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 *compilers* for efficient disjunctive composition (*e.g.* an OR of multiple clauses) of zero-knowledge proofs (*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 [CR-YPTO'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.

1 Introduction

Zero-knowledge proofs and arguments [30] 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 [29], and recent work has proposed zero-knowledge proofs that are more practically efficient [36,12,31,37,19,10,34]. Many of these efficient zero-knowledge proofs are now being used in practice [9,48,23], 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 optimized 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 [22,1,24] and recently there has been renewed interest in optimizing cryptographic protocols for disjunctions, both in the context of zero-knowledge [32,21,41,34,26,4,7] and secure multiparty computation [33,35,27].

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 [32,34,4] 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 [34,4]. 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* [22,1,7,26], generic approaches that automatically transform large classes of zero-knowledge protocols into disjunctive zero-knowledge protocols. The seminal work in this area is [22], 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. [7] and Goel et al. [26] 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 poly-logarithmic in the size of the computation being proven. Succinct zero-knowledge proofs are the subject of a long and active line of research ([39,25,12,31,11,17] 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 to 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 [26]) 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, [22], the prover in a succinct proof system can run the (more efficient) simulator for the inactive clauses instead of the (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. In addition to being a more technically challenging statement structure, true disjunctions are also important for many applications. For example, Heath and Kolesnikov studied showing the existence of a bug in a code base [34] in zero-knowledge, which implicitly embeds a true disjunction. It is easy to imagine many other such applications: a prover could want to demonstrate that a image is the product of applying one of a number of sanctioned image modification algorithms (eg. blur, red eye, etc...) to some committed photograph, or a financial institution might want to demonstrate that a transaction satisfies one of a number of policies that would make it compliant.

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 yield significant improvements in prover time. We do this by extending the notion of a "stackable" Σ-protocol, introduced by Goel et al. [26], to a more general notion of a "stackable" interactive protocol. At a high level, a protocol is stackable if it has a zero-knowledge simulator which can be decomposed into a randomized, statement-independent part $\mathcal{S}_{\text{RAND}}$, and a deterministic part $\mathcal{S}_{\text{DET}}$ that completes the work of $\mathcal{S}_{\text{RAND}}$ for some specific statement. 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 (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 $\mathcal{R}'((\mathbb{x}_1, \ldots, \mathbb{x}_\ell), \mathbb{w}) \coloneqq \exists i, \mathcal{R}_i(\mathbb{x}_i, \mathbb{w}) = 1$ with communication complexity proportional to $\mathbb{C}(\Pi) + O(\log(\ell))$ 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.

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

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 significant, 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) [11], a generalization of interactive proofs that underlies various efficient succinct argument constructions. We show how to adapt the [11,20] 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** [10] 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 $O_{\mathbb{F}}(\ell N \log(\ell N))$.[5]
- **Fractal** [20] 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" [17,19,3,5,4]. This class of protocols, best

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

represented by Compressed Σ-protocols [4] and Bulletproofs [19], 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** [3,4] 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 the circuit-satisfiability 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. [4] in that our approach supports *true disjunctions* as well.[6]
- **Bulletproofs** [19] 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 $\mathbb{x}_1 \in \mathcal{L}_1$, $\mathbb{x}_2 \in \mathcal{L}_2$, $\ldots$, 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 [22], 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

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

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 [46]), 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*.

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 [42] of Valiant's UC [47] shows that for a circuit implementing 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 [43], boolean UCs remain orders of magnitude larger than the circuits they can simulate, and arithmetic UCs would incur even higher constants [42].

Disjunctive Templates for Succinct Proofs. We now turn our attention to the disjunctive composition of succinct proofs for NP. We first observe that succinctness by itself implies communication-efficient disjunctive composition via statement combination. Specifically, if the size of the relation circuit is increased by a multiplicative factor of ℓ, a logarithmic-sized proof will only increase in size by an additive factor of $\log(\ell)$, resulting in a proof that is only marginally larger.

