfy the equation. This observation gives us a blueprint for designing the two simulators $\mathcal{S}_{\text{RAND}}$ and $\mathcal{S}_{\text{DET}}$: $\mathcal{S}_{\text{RAND}}$ will generate a transcript of appropriately distributed values $A_1, \dots, A_d, \{T_i\}_{i\notin\{e,j\}}, \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r}$ and $\mathcal{S}_{\text{DET}}$ completes the transcript by computing S (which can be done independently of the statement) and T_j (we arbitrarily choose to compute $\{T_i\}_{i\notin\{e,j\}}$ using $\mathcal{S}_{\text{RAND}}$ and T_j using $\mathcal{S}_{\text{DET}}$). We present a formal blueprint of these simulators in Figure 9.

Given the above observation, since $\hat{t}, \mathbf{l}, \mathbf{r}$ can be recycled, it is easy to see that the transcipt of $\Pi_{\text{innerprod}}$ is also recyclable. Using this approach, the additional cost of simulating transcripts for the inactive clauses depends on the time required to compute S and T_j, given $M^{\text{BULLET}}_{\text{RAND}}$. We observe that computing S requires multi-exponentiations dependent of the size of the function f, while computing T_j requires multi-exponentiations dependent of the length of the statement, i.e., $\{V_i\}_{i\in I}$. While the transcript of $\Pi_{\text{innerprod}}$ can be reused, the cost of computing $\Pi_{\text{innerprod}}$ is comparable to the cost of computing S in $\mathcal{S}_{\text{DET}}$. As a result, unfortunately, we do not get any significant computational savings (barring small constant factor improvements) when trying to stack Bulletproofs for disjunctions. However, when considering set-membership proofs, most of the heavy computation in S can also be re-used across all clauses. Therefore, the additional work required to simulate the inactive clauses only majorly depends on the computation of T. In this case, we do get some computational savings using our approach. However, as discussed in the introduction, this is only interesting for the case of Range proofs. For the case of circuit-satisfiability, this approach is unlikely to be any more efficient than the naïve approach used for computing set-membership proofs.

6.2 Speed-Stacking Range Proofs Version of Bulletproofs

Using the high-level ideas from the previous subsection, we now show how to speed-stack the first kind of relation considered in [BBB+18], namely range proofs. Range proofs are relations, where a prover wishes to convince the verifier that a committed value lies within a specific range. In particular, Bünz et al. consider relations of the form

$$\mathcal{R}_{\text{range}} = \left\{(g, h \in \mathbb{G}, V \in \mathbb{G}, N > 0; v, \gamma \in \mathbb{Z}_p) : V = h^{\gamma} g^{v} \wedge v \in [0, 2^N - 1]\right\}.$$

```
$M^{\text{BULLET}}_{\text{RAND}} \leftarrow \mathcal{S}^{\text{BULLET}}_{\text{RAND}}(1^\lambda, C = (x, y, z))$
$\mu, \tau_x \xleftarrow{\$} \mathbb{Z}_p$
$A_1, \ldots, A_d \xleftarrow{\$} \mathbb{G}$
$\mathbf{l}, \mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^N$
$\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p$
$\{T_i\}_{i \neq \{j,e\}} \xleftarrow{\$} \mathbb{G}$
return $A_1, \ldots, A_d, \{T_i\}_{i \neq \{j,e\}}, \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r}$

$T_2 \leftarrow \mathcal{S}^{\text{BULLET}}_{\text{DET}}(g, h, \{V_i\}_{i \in I}, (x, y, z), M^{\text{BULLET}}_{\text{RAND}} = (\{A_i\}_{i \in [d]}, \{T_i\}_{i \neq \{j,e\}}, \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r})$
Compute $S$ and $T_j$, such that the verification check is satisfied
return $S, T_j$
```

Figure 9: General Blue-Print for Simulators in (communication-inefficient version of) Bulletproofs.

Overview. We give a short overview of how this relation can be reformulated as a set of inner product and hadamard product constraints and how the polynomials $l(X)$ and $r(X)$ are defined. Let $\mathbf{a}_L = (a_1, \ldots, a_n) \in \{0,1\}^N$ be the binary representation of v, i.e., $\langle \mathbf{a}_L, \mathbf{2}^N \rangle = v$. Let $\mathbf{a}_R$ be such that $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{0}^N$ and $\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^N$. Intuitively, these constraints guarantee that $\mathbf{a}_L$ is a valid bit decomposition of v, such that each element of $\mathbf{a}_L$ is $\{0,1\}$. Note that if v can be represented with a N bit binary vector, then we can concluded that $v \in [0, 2^{N-1}]$. Using aforementioned ideas, these constraints can be reduced to the following single inner product constraint:

$$\left\langle \mathbf{a}_L - z \cdot \mathbf{1}^N, \mathbf{y}^N \circ (\mathbf{a}_R + z \cdot \mathbf{1}^N) + z^2 \cdot \mathbf{2}^N \right\rangle = z^2 \cdot v + \delta(y, z)$$

where

$$\delta(y, z) = (z - z^2) \cdot \langle \mathbf{1}^N, \mathbf{y}^N \rangle - z^3 \langle \mathbf{1}^N, \mathbf{2}^N \rangle \in \mathbb{Z}_p. \tag{1}$$

Finally, given the blinding terms $\mathbf{s}_L, \mathbf{s}_R \in \mathbb{Z}_p^N$, vector polynomials $l(X), r(X) \in \mathbb{Z}_p^N[X]$ can be defined as follows:

$$\begin{aligned} l(X) &= \mathbf{a}_L - z \cdot \mathbf{1}^N + \mathbf{s}_L \cdot X \\ r(X) &= \mathbf{y}^N \circ (\mathbf{a}_R + z \cdot \mathbf{1}^N + \mathbf{s}_R \cdot X) + z^2 \cdot \mathbf{2}^N \end{aligned}$$

Given $l(X)$ and $r(X)$, the rest of the proof proceeds exactly as described in the previous subsection. We now state the main Theorem from [BBB+18]:

Theorem 6.2 (Range Proof)**.** *There exists a protocol Π_{range} for relation $\mathcal{R}_{range}$ that has perfect completeness, perfect special honest verifier zero-knowledge, and computational witness extended emulation and requires the prover to send $2 \cdot \lceil \log_2(N) \rceil + 4$ group elements and 5 elements in $\mathbb{Z}_p$.*

For the sake of completeness, we give a formal description of protocol Π_{range} in Figure 10.

