Computation Verification for Noobs

Research thesis
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Michael Riabzev

Submitted to the Senate of the Technion - Israel Institute of Technology
Tishrei, 5780, Haifa, October, 2019
Instruction

The Research Thesis Was Done Under The Supervision of Professor Eli Ben-Sasson and Professor Yuval Ishai in the Faculty of computer-science.

Financial Aid

The Generous Financial Help Of the Technion, the Faculty of Computer-Science, Mel Berlin Cyber Scholarship, and Nehemia Levtzion Scholarship Is Gratefully Acknowledged.

List of publications

Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, Michael Riabzev: Scalable Zero Knowledge with No Trusted Setup. CRYPTO (3) 2019: 701-732

Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, Nicholas P. Ward: Aurora: Transparent Succinct Arguments for R1CS. EUROCRYPT (1) 2019: 103-128


Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, Madars Virza: Computational Integrity with a Public Random String from Quasi-Linear PCPs. EUROCRYPT (3) 2017: 551-579


Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, Nicholas Spooner: Zero Knowledge Protocols from Succinct Constraint Detection. TCC (2) 2017: 172-206
## Contents

1 Introduction
   1.1 Problem definition .................................................. 1
   1.2 Practical motivation .................................................. 1
   1.3 Proof systems classes ................................................. 2
   1.4 Verifying computations ................................................ 4
   1.5 Zero-Knowledge .......................................................... 7
   1.6 State-of-the-art solutions ............................................. 8
      1.6.1 Verifiable-computation with succinct verification ............... 8
      1.6.2 Verifiable computation for circuits ................................ 8

2 Research methods ......................................................... 10

3 Findings ................................................................. 11

4 Discussion ............................................................... 254
Abstract

Computational-integrity (also known as checking-computation [9], verifiable-computation [31, 49], and computation-delegation [37]) protocols eliminate the need to trust reporters of computation results, by allowing the latter to provide a cryptographic proof attesting to the validity of the results. The proof verification time is polylogarithmic with the length of the computation (succinct), introducing significant improvement over the naive solution for general computations, which is merely re-execution. Computational-integrity protocols were introduced around three decades ago [9]. This original work offered theoretical solutions to practical problems. An immediate practical motivation for taking such an approach comes from a different field of mathematical-proofs done by computer such as the celebrated example of [8] showing that any planar graph is four-colourable. The method used in [8] included a computer exhaustively verifying correctness of a finite set of graphs, requiring work beyond what is humanly achievable, and took over a thousand hours to complete using the computer. Obviously, anyone who did not want to repeat this heavy verification process had to put their trust in the researcher who claimed to execute it. This kind of a proof to a mathematical claim was novel, and the fact the proof cannot be easily verified by peer reviews disappointed many. Computational-integrity is a natural solution to such problems, as it can be used by researchers to generate a succinct proof for the integrity of any computation, verifiable by anyone. Unfortunately, the work of [8] predated the initial ideas of verifiable computation, introduced in [9], by roughly a decade. Even though roughly three decades have passed since the introduction of [9], it is still probably infeasible to use modern computational-integrity for problems of such complexity. Although there are implementations of succinct verifiers that can succinctly verify the correctness of a small proof, the computation overhead on the prover, compared to the naive computation, is still too high. The line of research work we present shows what we have done in order to advance the field of computational-integrity from theory to practice. Our results show how theory can be improved to provide concrete efficiency. A common technique we have used is adding interactivity to both improve the soundness of protocols, and reduce the proving computational load. In addition to theoretical improvements, we provide three POC implementations of systems used to measure the constructions’ concrete efficiency. Moreover, the implementations can be used as a reference for production grade system, as they introduce efficient data-structures and algorithms designed and tuned for implementation of algebraic proof systems.
Chapter 1

Introduction

This chapter provides a brief overview of the theoretical problem of computation verification. It then offers a few real world motivations for practical solutions to this problem, and finishes with a broad overview of state of the art solutions for the studied problem.

1.1 Problem definition

Verification of computations is a decision problem where, given as input a tuple \((P, I_{\text{explicit}}, O, T)\) in which (i) \(P\) is a deterministic program represented by some Turing complete language, (ii) \(I_{\text{explicit}} \in \{0,1\}^*\) is an explicit input to the program, (iii) \(O \in \{0,1\}^*\) is an output candidate and (iv) \(T \in \mathbb{N}\) is a bound on execution steps represented by \(O(\log T)\) bits, one must decide whether there exists a private input \(I_{\text{private}} \in \{0,1\}^*\) such that \(P\) halts on the concatenated input \(I_{\text{explicit}} || I_{\text{private}}\) within at most \(T\) steps and returns \(O\).

Although it has been shown that this decision problem is \(\text{NEXP}\)-complete, the celebrated work of Babai et al. [10] demonstrated that one could efficiently verify the existence of an \(I_{\text{private}}\) using auxiliary untrusted sources, which are often called provers or proofs. Systems using such solutions are called proof systems or protocols, and the decider in such systems is often called a verifier.

An important variant of this problem considers the knowledge complexity of the protocol, and more specifically, requires the verifier to “learn” nothing additional aside from the validity of the claim when engaged in a protocol with honest provers or accessing a valid proof, and in particular nothing about the private input \(I_{\text{private}}\). Such protocols are called Zero-Knowledge [38] protocols, and we say such protocols are Zero-Knowledge proof of knowledge [28] if the ability of the prover to convince a verifier of the existence of such an \(I_{\text{private}}\) implies an ability of the prover to extract such an \(I_{\text{private}}\) efficiently compared to the power required to convince the verifier.

1.2 Practical motivation

Many real world applications exist that could benefit from a practical solution to the problem of computation verification such as, for example, delegation of computations or solving issues involving a trusted party. We now elaborate on the benefits these application can gain from practical solutions to the problem.
Computation delegation  For this motivation, we focus on a scenario in which a strong service provider would execute very difficult computation jobs and provide their output in exchange for money. Such services are commonly called computation delegation services. Although there are some problems whose solution could be verified very efficiently, this is not the situation in the general case — there are many problems for which result verification is not known to have algorithms more efficient than the algorithms for problem solving. Consequently, there are many cases where a client has to believe the service provider is honest, and that the result offered by the provider is indeed the output of the computation, although no evidence is supplied. In contrast, the service provider may have some incentives to provide a phony output; for example, he may want to (i) replace an expensive execution with a cheaper approximation, or (ii) intentionally provide a misleading output in order to affect the client’s decisions. Despite these well-known problems, clients continue to use computation delegation services. We assume clients’ trust in the service providers is based on the provider’s reputation, and define this as the reputation based model of trust. In the reputation based model of trust, the probability of a potential client to use a service of some provider is highly dependant on the provider’s reputation. In this model, a cheating provider has a high probability of not getting caught, but the penalty if caught is the loss of their reputation.

In a reputation based model, cheating is unprofitable only in cases when the expected future profit is higher than the profit gained from cheating. The authors believe that in such a model, the clients would prefer to use a familiar service with a strong reputation even if a cheaper unfamiliar service is available. Modifying the trust model to be based on cryptography may remove the incentive to cheat, as a cheater would almost certainly get caught, obviating the need for a strong reputation for trust. The authors believe such a change could reduce the services’ cost. Additionally, the current model blocks certain jobs completely from being delegated to untrusted computing service providers that might be strongly motivated to fake results, e.g., jobs that might determine major financial decisions of a corporation. If there was a practical solution to the problem of computation verification, it would be very useful in designing such cryptographic based trust services in a straightforward manner.

Trusted parties  Many practical problems involve two parties that mutually distrust each other. Let’s call them Alice and Bob. Alice wants to convince Bob she knows some secret data that meets some conditions that could be verified by a computer program without revealing the secret to Bob, for example, a pre-image of some hash value. Today, the common solution to such a situation is adding Charlie to the protocol. Charlie is a trusted party, as both Alice and Bob are required to trust him. Alice allows Charlie to access her private data and trusts him not to reveal anything about it but whether it meets the required conditions, while Bob trusts Charlie to provide an honest answer. It is not a surprise that a trust model based on humans has many issues; on the other hand, a trust model based on cryptography could satisfy both Alice and Bob. This problem can be solved using practical implementations of verification of computations with Zero-Knowledge proof of knowledge, where the private input $I_{\text{private}}$ is Alice’s secret data, and the program $P$, which validates the secret, meets the required conditions.