While communication efficient, this approach, however, increases the running time of the prover by at least a multiplicative factor ℓ (potentially more, depending on the complexity of the proof system). 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.[7]

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 for each of the 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.[8] This observation means that applying

[7] We note that there do exist techniques generic techniques to minimize space complexity of provers, *e.g.* [14,15]

[8] A similar observation was recently used in a concurrent and independent work of Kim et al. [40] for designing efficient non-malleable zero-knowledge proofs.

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 [22] yields a protocol whose communication complexity is linear in ℓ. In a recent work, Goel et al. [26] 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 it 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 [22].

Taken together, these observations 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 summarize the approach of Goel et al. [26] and then proceed to discuss sublinear-sized proofs.

2.2 Stacking Sigmas for Sublinear-sized Proofs

The Approach of Stacking Sigmas [26]. Goel et al. [26] 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 (not conditioned on the first 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.[9]

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 multi-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 considers 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. The deterministic simulator $\mathcal{S}_{\text{DET}}$ can then "complete the transcript," by computing the remaining, statement-dependent parts of each message. 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.

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.

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) [38,44,11,45]. 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

[9] Note that to compile the resulting protocol with Fiat-Shamir, the prover passes the *partially-binding commitment* into the random oracle, as the challenge cannot depend on first-round messages that have not yet been computed.

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) [6]. 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 [11]: 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 [10] and a variant of Fractal [20] 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.

Intuitively 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 also require 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 [26] 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, and 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 [10] and Fractal [20]. 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-knowledge 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 [3,5,4] and Bulletproofs [17,19].

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 a 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-IPs 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 equivalent 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.[10] 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.[11] Specifically, this holds for the final round message in the "base" protocol in both Bulletproofs [17,19] and Compressed Σ-protocols [3,5,4]. 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 verifier. 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 reused/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

[10] We formalize this claim in the full version [28].

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

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.

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 multiexponentiations. 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 lies 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 suggests 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 stackability.

Compressed Σ-Protocol. Compressed Σ-protocols [3,5,4] 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 statement, 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 we are 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 [4], 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 [19] 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 achieve efficient zero-knowledge for range proofs and circuit

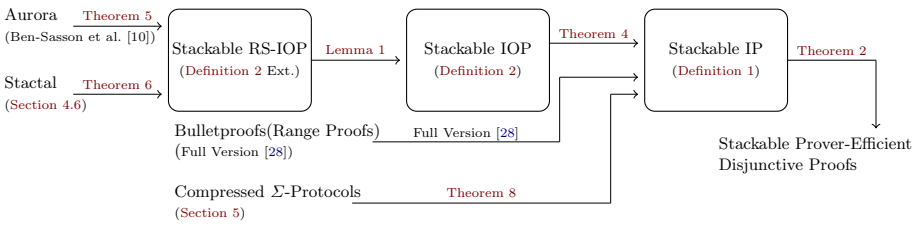

Fig. 1. A roadmap for the results in our paper. Several Theorems are contained in the full version of the paper.

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 statement dependent computation (that must be run separately for each clause) is remarkably inexpensive (involving only one group exponentiation 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.

Roadmap to Our Results. We give an overview of how we reach our technical results in Figure 1.

2.5 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 communication 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 $\mathtt{x}$ and the witness as $\mathtt{w}$. 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 $\mathtt{x}$. We use multiplicative notation for groups and group operations.

3 Stacking Zero-Knowledge Interactive-Proofs

In this section, we extend the notion of "stackability" introduced in Goel et al. [26] 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 formally define Stackable ZK-IP in Section 3.1, present our stacking compiler in Section 3.2, and provide a heuristic mechanism for preparing ZK-IP protocols for stacking in the full version of our paper [28].