Range Proofs Version of Bulletproofs are Stackable. We now show how to apply our compiler in Theorem 3.5 to Bulletproofs. In particular, we consider relations of the form:

$$\mathcal{R}_{\text{dis-range}} = \begin{Bmatrix} (g, h \in \mathbb{G}, \{V_i\}_{i \in [\ell]}, N > 0; \alpha \in [\ell], v_\alpha, \gamma \in \mathbb{Z}_p) : \\ V_\alpha = h^\gamma g^{v_\alpha} \wedge v_\alpha \in [0, 2^N - 1] \end{Bmatrix}$$

Note that the above statement in less generic than the ones considered in previous sections. Here we consider the same range across all branches, and therefore this is *set memebership*. As discussed in the technical overview, the benefits of applying the same techniques to true disjunction of the circuit satisfiability

BulletProofs Range Proofs. PoK$\{(v, \gamma) : \mathbf{g}, \mathbf{h}, \in \mathbb{G}^N, V = h^\gamma g^v \wedge v \in [0, 2^N - 1]\}$

Prover(v, γ)		Verifier
$\mathbf{a}_L \in \{0,1\}^N$ s.t. $\langle \mathbf{a}_L, \mathbf{2}^N \rangle = v$		
$\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^N \in \mathbb{Z}_p^N, \alpha, \rho \xleftarrow{\$} \mathbb{Z}_p, \mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_p^N$		
$A = h^\alpha \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}, S = h^\rho \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R} \in \mathbb{G}$	$\xrightarrow{\quad A, S \quad}$	
$t_0 = vz^2 + \delta(y, z)$ (see 1)	$\xleftarrow{\quad y, z \quad}$	$y, z \xleftarrow{\$} \mathbb{Z}_q^*$
$\tau_1, \tau_2 \xleftarrow{\$} \mathbb{Z}_p$		
$T_1 = g^{t_1} h^{\tau_1}, T_2 = g^{t_2} h^{\tau_2}$	$\xrightarrow{\quad T_1, T_2 \quad}$	
$\mathbf{l} = \mathbf{a}_L - z \cdot \mathbf{1}^N + \mathbf{s}_L \cdot x \in \mathbb{Z}_p^N$	$\xleftarrow{\quad x \quad}$	$x \xleftarrow{\$} \mathbb{Z}_q^*$
$\mathbf{r} = \mathbf{y}^N \circ (\mathbf{a}_R + z \cdot \mathbf{1}^N + \mathbf{s}_R \cdot x) + z^2 \cdot \mathbf{2}^N \in \mathbb{Z}_p^N$		
$\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p$		
$\tau_x = \tau_2 \cdot x^2 + \tau_1 \cdot x + z^2 \cdot \gamma$		
$\mu = \alpha + \rho \cdot x \in \mathbb{Z}_p$	$\xrightarrow{\tau_x, \mu, \hat{t}, \Pi_{\text{innerprod}}(\hat{t}, \mathbf{l}, \mathbf{r})}$	$h_i' = h_i^{(y^{-i+1})} \in \mathbb{G}, \ \forall i \in [1, n]$
		$\mathbf{h}' = (h_1', h_2', h_3', \ldots, h_n')$
		$g^{\hat{t}} h^{\tau_x} \overset{?}{=} V^{z^2} \cdot g^{\delta(y,z)} \cdot T_1^x \cdot T_2^{x^2}$
		$P = A \cdot S^x \cdot \mathbf{g}^{-z} \cdot (\mathbf{h}')^{z \cdot \mathbf{y}^N + z^2 \mathbf{2}^N}$
		$P \overset{?}{=} h^\mu \cdot \mathbf{g}^{\mathbf{l}} \cdot (\mathbf{h}')^{\mathbf{r}}$
		Verify $(P, \Pi_{\text{innerprod}}(\hat{t}, \mathbf{l}, \mathbf{r}))$

Figure 10: Bulletproofs Range Proof Protocol Description.

version of bulletproofs is marginal, so we omit the construction. Additionally, we show that speed stacked Compressed Σ-protocols support true disjunctive statements.

We now prove the following theorem, showing that the range proofs construction in Bulletproofs is stackable.

Theorem 6.3. *The range proof construction in Bulletproofs [BBB+18] (Figure 10), denoted as Π_{range}, is stackable.*

Proof. **Random Simulation $\mathcal{S}_{\text{RAND}}$.** We start by showing that there exists a statement-independent simulator $\mathcal{S}_{\text{RAND}}$ that produces the set of messages $M_{\text{RAND}}^{\text{BULLET}} = (A, T_1, \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r})$ such that those messages are indistinguishable from an honest execution of the protocol. The simulator can be found on the left side of Figure 11.To prove security, we show that the set of messages generated by $\mathcal{S}_{\text{RAND}}$ and the set of messages $M_{\text{RAND}}^{\text{BULLET}}$ generated from an honest execution of the protocol are indistinguishable. For this, we consider the following hybrid distributions:

- **Hybrid H_0:** This is the distribution of messages $M_{\text{RAND}}^{\text{BULLET}}$ computed in an honest execution of the protocol.
- **Hybrid H_1:** First lets consider a hybrid H_1, where the messages $M_{\text{RAND}}^{\text{BULLET}}$ are computed exactly as in H_0, except that T_1 and τ_x are both sampled at random. It easy to see that since T_1 is a hiding

```
$M^{\text{BULLET}}_{\text{RAND}} \leftarrow \mathcal{S}^{\text{BULLET}}_{\text{RAND}}(1^\lambda, C = (x, y, z))$
1 :   $\alpha, \mu, \tau_x \xleftarrow{\$} \mathbb{Z}_p$
2 :   $\mathbf{a}_L, \mathbf{a}_R \xleftarrow{\$} \mathbb{Z}_p^N$
3 :   $A = h^\alpha \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R} \in \mathbb{G}$
4 :   $\mathbf{l}, \mathbf{r} \xleftarrow{\$} \mathbb{Z}_p^N$
5 :   $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle \in \mathbb{Z}_p$
6 :   $T_1 \xleftarrow{\$} \mathbb{G}$
7 :   return $A, T_1, \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r}$
```

```
$T_2 \leftarrow \mathcal{S}^{\text{BULLET}}_{\text{DET}}(g, h, V, (x, y, z), M^{\text{BULLET}}_{\text{RAND}} = (A, S, T_1, \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r}))$
1 :   $S = h^\mu \cdot A^{-1} \cdot \mathbf{g}^{\mathbf{l}+z\cdot\mathbf{1}^N} \cdot (\mathbf{h}')^{\mathbf{r}-z\cdot\mathbf{y}^N - z^2\cdot\mathbf{2}^N} \in \mathbb{G}$
2 :   $\delta(y, z) = (z - z^2) \cdot \langle \mathbf{1}^N, \mathbf{y}^N \rangle - z^3 \langle \mathbf{1}^N, \mathbf{2}^N \rangle \in \mathbb{Z}_p$
3 :   $T_2 = \left( \frac{g^{\hat{\tau}} \cdot h^{\tau_x}}{V^{z^2} \cdot g^{\delta(y,z)} \cdot T_1^x} \right)^{-x^2}$
4 :   return $T_2$
```

Figure 11: Simulators for Bulletproofs.

commitment, the set of $M^{\text{BULLET}}_{\text{RAND}}$ generated this way is indistinguishable from an honestly generated set of messages $M^{\text{BULLET}}_{\text{RAND}}$.