1.3 Proof systems classes

This section introduces common proof systems and their strength in terms of complexity classes.
Non-interactive proofs  A classical proof for a claim of the form $x \in L$ is a string $\Pi$ that can be read and verified deterministically, similarly to standard mathematical proofs. Formally, a classical proof system for a language $L$ is a polynomial time machine $V$ called a verifier such that:

- For any $x \in L$, there exists a proof $\Pi_x$ of length $|\Pi_x| \leq \text{poly}(|x|)$ such that the verifier $V$ accepts the pair $(x, \Pi_x)$.
- For any $x \notin L$ and for any $\Pi$, the verifier $V$ rejects the pair $(x, \Pi)$.

Classical proof systems are well-known, and the class of languages that have classical proof systems is known as $NP$.

A native expansion of classical proof systems allows the verifier to use randomness and is defined as a $\mathbf{BPP}$ machine. The class of languages having such expanded classical proof systems is called $MA$ and is formally defined as all the languages $L$ having a probabilistic polynomial-time machine $V$ called a verifier such that:

- For any $x \in L$, there exists a proof $\Pi_x$ of length $|\Pi_x| \leq \text{poly}(|x|)$ such that the verifier $V$ accepts the pair $(x, \Pi_x)$ with probability of at least $\frac{2}{3}$.
- For any $x \notin L$ and for any $\Pi$, the verifier $V$ rejects the pair $(x, \Pi)$ with probability of at least $\frac{2}{3}$.

Obviously, $NP \subseteq MA$, but it is unknown whether the equivalence holds.

Interactive proofs  Another well-studied proof system model is the interactive proof introduced by Goldwasser et al. [38]. In such systems, the verifier $V$ can interact with a devilish party $P$ called a prover, willing to convince $V$ that some claim of the form $x \in L$ holds, and in order to do this, it agrees to answer the verifier's messages. This model is very similar to questioning of a witness as done in court. The witness is willing to convince the court that some claim is true, and agrees to answer the investigator's questions; however, the witness' reliability must also be validated. Similarly, there is doubt about the prover's honesty, and the latter can use any strategy to convince the verifier that its claim is true, even if it is not. Formally, an interactive proof system for a language $L$ is a pair $(P, V)$ where $P$ is a Turing machine having no bound on its complexity and is called the prover, and $V$ is a probabilistic polynomial-time machine called a verifier such that:

- For any $x \in L$, the verifier $V$ accepts $x$ after interacting with the prover $P$ with probability 1.
- For any $x \notin L$, the verifier $V$ rejects $x$ after interacting with any malicious prover $P^*$ with probability of at least $\frac{2}{3}$.

Shamir [52] showed that the class of languages having interactive proofs is exactly the class of languages that can be solved using polynomial space, and is known as $PSPACE$. Changing the acceptance model to allow the verifier to reject claims with a small probability of being true as in $BPP$, does not increase the power of the class. Obviously, $MA \subseteq PSPACE$, but it is unknown whether the equivalence holds.
Non adaptive provers  Two very powerful models were developed independently and later shown to be equivalent: (i) The multiple provers interactive proof or MIP, introduced by Ben-Or et al. [11], is similar to the interactive proof model, and differs only by allowing the verifier $V$ to interact with several isolated provers $P_1, P_2, \ldots, P_k$ that cannot communicate among themselves. (ii) The second model, probabilistically checkable proof or PCP, was introduced by Fortnow et al. [34]. In this model, the verifier $V$ has random access to a huge string $\Pi$, which is a proof that should convince $V$ of the truth of a claim of the form $x \in L$. The proof $\Pi$ is too big for the verifier to read entirely, thus the verifier accesses only a small fraction of it, which should be sufficient to verify the validity of the proof with high probability. The formal definition of MIP systems is very similar to that of interactive proofs. The formal definition of a PCP system for a language $L$ is a polynomial time machine $V$ called a verifier such that:

- For any $x \in L$, there exists a proof $\Pi_x$ of length $|\Pi_x| \leq 2^{\text{poly}(|x|)}$ such that the verifier $V$ accepts $x$ after using the random access to $\Pi_x$ with probability 1.
- For any $x \notin L$ and for any $\Pi$, the verifier $V$ rejects $x$ even after accessing $\Pi$ with probability of at least $\frac{2}{3}$.

The equivalence of these two models was shown by Ben-Or et al. [11]. Babai et al. [10] showed that they are both equivalent to the class of languages decidable in non-deterministic exponential time, and known as $\mathcal{NEXP}$. Similarly to the case of $\mathcal{PSPACE}$, changing the acceptance model in both $\mathcal{PCP}$ and $\mathcal{MIP}$ to allow the verifier to reject claims with a small probability of being true, as in $\mathcal{BPP}$, does not increase the power of the class. Obviously, $\mathcal{PSPACE} \subseteq \mathcal{NEXP}$, but it is unknown whether the equivalence holds.

Intuitively, the main difference of $\mathcal{PCP}$ compared to interactive proofs is the restriction of the single prover and not allowing it to answer adaptively to previous queries and answers. Further, the common constructions of protocols for $\mathcal{NEXP}$ using $\mathcal{MIP}$ use multiple provers to insure nonadaptiveness of one of the provers. This property is crucial to our discussion in practical constructions, as no known practical constructions can insure this property without using additional assumptions.

1.4 Verifying computations

This section starts by defining the problem we solve and briefly scans the advances made towards finding efficient solutions. The second part of this section explains the inherent challenges in constructing practical systems based on theoretical solutions, and explains the different attitudes to overcoming these challenges.

Problem definition  We recall the definition of computation verification given in the introduction. This problem was first introduced by Babai et al. [9]. One could notice it is $\mathcal{NEXP}$-complete as there is a trivial reduction from the problem of Oracle-3-satisfiability [10, Definition 4.1, p. 13], which was shown to be $\mathcal{NEXP}$-complete by Babai et al. in [10, Proposition 4.2, p. 13]. There are different approaches to solving less general problems efficiently, as, for example, verifying computation of programs that can be represented using shallow circuits [37], but we focus on solving the general problem.
**Theoretical solutions** We now review the different theoretical solutions and their complexities. The first solution to the problem of computation verification was introduced by Babai et al. when they introduced the problem itself [9]. They constructed a family of PCP protocols, having a protocol for each value of $\varepsilon > 0$, where the length of the proof $\Pi$ is $\Theta((T^2)^{1+\varepsilon})$ and both the query complexity and the verification time are poly-logarithmic: poly $\left((\log(T^2))^{\frac{1}{\varepsilon}}\right)$. Polishchuck and Spilman [50] improved the result of [9] by constructing a PCP system with the same properties but having a constant query complexity. The work of Ben-Sasson et al. [25] improved the proof size by showing an explicit construction of PCP with a proof size $T^2 \cdot 2^{\tilde{O}(\sqrt{\log T})}$, which was improved even more in [23] to $T^2 \cdot \exp(o(\log \log T)^2)$. Ben-Sasson and Sudan achieved a quasi-linear expansion of the proof in [24] – thereby, producing PCP systems with a proof size of $\tilde{O}(T^2)$, and [19] combined this technique with a novel method to verify RAM access to achieve proofs of length $\tilde{O}(T)$. Recently, the novel interactive oracle proof model [22], or IOP, which is a hybrid of PCP and interactive proofs, was introduced. In this model, the prover writes the huge proof during an interactive process, and in each interaction, it reveals additional parts of the proof to the verifier in response to its requests; the moment a part of the proof is revealed, it is fixed. While the strength of the IOP model is equivalent to that of the PCP model, its advantage lies in the observation that in many PCP systems, interactive writing of the proof could reduce significantly the resources required by the prover to convince the verifier, and in particular, reduce the proof size as well, without harming the soundness of the system. Moreover, this can produce short proof IOP systems based on PCP protocols with enormous proofs. Ben-Sasson et al. [13] achieved proofs of size $O(T \cdot \log^2 T)$.

**Challenges in practical constructions** While the classical and interactive proof systems have trivial practical implementations in the real world, this is not the case with proof system for $\mathsf{NEXP}$. There are two classes of proof systems for $\mathsf{NEXP}$: (i) one based on isolated provers as in MIP and (ii) the other based on huge proofs that are too long for the verifier to read, as in PCP.

In the first case, it is very challenging, or may even be impossible, to ensure isolation of two untrusted machines (provers), especially in the case of MIP when they both should have a communication channel with a shared third machine, the verifier. Based on today’s variety of different communication channels, it could be infeasible to make sure no information is passed between two machines that are not under the control of a trusted authority. Protocols that do not rely on isolation of provers are called single prover protocols.

