Scalable Zero Knowledge with no Trusted Setup

Eli Ben-Sasson[1], Iddo Bentov[2], Yinon Horesh[3], and Michael Riabzev[1]

[1] Technion & StarkWare Industries Ltd., Israel
[2] Cornell Tech, NY, USA
[3] Technion, Israel

Abstract. One of the approaches to constructing zero knowledge (ZK) arguments relies on "PCP techniques" that date back to influential works from the early 1990's [Babai et al., Arora et al. 1991-2]. These techniques require only minimal cryptographic assumptions, namely, the existence of a family of collision-resistant hash functions [Kilian, STOC 1992], and achieve two remarkable properties: (i) all messages generated by the verifier are public random coins, and (ii) total verification time is merely poly-logarithmic in the time needed to naïvely execute the computation being verified [Babai et al., STOC 1991].

Those early constructions were never realized in code, mostly because proving time was too large. To address this, the model of interactive oracle proofs (IOPs), which generalizes the PCP model, was recently suggested. Proving time for ZK-IOPs was reduced to *quasi-linear*, even for problems that require nondeterministic exponential time to decide [Ben-Sasson et al., TCC 2016, ICALP 2017].

Despite these recent advances it was still not clear whether ZK-IOP systems can lead to concretely efficient succinct argument systems. Our main claim is that this is indeed the case. We present a new construction of an IOP of knowledge (which we call a zk-STIK) that improves, asymptotically, on the state of art: for log-space computations of length T it is the first to $O(T \log T)$ arithmetic prover complexity and $O(\log T)$ verifier arithmetic complexity. Prior IOPs had additional $\mathsf{poly} \log T$ factors in both prover and verifier. Additionally, we report a C++ realization of this system (which we call libSTARK). Compared to prevailing ZK realizations, it has the fastest proving and (total) verification time for sufficiently large *sequential* computations.

1 Introduction

By the early 1990s, a combination of works [44, 6, 5, 54, 39, 7] showed the existence of proof systems that satisfy the following conditions, simultaneously:

1. **universality:** such systems can be constructed for any language $L \in \mathsf{NEXP}$;
2. **zero knowledge (ZK):** the proof for the membership of $x \in L$ reveals no meaningful information about the nondeterministic witness w provided to show $x \in L$;
3. **argument of knowledge (ARK):** the witness w can be "extracted" from a prover that succeeds in showing $x \in L$;
4. **scalable (succinct) verification:** for instances of size n, verifying membership in L requires time at most $\mathsf{poly}(\log \mathsf{T}(n), n)$, where $\mathsf{T}(n)$ is the running time of the nondeterministic machine[4] deciding membership in L on instances of size n;

[4] The machine could either be a Turing machine or a RAM machine.

5. **public coins:** all messages and queries sent by the verifier are public random coins ("Arthur-Merlin" protocols); we choose to refer to such protocols as *transparent* and this allows us to compress terminology (one word instead of two) while emphasizing the benefits of such systems.
6. **"simple" cryptographic assumptions:** the soundness of these constructions assumes only the existence of a family of collision resistant hash functions[5].

The early theoretical constructions that achieved the six properties above were based on the celebrated PCP Theorem [2, 3, 6, 5] and ZK variants of PCPs (ZK-PCPs) [39, 55, 52]. But these theoretical constructions were never *realized*[6] in code, mostly due to prover (in)efficiency problems. Recent advances in the study of quasilinear PCPs [27, 38, 25, 62, 17] and ZK Interactive Oracle Proofs (IOPs) [22, 13, 15, 67] have shown the existence of ZK-IOP systems that achieve all six properties along with the following property, simultaneously:

7. **scalable (quasilinear) proving:** the running time of the prover is $\tilde{O}(T(n)) := T(n) \cdot \log^{O(1)} T(n)$.

Nevertheless, the constructions that achieve all seven properties were inefficient in terms of both prover and verifier running times. Indeed, a proof-of-concept IOP-based system without ZK but with the remaining six properties, called SCI [9], was reported recently but was relatively inefficient, and the cost of adding ZK to it would further deteriorate its performance. The recent Aurora system [21] describes a ZK-IOP (along with an accompanying implementation) that is designed for arithmetic circuits and provides succinct proofs (poly-logarithmic in the size of the arithmetic circuit). However, verifier running time scales linearly with the input size, meaning the system is not (doubly) scalable according to our definition of the term. Therefore, a valid question to ask is whether IOPs are a viable approach to obtaining ZK systems for any concretely realizable computational setting? The main point of this paper is to provide a positive answer to this question.

Contributions We make four:

1. The first *strictly scalable* ZK-IOP for log-space computations, in arithmetic complexity (see Definition 3 and Theorem 1). In words, this is the first ZK-IOP for computations requiring $T(n)$ time and $O(\log T(n))$ space (on instances of size n) in which the arithmetic complexity of the prover is $O(T(n) \cdot \log T(n))$ and that of the verifier is $O(\log T(n))$. All prior ZK-IOP constructions had poly-log factors in the verifier and/or prover with an exponent (in the poly-log) that is strictly greater than 1.
2. A scalable ZK-IOP for general sequential computations (with no restrictions on memory access) in NEXP, which is more efficient in terms of asymptotic prover and

[5] In the "random oracle" model where all parties have access to the same random function, these systems can be made non-interactive [60, 22].

[6] Henceforth, a proof system *realization* refers to an implementation in code, along with reported measurements, of it.

verifier complexity than the prior state of the art (Theorem 2). It is the first scalable ZK-IOP system with *strictly* quasi-linear ($O(T(n)\log T(n))$) proof length (measured in field elements) and *strictly* logarithmic ($O(\log T(n))$) query complexity.

3. A code realization (in C++) of an argument system that implements this pair of IOP systems. The code base, called libSTARK, is published under the permissive MIT license [10]. Furthermore, the ZK-STARK prover is $\geq 10\times$ faster than prior ZK provers for general sequential computations (see Section 3). This reduction is significant because prover complexity is the main bottleneck encountered when scaling ZK proof systems to deal with large computations. Compared to SCI [9], the prior state-of-the-art scalable IOP system, our ZK-STARK reduces proving time by $7\times$–$40\times$ and communication complexity by $3\times$–$20\times$; the improved verifier complexity (but not prover compelxity) relies on a new set of algebraic conjectures — different than those relied upon by SCI (and other ZK constructions). These conjectures, which are of independent interest, are discussed in Section 4.3.
4. For the benefit of future and alternative constructions, we formally define the notions of a scalable and transparent IOP of knowledge (STIK) and a scalable and transparent argument of knowledge (STARK), which is a system that achieves, simultaneously, all seven properties listed earlier.

1.1 The virtues of transparent scalability

No prior ZK system realized in code has achieved both transparency *and* full (or double) scalability for general programs, meaning the simultaneous combination of quasilinear proving time *and* polylogarithmic (succinct) verification time. We briefly discuss the importance of the combined effect of scalability and transparency in ZK systems.

Transparency Non-transparent protocols require an elaborate *setup phase* that is hard to perform securely [20]. This phase constitutes a single point of failure that might be exploited by powerful parties to compromise the system (especially when that system carries significant value, as is the case with Zcash [64]). The complexity of performing the setup leads to another security threat: to minimize the number of times the setup is invoked, projects using non-transparent systems will batch together many system improvements within a single roll-out, adding to operational security risks; this is already the case with Zcash's recent "Sapling" upgrade.

A different benefit of transparency relates to decentralized open source code. It is far easier to build transparent systems in this manner, because they do not require an extra setup procedure, one that requires additional trust assumptions and governance structures (who will be trusted to perform and manage the setup phase?). For the reasons above, leading crypto-currencies that care about financial privacy (including Ethereum, Monero and Zcash) agree that a move to *transparent* ZK ARKs is inevitable.

Scalability An aspect of proof systems (with or without ZK) that was first noted by [6, 5] is their potential for truly scaling computation in a sound and trustless manner. As articulated by Babai et al.: *"a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware"* [5].

A STARK (even without ZK capabilities) can deliver on this promise in an extreme way, facilitating *exponential* savings in verification time and space (like compressing Bitcoin's blockchain to a logarithmic size proof that would attest to the validity of its latest UTXO set); notably, a *transparent* proof system achieves this exponential compression without any auxiliary key management issues and their associated trust assumptions and governance problems.

Organization of the paper In Section 2 we define the notions STIK and STARK and state the theorems backing our construction (proofs appear in the full online version [12]). Section 3 compares our work to other ZK solutions, theoretically and practically. Section 4 explains the main novel components in our IOP and STARK constructions, showing how the asymptotic efficiency of Theorems 1 and 2 is translated to concrete efficiency of the realized system. In Appendix A we provide a self-contained overview of the ZK-STARK protocol from start to end, along with an example "toy problem" to assist readers unfamiliar with ZK-IOP constructions. Full details appear in the online version [12].

Acknowledgements We thank Arie Tal, Yechiel Kimchi and Gala Yadgar for help optimizing code performance. We thank the Andrea Cerulli, Venkitasubramaniam Muthuramakrishnan, Madars Virza, and the other authors of [30, 1] for assistance in obtaining the data reported in Figure 2. We thank Alessandro Chiesa, Yuval Ishai and the anonymous referees for commenting on earlier drafts of this paper.

2 Theory — Definitions and Main Results

This section describes our theoretical contributions. After recalling the interactive oracle proof model, we define a particularly efficient class of IOP protocols called scalable and transparent IOPs of knowledge (STIK), present our main theorems for this model (proofs omitted due to space limitations) and define the notion of a STARK.

2.1 Interactive Oracle Proofs (IOP)

The IOP[7] model suggested in [22, 67] is a generalization of the IP [44], PCP [2], and interactive PCP (IPCP) [53] models. It is an information theoretic model in which soundness can be proven unconditionally, as in the PCP, IP and MIP models. But, like those earlier models, the IOP model is unrealistic. To realize it, additional cryptographic assumptions are needed, and those are discussed later.

Remark 1 (The computational integrity language). Our statements and constructions apply to large classes of languages (like NP and NEXP). But we advise the reader to focus on the specific *computational integrity* (CI) language L (also called the *universal language* and the *bounded-halting language*), comprised of quadruples $(\mathsf{C}, x, y, \mathsf{T})$

[7] Reingold et al. [67] use the name "Probabilistically Checkable Interactive Proofs" (PCIP).

such that the computation specified by a program C, on public input x and auxiliary (private) witness w, reaches output y within T cycles. In fact, to achieve scalable verification it is *necessary* to use succinctly represented instances, such as sequential programs that are short and require execution time that is greater than the program description.

Informally, during an IOP protocol for a nondeterministic language L the prover and verifier receive public input $\mathbb{x}$ and then interact over a number of rounds; the prover's goal is to establish in zero knowledge that it knows a nondeterministic witness $\mathbb{w}$ for the fact that $\mathbb{x}$ belongs to L. During each round the verifier sends a message (in the case of transparent IOPs, like ours, all messages are public random coins), and the prover replies with an oracle, a long message which the verifier may query at random locations and need not read in entirety (jumping ahead, these oracles will be implemented in our ZK-STARK using Merkle-tree commitments). The verifier may query these oracles at any time during the interaction but for transparent systems (like ours) all queries can be postponed to the very last stage, after all prover-side oracles have been sent. Once the interaction has terminated and the verifier has made the required queries, it posts a decision — whether to accept $\mathbb{x}$ as a member of L or to reject it. Completeness means that an honest prover knowing $\mathbb{w}$ will succeed in making the verifier accept with probability 1, soundness means that for $\mathbb{x} \notin \mathsf{L}$ the prover has only negligible probability ϵ of convincing the verifier to accept, and knowledge soundness means that a prover succeeding with probability $\gg \epsilon$ in convincing the verifier to accept $\mathbb{x}$ has provided oracles that, if opened, will be found to encode a witness $\mathbb{w}$ that shows $\mathbb{x} \in \mathsf{L}$ directly. We now present the formal definitions.