3.1 Defining Stackable ZK-IP

Recently, Goel et al. [26] 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 $\mathcal{S}_{\text{RAND}}$,[12] 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 1 (Stackable ZK-IP). *Let Π be a ZK-IP consisting of k prover messages and $k-1$ verifier messages 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 computable, well-behaved, deterministic simulator $\mathcal{S}_{\text{DET}}$, such*

[12] We depart from the notation introduced by Goel et al. [26], in which this first part is instead discussed as an efficiently samplable 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 $\mathcal{S}_{\text{RAND}}$)

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:*

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

$$\approx$$

$$\left\{((m_{\text{RAND},i} \| m_{\text{DET},i})_{i\in[k]}, C) \mid M_{\text{RAND}} \leftarrow \mathcal{S}_{\text{RAND}}(1^\lambda, C); M_{\text{DET}} \coloneqq \mathcal{S}_{\text{DET}}(\mathbb{x}, C, M_{\text{RAND}})\right\}$$

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

3.2 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. [26], the main idea behind this construction is to honestly 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 receiving 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 associated 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 2. *Let* Π *be a stackable ZK-IP (see Definition 1) 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 (as defined in [26]). 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* $\mathcal{R}'((\mathbb{x}_1, \ldots, \mathbb{x}_\ell), (\mathfrak{a}, \mathbb{w})) := \mathcal{R}(\mathbb{x}_\mathfrak{a}, \mathbb{w})$.

The proof for Theorem 2 can be found in the full version [28].

Complexity Discussion. Let $\mathtt{CC}(\Pi)$ be the communication complexity of Π. Then, the communication complexity of the Π' obtained from Theorem 2 is $(\mathtt{CC}(\Pi) + |\mathsf{ck}| + |\mathsf{com}| + |r'|)$, where the sizes of $\mathsf{ck}, \mathsf{com}$ 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. [26, 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 $O(\log \ell)$.

Finding recyclable messages requires manual effort. We discuss intuition for finding these messages, along with an informal procedure, in the full-version of our paper [28].

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 $\{m_{\text{DET},i,j}\}_{i\in[k]} \coloneqq \mathcal{S}_{\text{DET}}(\mathbb{x}_j, \{c_j\}_{j\in[k-1]}, \{m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]})$.
- 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 $\{m_{\text{DET},i,j}\}_{i\in[k]} \coloneqq \mathcal{S}_{\text{DET}}(\mathbb{x}_i, \{c_i\}_{i\in[k-1]}, \{m_{\text{RAND},i,\mathfrak{a}}\}_{i\in[k]})$.
- For each $i \in [k]$, set $\mathbf{v}'_i = (m_{\text{DET},i,1}, \ldots, m_{\text{DET},i,\ell})$.
- Compute and return:

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

Fig. 2. A compiler for stacking multiple instances of a stackable ZK-IP

4 Speed-Stacking Interactive Oracle Proofs

Interactive oracle proofs, originally proposed by [11,45], 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 [20] 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 [10] Fractal hIOP [20]. 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 4.3. 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 4.5, we describe our two constructions of stackable hIOPs, based on the Aurora IOP [10] and Fractal hIOP [20] constructions respectively.

4.1 Holographic IOPs

Holographic IOPs were originally defined in [20]. Here we describe special properties of holographic IOPs that we will make use of in this work; for a full definition of the model, see [28].

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 $(C, \Pi_1|_{Q_1}, \ldots, \Pi_k|_{Q_k})$.

4.2 Reed–Solomon encoded holographic IOPs

Reed–Solomon encoded IOPs (RS-IOPs) were introduced in [10] and adapted to the holographic setting in [20, Section 4.1]. We refer the reader to [28]for a full definition of RS-IOPs; here we give an adapted definition of zero-knowledge for RS-IOPs that we will use later.

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
– $\mathrm{View}_{Q(C)}(\mathbf{P}(\mathtt{i}, \mathtt{x}, \mathtt{w}), \mathbf{V}^{\mathbf{I}(\mathtt{i})}(\mathtt{x})) = (C, \Pi_1|_{Q(C)}, \ldots, \Pi_k|_{Q(C)})$ 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 $Q(C) \subseteq L$.

4.3 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 4.4. Looking ahead, we will give modifications of Aurora IOP [10] and Fractal hIOP [20] 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 and IOP. 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 1), as follows.