In the second case, the one that involves huge immutable proofs, the core problem is the storage of the proof. If it is stored on a device trusted by the verifier, then the verifier must be powerful enough to store it, which is a very expensive solution, and makes implementations of such protocols over the Internet worthless, as the verifier has to read the proof completely in order to download it. The other option is keeping the proof at a location that is untrusted by the verifier and allowing it access to it. In such a naive implementation, the verifier cannot ensure the proof is really fixed, and the responses from the server holding the proof are not adaptive.

In order to allow practical implementations of PCP systems, Kilian and Micali [43, 44, 48] introduced argument systems. Argument systems are similar to proof systems and differ only in the computational assumptions about the prover. Soundness of proof systems do not assume anything about the prover’s computational power; they rely on the assumption that the prover’s power is bounded, and in particular, that there are solvable problems that it cannot solve as it is not strong enough. Their work relied on a commitment scheme based on the work of Merkle.
[47], called the Merkle commitment, which assumes the existence of a pseudorandom function. The Merkle commitment is a computationally binding scheme that is designed to commit huge data while its revealing phase does not require revealing the huge data completely and could be used to reveal only a small fraction of it, with an overhead proportional to the number of bits revealed. The size of the commitment could be tiny and depends on the computational power of the prover – stronger provers require longer commitments. An important disadvantage of this commitment scheme is the communication complexity overhead incurred when revealing, which grows by a factor of $\theta(\log(n))$, where $n$ is the length of the huge committed data, whenever only small fractions of data are accessed. The above-mentioned work of Kilian and Micali introduced general compilers that can construct interactive argument systems for any PCP system, where the prover first commits the huge proof, and then reveals fractions of it in response to the verifier’s demand. They even showed that based on the Fiat-Shamir heuristic [33], it is possible to construct noninteractive arguments, which are small arguments that could be read entirely by the verifier, similar to classical proofs. The work of [22] showed that IOP systems could be compiled into noninteractive arguments in a similar way. For most practical uses, it is believed that common cryptographic hash functions, as, for example, SHA-256, which has both succinct representation and can be computed efficiently.

Another interesting solution, suggested by Biehl et al. [26], is to use private information retrieval (PIR) protocols instead of commitment schemes to ensure nonadaptiveness of the prover. PIR protocols were introduced in the celebrated work of Chor et al. [30]. We say a protocol is a PIR protocol if it allows a client to retrieve a record in a database held by a server without revealing to the server which record it retrieved. Practical PIR protocols can be used to construct a practical PCP protocol, where the client is the verifier, the server is the prover, and its database is the proof itself. The verifier’s queries are obfuscated using the PIR protocol. Obviously, if the prover is honest, the verifier would always accept the proof in such system; what is left to verify is the nonadaptiveness of the prover. Intuitively, while the prover can, in fact, be adaptive, in the case where both the queries and responses are obfuscated using the PIR protocol, it seems adaptiveness is useless for a devilish prover. The work of Aiello et al. [6] showed that construction of a practical PCP using a practical PIR is indeed possible. Unfortunately, the only possible PIR single-server protocol that gives information-theoretic privacy is the trivial protocol where the server sends the entire database to the client – which is, of course, unacceptable for our requirement of succinct verifications. There are some single-server PIR protocols with privacy based on computational assumptions, but as far as the authors know, these all require great super-poly-logarithmic execution of the verifier, relative to the database size, thus not aiding in constructing efficient and practical PCP systems.

Ishai et al. [39] presented a weaker variant of PIR where the database must represent a linear function. They used homomorphic encryption to construct super-efficient PIR systems for the special case where the database is a linear function. In [39], the client can be succinct. The construction of [39] does not only supply a PIR system for linear functions. It also enforces the committed database to be a linear function – a useful property in many PCP protocols that implement some form of low degree testing. This work was the inspiration for many practical constructions of solutions for computation verification, both general and for a specific purpose, e.g., [18, 20, 49].

Similarly to the Merkle commitment based versions, PCP implementations based on [39] can be compiled into noninteractive arguments, much smaller than those possible in the version that
relies on Merkle commitments, mostly because of the revealing phase overhead the Merkle commitment scheme introduces. A disadvantage of systems based on [39] is the fact that they rely on stronger computational assumptions than those of systems based on Merkle commitments, specifically assumptions from the field of algebraic geometry. Moreover, they are based on public key cryptography, thus requiring private randomness in order to construct two public keys: a proving key that is used by the prover and a verification key that is used by the verifier. While both these keys are public, the randomness used to construct them must be kept secret from the prover, since if it is revealed, a proof can be forged for any statement, even false statements. Given a verifier and prover key pair, only verifiers that know the verification key can verify the proof, which leads to proofs that are designated-verifier. Moreover, the process of key generation requires great computational power and thus cannot be done by a weak (succinct) verifier, although it is possible to use a single pair of keys for many different proofs. The process of key generation is called the initialization process. A common solution used to enable both publicly verifiable proofs and succinct verification is delegation of the initialization process to a global trusted party that is expected to publish to the world the verification and proving keys while eliminating any reminder of the randomness used to construct these keys. Such solutions raise many sociological and security based issues, e.g., how could this party or its hardware really be trusted, especially in cases where the gain from cheating can be enormous, as, for example, in the case of cryptographic currencies[18].

1.5 Zero-Knowledge

**General introduction to Zero-Knowledge**  The concept of Zero-Knowledge was introduced independently by Goldwasser et al. [38] and Brassard et al. [28]. We say a proof system is Zero-Knowledge if it is possible to efficiently construct transcripts that are indistinguishable from possible transcripts of communication between the prover and the verifier. More formally, we say a protocol is Zero-Knowledge if there is an efficient random algorithm called a simulator that generates valid transcripts with an indistinguishable distribution to that of real transcripts. Intuitively, we say that if a protocol is Zero-Knowledge, then the verifier did not learn anything from the prover, as otherwise the verifier could use the efficient simulator to construct valid traces and learn something directly from them without using the prover at all. If there exists a simulator only for the honest verifier that executes the real protocol, although it might be curious and try to learn additional information from the prover’s responses, we say that the protocol is an honest verifier Zero-Knowledge protocol. Otherwise, if there exists a simulator for any verifier, even a malicious verifier that tries to gain extra information by cheating and not implementing the protocol, we say it is simply a Zero-Knowledge protocol, and the prover’s privacy is preserved even against malicious verifiers. Although we will focus our discussion on Zero-Knowledge protocols that do not assume that the verifier is honest, we would like to mention honest verifier Zero-Knowledge protocols are sufficient whenever a proof is compiled into noninteractive arguments, and could provide noninteractive Zero-Knowledge arguments.

**Distribution indistinguishability sorts**  As defined above, the formal definition of Zero-Knowledge requires distribution indistinguishability. There are three different definitions of such indistinguishability, implying three different sorts of Zero-Knowledge protocols. The first is (i) perfect Zero-Knowledge, i.e., the case when the distributions are identical. A slightly weaker version is (ii) statistical Zero-Knowledge, which describes the case where the distributions are statistically
close. The last and the weakest sort is (iii) computational Zero-Knowledge, which describes the case where the distributions are computationally indistinguishable.

**Zero-Knowledge for verifying computations** Computation verification Zero-Knowledge protocols imply that the verifier cannot learn anything but the fact of the existence of a valid private input $I_{\text{private}}$. As we already mentioned, this problem is $\mathcal{NEXP}$-complete; thus, in order to achieve a Zero-Knowledge protocol, it has to be a protocol for $\mathcal{NEXP}$-complete problems. It is known there is a perfect Zero-Knowledge $\mathcal{MIP}$ protocol for any language in $\mathcal{NEXP}$ [11, 32, 46], but no perfect Zero-Knowledge protocol for $\mathcal{NEXP}$ using a single-server prover were known until the recent work of Ben-Sasson et al. [16] using the interactive oracle proof model. Previous work [40–42, 45] achieved statistical Zero-Knowledge for $\mathcal{NEXP}$ using the PCP model, with their main technique being locking schemes. While statistical Zero-Knowledge seems good enough for any practical use, the main practical advantage of [16] is in the length of the proof. Previous work compiled general PCP systems into Zero-Knowledge PCP systems, and expanded the proof polynomially. For example, applying this technique to the efficient PCP system for computation verification, which is based on [24] and achieves proof length of $O(T)$, would result in a proof of length $\omega(T^c)$ for some $c > 1$. The work of [16] modified the specific IOP construction of [13] to ensure Zero-Knowledge. The addition of Zero-Knowledge does not change the asymptotic properties of the IOP system and the proof length remains $O(T \cdot \log^2(T))$. The only overhead is a small multiplicative constant, namely, a factor of approximately 2 is sufficient in order to ensure perfect Zero-Knowledge against any verifier querying less than $T$ bits.