A nondeterministic machine[4] M that decides a language $\mathsf{L} \in \mathsf{NTIME}(T(n))$ in time $T(n)$ (n denotes instance size) *induces* a binary relation R_M consisting of all pairs $(\mathbb{x}, \mathbb{w})$ where $\mathbb{x} \in \mathsf{L}$ and $\mathbb{w}$ is a sequence of nondeterministic choices of $M(\mathbb{x})$ that lead to an accepting state. In this case we say $\mathsf{R} = \mathsf{R}_M$ is *induced* by L and implicitly assume M is fixed and known. We recall the IOP definition from [22].

Definition 1 (Interactive Oracle Proof (IOP)). *Let* R *be a binary relation induced by a nondeterministic language* L *and let* $\epsilon \in [0, 1]$ *denote* soundness error. *An* Interactive Oracle Proof (IOP) *system* S *for* R *with soundness* ϵ *is a pair of interactive randomized algorithms* $\mathsf{S} = (\mathsf{P}, \mathsf{V})$ *that satisfy the properties below;* P *is the* prover *and* V *is the* verifier.

- **operation:** *The input of the verifier is* $\mathbb{x}$*, and the input of the prover is* $(\mathbb{x}, \mathbb{w})$ *for some string* $\mathbb{w}$*. The number of interactive rounds, denoted* $\mathsf{r}(\mathbb{x})$*, is called the* round complexity *of the system. During a single round the prover sends a message (which may depend on* $\mathbb{w}$ *and prior messages) to which the verifier is given oracle access, and the verifier responds with a message to the prover. We denote by* $\langle \mathsf{P}(\mathbb{x}, \mathbb{w}) \leftrightarrow \mathsf{V}(\mathbb{x}) \rangle$ *the output of* V *after interacting with* P*; this output is either* accept *or* reject.
- **completeness** *If* $(\mathbb{x}, \mathbb{w}) \in \mathsf{R}$ *then*

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

- **soundness** *If* $\mathbb{x} \notin \mathsf{L}$ *then for any* P^**,*

$$\Pr\left[\langle \mathsf{P}^* \leftrightarrow \mathsf{V}(\mathbb{x}) \rangle = \mathsf{accept}\right] \leq \epsilon$$

The proof length*, denoted $\ell(\mathbb{x})$, is the sum of lengths of all messages sent by the prover. The* query complexity *of the protocol, denoted* $\mathsf{q}(\mathbb{x})$*, is the number of entries read by* V *from the various prover messages. Given witness* $\mathbb{w}$ *such that* $(\mathbb{x}, \mathbb{w}) \in \mathsf{R}$*,* prover complexity*, denoted* $\mathsf{tp}(\mathbb{x}, \mathbb{w})$*, is the complexity required to generate all prover messages, and* verifier complexity*, similarly defined, is denoted* $\mathsf{tv}(\mathbb{x})$.

2.2 ZK-STIK

Next, we introduce the definition of a scalable and transparent IOP of knowledge (STIK). Most of the work described in later sections is related to constructing a new, concretely efficient, ZK-STIK; soundness is proved *information-theoretically*, with no cryptographic assumptions.

Definition 2 (Scalable Transparent IOP of Knowledge (STIK)). *Let* R *be a binary relation induced by a nondeterministic language* $\mathsf{L} \in \mathsf{NTIME}(T(n))$ *for* $T(n) \geq n$ *and let* $\mathsf{S} = (\mathsf{P}, \mathsf{V})$ *be an IOP for* L *with soundness error* $\epsilon(n) < 1$*. We say* S *is*

- **transparent** *if all verifier messages and queries are public random coins.*
- **(doubly) scalable** *if for every instance* $\mathbb{x}$ *of length* n*, both of the following hold:*
  1. **scalable verifier:** $\mathsf{tv}(n) = \mathsf{poly}(n, \log T(n), \log 1/\epsilon(n))$
  2. **scalable prover:** $\mathsf{tp}(n) = T(n) \cdot \mathsf{poly}(n, \log T(n), \log 1/\epsilon(n))$
- **a proof of knowledge** *if there exists a* knowledge error function $\epsilon'(n) \in [0, 1]$ *and a randomized extractor* E *that, given oracle access to any prover* P^* *that causes the verifier to accept* $\mathbb{x}$ *with probability* $p(n) > \epsilon'(n)$*, outputs in expected time* $\mathsf{poly}\left(\frac{T(n)}{p(n)-\epsilon'(n)}\right)$ *a witness* $\mathbb{w}$ *such that* $(\mathbb{x}, \mathbb{w}) \in \mathsf{R}$.
- **witness indistinguishable (privacy preserving)** *if there exists a randomized simulator* Sim *that samples (perfectly) the distribution on transcripts of interactions between* V *and* P*, and runs in time* $\mathsf{poly}(T(n))$.

A (doubly) scalable and transparent IOP of knowledge *will be denoted by* STIK*. A witness indistinguishable* STIK *is denoted by wi-*STIK*, and when* $T(n) = \mathsf{poly}(n)$ *it will be called a* zero knowledge STIK*, denoted ZK-*STIK.

Remark 2 (Zero knowledge vs. witness indistinguishability). In this work we construct (ZK) simulators that run in time that is polynomial in the *prover's* running time. For languages in NP, prover and verifier running times are both polynomial in the input size, so our simulator gives perfect zero knowledge. However, for languages in super-polynomial time, as stated in Theorem 2, our simulator only shows that the system is witness indistinguishable. The question of presenting a *succinct* simulator is left as an interesting open question; cf. [14] where a similar ZK simulator of NEXP is presented for a different IOP construction.

Remark 3 (History). PCP systems are, by definition, transparent (1-round) IOP systems. The first such system with a scalable verifier was given in the works[8] of Babai et al. [6, 5] and the first doubly scalable PCP, i.e., the first STIK construction, appears in

[8] The first work [6] shows this for NEXP and the second [5] scales it down to NP.

the works[9] of Ben-Sasson et al. [25, 17]. The first $\mathsf{ZK\text{-}STIK}$ for NP appears in the work of Ben-Sasson et al. [16], later extended to a $\mathsf{ZK\text{-}STIK}$ for NEXP [13].

For languages with logarithmic space our construction in Theorem 1 has prover and verifier complexity that are asymptotically better than previous constructions, and lead to a *strictly scalable* construction in arithmetic complexity, as defined next.

Definition 3 (Strictly scalable IOPs). *Using the notation of Definition 2, we say that* S *is a* strictly scalable transparent IOP of Knowledge (strict STIK) *if for every instance* $\mathbb{x}$ *of length* n*, both of the following hold:*

1. **strictly scalable verifier:** $\mathsf{tv}(n) = O(\log T(n)) + \mathsf{poly}(n, \log 1/\epsilon(n))$
2. **strictly scalable prover:** $\mathsf{tp}(n) = O(T(n) \log T(n)) + \mathsf{poly}(n, \log 1/\epsilon(n))$

When the complexity of prover and verifier is measured as the number of arithmetic operations over a finite field of size $O(T(n))$*, we say that* S *is a* strict arithmetic STIK.

2.3 Main Theorems

We now state the two main theorems regarding IOP systems that underlie our construction. IOP constructions use finite fields, so prover and verifier complexity are most naturally stated using arithmetic complexity over the ambient field, the size of which is derived from the size of the instance $\mathbb{x}$; we use $\mathsf{tv}^{\mathbb{F}}$ and $\mathsf{tp}^{\mathbb{F}}$ to denote arithmetic complexity, assuming the field $\mathbb{F}$ is understood from context. In contrast to other ZK approaches, the size of the field does not need to grow with the security parameter. In particular, our $\mathsf{libSTARK}$ implementation [10] uses the finite field of size 2^{64}, and could use even smaller fields, yet achieves soundness error $2^{-128} \ll 1/|\mathbb{F}|$. This unlinking of the security parameter from the ambient field size is one reason (out of several) our $\mathsf{libSTARK}$ prover is fast.

Let $\mathsf{NTimeSpace}(T(n), S(n))$ denote the class of nondeterministic languages that are decidable in simultaneous time $T(n)$ and space $S(n)$. Our first theorem applies to space bounded sequential computations.

Theorem 1 (ZK-STIK for space bounded computations). *Let* L *be a language in* $\mathsf{NTimeSpace}(T(n), S(n)), T(n) \geq n$ *and let* R *be induced by* L*. Then* R *has a transparent witness indistinguishable IOP of knowledge with the following parameters, stated for soundness error* $\mathsf{err} = 2^{-\lambda}$ *(that may depend on* n*)*

- *perfect completeness and soundness error at most* $\mathsf{err}(n)$ *for instances of size* n
- *knowledge error bound* $\mathsf{err}'(n) = O(\mathsf{err}(n))$
- *round complexity* $\mathsf{r}(n) = \frac{\log T(n)}{2} + O(1)$
- *query complexity* $\mathsf{q}(n) = 36(\lambda + 2) \cdot (\log T(n) + \frac{S(n)}{\log T(n)} + O(1))$
- *alphabet size: each query answer belongs to a binary field* $\mathbb{F}, |\mathbb{F}| = 2^{\mathsf{n}}$ *for* $\mathsf{n} = \lambda + \log T(n) + O(1)$

[9] The first work [25] presents a PCP with scalable verification and quasi-linear *proof length*, the second work [17] bounds the prover running time and also proves the proof of knowledge property.

- *verifier arithmetic complexity* $\mathsf{tv}^{\mathbb{F}}(n) = \tilde{O}(n) + O(\lambda \cdot (\frac{S(n)}{\log T(n)} + \log T(n))$
- *prover arithmetic complexity* $\mathsf{tp}^{\mathbb{F}}(n) = O(S(n) \cdot T(n))$
- *proof length* $O(S(n) \cdot T(n)/\log T(n))$*, measured in field elements.*

In particular, for $S(n) = \mathsf{poly}\log T(n)$*, this IOP is doubly scalable, i.e., the system is a wi-*STIK *(see 2). Moreover, for* $S(n) = O(\log T(n))$ *the IOP is a* strict arithmetic STIK *(see Definition 3), meaning the prover arithmetic complexity is* $O(T(n)\log T(n))$ *and verifier arithmetic complexity is* $O(\log T(n)) + \mathsf{poly}(n)$*. Finally, when* $T(n) = \mathsf{poly}(n)$*, the system has perfect ZK, i.e., it is a ZK-*STIK*.*

For computations with super-poly-logarithmic space the theorem above is not scalable, neither for prover nor for verifier. The following theorem is doubly scalable for any nondeterministic language, i.e., it can be said to be a *universal* wi-STIK (see Remark 1). Comparing Theorem 2 to the previous Theorem 1, the following result is more general, as it makes no assumptions regarding space. For computations requiring space $S(n) = o(\log^2 T(n))$ Theorem 1 has lower asymptotic prover complexity, but for $S(n) = \omega(\log^2 T(n))$ the more general Theorem 2 has more efficient prover complexity.