- **Hybrid H_2:** Next, we consider a distribution where the messages $M^{\text{BULLET}}_{\text{RAND}}$ are computed exactly as in $\mathcal{S}_{\text{RAND}}$, except that in Step 2, instead of sampling $\mathbf{a}_L$ and $\mathbf{a}_R$ randomly, they are computed as follows: $\mathbf{a}_L \in \{0, 1\}^N$ s.t. $\langle \mathbf{a}_L, \mathbf{2}^N \rangle = v$ and $\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^N$. It is easy to see that hybrids H_1 and H_2 are identically distributed. Indeed, the only difference between the two is that in H_1, we first sample random $\mathbf{s}_L, \mathbf{s}_R$ and then compute the corresponding $\mathbf{l}, \mathbf{r}$. Whereas, in H_2 we first sample random $\mathbf{l}, \mathbf{r}, \mu$ and then compute the corresponding S directly.

- **Hybrid H_3:** This is identical to the messages simulated by $\mathcal{S}_{\text{RAND}}$. The only difference between H_2 and H_3 is in the way $\mathbf{a}_L, \mathbf{a}_R$ and hence A is generated. Indistinguishability between these distributions, follows from the hiding property of Pederson vector commitments.

Deterministic Simulation $\mathcal{S}^{\text{BULLET}}_{\text{DET}}$. We now construction a *deterministic* simulator $\mathcal{S}^{\text{BULLET}}_{\text{DET}}$ that takes in a set of messages $M^{\text{BULLET}}_{\text{RAND}}$ and completes the transcript. The simulator can be found on the right side of Figure 11. It is easy to see that $\mathcal{S}^{\text{BULLET}}_{\text{DET}}$ produces the exact same transcript that an honest execution would produce. Moreover, the computations performed by this simulator are all deterministic.

Optimizing $\mathcal{S}^{\text{BULLET}}_{\text{DET}}$ By Passing State. We note that the runtime of $\mathcal{S}^{\text{BULLET}}_{\text{DET}}$ can be further improved by having the randomized simulator compute *most* of T_2. Specifically, compute $\mathsf{st} = \left(\frac{g^{\hat{\tau}} \cdot h^{\tau_x}}{g^{\delta(y,z)} \cdot T_1^x} \right)^{-x^2}$ and output st as part of the transcript. $\mathcal{S}^{\text{BULLET}}_{\text{DET}}$ then computes $\mathsf{st}/V^{z^2 - x^2}$.

□

Combining Theorems 3.5 and 6.3, we get the following Corollary.

Corollary 6.4. *Let $\Pi_{\textit{speed-range}}$ be the output of the compiler in Figure 2 recursively applied to $\Pi_{\textit{range}}$ from Bulletproofs using $\mathcal{S}_{\text{RAND}}$, $\mathcal{S}^{\text{BULLET}}_{\text{DET-I}}$, and $\mathcal{S}^{\text{BULLET}}_{\text{DET-D}}$ as defined in the proof of Theorem 6.3. Then $\Pi_{\textit{speed-bullet}}$ is a stackable ZK-IP for $\mathcal{R}_{dis-range}$ with logarithmic communication complexity, and prover computational complexity $O(\mathsf{Time}(\Pi_{\textit{range}}) + \ell \cdot \mathsf{Time}(\mathcal{S}^{\text{BULLET}}_{\text{DET}}))$.*

Proof. The only remaining observation necessary is that the prover can replace the messages $\mathbf{l}$ and $\mathbf{r}$ produced by $\mathcal{S}^{\text{BULLET}}_{\text{RAND}}$ with the protocol $\Pi_{\text{innerprod}}$ from [BBB+18] to achieve logarithmic communication complexity. □

Efficiency of Speed-Stacked Bulletproofs. Concretely, the prover's computational cost for running the compiled protocol is $\mathsf{Time}(\Pi_{\text{compressed}}) + \ell \cdot \mathsf{Time}(\mathcal{S}^{\text{BULLET}}_{\text{DET}}) + \mathsf{Time}(\mathsf{Gen}) + \mathsf{Time}(\mathsf{EquivCom}) + \mathsf{Time}(\mathsf{Equiv})$. It is easy to see that $\mathcal{S}^{\text{BULLET}}_{\text{DET}}$ only requires a constant number of group exponentiations and no multi-exponentiations. In the commitment scheme proposed by Goel et al. [GGHAK22], both key generation and committing require ℓ exponentations and group operations, while equivocation requires only field operations. Thus the overhead (when counting group operations) introduced from running a disjunction with ℓ clauses is only ℓ group exponentiations and ℓ group multiplications, plus the cost of the commitment scheme.

Acknowledgements

The first author was supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. This work was done in part while the first author was a student at Johns Hopkins University and while they were visiting University of California, Berkeley. The second author is funded by Concordium Blockchain Research Center, Aarhus University, Denmark. The third author is supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project and is supported by DARPA under Agreement No. HR00112020021. This work was completed in part while the fourth author was at Boston University and was supported by DARPA under Agreement No. HR00112020023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

[AC20] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.

[ACF21] Thomas Attema, Ronald Cramer, and Serge Fehr. Compressing proofs of k-out-of-n partial knowledge. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 65–91, Virtual Event, August 2021. Springer, Heidelberg.

[ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 549–579, Virtual Event, August 2021. Springer, Heidelberg.

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 415–432. Springer, Heidelberg, December 2002.

[AR20] Shashank Agrawal and Srinivasan Raghuraman. KVaC: Key-Value Commitments for blockchains and beyond. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 839–869. Springer, Heidelberg, December 2020.

[BBB$^+$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*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BCC$^+$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 Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

[BCG$^+$14] 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*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *23rd ACM STOC*, pages 21–31. ACM Press, May 1991.

[BGKS20] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 5:1–5:32. LIPIcs, January 2020.

[BM74] Allan Borodin and R. Moenck. Fast modular transforms. *J. Comput. Syst. Sci.*, 8(3):366–386, 1974.

[BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 2021. Springer, Heidelberg.

[CDP12] Ronald Cramer, Ivan Damgård, and Valerio Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. In Adam Smith, editor, *ICITS 12*, volume 7412 of *LNCS*, pages 62–79. Springer, Heidelberg, August 2012.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

[CPS$^+$16] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Online/offline OR composition of sigma protocols. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 63–92. Springer, Heidelberg, May 2016.

[GGHAK22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In *EUROCRYPT 2022, Part II*, LNCS, pages 458–487. Springer, Heidelberg, June 2022.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 253–280. Springer, Heidelberg, April 2015.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986.