The work of [36] showed how statistical Zero-Knowledge can be achieved with noninteractive argument systems based on [39] with negligible overhead, and without enlarging the size of the noninteractive argument. This was in contrast to argument systems based on Merkle commitments where addition of Zero-Knowledge slightly extends the argument size.

### 1.6 State-of-the-art solutions

Several solutions to the verifiable-computation problem have been published.

#### 1.6.1 Verifiable-computation with succinct verification

The work of [12] implements a system based on the IOP protocol of [13], which is compiled into an argument system using the technique of [22, 44, 48]. The main characteristics of this work are $O(T \log^2(T))$ proving arithmetic complexity, and $O(\log(T)^2)$ verification arithmetic-complexity. It does not feature zero-knowledge and is plausibly quantum-secure.

Similar methods were used in the construction of [15], which was developed based on additional ideas from [14, 17], adding perfect zero-knowledge (in the IOP model) and improving the efficiency of the prover to $O(T \log^2(T))$ and the efficiency of the verifier to $O(\log(T))$ arithmetic complexity. For the special case when RAM (random access memory) is not required, the prover’s complexity is improved to $O(T \log(T))$, and the system in this case is called scalable.

#### 1.6.2 Verifiable computation for circuits

When programs are described as circuits, the complexity of the description is the same as the complexity of programs’ execution ($|P| \sim T$). For simplicity, we use $|C|$ to represent the circuit size
\(|\mathcal{C}| = \Theta(|\mathcal{P}|) = \Theta(\mathcal{T})\). The main motivation for such cases is systems with zero-knowledge, and the different solutions differ by their cryptographic assumption, and their communication complexity.

The work of [7] introduced a system that assumed only the existence of a collision-resistant hash function, providing \(O(\sqrt{|\mathcal{C}|})\) communication complexity. The work of [21] used similar techniques as both [7, 15], reaching communication complexity of \(O(\log(|\mathcal{C}|))\) in the IOP model. Both these systems are plausibly quantum-secure. The work of [29] assumes the hardness of Elliptic-Curve Discrete-Logarithm Problem (ECDLP), achieving communication complexity of \(O(\log(|\mathcal{C}|))\) as well.

Based on the protocol of [36, 39], the notable implementations are found in [20, 31, 49]. There are two different approaches to using the systems mentioned above: (i) non-succinct verification, where the verifier generates proving and verification keys, and in particular designed-verifier, or (ii) trusted-setup requiring systems, where the verifier delegates construction of the keys to a trusted party, offloading the load and responsibility of key generation to it. In such cases, the soundness of the system relies on the trusted party not revealing the randomness used for key generation. The protocol of [27] introduces a distributed setup for such protocols, used to reduce the trust assumptions in such systems.
Chapter 2

Research methods

Our research, at its highest level, is an endeavor to move ideas from theory into practice. The objective of the research is to take the ideas of computational-integrity proofs with zero-knowledge, and show that they can be implemented in systems whose performance appeals to and suits industrial applications. To measure the performance we used simulation and reported the systems' performance. We implemented our simulator in C++, a programming language both rich enough to easily describe complex systems and data structures, and known to compile to efficient executables, providing measurements as close to industrial systems as we could achieve.

Initial simulation trials showed that the theory does not support implementations sufficiently performant for industrial use cases, thus our research involved theoretical research aimed at improving existing theoretical constructions. Our theoretical research focuses on both improving efficiency and expanding the feature set of the existing theory, e.g., by allowing privacy using zero-knowledge.

In the theoretical research we used various known methods such as analysis of zero-knowledge protocols using simulators, using low-degree testing, and probabilistic analysis of protocol soundness. One interesting method we developed in Zero Knowledge Protocols from Succinct Constraint Detection is the Succinct Constraint Detection method. The Succinct Constraint Detection method shows how, given a linear code having long block length $n$ and a set of indices between 0 and $n$, to determine all linear dependencies over values of codewords over the specified indices. Whenever it is possible to compute all such dependencies efficiently, this method enables efficient simulation of reading small parts of a random codeword. Such simulation is essential for succinct zero-knowledge simulation of a proofs of proximity protocols such as common low-degree testing protocols, as they require the verifier to sample a codeword exponentially longer than its computational complexity bound. The main difference between the analysis of [17] and the one in Zero Knowledge Protocols from Succinct Constraint Detection is that the former required the simulator to draw a random Reed-Solomon codeword and compute for it its proof of proximity, achieving zero-knowledge only for NP, while in the latter, the simulator used Succinct Constraint Detection to directly access an unknown random proof of proximity, for some unknown random Reed-Solomon codeword.
Chapter 3

Findings

In this section we briefly highlight the main contribution of each published work. Computational Integrity with a Public Random String from Quasi-Linear PCPs, aka SCI, provides the first implementation of a computational-integrity system, assuming only the existence of a collision-resistant hash function (CRH) and one-shot universal scalability. SCI is a first of a kind proof-of-concept implementation. The protocol neither features zero-knowledge nor provides appealing performance—although the verification has pleasing concrete efficiency. Its communication complexity is on the order of dozens of megabytes and the proving overhead is approximately eight orders of magnitude. The following works aims to improve the result in Computational Integrity with a Public Random String from Quasi-Linear PCPs, using various approaches.

In Zero Knowledge Protocols from Succinct Constraint Detection, we showed how the SCI protocol can be made perfect zero-knowledge, even with only a succinct verifier, using the fine analysis of the low-degree test used \cite{24}. This result is based on the work of \cite{17}, which shown a protocol construction based on SCI featuring perfect zero-knowledge in the interactive-oracle proof model, using a simulator having computational complexity proportional to the computational complexity of the prover and, in particular, is not succinct. Thus the result in \cite{17} implies only witness indistinguishably for succinct verifiers. In Zero Knowledge Protocols from Succinct Constraint Detection, we introduced fine analysis of the protocol suggested in \cite{17}, and a succinct simulator showing it features perfect zero-knowledge even for succinct verifiers.

In Interactive Oracle Proofs with Constant Rate and Query Complexity, we demonstrated how interactivity can be used to improve proving complexity, specifically applied on the sublinear sumcheck protocol. In Fast Reed-Solomon Interactive Oracle Proofs of Proximity, aka FRI, we showed how the low-degree test of \cite{24} can be improved using interactivity, thereby providing both better asymptotic query-complexity, and concrete performance for prover and verifier. These are good examples of the way interactivity contributes both to efficiency of provers and protocol soundness, for protocols classically described using PCP proof composition. The prover efficiency is improved by not requiring it to generate the entire proof tree, and to progress only when requested by the verifier. The soundness is improved as the range of possible subproofs no longer need to be bounded, what was a common methodology in PCP constructions used to preserve the prover’s efficiency.

In Scalable, Transparent, and Post-Quantum Secure Computational Integrity, aka zkSTARK, we improved the construction of computational-integrity solution further, featuring zero-knowledge, and improving the concrete efficiency both by using FRI, and additional improvements.
In *Aurora: Transparent Succinct Arguments for R1CS*, we showed how ideas similar to those used in zkSTARK can be fine-tuned to solve computational-integrity with non-succinct verifiers, e.g., circuit satisfaction.
Computational integrity with a public random string from quasi-linear PCPs.

Eli Ben-Sasson¹, Iddo Bentov², Alessandro Chiesa³, Ariel Gabizon⁴, Daniel Genkin⁵, Matan Hamilis¹, Evgenya Pergament¹, Michael Riabzev¹, Mark Silberstein¹, Eran Tromer⁶ and Madars Virza⁷

¹ Technion — Israel Institute of Technology
² Cornell University
³ University of California, Berkeley
⁴ Zerocoin Electric Coin Company (Zcash)
⁵ University of Pennsylvania and University of Maryland, College Park
⁶ Tel Aviv University
⁷ Massachusetts Institute of Technology

A party executing a computation on behalf of others may benefit from misreporting its output. Cryptographic protocols that detect this can facilitate decentralized systems with stringent computational integrity requirements. For the computation’s result to be publicly trustworthy, it is moreover imperative to use publicly verifiable protocols that have no “backdoors” or secret keys that enable forgery. Probabilistically Checkable Proof (PCP) systems can be used to construct such protocols, but some of the main components of such systems — proof composition and low-degree testing via PCPs of Proximity (PCPPs) — have been considered efficiently only asymptotically, for unrealistically large computations. Recent cryptographic alternatives suffer from a non-public setup phase, or require large verification time.