Theorem 2 (wi-STIK for NEXP). *Let* L *be a language in* $\mathsf{NTIME}(T(n)), T(n) \geq n$ *and* R *be induced by* L*. Then* R *has a doubly scalable, transparent, and witness indistinguishable (see 2) IOP of knowledge (wi-*STIK*) with the following parameters, stated for soundness error* $\mathsf{err} = 2^{-\lambda}$ *(that may depend on* n*)*

- *perfect completeness and soundness error* $\mathsf{err}(n)$ *for instances of size* n
- *knowledge extraction bound* $\mathsf{err}'(n) = O(\mathsf{err}(n))$
- *round complexity* $\mathsf{r}(n) = \frac{\log T(n)}{2} + O(1)$
- *query complexity* $O(\lambda \cdot \log T(n))$
- *alphabet size: each query answer belongs to a binary field* $\mathbb{F}, |\mathbb{F}| = 2^{\mathsf{n}}$ *for* $\mathsf{n} = \lambda + \log T(n) + \log\log T(n) + O(1)$
- *verifier arithmetic complexity* $\mathsf{tv}^{\mathbb{F}}(n) = \tilde{O}(n) + O(\lambda \cdot \log T(n))$*,*
- *prover arithmetic complexity* $\mathsf{tp}^{\mathbb{F}}(n) = O(T(n)\log^2 T(n))$*,*
- *proof length* $O(T(n)\log T(n))$*, measured in field elements.*

For $T(n) = \mathsf{poly}(n)$ *the system has perfect ZK, i.e., it is a ZK-*STIK*.*

We point out that this is the first construction of a scalable ZK-IOP system with strictly quasi-linear ($O(T(n)\log T(n))$) proof length and strictly logarithmic ($O(\log T(n))$) query complexity. Prior IOP systems, even without ZK, required query complexity $\log^c T(n)$ for exponent $c > 1$ for any quasi-linear length proofs [17, 9, 13].

2.4 STARK as a realization of STIK

Definition 2 refers to the IOP model, in which results can be proved with no cryptographic assumptions. A number of fundamental transformations have been suggested in the past to realize PCP systems using various cryptographic assumptions, and these transformations were adapted to the IOP model [22]. In all such realizations the prover must be computationally bounded, and such systems are commonly called *argument systems*, and, consequently, the realization of a STIK results in a *Scalable Transparent ARgument of Knowledge* (STARK).

The two main transformations of proof systems into realizable argument systems are:

- **Interactive** STARK (iSTARK) As shown by Kilian [54] for the PCP model, a family of collision-resistant hash functions can be used to convert a STIK into an interactive argument of knowledge system; if the STIK has perfect ZK, then the argument system has computational ZK. Any realization of a STIK using this technique will be called an *interactive* STARK (iSTARK); when one wants to emphasize that the STIK is zero knowledge, the term ZK-iSTARK will be used.
- **Non-interactive** STARK (nSTARK) As shown by Micali [61] and Valiant [77] for the PCP model, and by Ben-Sasson et al. [22] for the IOP model, any STIK can be compiled into a non-interactive argument of knowledge in the random oracle model (called a *non-interactive random-oracle proof (NIROP)* there); if the STIK had perfect zero knowledge then the resulting construction has computational zero knowledge. Any realization of a STIK using this technique will be called an *non-interactive* STARK (nSTARK); when one wants to emphasize that the STIK is zero knowledge, the term ZK-nSTARK will be used.

While non-interactive STARKs have the advantage of being comprised of a single message from the prover, they also rely on stronger assumptions. Thus, we leave the choice of which particular realization mode to use for a (ZK)-STIK— (ZK)-iSTARK vs. (ZK)-nSTARK— to be made by system designers based on particular use cases, and refer to both realization modes of a STIK as a STARK; to emphasize the ZK aspect of the STIK we may refer to the realization as a ZK-STARK.

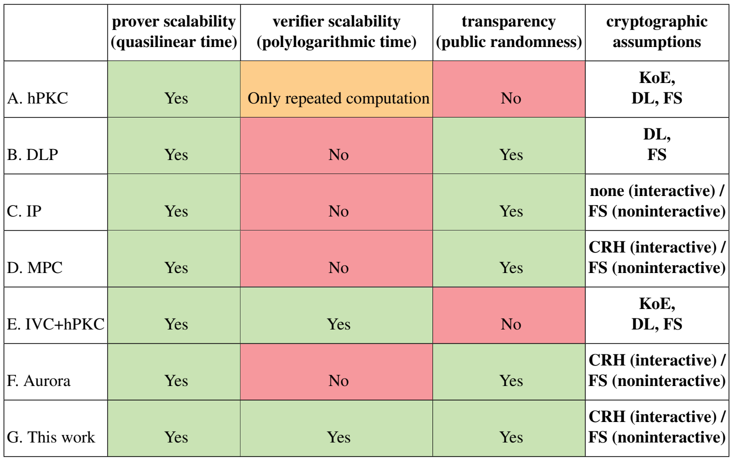
3 Evaluation and comparison

In this section we compare our ZK-STARK to other implemented systems. We start in Section 3.1 by comparing our approach to other implemented ZK approaches from a purely *asymptotic* and *theoretical* point of view, and show that the combination of full scalability, transparency and lean cryptographic assumptions for *universal* computations is *unique* to our system. We continue in Section 3.2, where we measure implemented systems for similar circuit size and topology as that which our system deals with. In Section 3.3 we compare our system to the previous state-of-the-art IOP system, called SCI [9], and show our system is faster while also adding ZK, which SCI did not obtain (see Remark 4 for a discussion of performance compared to the recent Aurora system [21]).

3.1 Comparison to prior works — theory

The literature on ZK realizations is vast, and rapidly expanding, so we limit the discussion to approaches that are *ZK and universal*, i.e., apply to any language in NP (thus, we sadly omit reference to many *verifiable computation* approaches that do not achieve ZK, like the recent [81]). For the purposes of this discussion, we consider four properties: asymptotic (i) prover scalability (quasilinear running time), (ii) asymptotic verifier scalability (poly-logarithmic verification time, including setup/parameter generation time), (iii) transparency (public randomness), and (iv) cryptographic assumptions.

Figure 1 summarizes our discussion, and we provide details next. Later, when we evaluate the performance of our system against other methods (Section 3) we will use the classification below.



	prover scalability (quasilinear time)	verifier scalability (polylogarithmic time)	transparency (public randomness)	cryptographic assumptions
A. hPKC	Yes	Only repeated computation	No	KoE, DL, FS
B. DLP	Yes	No	Yes	DL, FS
C. IP	Yes	No	Yes	none (interactive) / FS (noninteractive)
D. MPC	Yes	No	Yes	CRH (interactive) / FS (noninteractive)
E. IVC+hPKC	Yes	Yes	No	KoE, DL, FS
F. Aurora	Yes	No	Yes	CRH (interactive) / FS (noninteractive)
G. This work	Yes	Yes	Yes	CRH (interactive) / FS (noninteractive)

*Fig. 1: Theoretical comparison of universal (*NP *complete) realized ZK systems. KoE stands for “knowledge of exponent” assumptions, DL for “hardness of discrete log”, CRH for “collision resistant hash” and FS for Fiat-Shamir heuristic.*

A. Homomorphic public-key cryptography (hPKC): This approach, initiated by Ishai et al. [50] (for the “designated verifier” case) and Groth [45] (for the “publicly verifiable” case), uses an efficient information-theoretic model called a “linear PCP” that is then “compiled” into a cryptographic system using hPKC. An extremely efficient instantiation, based on Quadratic Span Programs, was introduced by Gennaro et. al [41] (see [49, 40, 58, 29, 47, 48] for related work and further improvements). It serves, e.g., as the proof system behind Zerocash and Zcash™. The first implementation of a QSP based system is called Pinocchio [63], with subsequent implementations including libSNARK [19, 68] which is used in the Zerocash and Zcash™ implementations; additional implementations appear in [70, 73, 72, 71, 24, 79, 37, 82].

The theoretical differences between hPKC and ZK-STARK are the lack of transparency and the reliance on number-theoretic knowledge of exponent assumptions (which are vulnerable to attacks by quantum computers). Verification time in hPKC is scalable only for computations that are repeated many times, because the hPKC “setup phase” requires time $\geq \mathsf{T}$, where T denotes running time of the nondeterministic computation[4] being verified.

B. Discrete logarithm problem (DLP): An approach initiated by Groth [46] (cf. [69]) and implemented in [30], relies on the hardness of the DLP to construct a system that is transparent. Shor's quantum factoring algorithm solves the DLP efficiently, rendering this approach quantum-susceptible. Additionally, verifier complexity in the DLP approach requires time $\geq \mathsf{T}_{\mathsf{C}}$ hence it is non-scalable (according to our definition of the term), although communication complexity in the DLP approach is logarithmic. We refer to the initial implementation of this system as BCCGP [30], and a recent improved version is called BulletProofs [31].

C. Interactive Proofs (IP) based: IP protocols can be performed with zero knowledge [8] but only recently have IP protocols been efficiently "scaled down" to small depth (non-sequential) computations via so-called "proofs for muggles" of Goldwasser et al. [43, 67]. This led to a line of realizations in code, early works lacked ZK [36, 35, 76, 78], but the state-of-the-art ones, like [82] and Hyrax [80], do have it.

Like ZK-STARK, most of these IP-based proofs (but for [82]) are transparent and have a scalable prover, but their verifier is not scalable, as its running time grows linearly with computation time for "standard" (i.e., sequential) computations. In terms of cryptographic assumptions, some are plausibly post-quantum secure while others rely on number theoretic assumptions that are susceptible to quantum attacks.

D. Secure multi-party computation (MPC): This approach, suggested by Ishai et al. [51] and implemented first in the ZKBoo [42] system, and more recently, in Ligero [1], "compiles" secure MPC protocols into ZK-PCP systems, by requiring the prover to commit to the transcript of a secure MPC protocol, and then reveal the view of one of the parties.

Like ZK-STARK, the MPC-based proofs are transparent and have scalable (quasi-linear) proving time. However, MPC based systems have a non-scalable verifier, one that runs in time $\geq \mathsf{T}$. Additionally, their communication complexity is non-scalable, it is $\sqrt{\mathsf{T}}$ in the state of the art system [1]; nevertheless, for concrete circuits and amortized computations verification time and communication complexity are extremely efficient.

E. Incrementally Verifiable Computation (IVC): This approach, suggested by Valiant [77] (cf. [34, 28]) reduces prover space consumption by relying on knowledge extraction assumptions; this approach can be applied on top of other proof systems with succinct (sub-linear) verifiers, including ZK-STARK, but thus far has been realized only for a single hPKC system [23].

Compared with ZK-STARK, systems built this way inherit most properties from the underlying proof system. In particular, the hPKC-based IVC is non-transparent and quantum-susceptible; however the verifier is scalable even for a computation executed only once, because the setup phase runs in poly-logarithmic time.

F. Aurora: The Aurora system is a recently posted ZK-IOP by Ben-Sasson et al., that is optimized for arithmetic circuits [21]. For a circuit with N gates, prover running time is scalable — $O(N \log N)$ arithmetic operations over the ambient field — and proof length scales succinctly, poly-logarithmically in N. However, verification time scales *linearly* in N. Aurora shares many similarities with our ZK-STARK: both are IOP-based, plausibly post-quantum secure and require only symmetric cryptographic assumptions (for the interactive setting; the non-interactive one relies on the Fiat-Shamir heuristic). Furthermore, both use the FRI protocol for asserting proximity to RS codes. The main difference between Aurora and our system regards verifier time: Aurora's verifier scales linearly with the computation size whereas our system has poly-logarithmic verification time.