Definition 2 (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 $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$, the following algorithm is a special honest-verifier zero-knowledge simulator for HOL:*

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

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

and for all $\lambda \in \mathbb{N}$ and $(\mathtt{i}', \mathtt{x}', \mathtt{w}')$ (whether in $\mathcal{R}$ or not), $\mathbf{S}(1^\lambda, \mathtt{i}', \mathtt{x}')$ 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 $\mathbf{S}$ be an SSHVZK simulator.

4.4 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 the full version [28], we give a formal definition for the key-value commitment schemes that we require. In this section we provide both compilers (in Lemma 1 and Theorem 4 respectively).

Hiding Key-Value Commitments. Key-value commitments, described by Boneh, Bünz and Fisch [16] and Agrawal and Raghuraman [2] 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 polynomial 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. The exact definition and constructions can be found in the full-version of our paper [28].

Compiling RS-IOP to Stackable IOP. We now show that, by slightly tweaking the RS-IOP to IOP transformation presented in [10, Section 8.1], we can preserve stackability. The compiler of [10, Section 8.1] converts an RS-IOP into an IOP using a (IOP) proximity test for Reed-Solomon codes [8] [13] (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 $1/|\mathbb{F}|$[13]. 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 distribution 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 [10, Protocol 8.6] there is no need for the prover to send the oracle q explicitly[14], however we need this to efficiently stack.

Lemma 1 (From Stackable RS-IOP to Stackable IOP). *There is a transformation (an adaptation of [10, 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 full version [28] for details.*

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

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

Compiling Stackable IOP to Stackable IP. Next we show how to compile stackable IOPs into stackable interactive arguments using hiding key-value commitments. 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 3 (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. [11, Section 6] (from IOPs to IPs) but replacing Merkle trees with the related notion of a key-value commitment.

Theorem 4 (Correctness of Construction 3). *Given a key-value commitment scheme: 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* $\mathcal{S}_{\text{DET}}$ *is that of* $\mathcal{S}_{\text{DET}}^{\mathbf{I}(\mathbb{i})}$ *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 full version [28] for the proof.*

4.5 Stackable RS-IOPs

In this section we show that two key IOP protocols from the literature, Aurora [10] and Fractal [20] can be made stackable. These protocols are proof systems for the R1CS relation, defined formally below.

Definition 3. *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 2 (By Ben-Sasson et al. [10]). *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 $\hat{q} \coloneqq \hat{r} + \beta \hat{f}$ 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 $\hat{e} \coloneqq \Sigma^{-1}_{H,\zeta}(\hat{q} - \hat{h} \cdot v_H)$.

The zero-knowledge simulator given by [10] 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 5. *The Aurora zero-knowledge RS-IOP for R1CS [10, Protocol 7.5] is stackable (after a small modification) with $\mathcal{S}_{\text{DET}}$ 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 $\mathcal{S}_{\text{RAND}}$ (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 $\mathcal{S}_{\text{DET}}$ to compute r as $\Sigma_{H,\zeta}(g) + h \cdot v_H - \beta f$.

As a result, the time complexity of $\mathcal{S}_{\text{DET}}$ 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 [18]. □

4.6 Stactal

Next we describe "stackable Fractal", or Stactal, a variant of the Fractal protocol [20] 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, [20] 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 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 polylog($\|M\|$) and makes a small number of queries to the encoding of M.We prove the following theorem in the full version of the paper [28]:

Theorem 6 (Stactal). *The protocol obtained from Fractal by replacing the holographic lincheck protocol with a stackable holographic lincheck (as described in the full version of our paper [28]) is stackable, with $\mathcal{S}_{\text{DET}}$ running in time $O(\mathsf{b} \cdot (|x| + \text{polylog}(\|A\| + \|B\| + \|C\|)))$ (measured in field operations).*

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 systems 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 [19], which give a folding argument for inner products, and Compressed Σ-Protocols [3,4,5], 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 full-version [28], we demonstrate how to stack Bulletproofs, which as discussed earlier are less amenable to computational savings from our stacking approach.

Compressed Σ-protocols were proposed in a series of recent works by Attema, Cramer, Fehr and Kohl [3,4,5]. In this section, we focus on the specific instantiation of this approach proposed by Attema, Cramer, and Fehr [4], as it has a clean presentation.