This work introduces SCI, the first implementation of a scalable PCP system (that uses both PCPPs and proof composition). We used SCI to prove correctness of executions of up to $2^{20}$ cycles of a simple processor, and calculated its break-even point: the minimal input size for which naïve verification via re-execution becomes more costly than PCP-based verification.

This marks the transition of core PCP techniques (like proof composition and PCPs of Proximity) from mathematical theory to practical system engineering. The thresholds obtained are nearly achievable and hence show that PCP-supported computational integrity is closer to reality than previously assumed.

⋆ ⋆ ⋆ Work done while at Technion
† Work done while at Technion and Tel Aviv University
1 Introduction

Computational Integrity An unobserved party is often required to execute a program \( P \) on data \( x \), using auxiliary data \( w \). Yet, that party might benefit from misreporting the output \( y \). For example:

1. Individuals and companies may benefit financially from reporting lower tax payments; in this case \( P \) is the program that computes tax, \( x \) is the tax-relevant data (\( w \) is the empty string) and \( y \) is the resulting tax.
2. Criminals may benefit if an innocent individual (or no individual) is prosecuted based on faulty crime-scene data analysis, and corrupt law enforcement officials to reach this outcome. In this case \( P \) is the program that analyzes crime-scene data, \( x \) may contain the cryptographic hashes of (i) a criminal DNA database and (ii) DNA fingerprints taken from the crime-scene, \( w \) is the preimage of (i), (ii) and \( y \) would be the name of a suspect.
3. Health-care and other insurance companies may benefit from mis-computing policy rates. In this case \( P \) may be a government-approved program that computes policy rates, \( x \) is the identifying number of a patient, \( w \) would be her medical history (including, perhaps, her DNA sequence) and \( y \) is the policy rate.

Naturally, correctness and integrity of the input data \( x, w \) are preliminary requirements for obtaining a correct output \( y \); These inputs often arrives from third parties and can be digitally signed by them, hence changing \((x, w)\) maliciously to \((x', w')\) would require their collusion. Instead, the main focus of this work is on ensuring the integrity of the computation \( P \) itself, e.g., ensuring that the reported tax \( y \) is correct with respect to the explicit input \( x \), program \( P \) and some auxiliary input \( w \). In spite of incentives to cheat, we often assume that unobserved parties operate with computational integrity (CI) meaning that CI statements like

\[
\tau_{(P,x,y,T)} := \exists w \text{ such that } y = \text{ output of } P \text{ on inputs } x, w \text{ after } T \text{ steps} \quad (*)
\]

are considered true, even when the party making the statement could benefit from replacing \( y \) with \( y' \neq y \). The assumption that parties operate with computational integrity is backed by (i) legislation and (ii) regulation, and also relies on (iii) the economic value of “integrity” to individuals, businesses and government. Manual enforcement of CI via audits and reports by trusted third parties is labor-intensive, and yet leaves the door open to corruption of those third parties. Automated CI based on cryptography (also called delegation of computation [44], certified computation [33] and verifiable computation [41]) could potentially replace this manual labor and, more importantly, introduce integrity to settings in which it is currently too costly to achieve.

Interactive proof (IP) systems [5, 45] revolutionized cryptographic CI by initiating an approach that led (see below) to a viable theoretical solution to the problem of discovering false CI statements. In such systems the party that makes the CI statement (*) is represented by a prover which is a (randomized)
algorithm. The prover tries to convince a verifier — an efficient randomized algorithm — that (*) is true via a court-of-law-style interactive protocol in which the verifier “interrogates” the prover over several rounds of communication. The protocol ends with the verifier announcing its verdict which is either to “accept” \( \tau(P,x,y,T) \) as true, or to “reject” it. The systems we focus on have only one-sided error: all true statements can be supported by a prover that causes the verifier to accept them but the verifier may err and accept falsities; the probability of error is known as the soundness-error.

**Probabilistically checkable proof (PCP) systems**\(^8\) [4, 3, 2, 1] are a particularly efficient multi-prover interactive proof (MIP) system [8] in terms of the amount of communication between prover and verifier, verification time, the number of rounds of interaction and soundness-error. Assuming \( T \) is given in binary, the set of true CI statements (eq:statement) is a NEXP-complete language and PCPs are powerful enough to prove membership in this language. Here, the prover writes once a string of bits \( \pi(P,x,y,T) \) known as a PCP; its length is polynomial in the execution time \( T \). Total verifier running time is poly log \( T \), which is (i) negligible compared to the naïve solution of re-executing \( P \) at a cost of \( T \) steps and (ii) nearly-optimal because every proof system for general CI statements must have the verifier running time be at least \( O(\log T) \). Using a single round, the verifier asks to read a small (randomly selected) number of bits of \( \pi(P,x,y,T) \); clearly the verifier cannot read more bits than its running time (poly log \( T \)) allows, and this amount can be further reduced to a small constant that is independent of \( T \) (cf. [67, 50, 35, 64]). Initial constructions required proofs of length poly(\( T \)) but length has been reduced since then [49, 43, 24, 21] and state-of-the-art proofs are of quasi-linear length in \( T \), i.e., length \( T \cdot \text{poly log} \ T \) [23, 35, 20, 63] and can be computed in quasi-linear time as well [13]. The system reported — called Scalable Computational Integrity (SCI) — implements the quasi-linear PCP system [23, 13] with certain improvements (described later).

In many cases the prover needs to preserve the privacy of the auxiliary input \( w \) (as is the case with examples 2, 3 above) while at the same time proving that it “knows” \( w \), as opposed to merely proving that \( w \) exists. Privacy-preserving, or zero knowledge (ZK) proofs [45] and ZK proofs of knowledge [7] can be constructed from any PCP system in polynomial time [56, 37, 57] (cf. [53, 54, 61, 55]). Certain “algebraic” PCP systems, including SCI, can be converted to ZK proofs of knowledge with only a quasilinear increase in running time [11]; implementing this enhancement is left to future work.

A PCP verifier requires random access to bits of \( \pi(P,x,y,T) \); a naïve implementation in which prover sends the whole proof to the verifier would cost poly(\( T \)) communication (and verification time) but a collision-resistant hash function can be used to reduce communication and verifier running time to poly log \( T \) [56]. The three messages transmitted between prover and verifier ((1) prover sends proof; (2) verifier sends queries; (3) prover answers queries) can be reduced to a single message from the prover, if both parties have access to the same random function [62]; this can be realized using a standard cryptographic hash function such as

\(^8\) PCPs are also known as holographic, and transparent proof systems.
SHA-3, via the Fiat-Shamir heuristic [39] (or via an extractable collision resistant hash function [27]). The single message (published by the prover) is known as a succinct computationally sound (CS) proof \( \hat{\pi} \); its length is poly log \( T \) and it can now be appended to \( \tau_{P,x,y,T} \) and then publicly verified in time poly log \( T \) with no further interaction with the prover. We refer to \( \hat{\pi} \) as a hash-based (CI) proof to emphasize that the only cryptographic primitive needed to implement it is a hash function.

**Prior CI solutions** In spite of the asymptotic efficiency of PCPs, prior CI approaches (recounted below) did not implement a PCP system. To quote from the recent survey [78], the reason for this was that “the proofs arising from the PCP theorem (despite asymptotic improvements) were so long and complicated that it would have taken thousands of years to generate and check them, and would have needed more storage bits than there are atoms in the universe”. Due to this view (which this work challenges), five main alternatives have been explored recently, described below. Like SCI, all rely on arithmetization [60], the reduction of computational integrity statements (*) to systems of low-degree polynomials over finite fields. But in contrast to SCI, all previous solutions circumvent the use of core PCP techniques like proof composition [2], low-degree testing and the use of PCPs of proximity (PCPP) [20, 36]; these techniques are crucial for obtaining succinctly verifiable proofs with a public setup process, which SCI is the first to implement.

**IP-based:** The proofs for muggles approach [44] scales down Interactive Proofs (IP) from PSPACE to P and leads to excellent solutions for a limited yet interesting class of programs: those with high parallelism and small memory consumption; prover time for IP-based systems was reduced to quasi-linear [34] and implemented in a number of works [33, 74, 76].