Summary

ZK-IOPs have a combination of beneficial attributes not achieved by any other code-realized approach; these are *full* scalability (prover- and verifier-side) and transparency. Additionally, the cryptographic assumptions needed by the ZK-IOP approach are rather minimal, although obtained by other approaches — MPC and IP. As we shall see later, the theoretical attributes are complemented by practical benefits, like the *fastest* proving time for ZK proofs of sequential computations.

3.2 Comparison to prior works — concrete performance

In this section we compare measurements of different ZK systems on the same hardware, a server with 32 AMD cores at clock speed of 3.2GHz, and 512GB of DDR3 RAM. Each pair of cores shares memory; this roughly corresponds to a machine with 16 cores and hyper-threading.

Comparison method All *prior* realized ZK systems we are aware of use arithmetic circuits over prime fields, and their complexity is mostly affected by arithmetic circuit (i) depth and (ii) size — the number of addition and multiplication gates; typically multiplication complexity dominates addition complexity. (See Remark 4 for a discussion of our system compared to the recent Aurora system [21].) Since these systems are affected mostly by the circuit topology — size and depth — the exact nature of the computation (beyond these parameters) does not significantly affect their complexity measures.

To generate circuits for other systems, we started with a program written in TinyRAM assembly [18] — the exhaustive subset-sum program reported for SCI in [9]. This computation does not access RAM memory, which is a requirement when comparing to other ZK systems that deal with circuits, not RAM machines (in the next section we shall also discuss RAM computations, when comparing ZK-STARK to SCI). This program was compiled into a ZK-STARK system, and also into a set of quadratic arithmetic program (QAP) constraints by libSNARK. This offers a rather direct comparison between the following three systems — SCI (an IOP system with no ZK), libSNARK (an hPKC system, with ZK) and our ZK-STARK. All three apply to the same computation, running on the same machine, and use multi-threading (see Remark 5 for a more thorough discussion of the comparison method).

We extracted depth and multiplication complexity numbers from the libSNARK compiler and requested the authors of the following systems to measure them on our server for arbitrary circuits with similar depth and multiplication complexity. Figure 1 shows the resulting proving time, verifying time and communication complexity. Since several of the systems operate only in single-threaded proving mode (all systems use single-thread for verification), we have a separate comparison of single-threaded ZK-STARK vs. the other single-treaded systems. Recall the classification of ZK approaches

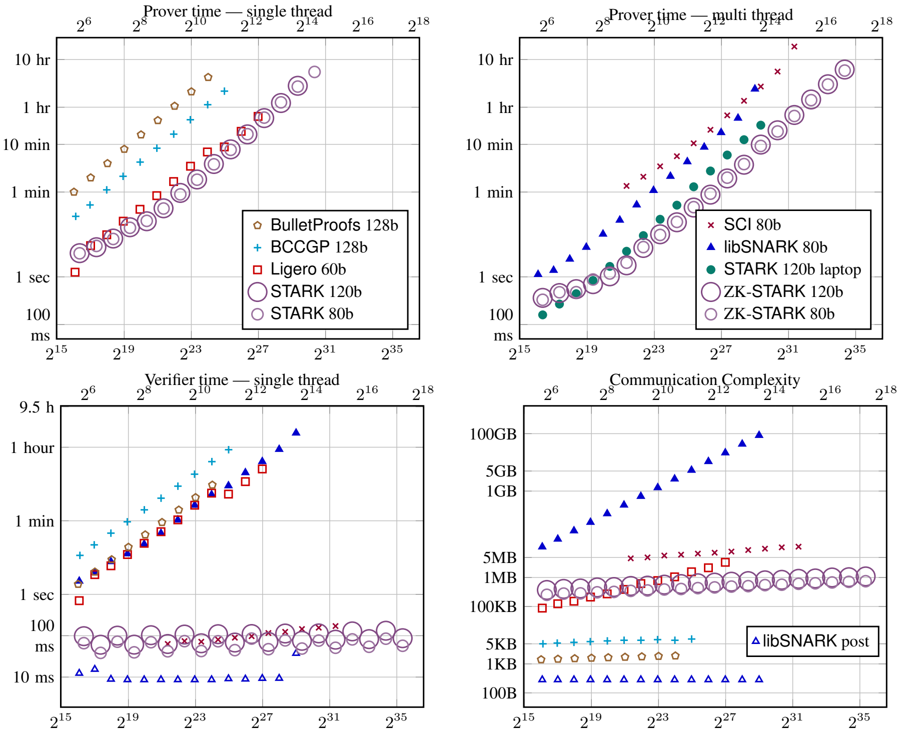


Fig. 2: A comparison of different realized proof systems as a function of the number of machine cycles (top axis) and multiplication gates (bottom axis); each cycle of the TinyRAM program corresponds to $\approx 2000 \approx 2^{11}$ multiplication gates. The estimated level of security of each system is denoted on the legend above (e.g., "STARK 80b" means estimated soundness error of $\leq 2^{-80}$). All systems were tested on the same server (specs below) and executed a computation of size and structure corresponding to the "exhaustive subset-sum" program from [9, Section 3]; our ZK-STARK was also *executed on the same program on a weaker laptop (quad core i7-8550U CPU @ 1.80GHz clock with 32GB of DDR4 RAM), see right top plot. Notice that even on this weaker machine the ZK-STARK prover is faster, and reaches larger circuit size, than all other systems.*

from Section 3.1. The systems that have performed the above testing procedure on our machine are:

- hPKC-based: libSNARK with 80 bits (80b) of security (commit dc78fd, September 7, 2017);
- DLP-based: The system of BCCGP with logarithmic communication complexity [30], and the BulletProofs system of [31]; both systems are single-threaded and have 128b security.
- MPC-based: Ligero strong with 60b security, single-threaded [1] (this system has sublinear communication complexity, compared with linear complexity of ZKBoo, hence we include only it in our measurements).

Regarding ZK-STARK, we evaluated it in single- and multi-theard mode, for 80 and 120 bits of security, using Blake2s (with 128-bits of security) as our CRH for constructing the Merkle tree commitments to oracles. To address concerns about the ability to execute ZK-STARK on weaker machines, we also plot the measured proving time on a Lenovo T440 laptop with 32GB of DDR4 RAM and a quad-core Intel i7-8550U CPU 1.80GHz clock speed.

Let us discuss prover time, verifier time, and communication complexity, addressing the systems above. We hope to add measurements for IP based systems like Hyrax in the future [80].

Prover complexity All systems surveyed here have prover complexity that scales either linearly or nearly-linearly in computation size. However, as shown in Figure 2, our ZK-STARK prover is the fastest among the single-threaded systems (though not by a large margin) and is at least $10\times$ faster than the second fastest prover (that of libSNARK) when multi-threading is allowed; all systems were tested up to maximal proving time of 12 hours. Notice that even when executed not on a large server but on a weaker laptop with 32 GB of RAM, our ZK-STARK prover is noticeably faster, and reaches larger circuit size, than all other systems (which were measured only on the stronger and bigger server). This shows that ZK-STARK proving efficiency is not an artifact of using a strong machine, but rather follows from the efficiency of the underlying protocol (the interested reader is welcome to test libSTARK on her laptop, using the runSubsetsumTests.sh procedure there [10].)

The speedup of multi-threaded over single-threaded execution of libSTARK on the server is plotted in Figure 3. For very small instances multi-threading gives moderate improvements, possibly due to short running time and cost of opening many threads, and for very large instances it drops somewhat, perhaps because memory swapping contributes more significantly to running time.

Verifier complexity The *total* verifier running time (including setup/parameter generation and post-processing) of all prior works grows at least like $\sqrt{\mathsf{T}}$, and, often, like T; in contrast, our ZK-STARK scales like $\mathsf{a} + \log \mathsf{T}$ (see Theorems 1 and 2). Consequently, for medium- and large-scale *sequential* computations our ZK-STARK total verification time is better than all prior solutions, as shown by Figure 2. The efficiency of ZK-IOP systems tailored specifically for small depth, parallel computations (the setting which Hyrax is tailored to) is left to future work.

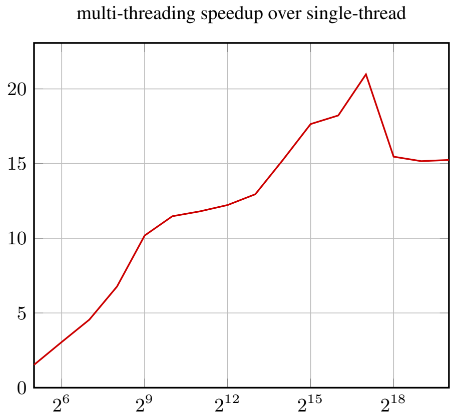


*Fig. 3: The ratio of multi-threaded to single-threaded proving time of ZK-*STARK *for the exhaustive subset-sum computation, as a function of the number of cycles. Recall that the server used for testing has 32 AMD cores, which correspond to 16 cores with hyperthreading.*

hPKC-based systems like Pinocchio and libSNARK, and IVC+hPKC systems like that of [23] are different in this respect. They have a setup that is performed only once per circuit. For Pinocchio and libSNARK pre-processing time grows *linearly* with circuit size. E.g., the libSNARK system requires $\approx$ 16 seconds for a computation with 2^{20} gates. In Figure 1 we plot both post-processing verification time (and CC) using open blue triangles and total time/CC (including setup) using filled blue triangles. For the IVC+hPKC system, pre-processing time is *constant* and does not depend on circuit size; this constant ($\approx$ 10 seconds) is quite large compared to our verifier time, but on the other hand is needed only once, so amortized over many computations it approaches 0.

Communication complexity (CC) The use of a pre-processing phase in the hPKC and IVC+hPKC systems leads to extremely small post-processing CC; the BCCGP and BulletProofs systems also enjoy extremely short CC and, because pre-processing is transparent, can be effectively replaced with a short seed to a pseudo-random generator. Concretely, for all computations measured in practice, post-processing CC of Pinocchio, libSNARK and the IVC+hPKC system are less than 300 bytes, that of BCCGP is less than 7KB, and BulletProofs is roughly $3\times$ smaller, less than 2.5KB [30, 31] (see also Figure 2). However, pre-processing key length scales linearly with circuit size for hPKC; the IVC+hPKC system is different in this respect, it has succinct pre-processing length even for large computation size, but once again, this length is concretely large — more than 40 MB for our computation. For Ligero, communication complexity scales like $70\sqrt{\mathsf{mult}_n}$ field elements [1, Section 5.3].

Discussion

Among all ZK systems compared above, our ZK-STARK has the fastest prover in single- and multi-thread modes; in particular, it is $\approx 10\times$ faster than the second fastest measured system — libSNARK. Other systems perform better (shorter communication, faster verification) on small circuits (ZKBoo, Ligero), small-depth circuits (Hyrax), and on computations repeated many times with the same fixed circuit (BulletProofs, Pinocchio, libSNARK). However, for general large scale sequential computations our ZK-STARK has verification time and communication complexity that outperforms all other *transparent* systems published thus far for this range of parameters. In other words, our particular ZK-STARK realization shows that the asymptotic benefits of full scalability and transparency are manifested already for concrete computations, and suggest that ZK-IOP systems are of interest not merely as a theoretical construct but also as a viable approach to building future ZK-systems.

Remark 4 (Runtime comparison to Aurora). For computations that are specified simply as arithmetic circuits, Aurora out-performs our ZK-STARK (and Ligero) (see [21, Figures 10–12]). However, for sequential computations specified by succinct programs, verification time in our ZK-STARK out-performs that of Aurora. Concretely, Aurora verification time for a circuit with a million gates requires ~ 1 second (see Figure 12 there) and scales linearly with N, whereas our ZK-STARK verifier scales quite slowly and requires less than 0.1 seconds even for a circuit with 34 billion gates (see Figure 2).