Notation. We slightly modify some of the notation presented by Attema, Cramer, and Fehr in [4] 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 $\mathbb{G}_T$.[15] 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 $f(0, \ldots, 0, \mathbf{x}_R)$. Upper-case 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 multi-exponentation by $\mathbf{g}^{\mathbf{x}} = \prod_i g_i^{x_i}$.

Compressed Σ-Protocols. Attema et al. [4] 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 homomorphism from $\mathbb{Z}_q^N$ to $\mathbb{G}_T$. Intuitively, their protocol is a "standard" (Schnorr-type) Σ-protocol, 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: $\mathbf{g}^{\mathbf{z}} \stackrel{?}{=} TP^c$ and $f(\mathbf{z}) \stackrel{?}{=} cy + t$ (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 work 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 *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 [4].

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

[15] 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 homomorphism; 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.

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 of 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 discuss the generalization in the full version of our paper [28]. We now state the main Theorem from [4].

Theorem 7 ([4]). *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 a perfectly complete, special honest-verifier zero-knowledge and unconditionally (2,3,3, ..., 3)-special sound.*

5.1 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 [4], 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-exponentiations 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. [26], 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 multi-round protocol should contain all recyclable messages besides t, T, and indeed it does.

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 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 reusable. 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 to maximize stacking:

(1) **Sending Q_1:** In the original protocol described in [4], 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 in the full version [28]. 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 $f_{1,R}(\mathbf{x}_{\mathbf{i}L})$ (when computing a_i) or $f_{1,L}(\mathbf{x}_{\mathbf{i}R})$ (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 $f_{1,R}(\mathbf{x}_{\mathbf{i}L})$ for a_i and $f_{1,L}(\mathbf{x}_{\mathbf{i}R})$ 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 8. *Compressed Σ-protocols [4], denoted as $\Pi_{compressed}$, is stackable.*

We give a proof for Theorem 8 in the full version of the paper. Combining Theorems 8 and 2, we get the following Corollary.

Corollary 1. *Let $\Pi_{speed\text{-}compressed}$ be output of the compiler in Figure 2 recursively applied to $\Pi_{compressed}$ using $\mathcal{S}^{\text{COMP}}_{\text{RAND}}$ and $\mathcal{S}^{\text{COMP}}_{\text{DET}}$ as defined in the proof of Theorem 8. Then $\Pi_{speed\text{-}compressed}$ is a stackable ZK-IP for $\mathcal{R}_{dis\text{-}compressed}$ with logarithmic communication complexity, and prover computational complexity $O(\mathsf{Time}(\Pi_{compressed}) + \ell \cdot \mathsf{Time}(\mathcal{S}^{\text{COMP}}_{\text{DET}}))$.*

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. [26], both key generation and committing require ℓ exponentiations 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-exponentiations 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 do this using Gauss-Jordan elimination, which will take $N \log^2(N)$ field operations.

Extension to Circuit Satisfiability. Due to space constraints, we discuss the details of speed-stacking the circuit satisfiability in the full paper [28].

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

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 415–432. Springer, Heidelberg (Dec 2002). https://doi.org/10.1007/3-540-36178-2_26
2. Agrawal, S., Raghuraman, S.: KVaC: Key-Value Commitments for blockchains and beyond. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 839–869. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64840-4_28
3. Attema, T., Cramer, R.: Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 513–543. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56877-1_18
4. Attema, T., Cramer, R., Fehr, S.: Compressing proofs of k-out-of-n partial knowledge. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 65–91. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_3
5. Attema, T., Cramer, R., Kohl, L.: A compressed Σ-protocol theory for lattices. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 549–579. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_19
6. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: 23rd ACM STOC. pp. 21–31. ACM Press (May 1991). https://doi.org/10.1145/103418.103428
7. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 92–122. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_4
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018), https://eprint.iacr.org/2018/046

9. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.36
10. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_4
11. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53644-5_2
12. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 781–796. USENIX Association (Aug 2014)
13. Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: Sampling outside the box improves soundness. In: Vidick, T. (ed.) ITCS 2020. vol. 151, pp. 5:1–5:32. LIPIcs (Jan 2020). https://doi.org/10.4230/LIPIcs.ITCS.2020.5
14. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 168–197. Springer, Heidelberg (Nov 2020). https://doi.org/10.1007/978-3-030-64378-2_7
15. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 123–152. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_5
16. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 561–586. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_20
17. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_12
18. Borodin, A., Moenck, R.: Fast modular transforms. J. Comput. Syst. Sci. **8**(3), 366–386 (1974)
19. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
20. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 769–793. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45721-1_27
21. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Online/offline OR composition of sigma protocols. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 63–92. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_3
22. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48658-5_19

23. swisspost evoting: E-voting system 2019. https://gitlab.com/swisspost-evoting/e-voting-system-2019 (2019)
24. Garay, J.A., MacKenzie, P.D., Yang, K.: Strengthening zero-knowledge protocols using signatures. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 177–194. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_11
25. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_37
26. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Stacking sigmas: A framework to compose Σ-protocols for disjunctions. In: EUROCRYPT 2022, Part II. pp. 458–487. LNCS, Springer, Heidelberg (Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_16
27. Goel, A., Hall-Andersen, M., Hegde, A., Jain, A.: Secure multiparty computation with free branching. In: EUROCRYPT 2022, Part I. pp. 397–426. LNCS, Springer, Heidelberg (Jun 2022). https://doi.org/10.1007/978-3-031-06944-4_14
28. Goel, A., Hall-Andersen, M., Kaptchuk, G., Spooner, N.: Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. IACR Cryptol. ePrint Arch. p. 1419 (2022), https://eprint.iacr.org/2022/1419
29. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: 27th FOCS. pp. 174–187. IEEE Computer Society Press (Oct 1986). https://doi.org/10.1109/SFCS.1986.47
30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th ACM STOC. pp. 291–304. ACM Press (May 1985). https://doi.org/10.1145/22145.22178
31. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
32. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 253–280. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_9
33. Heath, D., Kolesnikov, V.: Stacked garbling - garbled circuit proportional to longest execution path. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 763–792. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_27
34. Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 569–598. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45727-3_19
35. Heath, D., Kolesnikov, V.: LogStack: Stacked garbling with $O(b \log b)$ computation. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part III. LNCS, vol. 12698, pp. 3–32. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77883-5_1
36. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 955–966. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516662

37. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 525–537. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243805
38. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press (May 1992). https://doi.org/10.1145/129712.129782
39. Kilian, J.: On the complexity of bounded-interaction and noninteractive zero-knowledge proofs. In: 35th FOCS. pp. 466–477. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365744
40. Kim, A., Liang, X., Pandey, O.: A new approach to efficient non-malleable zero-knowledge. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13510, pp. 389–418. Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_14, https://doi.org/10.1007/978-3-031-15985-5_14
41. Kolesnikov, V.: Free IF: How to omit inactive branches and implement S-universal garbled circuit (almost) for free. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 34–58. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_2
42. Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant's universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017 (2016), https://eprint.iacr.org/2016/017
43. Liu, H., Yu, Y., Zhao, S., Zhang, J., Liu, W., Hu, Z.: Pushing the limits of valiant's universal circuits: Simpler, tighter and more compact. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 365–394. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_13
44. Micali, S.: CS proofs (extended abstracts). In: 35th FOCS. pp. 436–453. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365746
45. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 49–62. ACM Press (Jun 2016). https://doi.org/10.1145/2897518.2897652
46. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_22
47. Valiant, L.G.: Universal circuits (preliminary report). In: Proceedings of the Eighth Annual ACM Symposium on Theory of Computing. p. 196–203. STOC '76, Association for Computing Machinery, New York, NY, USA (1976). https://doi.org/10.1145/800113.803649, https://doi.org/10.1145/800113.803649
48. Zaverucha, G.: The picnic signature algorithm. Tech. rep. (2020), https://github.com/microsoft/Picnic/raw/master/spec/spec-v3.0.pdf