**LPCP-based:** [52] proposed using additively homomorphic encryption (AHE) and linear PCPs (LPCP) to build CI proof systems that are interactive, and where the verifier’s work is amortized over multiple statements; cf. [70, 73, 72] for implementations of LPCP-based systems.

**KOE-based:** A sequence of works [47, 41, 59, 29, 42] improved on [52] by relying on Knowledge Of Exponent (KOE) assumptions and bilinear pairings over elliptic curves. KOE-based systems were implemented in [66, 71, 15, 19, 77], and further optimizations of this latter system for specific applications related to Bitcoin [65] such as smart contracts [58] and anonymous payment systems [12] are already being evaluated by commercial entities [46].

**IVC-based:** KOE-based systems require a proving key \( k_P \) (discussed below) that is longer than \( T \), the number of computation cycles. Incrementally verifiable computation (IVC) [75] and bootstrapping [28] shorten the length of \( k_P \) to poly log \( T \) and an IVC-based system has been implemented recently [18].

**DLP-based:** KOE/IVC-based systems require a private setup phase that is discussed below. [48] (cf. [69]) assumes hardness of the Discrete Logarithm Problem (DLP) to build a system that requires only a public setup, like SCI.
Proof length in the initial works above was \( \Theta \left( \sqrt{T} \right) \) and this was reduced to \( \text{poly log } T \) in [30], which also implemented both versions; verifier running time in both variants is \( \Omega(T) \).

**Comparing SCI to prior CI solutions** SCI is the first CI solution that achieves both (1) a short public randomness setup phase and (2) universal scalability for one-shot computation. We discuss the significance of these properties after explaining them. (A quantitative comparison of the running time, memory consumption and communication complexity of SCI to prior systems appears in Section 2 and Table 1.)

**One-shot universal scalability (OSUS)** A CI system is *universally scalable* if for any fixed program \( P \), prover running time is bounded by \( T \text{poly log } T \) and verification time is at most \( \text{poly log } T \) where \( T \) is the number of machine cycles\(^9\). If the same asymptotic running times hold even for a single execution of \( P \), and where the setup (“preprocessing”) is carried out by the verifier (and hence setup-cost is part of the total verification-cost), we shall say that CI solution is *one-shot universally scalable* (OSUS). DLP-based systems have super-linear verification time, hence are not *scalable* for any program. IP-based systems are efficient only for highly-parallel computations, thus are not *universally scalable*. LPCP- and KOE-based systems are universally scalable but not OSUS because they require a proving key \( k_P \) that is longer than \( T \) which must be generated by the verifier (in the one-shot setting). Of all prior solutions, only the IVC-based one is OSUS, like SCI.

**Public setup** All implemented solutions but for DLP-based and SCI, if instantiated as publicly verifiable CI systems, require a setup phase (“preprocessing”), the output of which is a pair of keys \((k_P, k_V)\), one needed for proving statements, the other for verifying them. A “trapdoor key” \( k_{tpdr} \) is associated with \((k_P, k_V)\) and can be used to forge pseudo-proofs of false statements. Furthermore, \( k_{tpdr} \) can be recovered by the parties that run the preprocessing phase. Secure multi-party computation can boost security by “distributing knowledge” of the trapdoor among several parties [17] so that all of them have to be compromised to recover \( k_{tpdr} \); but this does not remove the concern that \( k_{tpdr} \) has been recovered by collusion of all parties, or retrieved by a central party eavesdropping to all of them. Even if \( k_{tpdr} \) has not been recovered by anyone, its mere existence may erode trust in such systems. (Cf. [6] for a recent discussion of setup-attacks and their implications and mitigations.) In contrast, SCI and DLP-based systems require only a short public random string when instantiated as a publicly verifiable noninteractive CI system.

**Discussion** The combination of OSUS and public setup which is unique to SCI has three implications: (i) the ease of setting up and modifying CI systems based on it is relatively small, (ii) the trust assumptions made by parties using it are comparatively minor and hence (iii) it seems more suitable than existing

\(^9\) Formally, a CI system is *universally scalable* if for any language \( L \in \text{NTIME}(T(n)) \) prover running time is \( T(n) \text{poly log } T(n) \) and verifier running time is \( \text{poly log } T(n) \) where \( n \) denotes input length.
solutions for use in decentralized and public settings, like Bitcoin. We repeat and stress that many such applications require zero-knowledge proofs, a property achieved by prior solutions and not achieved by SCI; augmenting SCI to obtain zero knowledge seems within reach [11] but is outside the scope of our work.

SCI—Main technical contributions We faced three major challenges when attempting to construct PCP systems that scale well and apply to general programs, and SCI is the first implementation to contain scalable solutions to each of them, reported here for the first time: (i) implementing the recursive proof composition [2] technique applied to PCPs of proximity (PCPPs) [20, 36] (ii) constructing quasi-linear PCPP systems for Reed-Solomon (RS) error correcting codes [68] of huge message length [23] that require, in particular, quasi-linear time algorithms for interpolation and multi-point evaluation of large-degree polynomials over finite fields of characteristic 2; and (iii) reducing general programs that include jumps, loops, and random access memory (RAM) instructions to succinct Algebraic Constraint Satisfaction Problem (sACSP) instances that “capture” the corresponding CI statement (*); prior arithmetization solutions require the verifier, or a party trusted by it, to “unroll” a $T$-cycle computation to obtain an arithmetic circuit of size $\Omega(T)$, whereas SCI’s verifier is succinct and does not perform this unrolling. (All prior solutions arithmetize over large prime fields; SCI is also novel in its being the first arithmetization over large binary fields, which poses new challenges, especially for integer operations like addition and multiplication, cf. Section B.1.)

To overcome the blowup (i) that is due to recursive PCPP composition, we replace PCPPs with interactive oracle proofs of proximity (IOPPs) [38, 9, 10], implemented here for the first time, and increase the number of rounds of interaction between prover and verifier; the extra rounds can be removed in the random oracle model [38]. To address (ii) we built a dedicated library that implements finite field arithmetic efficiently (reported in [22]) and used it to further implement additive Fast Fourier Transforms (aFFT) [40] that perform interpolation and multi-point evaluation in quasi-linear time and in parallel (via multi-threading); the large-scale additive FFTs are reported here for the first time. To solve (iii) and reduce general programs to PCP systems efficiently, we devise a novel reduction from general programs for random access machines to sACSP instances. We describe these three contributions in more detail in Section 3 and the appendix.

2 Measurements

SCI can be applied to any language in \textbf{NEXP}; for concreteness we picked two programs computing the \textbf{NP-complete} subset-sum problem (cf. Appendix C); we explain this choice after introducing the two programs. The input to the subset-sum problem is an integer array $A$ of size $n$ and a target integer $t$; the problem is to decide whether there exists a subset $A' \subset A$ that sums to $t$. The CI statement addressed here is the co-NP version of the problem, stating “no
subset of $A$ sums to $t$ and denoted by $\tau(A,n,t)$. The two programs differ in their time and space consumption. The first one exhaustively tries all possible subsets, requiring $2^n$ cycles but only $O(1)$ memory, hence can be executed using only the local registers of the machine and with no random access to memory. The second program uses sorting and runs in time $O(2^{n/2})$, a quadratic improvement over the exhaustive solution but it also requires $\Theta(2^{n/2})$ memory and hence uses the random access memory. We denote the two programs by $P_{exh}$ and $P_{sort}$, respectively.

On choice of programs We would like to run SCI on “real-world” applications like the examples given in the introduction but our current scalability is not up to par. This situation is similar to that of the very first works on other CI solutions (cf. [34, 70, 66, 15]): initial reports discussed only small word-size machines, restricted functionality and simple programs. Like some of those works (most notably, [19]) we use the 16-bit version of the TinyRAM architecture as our model of computation, and support all of its assembly code even though these two programs use only a subset of it. We focus on subset-sum for two reasons: (i) it is a natural NP-complete problem that is often used in cryptographic applications but more importantly (ii) it allows us to display the effect of time–space tradeoffs on our CI solution (cf. Figure 2). Since SCI supports non-determinism, we could have used the non-deterministic version of the subset-sum statement. In fact, this would have reduced prover and verifier complexity because fewer boundary constraints are imposed on the input. However, the resulting statement seems less interesting, saying “there exists $A$ such that no subset of it sums to $t$”.