Summarizing, we view Aurora and our ZK-STARK as complementary: both are IOP-based, transparent, plausibly post-quantum secure and have concretely efficient provers. Arora is better when dealing with computations specified as generic arithmetic circuits but does not offer full scalability, while our ZK-STARK is better when dealing with sequential programs because its verification time scales poly-logarithmically with computation time.

Remark 5 (On validity of the comparison method). The reader might ask whether the method outlined above — compiling the particular exhaustive subset-sum program into (i) arithmetic circuits over prime fields and (ii) AIRs over binary fields, is fair and valid. Wouldn't it better to "hand optimize" the circuit/AIR for a particular computation, and perhaps do it over the same ambient field?

The choice of program — the exhaustive subset-sum — was dictated by the constraint of including a comparison to SCI, the prior IOP state of the art; this limited us to choosing one of the programs provided there. Hand-optimizing AIRs and arithmetic circuits for the same computation for all the various proof systems surveyed here is beyond the scope of this work, as these systems are provided by different teams and some of the code-bases (SCI, for instance) are not updateable.

The compilation process that converts a program (in our case, written in TinyRAM assembly) to an arithmetic circuit, and to an AIR, leads to a construction that is less efficient than a "hand-written" circuit/AIR of the very same computation. It is hard to estimate which approach (AIR vs. circuits) suffers more from compilation inefficiency but the fundamental complexity measures for circuits and STARKs — number of gates per cycle (for arithmetic circuit), and "total degree" per cycle (= state width $\times$ constraint degree / code rate) — are roughly similar for this particular choice of program and compilation: roughly 2,000 multiplication gates per cycle (for arithmetic circuits), and total degree roughly 9,000 per cycle (because our program leads to 94 state width, the constraint degree is 12 and the code rate is $1/8$).

3.3 SCI vs. ZK-STARK

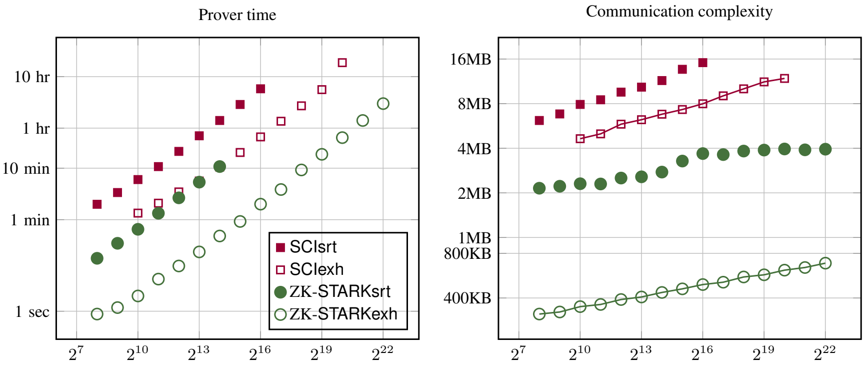


Fig. 4: SCI *vs. ZK-*STARK *comparison of prover time and communication complexity. Both systems measured at 80 bits of security on the same machine.*

To compare SCI and ZK-STARK we use the exact same pair of TinyRAM programs used by SCI and reported in [9], namely:

- exh: the exhaustive-search subset-sum program which does not require RAM access (no use of LOAD/STORE TinyRAM opcodes); this corresponds to Theorem 1
- srt: the sorted subset-sum program which does require RAM access (with LOAD-/STORE opcodes), corresponding to Theorem 2

Both systems were executed with an 80-bit security level and measured on the machine specified at the beginning of Section 3.2. Figure 4 shows that ZK-STARK prover time is $7\times$–$40\times$ faster than that of SCI and has communication complexity that is $3\times$–$20\times$ smaller than that of SCI. Notably, ZK-STARK has ZK, which SCI does not (the cost of adding ZK increases computational complexity across the board).

As pointed out earlier (Section 4), this improvement is due to the better arithmetization which uses many RS codewords (one per register), tighter soundness analysis, the use of the more efficient FRI protocol and the efficient additive FFTs of [57].

The improvement of ZK-STARK over SCI is more noticeable for the program that does not use RAM. The reason for this is verifying correct RAM requires certain tools that incur large blow-ups in communication complexity and prover time. These blowups are due to the need to verify that an arbitrary RAM access pattern was executed correctly. This is solved in both SCI and ZK-STARK using switching networks to “route” accesses to memory, following the method of [17]. We refer the reader to Appendices C.3 and G in the online version of this paper [12] for full details.

4 Novel ingredients in the construction

Our new ZK-STARK builds significantly on recent ZK-IOP research [22, 13, 15, 11], and its main advantage is *improved efficiency*, leading to it being the first strictly scalable IOP for space bounded computation (Theorem 1). Our main improvements are four, listed below. We briefly recount the prior state of the art as background and then explain how ZK-STARK improves on it.

Background — SCI and FRI The SCI system [9] is an IOP without zero knowledge. It uses an arithmetization process that reduces a witness of membership in a language to a *pair* of univariate polynomial, and reduces the transition function of the computation to a *single* low-degree multivariate polynomial. Then, it employs an IOP version of the quasilinear PCP of Proximity (PCPP) of [27] to solve the low-degree testing problem. This PCPP, and the IOP emerging from it, require quasi-linear proving time and poly-logarithmic verification time, but both algorithms are not *strictly* quasi-linear (cf. Definition 3). Due to the reliance on bivariate polynomials in that IOP, when converting it to an argument system via Merkle trees, different queries to the proof oracles led to different authentication paths, resulting in increased communication complexity.

Another component that is used in ZK-STARK (and in Aurora [21]) is the recent stricly quasi-linear IOP of proximity (IOPP) for univariate polynomials called FRI and discussed further below [11].

Improvements In addition to the qualitative improvement over SCI of adding ZK, our system is asymptotically and concretely more efficient in terms of verifier complexity and communication complexity than SCI, and has a prover that is more efficient, for sequential computations, than all other existing systems. The main novel components in ZK-STARK that facilitate this are:

1. ZK-STARK uses the FRI protocol of [11], which is vastly more efficient, both asymptotically and concretely, than the Ben-Sasson–Sudan PCPP used by SCI. Asymptotically, FRI has prover arithmetic complexity that is strictly linear in block-length (prior IOPPs required quasi-linear proving time) and strictly logarithmic verifier arithmetic complexity (prior verifiers required poly-logarithmic complexity, with an exponent greater than 1).
2. The FRI oracle structure is used by our ZK-STARK to significantly reduce Merkle-tree authentication path complexity; this aspect is explained in Section 4.1;
3. Our ZK-STARK uses an arithmetization with one RS codeword per register, as opposed to one RS codeword for *all* registers; we then use a round of interaction to solve the RPT problem only once over all different RS codewords; see Section 4.2.
4. in similar fashion to the step above, our new *algebraic linking IOP* (ALI) protocol "compresses" all of the constraints that enforce the computational integrity of the transition function, into a *single* random combination of them all. This dramatically reduces the memory and computational complexity of the prover. The specification of the ALI protocol and its analysis appear in the full version of the paper [12, Sections B.5, D].

Below we elaborate on the second and last items of the list above.

4.1 Reduced Authentication Path Complexity

The largest contributor to communication complexity, and to verifier time and space complexity in our ZK-STARK (and prior related works [27, 17, 33, 9]) is the cost of checking authentication paths. We now discuss the way our ZK-STARK reduces this cost. Let λ denote the number of output bits of the cryptographic hash function used to construct a Merkle tree in our system; let AP_{total} denote the total number of authentication path nodes in all subtrees of Merkle trees whose leaves are query answers, and let q_{total} denote the total number of queries, made to all proof oracles. The total communication complexity (CC) of the proof system is

$$\mathsf{CC} = \mathsf{q}_{total} \cdot \log |\mathbb{F}| + \mathsf{AP}_{total} \cdot \lambda \tag{1}$$

Compared to prior works, most notably SCI, our ZK-STARK reduces the second summand in two separate ways:

1. The ZK-STARK verifier queries *rows* of the (low degree extension of the) execution trace, each row comprises a field elements that represent the state at some point in the computation (or its low degree extension). To reduce communication complexity, the ZK-STARK prover places each such row in a *single* sub-tree of the Merkle tree, and therefore only *one* authentication path is required per row (as opposed to a many paths in prior solutions).
2. The verifier of the FRI protocol queries functions on cosets of a fixed subspace; i.e., the entries of each oracle accessed by the verifier can be *partitioned*, so that a single authentication path covers all entries required by the verifier in a single test. Accordingly, the ZK-STARK prover places each member of the partition in a *single* sub-tree of the Merkle tree, thereby reducing the number of authentication paths to one-per-coset (as opposed to one per field element).

4.2 Algebraic Linking Interactive Oracle Proof (ALI)

The main bottleneck for prover time and space complexity is the cost of performing *polynomial interpolation* and its inverse operation — multi-point *polynomial evaluation*. The complexity measure that dominates this bottleneck is the *maximal degree* of a polynomial which the prover must interpolate and/or evaluate; for a computation involving a $\mathsf{T} \times \mathsf{a}$ execution trace specified by s constraints of degree at most d, we denote this degree by $\mathsf{d}^{\max} = \mathsf{d}^{\max}(\mathsf{T}, \mathsf{a}, \mathsf{s}, \mathsf{d})$. Prior state-of-the-art [27, 17, 33, 9] gave

$$\mathsf{d}^{\max}_{\mathsf{old}}(\mathsf{T}, \mathsf{a}, \mathsf{s}, \mathsf{d}) = \mathsf{T} \cdot \mathsf{a} \cdot \mathsf{d} + \mathsf{T} \cdot \mathsf{s}. \tag{2}$$

which leads to concretely large values. Our ZK-STARK reduces $\mathsf{d}^{\max}$ to

$$\mathsf{d}^{\max}_{\mathsf{ZK-STARK}}(\mathsf{T}, \mathsf{a}, \mathsf{s}, \mathsf{d}) = \mathsf{T} \cdot \mathsf{d} \tag{3}$$

The improved efficiency of our ZK-STARK is due to two reasons, explained next. The first one completely removes the second summand of (2) and the second one removes a from its first summand.

*Algebraic linking IOP (*ALI*)* The second summand of (2) arises because our prover needs to apply a "local map" induced by the AIR constraint system. Prior state-of-the-art systems, like [9], used a local map that checks each constraint of the AIR separately, leading to this second summand. Instead, our ZK-STARK uses a single round of interaction to reduce all s constraints to a *single constraint* that is a *random linear combination* of all AIR constraints, thereby completely removing the second summand of (2). See [12, Sections B.5, D] for a specification of the protocol.

Register-based encoding Prior systems, like [9], encoded the *full* execution trace by a *single* Reed-Solomon codeword, leading to degree $\mathsf{T}\cdot\mathsf{a}$; this degree is then multiplied by d to account for application of the AIR constraints to this codeword, resulting in the first summand of (2). Our ZK-STARK uses a *separate* Reed-Solomon codeword for each register[10], leading to a many codewords, each of lower degree T. At first glance this tradeoff may seem wasteful, because we now have to solve an RPT problem for each of these a codewords. However, the interaction and use of randomness allowed by the IOP model once again come to our aid: it suffices to solve a *single* RPT problem, applied to a *random* linear combination of all a codewords. The use of a single codeword per register also helps with reducing communication complexity, as explained in Section 4.1.