[GMY03] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 177–194. Springer, Heidelberg, May 2003.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[HK20a] David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 763–792. Springer, Heidelberg, August 2020.

[HK20b] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 569–598. Springer, Heidelberg, May 2020.

[HK21] David Heath and Vladimir Kolesnikov. $\mathsf{LogStack}$: Stacked garbling with $O(b \log b)$ computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 3–32. Springer, Heidelberg, October 2021.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

[Kil94] Joe Kilian. On the complexity of bounded-interaction and noninteractive zero-knowledge proofs. In *35th FOCS*, pages 466–477. IEEE Computer Society Press, November 1994.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

[Kol18] Vladimir Kolesnikov. Free IF: How to omit inactive branches and implement S-universal garbled circuit (almost) for free. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 34–58. Springer, Heidelberg, December 2018.

[LMS16] Helger Lipmaa, Payman Mohassel, and Saeed Sadeghian. Valiant's universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017, 2016. https://eprint.iacr.org/2016/017.

[LYZ+21] Hanlin Liu, Yu Yu, Shuoyao Zhao, Jiang Zhang, Wenling Liu, and Zhenkai Hu. Pushing the limits of valiant's universal circuits: Simpler, tighter and more compact. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 365–394, Virtual Event, August 2021. Springer, Heidelberg.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[se19] swisspost evoting. E-voting system 2019. https://gitlab.com/swisspost-evoting/e-voting-system-2019, 2019.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, page 196–203, New York, NY, USA, 1976. Association for Computing Machinery.

[Zav20] Greg Zaverucha. The picnic signature algorithm. Technical report, 2020. https://github.com/microsoft/Picnic/raw/master/spec/spec-v3.0.pdf.

A Stackable holographic lincheck

In this section we describe a stackable variant of the holographic lincheck protocol of [COS20]. Before giving the formal construction, we give a high-level overview of the main idea.

Recall that the original holographic lincheck protocol for a matrix M fails to be efficiently stackable because the prover sends the evaluation of a polynomial $\hat{u}_{\alpha,M}$. Computing a single point in this evaluation takes time linear in the number $\|M\|$ of nonzero entries in M, and so the instance-dependent part of the simulator must run in time $\Omega(\|M\|)$.

The essence of our solution to this problem is in the following observation. Let $R \in \mathbb{F}^{k \times k}$ be a uniformly random matrix, and let $[M|R]$ denote the block matrix

$$\begin{pmatrix} M & 0 \\ 0 & R \end{pmatrix} \in \mathbb{F}^{(n+k)\times(n+k)} .$$

Note that $Mx = y$ if and only if there exist $z, z' \in \mathbb{F}^k$ such that $[M|R](x|z) = (y|z')$, where $(u|v)$ denotes the concatenation of vectors u, v. Hence we can solve the lincheck problem for M by solving the lincheck problem for $[M|R]$, ignoring the last k coordinates. The reduction to sumcheck allows this directly.

The advantage of the matrix $[M|R]$ over M itself is that for random R, $\hat{u}_{\alpha,[M|R]}$ is a k-wise independent function; that is, for any k distinct points $\beta_1, \ldots, \beta_k \in \mathbb{F}$, $(\hat{u}_{\alpha,[M|R]}(\beta_i))_{i=1}^{k}$ is distributed as a uniformly random vector in $\mathbb{F}^k$. Hence provided that the verifier makes at most k queries to $\hat{u}_{\alpha,[M|R]}$, the answers to these queries can be simulated with uniform randomness.

Most of the technical work in this construction is in enabling the prover to commit to the matrix R in a way that allows it to verifiably open to evaluations of $\hat{u}_{\alpha,[M|R]}$. This is achieved by having the prover send a univariate polynomial val_R encoding the entries of R; this is similar to the encoding used for M except that the former is dense while the latter is sparse. We design a protocol that combines the encodings of M and R to achieve our goal.

Appendix A.1 gives preliminaries for readers unfamiliar with [COS20]. Appendix A.2 describes the stackable lincheck protocol in detail as an RS-IOPP. Finally, Appendix A.3 proves soundness and stackability.

A.1 Preliminaries

Notation. We denote polynomials with "hats", e.g. $\hat{f}, \hat{g}$, the evaluation of a polynomial $\hat{f}$ on a domain S by $\hat{f}|_S$ and Reed–Solomon codewords corresponding to those polynomials by plain letters, so $f = \hat{f}|_L$. In this formalism, the lincheck problem is the problem of determining, given oracles f_1, f_2, whether $\hat{f}_1|_H = M \cdot \hat{f}_2|_H$ for $H \subseteq \mathbb{F}$ and $M \in \mathbb{F}^{H \times H}$. The bivariate low-degree extension of a matrix $M \in \mathbb{F}^{H \times H}$ is denoted by $\hat{M}(X, Y)$, and is defined to be the unique polynomial of degree $|H| - 1$ in each variable such that $\hat{M}(a, b) = M_{ab}$ for $a, b \in H$.

In the construction below, $H, K, H_0 \subseteq \mathbb{F}$. We assume that $\mathsf{b} = |H_0| \leq \sqrt{|K|}$ and that H, H_0 are disjoint. Let $S := H \cup H_0$. Given a matrix $M \in \mathbb{F}^{H \times H}$ and a matrix $C \in \mathbb{F}^{H_0 \times H_0}$, we define $[M|C] \in \mathbb{F}^{S \times S}$ to be the block matrix $\begin{pmatrix} M & 0 \\ 0 & C \end{pmatrix}$. Fix maps $\Phi_1, \Phi_2 \colon K \to S$. We require that the map $\Phi \colon K \to S \times S$ given by $k \mapsto (\Phi_1(k), \Phi_2(k))$ is bijective when viewed as a map $\Phi^{-1}(H_0 \times H_0) \to H_0 \times H_0$. We write $\hat{\Phi}_1, \hat{\Phi}_2$ for the low-degree extensions of Φ_1, Φ_2. These have degree at most $|K|$.[13] We will assume that the prover and the verifier can evaluate $\hat{\Phi}_1, \hat{\Phi}_2$ at points in L in time polylogarithmic in $|K|$, e.g. by precomputation or by the algebraic structure of K and H_0.

Sparse representations of matrices. As in [COS20], our holographic lincheck protocol uses sparse representations of matrices, following the definition below.

Definition A.1. *Let $H, K \subseteq \mathbb{F}$. A* sparse representation *of a matrix is a function $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ that is injective when its output is restricted to $H \times H$. The matrix $M \in \mathbb{F}^{H \times H}$ is obtained from $\langle M \rangle$ by setting, for $a, b \in H$, $M_{a,b} := \gamma$ if there exists $k \in K$ such that $\langle M \rangle(k) = (a, b, \gamma)$ and $M_{a,b} := 0$ otherwise.*

Note that a matrix $M \in \mathbb{F}^{H \times H}$ has many possible sparse representations. In particular, we may choose any large enough K and any injection from K to $H \times H$ that "covers" the nonzero entries of M.