Measurement range Input array size $n$ ranged between 3–16. Prover data was measured on a “large” server with 32 AMD Opteron cores at clock rate 3.2 GHz and 512 Gigabytes of RAM, running with two threads per core (total of 64 threads); to bound the single-core/thread prover time one may multiply the stated times by $\times 32 / \times 64$ respectively. Verifier data was measured on a “standard” laptop, a Lenovo T440s with Intel core i7-4600 at clock rate 2.1 GHz and 12 Gigabytes RAM. We stress that verifier succinctness for one-shot programs allows us to measure verifier running time independently of prover running time, all the way up to $2^{47}$ machine cycles. Both prover and verifier were measured for 1-bit security and 80-bit security using state-of-the-art PCPP and IOPP security estimates [9].

Prover time and memory The left column of Figure 1 presents the running time (top) and memory consumption (bottom) of the Prover for both $P_{exh}$ and $P_{sort}$ as a function of the number of machine cycles of the simulated machine for both 1-bit and 80-bit security level. The two main observations from these figures are that (i) resources scale quasi-linearly with number of cycles and (ii) $P_{sort}$ is more costly than $P_{exh}$ due to its random access memory usage, which increases proof length by $\times \log O(1) T$ factor for a $T$-cycle execution (cf. Section 3). Figure 2 compares time and memory as a function of the size on the input array $n$ and shows that for $n \geq 8$ the quadratic running-time improvement of $P_{sort}$ over $P_{exh}$.
outweighs the $\times O(\log T)$ factor required by random access to memory, both for 1-bit and 80-bit security level.

Verifier time and query complexity The right column of Figure 1 shows verifier running time (top) and query complexity (bottom) for both programs for both 1-bit and 80-bit security levels. Notice the $\approx 2^{13} - 2^{23}\times$ factor improvement of verifier over prover in both parameters (recall $1MB = 2^{10}KB$) and the increase in running time as a function of security due to repetition. For small $n$ verifier running time is greater than that of the naive verifier which re-runs the program. However, since naive verification grows like $2^n$ for $\mathbb{P}_{\text{exh}}$ and like $2^{n/2}$ for $\mathbb{P}_{\text{sort}}$, for $n \geq 22$ (at 80-bit security) our verifier is more efficient than the naive one for $\mathbb{P}_{\text{exh}}$, and for $n \geq 48$ the verifier for $\mathbb{P}_{\text{sort}}$ is more efficient than the naive one (cf. Figure 3).

Table 1: Quantitative comparison of SCI with KOE-based [15], IVC-based [18] and DLP-based [48] solutions. Data measured on executions of $2^{16}$ cycles of $\mathbb{P}_{\text{exh}}$ at an 80-bit security level on the same machine with 32 AMD Opteron cores at clock rate 3.2 GHz and 512 Gigabytes of RAM. The DLP-based column is extrapolated from [48, Table 2], accounting for (i) the larger circuit size of our computation (which has $\approx 132M$ gates compared with maximal size of 1.4M gates there) and different compute architectures (single threaded Intel 4690K core vs. 64 threaded AMD Opteron). Notice the proving time of SCI is $\approx \times 2 - 4$ slower than KOE- and DLP-based and $\approx \times 150$ faster than IVC-based. Regarding total communication complexity, SCI is more efficient than prior solutions but less efficient when measuring only post-processing communication.

<table>
<thead>
<tr>
<th>KOE-based</th>
<th>IVC-based</th>
<th>DLP-based</th>
<th>SCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verifier setup</td>
<td>time</td>
<td>~ 28 min</td>
<td>~ 18.9 GB</td>
</tr>
<tr>
<td></td>
<td>key length</td>
<td>~ 10 sec</td>
<td>43 MB</td>
</tr>
<tr>
<td>Prover</td>
<td>time</td>
<td>~ 18 min</td>
<td>4.2 days</td>
</tr>
<tr>
<td></td>
<td>memory</td>
<td>~ 216 GB</td>
<td>2.9 GB</td>
</tr>
<tr>
<td>Verifier postprocessing</td>
<td>time</td>
<td>&lt; 10 ms</td>
<td>~ 25 ms</td>
</tr>
<tr>
<td></td>
<td>communication complexity</td>
<td>230 bytes</td>
<td>374 bytes</td>
</tr>
<tr>
<td>Verifier total</td>
<td>time</td>
<td>~ 28 min</td>
<td>~ 10 sec</td>
</tr>
<tr>
<td></td>
<td>communication complexity</td>
<td>~ 154 MB</td>
<td>~ 42.5 MB</td>
</tr>
</tbody>
</table>

Quantitative comparison with other CI implementations Table 1 compares SCI to three recent CI systems, the KOE-based [15], the IVC-based [18], and the DLP-based [48], using the version with poly log($T$) communication complexity. One sees that SCI has the shortest and fastest setup but larger post-setup communication complexity; post-setup verification is faster than DLP-based but slower than KOE/IVC-based, as predicted by theory. Two other important points are: (i) proofs in SCI are not zero-knowledge whereas the other solutions are, and (ii) the setup of the last two columns (DLP-based and SCI) is comprised only of
a public random string, whereas KOE/IVC-based solutions require private setup and involve a trapdoor that can be used to forge proofs of false statements.

Fig. 1: Comparison of prover (left) and verifier (right) running time (top) and memory consumption (bottom). The sharp drop in query complexity is due to transition from 2 to 3 levels of recursion in the RS-PCPP; as seen in the top-right, this has little effect on overall verifier running time, which is significantly smaller than prover running time, and also grows at a considerably slower rate as a function of \# cycles. Answers to verifier queries provided by random strings which simulates accurately actual proofs because verifier is non-adaptive, i.e., its running time is independent of the proof content.
Fig. 2: Prover running time (left) and memory consumption (right) as a function of input array size $n$. For $n \geq 8$ the quadratic running-time improvement of $P_{\text{sort}}$ over $P_{\text{exh}}$ overcomes the $\times \text{poly log} T$ factor overhead of $P_{\text{exh}}$ due to random memory access; this holds for both 1-bit and 80-bit security level.

Fig. 3: Computation of the break-even point [73, 72], the minimal input size $n$ for which naïve verification via re-execution becomes more costly than PCP-based verification. For $P_{\text{exh}}$ at 80-bit security this threshold is at $n = 22$ and for $P_{\text{sort}}$ it is significantly higher, estimated around $n = 48$, due to quadratic improvement in running time of the latter program.

3 Overview of construction

The construction of the PCP $\pi_{(P,x,y,T)}$, for the computational statement $\tau_{(P,x,y,T)}$ follows the rather complex process detailed in [23, 21, 14, 13] which we summarize next (see Appendix A). The statement $\tau_{(P,x,y,T)}$ is converted into an instance $\psi_{(P,x,y,T)}$ of an algebraic constraint satisfaction problem (ACSP) over a finite
field of characteristic 2 and $\tau(P,x,y,T)$ is used by prover and verifier as described next.

**Prover** To construct the PCP, the prover executes $P$ on input $x$ and encodes the execution trace by a Reed-Solomon [68] codeword $a(P,x,y,T)$ evaluated over an additive sub-group of $\mathbb{F}$. The ACSP instance $\psi(P,x,y,T)$ is applied to $a(P,x,y,T)$ as described in [23, Equation (3.2)] to obtain an additional RS-codeword, denoted $b(P,x,y,T) = \psi(P,x,y,T)(a(P,x,y,T))$, that “attests” to the fact that $a(P,x,y,T)$ encodes a valid execution trace, and hence, in particular, its output is correct. Each of the two codewords is appended with a PCP of proximity (PCPP) for the RS-code [23], denoted $\pi_a, \pi_b$, respectively. The PCP $\pi(P,x,y,T)$ is defined to be the concatenation of $a(P,x,y,T), b(P,x,y,T), \pi_a$ and $\pi_b$.

**Verifier** The verifier queries the four parts of the PCP in the following manner: First it invokes an RS-PCPP verifier that queries $a(P,x,y,T)$ and $\pi_a$ to “check” that $a(P,x,y,T)$ is close in Hamming distance to a codeword of the RS-code; it repeats this process with respect to $b(P,x,y,T)$ and $\pi_b$. Second and last, the verifier queries $a(P,x,y,T)$ and $b(P,x,y,T)$ and uses $\psi(P,x,y,T)$ to check that the two codewords encode a valid computation of $P$ that starts with $x$ and reaches $y$ within $T$ cycles. In this process we rely on the “locality” of the mapping $\psi(P,x,y,T) : a(P,x,y,T) \rightarrow b(P,x,y,T)$ which means that each entry of $b(P,x,y,T)$ depends on a small number of entries of $a(P,x,y,T)$. In what follows we elaborate on the novel aspects of this reduction as implemented in SCI.