4.3 Algebraic security assumptions

In our measurements (Section 3.2) we rely on two conjectures. Informally, the first, which appears in the full version [12, Conjecture B.17] due to space limitations, says that any efficient attacker will be presenting proof oracles $f^{(0)}, g^{(0)}$ that are maximally far from the respective RS codes, and the second, stated below, says that δ-far words are rejected by the FRI protocol with probability $\approx \delta$. Both conjectures match our current understanding of the best possible attacks against the ZK-STIK system; it is reasonable to use such an approach when running comparisons to other implemented systems, because all other systems use a similar "security-based" approach when setting parameters (group size in an elliptic curve, field size in a discrete-log based approach, bit-length in a cryptographic hash function, etc.). To be fair, these other assumptions have received more scrutiny than ours but by stating this conjecture we hope it, too, will be further inspected by the research community.

Conjecture 1 (FRI soundness — informal). For *any* rate parameter ρ and constant δ, if $f : S \to \mathbb{F}$ is δ-far from $\mathsf{RS}[\mathbb{F}, S, \rho]$, then the FRI protocol rejects f with probability at least $\delta - \frac{O(1)}{|\mathbb{F}|}$.

For a code of rate $\rho = 2^{-\mathcal{R}}$, the conjecture implies that to reach a security level of λ bits (or error probability $< 2^{-\lambda}$), the QUERY phase of the FRI protocol should be invoked $\lambda/\mathcal{R}$ times. See [11, 26] for a discussion of the conjecture.

Without Conjecture 1 and [12, Conjecture B.17], the number of FRI-verifier tests would increase at most *three-fold*, to $3 \cdot \lambda/\mathcal{R}$ (to achieve λ bits of security). This would entail a $\times 3$ increase in communication complexity and verifier running time (both scale linearly with the number of FRI-verifier tests), however, there would be no other change

[10] For simplicity, the current description discusses the case of space bounded computations; the case of computations with large space also uses multiple codewords but the reduction is more complicated, and discussed in the online version of the paper.

to the system parameters, such as field size, the schedule of reductions, etc. Regarding prover time — the main bottleneck in proof systems — the impact would be negligible ($< 1\%$ for all reasonable sized computations) because producing query answers requires only poly-logarithmic running time (whereas producing the proof requires quasi-linear running time and vastly dominates overall proving time).

We stress that in terms of security, our ZK-STARK is *qualitatively better* than most prior ZK approaches (but for Ligero and Aurora that are similar in this respect). Consider the effect of refuting, in the strongest possible way, either of the Knowledge of Exponent (KoE) or Discrete Log Problem (DLP) hardness assumptions discussed in Section 3.1, say, by an efficient algorithm that breaks them (or by a large scale quantum computer). In such a case, the systems relying on KoE/DLP would be rendered completely broken and useless. In stark contrast (pun intended), if Conjecture 1 and [12, Conjecture B.17] were to be refuted in the strongest possible way, the effect on ZK-STARK would only be to increase communication complexity and verifier complexity by a factor of $\leq \times 3$. This is thanks to *proven, information-theoretic* bounds that show that for any $\delta \leq 1 - \sqrt[3]{\rho} = 1 - 2^{-\mathcal{R}/3}$ the conjecture above is in fact a theorem (see [26] for more details)[11].

References

[1] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. 2017.

[2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof verification and the hardness of approximation problems". In: *Journal of the ACM* 45.3 (1998). Preliminary version in FOCS '92., pp. 501–555.

[3] Sanjeev Arora and Shmuel Safra. "Probabilistic checking of proofs: a new characterization of NP". In: *Journal of the ACM* 45.1 (1998). Preliminary version in FOCS '92., pp. 70–122.

[4] László Babai and Lance Fortnow. "Arithmetization: A new method in structural complexity theory". In: *computational complexity* 1.1 (1991), pp. 41–66. ISSN: 1420-8954. DOI: `10.1007/BF01200057`. URL: `http://dx.doi.org/10.1007/BF01200057`.

[5] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. "Checking computations in polylogarithmic time". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. STOC '91. 1991, pp. 21–32.

[6] László Babai, Lance Fortnow, and Carsten Lund. "Nondeterministic exponential time has two-prover interactive protocols". In: *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. FOCS '90. 1990, pp. 16–25.

[7] Mihir Bellare and Oded Goldreich. "On Defining Proofs of Knowledge". In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '92. 1993, pp. 390–420.

[8] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Hr astad, Joe Kilian, Silvio Micali, and Phillip Rogaway. "Everything Provable is Provable in Zero-Knowledge". In: *Proceedings of the 8th Annual International Cryptology Conference*. CRYPTO '89. 1988, pp. 37–56.

[9] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. "Computational integrity with a public random string from quasi-linear PCPs". In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 646. URL: `http://eprint.iacr.org/2016/646`.

[10] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. *libSTARK: a library for zero knowledge (ZK) scalable transparent argument of knowledge (STARK)*. `https://github.com/elibensasson/libSTARK`. URL: `https://github.com/elibensasson/libSTARK`.

[11] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Fast Reed-Solomon Interactive Oracle Proofs of Proximity". In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. 2018, 14:1–14:17. DOI: `10.4230/LIPIcs.ICALP.2018.14`. URL: `https://doi.org/10.4230/LIPIcs.ICALP.2018.14`.

[11] Our ZK-STARK still requires a collision resistant hash function, and in the interactive setting even the Fiat-Shamir heuristic, and, obviously, we make no information-theoretic claims on those.

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

[13] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. "On Probabilistic Checking in Perfect Zero Knowledge". In: *Electronic Colloquium on Computational Complexity (ECCC)* 23 (2016), p. 156. URL: `http://eccc.hpi-web.de/report/2016/156`.

[14] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. "Zero Knowledge Protocols from Succinct Constraint Detection". In: *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*. 2017, pp. 172–206. DOI: `10.1007/978-3-319-70503-3\_6`. URL: `https://doi.org/10.1007/978-3-319-70503-3\_6`.

[15] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. "Short Interactive Oracle Proofs with Constant Query Complexity, via Composition and Sumcheck". In: *Electronic Colloquium on Computational Complexity (ECCC)* 23 (2016), p. 46.

[16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. "Quasilinear-Size Zero Knowledge from Linear-Algebraic PCPs". In: *Proceedings of the 13th Theory of Cryptography Conference*. TCC '16. 2016, pp. 33–64.

[17] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. "On the Concrete Efficiency of Probabilistically-Checkable Proofs". In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC '13. 2013, pp. 585–594.

[18] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. "TinyRAM architecture specification v2. 00, 2013". In: *URL: http://scipr-lab. org/tinyram* ().

[19] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge". In: *Proceedings of the 33rd Annual International Cryptology Conference*. CRYPTO '13. 2013, pp. 90–108.

[20] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs". In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 2015, pp. 287–304. DOI: `10.1109/SP.2015.25`. URL: `http://dx.doi.org/10.1109/SP.2015.25`.

[21] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. *Aurora: Transparent Succinct Arguments for R1CS*. Cryptology ePrint Archive, Report 2018/828. To appear in Eurocrypt 2019, `https://eprint.iacr.org/2018/828`. 2018.

[22] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 31–60. ISBN: 978-3-662-53644-5. DOI: `10.1007/978-3-662-53644-5_2`. URL: `http://dx.doi.org/10.1007/978-3-662-53644-5_2`.

[23] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *Proceedings of the 34th Annual International Cryptology Conference*. CRYPTO '14. Extended version at `http://eprint.iacr.org/2014/595`. 2014, pp. 276–294.

[24] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture". In: *Proceedings of the 23rd USENIX Security Symposium*. Security '14. Extended version at `http://eprint.iacr.org/2013/879`. 2014, pp. 781–796.

[25] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. "Short PCPs Verifiable in Polylogarithmic Time". In: *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*. CCC '05. 2005, pp. 120–134.

[26] Eli Ben-Sasson, Swastik Kopparty, and Shubhangi Saraf. "Worst-Case to Average Case Reductions for the Distance to a Code". In: *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*. 2018, 24:1–24:23. DOI: `10.4230/LIPIcs.CCC.2018.24`. URL: `https://doi.org/10.4230/LIPIcs.CCC.2018.24`.

[27] Eli Ben-Sasson and Madhu Sudan. "Short PCPs with Polylog Query Complexity". In: *SIAM Journal on Computing* 38.2 (2008). Preliminary version appeared in STOC '05., pp. 551–607.

[28] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC '13. 2013, pp. 111–120.

[29] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. "Succinct Non-Interactive Arguments via Linear Interactive Proofs". In: *Proceedings of the 10th Theory of Cryptography Conference*. TCC '13. 2013, pp. 315–333.

[30] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. 2016, pp. 327–357. DOI: `10.1007/978-3-662-49896-5_12`. URL: `http://dx.doi.org/10.1007/978-3-662-49896-5_12`.

[31] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. *Bulletproofs: Efficient Range Proofs for Confidential Transactions*. Cryptology ePrint Archive, Report 2017/1066. `https://eprint.iacr.org/2017/1066`. 2017.

[32] Vitalik Buterin. 2017. URL: `https://vitalik.ca/`.

[33] Alessandro Chiesa and Zeyuan Allen Zhu. "Shorter arithmetization of nondeterministic computations". In: *Theor. Comput. Sci.* 600 (2015), pp. 107–131.

[34] Alessandro Chiesa and Eran Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: *Proceedings of the 1st Symposium on Innovations in Computer Science*. ICS '10. 2010, pp. 310–331.

[35] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. "Practical Verified Computation with Streaming Interactive Proofs". In: *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*. ITCS '12. 2012, pp. 90–112.

[36] Graham Cormode, Justin Thaler, and Ke Yi. "Verifying computations with streaming interactive proofs". In: *Proceedings of the VLDB Endowment* 5.1 (2011), pp. 25–36.

[37] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. "Square Span Programs with Applications to Succinct NIZK Arguments". In: *Advances in Cryptology – ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 532–550. ISBN: 978-3-662-45611-8. DOI: 10.1007/978-3-662-45611-8_28. URL: http://dx.doi.org/10.1007/978-3-662-45611-8_28.

[38] Irit Dinur. "The PCP theorem by gap amplification". In: *Journal of the ACM* 54.3 (2007), p. 12.

[39] Cynthia Dwork, Uriel Feige, Joe Kilian, Moni Naor, and Shmuel Safra. "Low Communication 2-Prover Zero-Knowledge Proofs for NP". In: *Proceedings of the 11th Annual International Cryptology Conference*. CRYPTO '92. 1992, pp. 215–227.

[40] Rosario Gennaro, Craig Gentry, and Bryan Parno. "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers". In: *Proceedings of the 30th Annual Conference on Advances in Cryptology*. CRYPTO'10. Santa Barbara, CA, USA: Springer-Verlag, 2010, pp. 465–482. ISBN: 3-642-14622-8, 978-3-642-14622-0. URL: http://dl.acm.org/citation.cfm?id=1881412.1881445.

[41] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '13. 2013, pp. 626–645.

[42] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. "ZKBoo: Faster Zero-Knowledge for Boolean Circuits". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 1069–1083. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/giacomelli.

[43] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*. STOC '08. 2008, pp. 113–122.

[44] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The knowledge complexity of interactive proof systems". In: *SIAM Journal on Computing* 18.1 (1989). Preliminary version appeared in STOC '85., pp. 186–208.

[45] Jens Groth. "Short Pairing-Based Non-interactive Zero-Knowledge Arguments". In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '10. 2010, pp. 321–340.

[46] Jens Groth. "Efficient Zero-Knowledge Arguments from Two-Tiered Homomorphic Commitments". In: *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*. 2011, pp. 431–448. DOI: 10.1007/978-3-642-25385-0_23. URL: http://dx.doi.org/10.1007/978-3-642-25385-0_23.