Given a sparse representation $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$, define $\hat{\mathsf{row}}_{\langle M \rangle}, \hat{\mathsf{col}}_{\langle M \rangle} \colon K \to H, \hat{\mathsf{val}}_{\langle M \rangle} \in \mathbb{F}[X]$ to be the unique polynomials of minimal degree such that for each $k \in K$, letting $(a, b, \alpha) := \langle M \rangle(k)$,

$$\hat{\mathsf{row}}_{\langle M \rangle}(k) := a \ , \quad \hat{\mathsf{col}}_{\langle M \rangle}(k) := b \ , \quad \hat{\mathsf{val}}_{\langle M \rangle}(k) := \frac{\alpha}{u_H(a, a) \cdot u_H(b, b)} \ .$$

Special polynomials. For a set $S \subseteq \mathbb{F}$, we denote by $v_S(X)$ the "vanishing polynomial" of S, the unique non-zero monic polynomial of degree at most $|S|$ that is zero on S. We denote by u_S the bivariate polynomial

$$u_S := \frac{v_S(X) - v_S(Y)}{X - Y}$$

of individual degree $|S| - 1$.

For a matrix $M \in \mathbb{F}^{S \times S}$, we define the polynomial $u_M(X, Y)$ to be the low-degree extension (in X) of the vector of polynomials $\mathbf{u}_S^\top M \in \mathbb{F}[Y]^S$, where $\mathbf{u}_S := (u_S(a, Y))_{a \in S} \in \mathbb{F}[Y]^S$. We recall the following useful fact about u_M from [COS20].

[13] If K and H_0 are algebraically related then this can be reduced.

Claim A.2. *For any matrix $M \in \mathbb{F}^{S \times S}$, let $M^* \in \mathbb{F}^{S \times S}$ be the matrix given by $M^*_{a,b} := M_{b,a} \cdot u_S(b,b)$ for all $a, b \in S$. Then*

$$u_M(X,Y) \equiv \widehat{M^*}(X,Y) \ .$$

It will later be useful to observe that $[M|C]^* = [M^*|C^*]$ for all M, C.

A.2 Construction

Construction A.3 (stackable holographic lincheck). *The indexer $\mathbf{I}$ receives as input an index $\mathbb{i} = (\mathbb{F}, L, H, K, d, \langle M \rangle)$, computes a sparse representation $\langle M^* \rangle$ of the matrix M^*, computes $\hat{\mathsf{row}}_{\langle M \rangle}, \hat{\mathsf{col}}_{\langle M \rangle}, \hat{\mathsf{val}}_{\langle M \rangle}$ as described above, and then outputs their evaluations $\mathsf{row}_{\langle M \rangle}, \mathsf{col}_{\langle M \rangle}, \mathsf{val}_{\langle M \rangle} \in \mathrm{RS}[L, |K|-1]$.*

Subsequently, given an instance $\mathbb{x} = 1^{\log |K|}$ and witness $\mathbb{w} = (f_1, f_2)$, the honest prover $\mathbf{P}$ receives as input $(\mathbb{i}, \mathbb{x})$, the honest verifier $\mathbf{V}$ receives as input $\mathbb{x}$ and oracle access to $\mathbf{I}(\mathbb{i})$, and they engage in the following protocol.

(P1) $\mathbf{P}$ *chooses a matrix $C \in \mathbb{F}^{H_0 \times H_0}$ uniformly at random and sends $c \in \mathrm{RS}[L, |K| + \mathsf{b} - 1]$ to the verifier, where $\hat{c}$ is a uniformly random polynomial of degree $|K| + \mathsf{b} - 1$ such that for all $k \in K$:*

$$\hat{c}(k) = \begin{cases} \frac{C^*_{\Phi_1(k),\Phi_2(k)}}{u_S(\Phi_1(k),\Phi_1(k)) u_S(\Phi_2(k),\Phi_2(k))} & \text{if } \Phi(k) \in H_0 \times H_0\text{, and} \\ 0 & \text{otherwise.} \end{cases}$$

(V1) $\mathbf{V}$ *sends $\alpha \in \mathbb{F} \setminus H$ chosen uniformly at random.*

(P2) $\mathbf{P}$ *sends the evaluation $t \in \mathrm{RS}[L, |H| + |H_0| - 1]$ of the polynomial $\hat{t}(X) := u_{[M|C]}(X, \alpha) = [\widehat{M^*|C^*}](X, \alpha)$.*

(P3,V3,P4) $\mathbf{P}, \mathbf{V}$ *engage in the zero knowledge sumcheck protocol to show that*

$$\sum_{b \in H} u_H(b, \alpha) \hat{f}_1(b) - \hat{t}(b) \hat{f}_2(b) = 0 \ .$$

(V4) $\mathbf{V}$ *sends $\beta \in \mathbb{F} \setminus H$ uniformly at random.*

(P5) $\mathbf{P}$ *sends the field element $\gamma := u_{[M|C]}(\beta, \alpha) = \hat{t}(\beta)$, and $\mathbf{V}$ outputs the boundary constraint "$\hat{t}(\beta) = \gamma$".*

(P6) $\mathbf{P}$ *sends a random $f \in \mathrm{RS}[L, |K| + \mathsf{b}]$ such that for all $k \in K$,*

$$\hat{f}(k) = \frac{v_S(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M \rangle}(k))} \cdot \frac{v_S(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M \rangle}(k))} \cdot \hat{\mathsf{val}}_{\langle M \rangle}(k) + \frac{v_S(\alpha)}{(\alpha - \hat{\Phi}_1(k))} \cdot \frac{v_S(\beta)}{(\beta - \hat{\Phi}_2(k))} \cdot \hat{c}(k) \ .$$

$\mathbf{V}$ *outputs the constraint "$\deg(e) \leq 2|K| + \deg(\hat{\Phi}_1) + \deg(\hat{\Phi}_2) + \mathsf{b}$" where:*

$$\begin{aligned} e = \frac{1}{v_K} \cdot \Big(& (\alpha - \hat{\mathsf{row}}_{\langle M \rangle})(\beta - \hat{\mathsf{col}}_{\langle M \rangle})(\alpha - \hat{\Phi}_1)(\beta - \hat{\Phi}_2)\hat{f} \\ & - v_S(\alpha) \cdot v_S(\beta) \cdot \big((\alpha - \hat{\Phi}_1)(\beta - \hat{\Phi}_2)\hat{\mathsf{val}}_{\langle M \rangle} + (\alpha - \hat{\mathsf{row}}_{\langle M \rangle})(\beta - \hat{\mathsf{col}}_{\langle M \rangle})\hat{c} \big) \Big) \end{aligned}$$

(P7,V7,P8) $\mathbf{P}$ *and* $\mathbf{V}$ *engage in the zero knowledge sumcheck protocol to show that $\sum_{k \in K} \hat{f}(k) = \gamma$.*

A.3 Proof of correctness

We first state two necessary technical claims.

Claim A.4. *Let $\langle M \rangle$ be a sparse representation of $M \in \mathbb{F}^{K \times K}$, and let $C \in \mathbb{F}^{H_0 \times H_0}$. Then*

$$\begin{aligned} [\widehat{M|C}](X,Y) \equiv \sum_{k \in K} & \frac{v_S(X)}{(X - \hat{\mathsf{row}}_{\langle M \rangle}(k))} \cdot \frac{v_S(Y)}{(Y - \hat{\mathsf{col}}_{\langle M \rangle}(k))} \cdot \hat{\mathsf{val}}_{\langle M \rangle}(k) \\ & + \frac{v_S(X)}{(X - \hat{\Phi}_1(k))} \cdot \frac{v_S(Y)}{(Y - \hat{\Phi}_2(k))} \cdot \hat{c}(k) \end{aligned}$$

where $c \in \mathbb{F}^K$ is given by $c_k := \frac{C_{\Phi_1(k),\Phi_2(k)}}{u_S(\Phi_1(k),\Phi_1(k)) u_S(\Phi_2(k),\Phi_2(k))}$ for $k \in K$ such that $\Phi_1(k), \Phi_2(k) \in H_0$, and $c_k := 0$ otherwise.

The proof is very similar to the proof of [COS20, Claim 6.4], and so we omit it.

Claim A.5. *For uniformly random matrix $C \in \mathbb{F}^{H_0 \times H_0}$, and for all $(\alpha_1, \beta_1) \neq \ldots \neq (\alpha_{n_0}, \beta_{n_0}) \in (\mathbb{F} \setminus H)^2$ where $n_0 := |H_0|$, writing $C' := [0|C]$,*

$$\left(\hat{C}'(\alpha_1, \beta_1), \ldots, \hat{C}'(\alpha_{n_0}, \beta_{n_0})\right) \sim U(\mathbb{F}^{n_0}) .$$

Proof. It suffices to show that, for all $\gamma_1, \ldots, \gamma_{n_0} \in \mathbb{F}$ not all zero, there exists a matrix C such that

$$\sum_{i=1}^{n_0} \gamma_i \hat{C}'(\alpha_i, \beta_i) \neq 0 \ .$$

Let $A := \{\alpha_1, \ldots, \alpha_{n_0}\}$ and $B := \{\beta_1, \ldots, \beta_{n_0}\}$. Then $A \times B$ can be extended to an interpolating set for bivariate polynomials of individual degree $n_0 - 1$. In particular, there exists a bivariate polynomial Q of degree $n_0 - 1$ for which $Q(\alpha_i, \beta_i) = 1$ for i the smallest such that $\gamma_i \neq 0$, and $Q(\alpha_j, \beta_j) = 0$ for all $j \neq i$.

Then $P(X, Y) := Q(X, Y) \cdot v_H(X) \cdot v_H(Y)$ is a bivariate polynomial of individual degree $n - 1$ whose restriction to $S \times S$ is equal to $[0|C]$ for some $C \in \mathbb{F}^{H_0 \times H_0}$; hence $P \equiv \hat{C}'$. Moreover, $P(\alpha_j, \beta_j) = Q(\alpha_j, \beta_j) \cdot v_H(\alpha_j) \cdot v_H(\beta_j)$ is nonzero if $j = i$ and zero otherwise, since $\alpha_j, \beta_j \notin H$ for all $j \in [n_0]$. □

Lemma A.6. *Construction A.3 is a stackable RS-IOPP for the lincheck problem with $\mathcal{S}_{\text{DET}}$ running in time $\mathsf{b} \cdot \operatorname{polylog}(|K|)$ given query access to $(f_1, f_2), f_1$.*

Proof. **Completeness.** Suppose that $\hat{f}_1|_H = M \cdot \hat{f}_2|_H$. Then

$$\begin{aligned}\sum_{b \in H} u_S(b, \alpha) \hat{f}_1(b) &= \mathbf{r}_\alpha^T(\hat{f}_1|_H\ 0_{H_0}) = \mathbf{r}_\alpha^T(M \cdot \hat{f}_2|_H\ 0_{H_0}) \\ &= \mathbf{r}_\alpha^T[M|C](\hat{f}_2|_H\ 0_{H_0}) = \sum_{b \in H} u_{[M|C]}(b, \alpha) \hat{f}_2(b) = \sum_{b \in H} \hat{t}(b) \hat{f}_2(b) \ .\end{aligned}$$

By Claim A.4, $\sum_{k \in K} \hat{f}(k) = \gamma$. Hence both sumchecks accept; the other rational constraints are satisfied by definition.

Soundness. Suppose that $\hat{f}_1|_H \neq M \cdot \hat{f}_2|_H$. Then with probability $1 - |S|/|\mathbb{F}|$, $\mathbf{r}_\alpha^T(M \cdot \hat{f}_2|_H\ 0_{H_0}) \neq \mathbf{r}_\alpha^T[M|C](\hat{f}_2|_H\ 0_{H_0})$. In this case, either $\sum_{b \in H} u_S(b, \alpha) \hat{f}_1(b) - \hat{t}(b) \hat{f}_2(b) \neq 0$, or $\hat{t}(X) \not\equiv u_{[M|C]}(X, \alpha)$. In the former case, by the soundness guarantee of the zero knowledge sumcheck protocol, the verifier accepts with probability at most $1/|\mathbb{F}|$. In the latter case, it holds that $\hat{t}(\beta) \neq u_{[M|C]}(\beta, \alpha)$ with probability $1 - |S|/|\mathbb{F}|$; we now consider this subcase.

If $\gamma \neq \hat{t}(\beta)$ then the boundary constraint is not satisfied, so suppose otherwise. Then the claim $\gamma = u_{[M|C]}(\beta, \alpha)$ is false, and so the zero knowledge rational sumcheck protocol will output an unsatisifed constraint with probability $1 - 1/|\mathbb{F}|$.