[47] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11. URL: http://dx.doi.org/10.1007/978-3-662-49896-5_11.

[48] Jens Groth and Mary Maller. "Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs". In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*. 2017, pp. 581–612. DOI: 10.1007/978-3-319-63715-0_20. URL: https://doi.org/10.1007/978-3-319-63715-0_20.

[49] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24. URL: http://dx.doi.org/10.1007/978-3-540-78967-3_24.

[50] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. "Efficient Arguments without Short PCPs". In: *Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity*. CCC '07. 2007, pp. 278–291.

[51] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Zero-knowledge from secure multiparty computation". In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM. 2007, pp. 21–30.

[52] Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. *On Zero-Knowledge PCPs: Limitations, Simplifications, and Applications*. Available at http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf. 2015.

[53] Yael Kalai and Ran Raz. "Interactive PCP". In: *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*. ICALP '08. 2008, pp. 536–547.

[54] Joe Kilian. "A note on efficient zero-knowledge proofs and arguments". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. STOC '92. 1992, pp. 723–732.

[55] Joe Kilian, Erez Petrank, and Gábor Tardos. "Probabilistically checkable proofs with zero knowledge". In: *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*. STOC '97. 1997, pp. 496–505.

[56] Sian-Jheng Lin, Tareq Y. Al-Naffouri, Yunghsiang S. Han, and Wei-Ho Chung. "Novel Polynomial Basis With Fast Fourier Transform and Its Application to Reed-Solomon Erasure Codes". In: *IEEE Trans. Information Theory* 62.11 (2016), pp. 6284–6299.

[57] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. "Novel Polynomial Basis and Its Application to Reed-Solomon Erasure Codes". In: *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. FOCS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 316–325. ISBN: 978-1-4799-6517-5. DOI: 10.1109/FOCS.2014.41. URL: http://dx.doi.org/10.1109/FOCS.2014.41.

[58] Helger Lipmaa. "Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments". In: *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*. TCC '12. 2012, pp. 169–189.

[59] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *Journal of the ACM* 39.4 (1992), pp. 859–868.

[60] Silvio Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS '94., pp. 1253–1298.

[61] Silvio Micali. "Computationally Sound Proofs". In: *SIAM J. Comput.* 30.4 (2000), pp. 1253–1298. DOI: 10.1137/S0097539795284959. URL: http://dx.doi.org/10.1137/S0097539795284959.

[62] Thilo Mie. "Polylogarithmic Two-Round Argument Systems". In: *Journal of Mathematical Cryptology* 2.4 (2008), pp. 343–363.

[63] Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. "Pinocchio: Nearly Practical Verifiable Computation". In: *Proceedings of the 34th IEEE Symposium on Security and Privacy*. Oakland '13. 2013, pp. 238–252.

[64] M. Peck. "A blockchain currency that beat s bitcoin on privacy [News]". In: *IEEE Spectrum* 53.12 (2016), pp. 11–13. ISSN: 0018-9235. DOI: 10.1109/MSPEC.2016.7761864.

[65] Evgenya Pergament. "Algebraic RAM". MA thesis. Technion — Israel Institute of Technology, 2017.

[66] Alexander A Razborov. "Lower bounds on the size of bounded depth circuits over a complete basis with logical addition". In: *Mathematical Notes of the Academy of Sciences of the USSR* 41.4 (1987), pp. 333–338.

[67] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-round interactive proofs for delegating computation". In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. 2016, pp. 49–62. DOI: 10.1145/2897518.2897652. URL: http://doi.acm.org/10.1145/2897518.2897652.

[68] SCIPR Lab. *libsnark: a C++ library for zkSNARK proofs*. https://github.com/scipr-lab/libsnark. URL: https://github.com/scipr-lab/libsnark.

[69] Jae Hong Seo. "Round-Efficient Sub-linear Zero-Knowledge Arguments for Linear Algebra". In: *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*. 2011, pp. 387–402. DOI: 10.1007/978-3-642-19379-8_24. URL: http://dx.doi.org/10.1007/978-3-642-19379-8_24.

[70] Srinath Setty, Andrew J. Blumberg, and Michael Walfish. "Toward practical and unconditional verification of remote computations". In: *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*. HotOS '11. 2011, pp. 29–29.

[71] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. "Resolving the conflict between generality and plausibility in verified computation". In: *Proceedings of the 8th EuoroSys Conference*. EuroSys '13. 2013, pp. 71–84.

[72] Srinath Setty, Michael McPherson, Andrew J. Blumberg, and Michael Walfish. "Making argument systems for outsourced computation practical (sometimes)". In: *Proceedings of the 2012 Network and Distributed System Security Symposium*. NDSS '12. 2012.

[73] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. "Taking proof-based verified computation a few steps closer to practicality". In: *Proceedings of the 21st USENIX Security Symposium*. Security '12. 2012, pp. 253–268.

[74] Adi Shamir. "IP = PSPACE". In: *Journal of the ACM* 39.4 (1992), pp. 869–877.

[75] Roman Smolensky. "Algebraic methods in the theory of lower bounds for Boolean circuit complexity". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM. 1987, pp. 77–82.

[76] Justin Thaler. "Time-Optimal Interactive Proofs for Circuit Evaluation". In: *Proceedings of the 33rd Annual International Cryptology Conference*. CRYPTO '13. 2013, pp. 71–89.

[77] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Proceedings of the 5th Conference on Theory of Cryptography*. TCC'08. New York, USA: Springer-Verlag, 2008, pp. 1–18. ISBN: 3-540-78523-X, 978-3-540-78523-1. URL: http://dl.acm.org/citation.cfm?id=1802614.1802616.

[78] Victor Vu, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. "A hybrid architecture for interactive verifiable computation". In: *Proceedings of the 34th IEEE Symposium on Security and Privacy*. Oakland '13. 2013, pp. 223–237.

[79] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. "Efficient RAM and control flow in verifiable outsourced computation". In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*. 2015.

[80] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. *Doubly-efficient zkSNARKs without trusted setup*. Cryptology ePrint Archive, Report 2017/1132. https://eprint.iacr.org/2017/1132. 2017.

[81] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. "vRAM: Faster Verifiable RAM With Program-Independent Preprocessing". In: *2018 IEEE Symposium on Security and Privacy (SP)*. Vol. 00, pp. 203–

220. DOI: 10.1109/SP.2018.00013. URL: doi.ieeecomputersociety.org/10.1109/SP.2018.00013.

[82] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. *A Zero-Knowledge Version of vSQL*. Cryptology ePrint Archive, Report 2017/1146. https://eprint.iacr.org/2017/1146. 2017.

A Standalone construction

In this section we give an overview of the process leading to the main theorems specified above (Section 2.3). For didactic reasons we accompany our description with a simple and concrete "toy" computation as an example, marked in boxed texts, and gloss over some of the (numerous) technicalities (a few examples are discussed in the last part in this section); nevertheless, the same steps apply to more complex computations. Further details and formal definitions appear in the full version of this paper [12].

Many ZK systems (including ours) use *arithmetization*, a technique introduced to prove circuit lower bounds [66, 75] and adapted later to interactive proof systems [4, 59]. Arithmetization is the reduction of *computational* problems to *algebraic* problems, that involve "low degree" polynomials over a finite field $\mathbb{F}$; in this context, "low degree" means degree is significantly smaller than field size.

The start point for arithmetization in all proof systems is a computational integrity statement which the prover wishes to prove, like the following instance of the CI language (see Remark 1):

> *"I know private input y, such that executing* C *for* T *steps on public input x and private input y leads to result z."* (*)

For our ZK-STIK and for related prior systems [27, 25, 9], the end point of arithmetization is a pair of *Reed-Solomon (RS) proximity testing (RPT)* problems[12], and the scalability of our ZK-STIK relies on a new solution to it — the FRI protocol discussed below [11]. For $S \subset \mathbb{F}$ and rate parameter $\rho \in (0,1)$, the RS code with evaluation domain S and *rate* ρ is the space of evaluations of low-degree functions over S,

$$\mathsf{RS}[\mathbb{F}, S, \rho] = \{f : S \to \mathbb{F} \mid \deg(f) < \rho|S|\}\,.$$

The RPT problem for $\mathsf{RS}[\mathbb{F}, S, \rho]$ is one of deciding, with a small number of queries, whether a function $f : S \to \mathbb{F}$ is a member of $\mathsf{RS}[\mathbb{F}, S, \rho]$ or far from all members of the code in relative Hamming distance.

[12] The other solutions described in Section 3.1 like those based on Homomorphic public-key cryptography (hPKC) have different end points.

Toy problem For concreteness, consider the following special case of (*), which computes the T entry in a "multiplicative modular Fibonacci sequence":

> *"I know initial values $y_0, y_1 \in \mathbb{F}$, such that $z \in \mathbb{F}^*$ is the Tth element in the sequence defined inductively by $y_i = y_{i-2} \cdot y_{i-1}$ for $i > 1$ (i.e., $z = y_{\mathsf{T}}$)"* (**)

We call this a multiplicative modular Fibonacci sequence because, fixing g to be a generator of $\mathbb{F}^*$, and setting $y_i = g^{j_i}$ one sees that the correct output z is $z = g^{F_{\mathsf{T}}}$ where F_{T} is the Tth element in the Fibonacci sequence that starts with j_0, j_1, and is computed modulo $|\mathbb{F}^*| = |\mathbb{F}| - 1$. We choose this simple computation as our toy problem because it is non-trivial to compute over all fields (the standard modular Fibonacci sequence is trivial over binary fields).

Our process has 4 parts (see Figure 5). When reading the description below, the main thing to notice is that from start to end, verification costs are logarithmic in T (and polynomial in the description of the computation C). To see this it is useful to think informally of $\mathsf{T} \gg |\mathsf{C}|$, like $\mathsf{T} = 2^{|\mathsf{C}|}$. In each of the reductions, the verifier receives only an instance (denoted $\mathbb{x}$) as its input, whereas the prover additionally receives a witness (denoted $\mathbb{w}$) for membership of $\mathbb{x}$ in the relevant language.

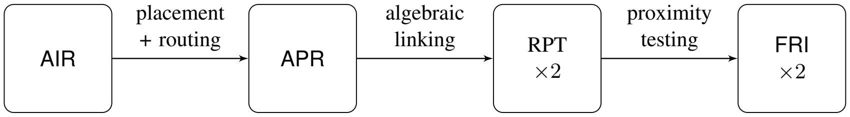


Fig. 5: The reduction from an AIR *instance to a pair of RPT problems, solved using the* FRI *protocol, explained later in this section. Briefly, the Algebraic Intermediate Representation (AIR) is converted via the Algebraic Placement and Routing (APR) reduction to an APR instance. This is reduced via the Algebraic Linking IOPP (ALI) protocol to a pair of RPT problems, which are solved using two applications of the FRI protocol.*

Part I The starting point is a natural *algebraic intermediate representation*[13] (AIR) of $\mathbb{x}$ and $\mathbb{w}$, denoted $\mathbb{x}_{\mathsf{AIR}}, \mathbb{w}_{\mathsf{AIR}}$. The verifier receives $\mathbb{x}_{\mathsf{AIR}}$ and the prover also receives $\mathbb{w}_{\mathsf{AIR}}$. Informally, $\mathbb{x}_{\mathsf{AIR}}$ corresponds to the statement (*) and $\mathbb{w}_{\mathsf{AIR}}$ corresponds to an execution trace witnessing correctness of (*), i.e., $\mathbb{w}_{\mathsf{AIR}}$ is a $\mathsf{T} \times \mathsf{a}$ array in which the ith row describes the state of the computation at time i and the jth column tracks the contents of the jth register over time (this column will later give rise to f_j). Each entry of this array is an element in the field $\mathbb{F}$. The transition relation of the computation is specified by a set of multi-variate polynomials over variables $X_1, \ldots, X_{\mathsf{a}}, Y_1, \ldots, Y_{\mathsf{a}}$ that correspond to the current state registers (X variables) and next state registers (Y variables). These constraints enforce the validity of the transition from one state to the next.