Stackability. The RS-IOPP described is stackable with query bound b. Rounds $R_{\mathsf{rec}} = \{1, 2, 4, 5, 7\}$ are recyclable: the simulator $\mathbf{S}_{\text{rec}}$ answers all queries in these rounds with uniformly random field elements. In particular, any set of fewer than $|H_0|$ queries to t can be answered with uniformly random field elements by Claim A.5. The remaining rounds (P3, P6) consist of zero knowledge sumcheck masks which can be back-computed deterministically in polylogarithmic time (given query access to f_1, f_2). □

B Merkle Trees as Hiding Key-Value Commitments

In this section we provide simple and concretely efficient hiding key-value commitments derived from Merkle trees.

```
$\mathsf{C}, \mathsf{o} \leftarrow \mathsf{KV.Com}(\mathsf{pp}, \mathcal{M})$

$S \leftarrow \emptyset$
**for** $(\mathbb{k}, \mathbb{v}) \in \mathcal{M}$ :    // Hiding commitment to every leaf
    $r_{\mathbb{k}} \leftarrow \{0,1\}^{\lambda}$
    $S \leftarrow S \cup \{(\mathbb{k}, \mathsf{H}(\mathbb{v} \| r_{\mathbb{k}}))\}$
**for** $i \in [c, 1]$ :    // Repeatedly merge internal nodes
    $S' \leftarrow \emptyset$
    **for** $(\mathbb{k}\|0, \mathbb{v}_0) \in S : \exists(\mathbb{k}\|1, \mathbb{v}_1) \in S$ :
        // Merge the two siblings
        $O_{\mathbb{k}\|0} \leftarrow \mathbb{v}_1; O_{\mathbb{k}\|1} \leftarrow \mathbb{v}_0; S' \leftarrow S' \cup \{(\mathbb{k}, \mathsf{H}(\mathbb{v}_0 \| \mathbb{v}_1))\}$
    **for** $(\mathbb{k}\|0, \mathbb{v}_0) \in S : \nexists(\mathbb{k}\|1, \mathbb{v}_1) \in S$ :
        // No right sibling: create a 'fake' one
        $\mathbb{v}_1 \xleftarrow{\$} \{0,1\}^{\lambda}$    // Sample 'fake' right node
        $O_{\mathbf{k}\|0} \leftarrow \mathbb{v}_1; S' \leftarrow S' \cup \{(\mathbb{k}, \mathsf{H}(\mathbb{v}_0 \| \mathbb{v}_1))\}$
    **for** $(\mathbb{k}\|1, \mathbb{v}_1) \in S : \nexists(\mathbb{k}\|0, \mathbb{v}_0) \in S$ :
        // No left sibling: create a 'fake' one
        $\mathbb{v}_0 \xleftarrow{\$} \{0,1\}^{\lambda}$    // Sample 'fake' left node
        $O_{\mathbb{k}\|1} \leftarrow \mathbb{v}_0; S' \leftarrow S' \cup \{(\mathbb{k}, \mathsf{H}(\mathbb{v}_0 \| \mathbb{v}_1))\}$
    $S \leftarrow S'$
$S = \{\mathsf{C}\};$ **return** $(\mathsf{C}, (O, r))$    // Return the root
```

```
$\mathbb{\pi} \leftarrow \mathsf{KV.Open}(\mathsf{pp}, \mathsf{o} = (O, r), K)$

**for** $\mathbb{k} \in K$ :
    $\hat{r}_{\mathbb{k}} \leftarrow r_{\mathbb{k}}$
    **for** $i \in [c, 1] : \hat{O}_{\mathbb{k}[:i]} \leftarrow O_{\mathbb{k}[:i]}$
**return** $(\hat{O}, \hat{r})$
```

```
$\mathsf{C} \leftarrow \mathsf{KV.Verify}(\mathsf{pp}, \mathbb{\pi} = (\hat{O}, \hat{r}), \mathcal{M})$

$n \leftarrow 0$
**for** $(\mathbb{k}, \mathbb{v}) \in \mathcal{M}$ :
    $n \leftarrow n + 1$
    $\hat{\mathsf{C}}_n \leftarrow \mathsf{H}(\mathbb{v} \| \hat{r}_{\mathbb{k}})$
    **for** $i \in [c, 1]$ :
        **if** $\mathbb{k}_i = 0 : \hat{\mathsf{C}}_n \leftarrow \mathsf{H}(\hat{\mathsf{C}}_n \| \hat{O}_{\mathbb{k}[:i]})$
        **if** $\mathbb{k}_i = 1 : \hat{\mathsf{C}}_n \leftarrow \mathsf{H}(\hat{O}_{\mathbb{k}[:i]} \| \hat{\mathsf{C}}_n)$
**if** $\hat{\mathsf{C}}_1 = \hat{\mathsf{C}}_2 = \ldots = \hat{\mathsf{C}}_n$ :
    **return** $\hat{\mathsf{C}}_1$
**return** $\perp$
```

Figure 12: A hiding key-value commit scheme in the random oracle model.

Construction B.1 (In The Random Oracle Model.). *We show that (ordinary) Merkle trees implement hiding key-value commitments (see Figure 12) for $\mathcal{K} = \{0,1\}^c$ (where c is any constant) and $\Sigma = \{0,1\}^*$ when modelling the hash function $\mathsf{H} : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$ as a random oracle. This is implemented by replacing the roots of empty subtrees with uniformly random strings, since the output of a random oracle is (statistically) indistinguishable from uniformly random bit-strings this hides the (lack of) values in the particular subtree – ensuring indistinguishablity between maps when opening the intersection.*

Statistical Hiding. For $b \in \{0,1\}$, let $\mathcal{H}_b$ be the distribution of the adversaries view in the hiding game when the bit b is 0/1 respectively. Consider the hybrid $\mathcal{H}_{(b,K)}$ in which the commitment C is replaced by $(\mathsf{C}', \mathsf{o}') \leftarrow \mathsf{KV.Com}(\mathsf{pp}, \mathcal{M}'_b = \{(\mathbb{k}, \mathbb{v})\}_{(\mathbb{k},\mathbb{v}) \in \mathcal{M}_b \wedge \mathbb{k} \in K})$ and the opening is computed as $\mathbb{o}' \leftarrow \mathsf{KV.Open}(\mathsf{pp}, \mathsf{o}', K)$. To see that $\mathcal{H}_{(b,K)} \approx_{\mathsf{stat}} \mathcal{H}_b$, observe that all the siblings off-path from the indexes in K are uniformly random strings in $\mathcal{H}_{(b,K)}$ and outputs of a random oracle evaluated on a uniformly random string in $\mathcal{H}_b$ – these two distributions are clearly statistically indistinguishable. Lastly to argue that $\mathcal{H}_{(0,K)} = \mathcal{H}_{(1,K)}$ simply observe (form the algorithm description) that the opening proof is independent of the values. In conclusion $\mathcal{H}_0 \approx_{\mathsf{stat}} \mathcal{H}_{(0,K)} = \mathcal{H}_{(1,K)} \approx_{\mathsf{stat}} \mathcal{H}_1$, hence (even an unbounded) adversary has negligible adversary in the hiding game. □

Computational Binding. Suppose the adversary opens the commitment C to two maps $\mathcal{M}_0, \mathcal{M}_1$ st. there exists a key $\mathbb{k}$ and values $\mathbb{v}, \mathbb{v}'$ with $\mathbb{v} \neq \mathbb{v}'$, $(\mathbb{k}, \mathbb{v}) \in \mathcal{M}_0$ and $(\mathbb{k}, \mathbb{v}') \in \mathcal{M}_1$, then in particular there must exists two distinct sequences of internal nodes from distinct leafs $\mathbb{v}, \mathbb{v}'$ with the same position $\mathbb{k}$ (and hence path) terminating in the same root C. This implies that the two sequences must contain equal (internal) nodes N, N' with distinct children $\mathbb{v}_0, \mathbb{v}_1$ and $\mathbb{v}'_0, \mathbb{v}'_1$ (i.e. $\mathbb{v}_0 \| \mathbb{v}_1 \neq \mathbb{v}'_0 \| \mathbb{v}'_1$), since $N = \mathsf{H}(\mathbb{v}_0 \| \mathbb{v}_1) = N' = \mathsf{H}(\mathbb{v}'_0 \| \mathbb{v}'_1)$, this yields a collision. Note that any adversary making q queries to the random oracle obtains a collision with probability at most $1 - \prod_{i<q}(1 - \frac{i}{2^{\lambda}}) = 1 - \left(\frac{2^{\lambda}!}{(2^{\lambda}-q)! \cdot 2^{\lambda q}}\right)$ – negligible when q is polynomial in λ. □

Construction B.2 (In The Standard Model.). *The construction shown in Figure 13 is key-value commitment from a compressing commitment scheme* Commit.

Statistical/Computational Hiding. The flavor (statistical/computational) of hiding is inherited from the commitment scheme. The argument is analogous to that of the random oracle based construction, except (statistical/computational) indistinguishably of the hybrids now follows from hiding of the commitment scheme: the inability of the adversary to distinguish a 'fake' node from a commitment to a subtree. □

Computational Binding. The proof is analogous to the proof of binding for the random oracle based construction: a collision (distinct committed values) trivially implies the ability to open the same commitment (some internal node) to two different values (pairs of children) – this violates computational binding of the commitment scheme. □

```
$\mathsf{C}, \mathsf{o} \leftarrow \mathsf{KV.Com}(\mathsf{pp}, \mathcal{M})$
$S \leftarrow \emptyset$
for $(\mathbb{k}, \mathbb{v}) \in \mathcal{M}$ :    // Commit to every leaf
    $r_{\mathbf{k}} \leftarrow \{0,1\}^{\lambda}$
    $S \leftarrow S \cup \{(\mathbb{k}, \mathsf{Commit}(\mathbb{v}; r_{\mathbb{k}}))\}$
for $i \in [c, 1]$ :    // Repeatedly merge nodes
    $S' \leftarrow \emptyset$
    for $(\mathbb{k}\|0, \mathbb{v}_0) \in S : \exists(\mathbb{k}\|1, \mathbb{v}_1) \in S$ :
        // Merge the two siblings
        $r \xleftarrow{\$} \{0,1\}^{\lambda}$
        $O_{\mathbb{k}\|0} \leftarrow (\mathbb{v}_1, r); O_{\mathbb{k}\|1} \leftarrow (\mathbb{v}_0, r)$
        $S' \leftarrow S' \cup \{(\mathbb{k}, \mathsf{Commit}((\mathbb{v}_0, \mathbb{v}_1); r))\}$
    for $(\mathbb{k}\|0, \mathbb{v}_0) \in S : \nexists(\mathbb{k}\|1, \mathbb{v}_1) \in S$ :
        // No right sibling: create a 'fake' one
        $r_1 \xleftarrow{\$} \{0,1\}^{\lambda}; \mathbb{v}_1 \leftarrow \mathsf{Commit}(\mathbf{0}; r_1)$
        $r \xleftarrow{\$} \{0,1\}^{\lambda}; O_{\mathbb{k}\|0} \leftarrow (\mathbb{v}_1, r)$
        $S' \leftarrow S' \cup \{(\mathbb{k}, \mathsf{Commit}((\mathbb{v}_0, \mathbb{v}_1); r))\}$
    for $(\mathbb{k}\|1, \mathbb{v}_1) \in S : \nexists(\mathbb{k}\|0, \mathbb{v}_0) \in S$ :
        // No left sibling: create a 'fake' one
        $r_0 \xleftarrow{\$} \{0,1\}^{\lambda}; \mathbb{v}_0 \leftarrow \mathsf{Commit}(\mathbf{0}; r_0)$
        $r \xleftarrow{\$} \{0,1\}^{\lambda}; O_{\mathbb{k}\|1} \leftarrow \mathbb{v}_0$
        $S' \leftarrow S' \cup \{(\mathbb{k}, \mathsf{Commit}((\mathbb{v}_0, \mathbb{v}_1); r))\}$
    $S \leftarrow S'$
$S = \{\mathsf{C}\}$; return $(\mathsf{C}, (O, r))$    // Return the root
```

```
$\mathsf{pp} \leftarrow \mathsf{KV.Setup}(1^{\lambda})$
$\mathsf{PP_{com}} \leftarrow \Pi_{\mathsf{com}}.\mathsf{Setup}(1^{\lambda})$
$\mathsf{pp} = (\mathsf{PP_{com}})$
return $\mathsf{pp}$
```

```
$\mathbb{o} \leftarrow \mathsf{KV.Open}(\mathsf{pp}, \mathsf{o} = (O, r), K)$
// Copy opening information for every index.
for $\mathbb{k} \in K$ :
    $\hat{r}_{\mathbb{k}} \leftarrow r_{\mathbb{k}}$
    for $i \in [c, 1] : \hat{O}_{\mathbb{k}[:i]} \leftarrow O_{\mathbb{k}[:i]}$
return $(\hat{O}, \hat{r})$
```

```
$\mathsf{C} \leftarrow \mathsf{KV.Verify}(\mathsf{pp}, \mathbb{o} = (\hat{O}, \hat{r}), \mathcal{M})$
$n \leftarrow 0$
for $(\mathbb{k}, \mathbb{v}) \in \mathcal{M}$ :
    $n \leftarrow n + 1$
    $\hat{\mathsf{C}}_n \leftarrow \mathsf{Commit}(\mathbb{v}, \hat{r}_{\mathbb{k}})$
    for $i \in [c, 1]$ :
        $(\bar{\mathsf{C}}, r) := \hat{O}_{\mathbb{k}[:i]}$
        if $\mathbb{k}_i = 0 : \hat{\mathsf{C}}_n \leftarrow \mathsf{Commit}((\hat{\mathsf{C}}_n, \bar{\mathsf{C}}); r)$
        if $\mathbb{k}_i = 1 : \hat{\mathsf{C}}_n \leftarrow \mathsf{Commit}((\bar{\mathsf{C}}, \hat{\mathsf{C}}_n); r)$
if $\hat{\mathsf{C}}_1 = \hat{\mathsf{C}}_2 = \ldots = \hat{\mathsf{C}}_n$ :
    return $\hat{\mathsf{C}}_1$
return $\perp$
```

Figure 13: A hiding key-value commit scheme from compressing commitments.