[13] AIRs are called algebraic constraint satisfaction problems (ACSPs) in prior works like [27, 9]; we prefer the mono-syllable term AIRs which also relates to the notion of an intermediate representation used in other areas of computer science.

In our toy problem (**), we shall use an execution trace of dimensions $\mathsf{T} \times 2$, where an honest prover is expected to fill the ith row with entries y_{i-1}, y_i. Using X_0, X_1 and Y_0, Y_1 to denote the registers in two consecutive sets, our toy transition relation is captured by the pair of polynomial constraints

$$C_0(Y_0, X_1) := Y_0 - X_1; \quad C_1(X_0, X_1, Y_1) := Y_1 - X_0 \cdot X_1.$$

Satisfying a constraint means assigning values to its variables as to make it vanish (evaluate to 0). The first constraint above ensures we move the latest element in the sequence to the first register and the second constraint ensures we compute the next element correctly. $\mathbb{x}_{\mathsf{AIR}}$ contains these two constraints, along with the boundary constraint that "forces" the $[\mathsf{T}, 2]$-entry of $\mathbb{w}_{\mathsf{AIR}}$ to equal z (the public input of the statement (**)).

Notice that $|\mathbb{x}_{\mathsf{AIR}}|$ can be much smaller than $|\mathbb{w}_{\mathsf{AIR}}|$; this is crucial for (full) scalability because tv must be bounded by a polynomial in $|\mathbb{x}_{\mathsf{AIR}}|$ and $\log \mathsf{T}$. Another point to bear in mind is that constructing an AIR for simple computations is straightforward (as shown in our toy example); additional examples appear in Vitalik Buterin's blog posts I and III on STARKs [32], in the examples in libSTARK [10], and in previous works like [9, Appendix B] and [65].

Part II We reduce the AIR representation into a different one, in which states of the execution trace are "placed" on nodes of an *affine graph*, so that consecutive states are connected by an edge in that graph. Informally, an affine graph is a "circuit" that has "algebraic" topology. The process of "placing" machine states on nodes of a circuit is roughly analogous to the process of *placement and routing* which is commonly used in computer and circuit design, although our design space is constrained by *algebra* rather than by physical reality. We refer to this particular transformation as the *algebraic placement and routing* (APR) reduction, and the resulting representation is an APR instance/witness pair $(\mathbb{x}_{\mathsf{APR}}, \mathbb{w}_{\mathsf{APR}})$. The affine graph will necessarily be quite large, larger than $|\mathbb{w}_{\mathsf{APR}}| \geq \mathsf{T}$, but the verifier requires only a *succinct* representation of this graph, via a constant size set of (edge) generators. This succinct representation is crucial for obtaining verifier scalability and avoiding the "computation unrolling" costs incurred by other ZK approaches. We first explain how a prover computes this transformation, and then address the verifier's transformation.

The (honest) prover interprets the jth column of the algebraic execution trace as a partial function $\hat{f}_j$ from a domain that is a subset of $\mathbb{F}$ and which maps into the field $\mathbb{F}$. Thus, the prover now interpolates this function $\hat{f}_j$ to obtain a polynomial $P_j(X)$, and then evaluates this polynomial on a different domain $S \subset \mathbb{F}$ of size $|S| = \beta \cdot \mathsf{T}$, to obtain a function f_j. The final step of this stage on the prover-side is providing the verifier with oracle access to the sequence $\boldsymbol{f} = (f_1, \ldots, f_{\mathsf{a}})$ where $f_i : S \to \mathbb{F}$, noticing this sequence is an encoding of columns (registers) of the execution trace via RS codewords. (in the ZK-STARK, this oracle access will be realized via Merkle-tree commitments to $\boldsymbol{f}$).

The verifier, on receiving $\mathbb{x}_{\mathsf{AIR}}$, computes the size $\beta \cdot \mathsf{T}$ and picks the same domain $S \subset \mathbb{F}$ as the prover (notice S does not depend on $\mathbb{w}_{\mathsf{AIR}}$). Then, the verifier computes the succinct set of affine transformations that correspond to edges in the affine graph, and obtains an APR instance, denoted $\mathbb{x}_{\mathsf{APR}}$.

In the toy problem (**) the APR reduction involves picking a multiplicative subgroup G of $\mathbb{F}^*$ of size $|G| = \mathsf{T}$ (for simplicity we assume such G exists; in $\mathsf{libSTARK}$ we use additive subgroups instead of multiplicative ones and pad the execution trace to size $|G|$). Let g denote a generator of G. The affine graph in this case has vertex set G and directed edges $(h, \mathsf{g} \cdot h)$. Using this, we now view the execution trace as a pair of mappings $\hat{f}_0, \hat{f}_1 : G \to \mathbb{F}$, one mapping per register/column of the execution trace. The prover interpolates each function to obtain a pair of polynomials $P_0(X), P_1(X)$ and evaluates them over a set S that is a union of cosets of G, creating the first proof oracle $\boldsymbol{f} = (f_0, f_1)$ (when constructing the ZK-STARK, this means the prover computes the Merkle root of $\boldsymbol{f}$ and sends it to the verifier).

The reduction in this step is deterministic on the verifier side, i.e., involves no verifier-side randomness and no interaction; as such, it also has perfect completeness and perfect soundness. On the prover side, randomness is used to create a zero knowledge version of the execution trace, by allowing the prover to use polynomials of degree slightly greater than T, as to allow for Shamir-style secret sharing techniques to hide individual entries of the execution trace.

Part III The APR representation is used to produce, via a 1-round IOP, a pair of instances of the Reed-Solomon proximity testing (RPT) problem. In our case, the two codes resulting from the reduction are over the same field $\mathbb{F}$ but may have different evaluation domains and different code rates. To maintain verifier scalability, we point out that specifying the code parameters — S and ρ, will be done in a succinct manner, one that requires space $\log |\mathsf{T}|$; thus, this part of our construction also supports verifier-side scalability.

The witness in this case is a pair of purported codewords $(f^{(0)}, g^{(0)})$. The first function $f^{(0)}$ is simply a random linear combination of $\boldsymbol{f}$ to which the prover committed in the previous step. The second function $g^{(0)}$ is obtained after the various constraints that enforce execution trace validity are randomly “linked” into a single (random) constraint. We thus refer to this step as the *algebraic linking IOP (*ALI*)* protocol.

For the toy problem (**) the ALI protocol works thus. After receiving oracle access to $\boldsymbol{f}$ (or its Merkle commitment), the verifier samples $r_0, r_1, r'_0, r'_1 \in \mathbb{F}$ and sends them to the prover. The prover is expected to compute $f^{(0)} = r_0 \cdot f_0 + r_1 \cdot f_1$. To construct $g^{(0)}$, the prover first constructs the single random constraint

$$C(X_0, X_1, Y_0, Y_1) := r'_0 \cdot C_0(Y_0, X_1) + r'_1 \cdot C_1(X_0, X_1, Y_1)$$

where C_0, C_1 are as defined in step 1. Then, the prover recalls the interpolating polynomials P_0, P_1 from step 2 and computes

$$Q(X) := C(P_0(X), P_1(X), P_0(\mathsf{g} \cdot X), P_1(\mathsf{g} \cdot X)).$$

Let $\mathsf{Zero}_G(X) := \prod_{\xi \in G}(X - \xi)$. The prover computes $g^{(0)} : S \to \mathbb{F}$ as the evaluation of $Q(X)/\mathsf{Zero}_G(X)$ on S. Notice that $g^{(0)}$ is well-defined because $G \cap S = \emptyset$. Recalling the verifier has oracle access to $\boldsymbol{f}$, notice that each entry of $f^{(0)}$ can be computed by querying a single row of the execution trace $\boldsymbol{f}$ (one query from f_0 and one from f_1; similarly, each entry of $g^{(0)}$ can be computed by reading two consecutive rows (4 entries) of $\boldsymbol{f}$. Thus, even though the next step will assume oracle access to $f^{(0)}, g^{(0)}$, the protocol does not require the prover to send another set of oracles during this step, the oracles can be "locally computed" from $\boldsymbol{f}$.
Finally, notice that if the prover is honest, then it holds that $f^{(0)}$ is a codeword of the RS code of rate $|G|/|S|$ over evaluation domain S. Similarly, since $Q(X)$ vanishes on all $\xi \in G$, we deduce that $Q(X)/\mathsf{Zero}_G(X)$ is a polynomial of degree at most $\deg(C) \cdot |S| - \deg(\mathsf{Zero}_G) = |S|$, so $g^{(0)}$ is also a codeword of $\mathsf{RS}[\mathbb{F}, S, |G|/|S|]$.

Part IV In the last step of our reduction, for each of the two functions (oracles) $f^{(0)}, g^{(0)}$, the prover and verifier interact according to the *fast RS IOP of proximity* (FRI) protocol from [11] (cf. [12, Appendix B.6]). That protocol has a scalable verifier and query complexity that is logarithmic in the size of the evaluation domain of the code, further establishing verifier scalability. And thus, from start to end, verifier side complexity remains scalable — logarithmic in T (and polynomial in $|\mathsf{C}|$).

In this last step our toy problem (**) behaves no differently than the general case. We apply the FRI protocol to each of $f^{(0)}, g^{(0)}$ described in the prior step, and compute the entries of each function by making oracle access to $\boldsymbol{f}$.

Regarding prover scalability, inspection reveals that the main bottleneck in the process is the low-degree extension part, in which each function $\hat{f}_j$ that encodes a register gets interpolated and then evaluated on a domain of size $\beta \cdot \mathsf{T}$. For this part we use so-called *additive FFTs*; in particular, libSTARK uses the recent innovative algorithm of [56] that performs this computation with $O(\beta\mathsf{T}\log(\beta\mathsf{T}))$ arithmetic operations. All other steps of the prover's computation are merely *linear* in $|\mathsf{T}|$; in particular, the FRI computation is such.

In closing we briefly mention some of the subtle issues that were glossed over in our toy example and are discussed at length in our formal proofs, and implemented in the code:

1. The toy construction is not zero knowledge, because each entry of $\boldsymbol{f}$ does reveal some information about y_0, y_1. To achieve zero knowledge we slacken the degree constraint on f_0, f_1, allowing the prover to sample a random polynomial that agrees with $\hat{f}_0, \hat{f}_1$ on G, and thus hide information regarding y_0, y_1 for query-limited verifiers (in a manner resembling Shamir secret sharing [74]).
2. We did not enforce the boundary condition stating that the last entry is z. To enforce this, the verifier interpolates a polynomial corresponding to all boundary constraints (in our toy example there is only one such constraint) and "incorporates it" in the proof oracle $\boldsymbol{f}$.
3. Verifier scalability requires that Zero_G be computed efficiently. This is indeed the case (because G is a subgroup of $\mathbb{F}$), and holds also for additive subgroups (as implemented by libSTARK [10]).
4. The toy computation does not make use of random memory access (RAM); maintaining scalability for programs that make significant use of RAM complicates the construction, requiring more elaborate affine graphs that embed DeBruijn switching networks; these issues are addressed by Theorem 2 and its proof.