*From assembly code to succinct ACSP* The efficiency of the ACSP instance $\psi(P,x,y,T)$ is measured by three parameters that we seek to minimize: circuit size, degree, and query complexity, denoted $C(P,x,y,T), D(P,x,y,T), Q(P,x,y,T)$ respectively. Circuit size affects both proving and verification time; degree affects PCP length and reducing it decreases running time and memory consumption on the prover side; query complexity affects the length of communication between prover and verifier (and the length of computationally sound (CS) proofs $\hat{\pi}$) as well as verifier running time. Each parameter can be optimized at the expense of the other two, and the challenge is to reach an efficient balance between all three.

Our starting point is a program $P$, i.e., a sequence of instructions for a random access machine (RAM). For simplicity we first focus on instructions that access only (local) registers; random access memory instructions are discussed below. Each instruction specifies the input and output register locations and an operation applied to the inputs, called the *opcode*. We build $\psi(P,x,y,T)$ bottom-up (cf. Appendix B for a detailed example). Each opcode $op$ appearing in $P$ (like xor, add, jump, etc.) is specified by an algebraic definition over $\mathbb{F}$; in other words, we specify a set of multi-variate polynomials $P_{op} \subseteq \mathbb{F}[X_1, X_2, \ldots, X_m]$ such that the set of common zeros of $P_{op}$ correspond to correct input-output tuples for $op$. Program flow is controlled by multiplying each polynomial in $P_{op}$ by a multivariate Lagrange “selector” polynomial that, based on the value $v$ of the

---

10 SCI uses the field of size $2^{64}$ which suffices for the computations measured here.
program counter (PC), annihilates all constraints that are irrelevant for enforcing the $v$th instruction of $P$. For a program with $\ell$ lines these selector polynomials have degree $\lceil \log \ell \rceil$. The resulting ACSP has circuit size $O(\ell)$ and degree and query complexity are $\log \ell + O(1)$; the constants hidden by asymptotic notation depend on the machine specification.

**Random access memory instructions** The execution trace of $P$ is the length–$T$ sequence of machine states that describes the computation. To verify the integrity of random access memory instructions (such as load and store) we follow [13, 14] and use a pair of execution traces. The first trace, $\text{trace}^{\text{time}}$, is sorted increasingly by time, and the second, $\text{trace}^{\text{mem}}$, is sorted lexicographically first by memory location, then by time. RAM-related execution validity is verified “locally” by inspecting pairs of consecutive elements in $\text{trace}^{\text{mem}}$, just like non-RAM related instructions are verified “locally” by inspecting pairs of consecutive elements in $\text{trace}^{\text{time}}$. To further reduce proof length and query complexity, each state of $\text{trace}^{\text{mem}}$ contains only the information needed to check memory consistency — an address, its content and the type of memory access (load/store); let $s$ denote the number of field elements in a single line of $\text{trace}^{\text{mem}}$.

To prove that $\text{trace}^{\text{mem}}$ and $\text{trace}^{\text{time}}$ refer to the same execution, the prover must describe a permutation between the two, and the verifier must check its validity. To achieve this SCI uses a non-blocking Benes switching network [32, 25] embedded in an affine graph over $\mathbb{F}$ (cf. [23, 14] for definitions). Using this method, adding RAM-related instructions to a program adds only $O(T \cdot \log T)$ field elements to the PCP and increases query complexity by a small constant.

**Reducing proof construction time via Interactive Oracle Proofs of Proximity (IOPP)** A significant portion of the prover running time and memory consumption are dedicated to the construction of the PCP of Proximity (PCPP) for $a_{(P,x,y,T)}$ and for $b_{(P,x,y,T)}$. The full PCPP for an RS-codeword of degree $N$ is of length $O(N \log^{2.6} N)$ which is quite large in our applications. Observing that (i) these PCPPs are built using recursive PCPP composition [21], and (ii) only a small fraction of recursive branches are explored by the verifier, we increase the number of rounds of interaction and use a notarized interactive proof of proximity (NIPP) [9], a special case of interactive oracle proofs of proximity (IOPP) [38, 10] to reduce proof length to $4N + O(\sqrt{N})$. The added rounds of interaction can be removed in the random oracle model to obtain computationally sound proofs [38].

**Parallel implementation of PCPPs for RS codes** To reduce the time required to encode the execution trace into a pair of RS-codewords, SCI uses parallel algorithms for finite field operations and for dealing with polynomials over finite fields of characteristic 2. To speed up basic field operations (most notably, multiplication) a dedicated algebraic library was built, that utilizes parallel hardware on multi-core CPU. Interpolation and evaluation of polynomials over affine spaces of size $N$ are computed in quasilinear time using so-called *additive Fast Fourier Transform* (aFFT) [40].
4 Concluding remarks

SCI is the first implementation of a system of computational integrity that achieves asymptotic one shot universal scalability (OSUS) with a setup key that is merely a public random string. Prior solutions either required super-linear verification time, or used a setup procedure that involves keys which could be used to forge proofs of falsities. While the computer programs on which SCI was tested are of limited applicability, the simpler setup assumptions of SCI make it a natural starting point for building further applications — most notably zero knowledge proofs — for use in decentralized networks.

Acknowledgements

We thank Ohad Barta, Lior Greenblatt, Shaul Kfir, Gil Timnat and Arnon Yogev for programming support in early stages of this work. The research reported here has received funding from the following sources, sorted alphabetically: the Blavatnik Interdisciplinary Cyber Research Center; the Center for Long-Term Cybersecurity at UC Berkeley; the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370; the Check Point Institute for Information Security; the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258; the Israeli Centers of Research Excellence I-CORE program (center 4/11); the Israeli Science Foundation (grants 1501/14,1138/14); and the Leona M. & Harry B. Helmsley Charitable Trust.

References


46. Greenberg, A.: Zcash, an untraceable bitcoin alternative, launches in alpha (January 2016), [Wired.com; posted online 20-January-2016]

A Detailed PCP construction

We describe the way a PCP is generated for $\tau(x,y,T)$, then discuss its verification.

Proof generation The PCP proof $\pi(x,y,T)$ for $\tau(x,y,T)$ is a concatenation of four sub-proofs: two codewords in a Reed-Solomon code [68] and two quasi-linear size PCPs of Proximity (PCPP) for the RS-codewords [23]. To obtain these four sub-proofs, the prover starts by executing the program $P$ on input $x$ for $T$ steps and records its execution trace — the length–$T$ sequence of machine states that the machine goes through during execution. Each state is converted to a sequence of elements in the finite field $F$ of size $2^6$; Auxiliary field elements are appended to each state to reduce the degree complexity of $\tau(x,y,T)$ as described in Section B. Let $s$ denote the total number of field elements per state. The resulting algebraic trace $\tau_{aug}$ is thus a table of $N = \log s$ elements of $F$, and is viewed as a function from $S \subseteq F$, $|S| = N$ to $F$, where $S$ is an affine space over the two-element field. Prover now computes the low-degree extension (LDE) of $\tau^{aug}$ by interpolating and then evaluating $\tau^{aug}$ on a set $S' \subset F$ that is significantly larger than $S$. This results in a codeword $a(x,y,T)$ of a Reed-Solomon (RS) code [68] over $F$ of degree $\log N - 1$ and rate $\rho = |S'|/|S|$. Next, the ACSP instance $\psi(x,y,T)$ is applied to $a(x,y,T)$ as described in [23, Equation (3.2)], producing another RS-codeword $b(x,y,T) = \psi(x,y,T)(a(x,y,T))$, of degree $D(x,y,T) \cdot (N - 1) + 1$ and rate $\rho = D(x,y,T) \cdot \rho$ (SCI uses $\rho = \frac{1}{2}$). Finally, a PCP of proximity (PCPP) for RS-codes $\pi(x,y,T)$ is appended to each of $a(x,y,T)$ and $b(x,y,T)$ to prove that indeed each belongs to the RS-code of the designated rate — $\rho$ for $a(x,y,T)$ and $\rho'$ for $b(x,y,T)$; denote these PCPPs by $\pi_a$, $\pi_b$, respectively. Summing up, the PCP proof $\pi(x,y,T)$ is the concatenation of the four strings $a(x,y,T)$, $\pi_a$, $b(x,y,T)$ and $\pi_b$.

Proof verification On the verifier side, given $\psi(x,y,T)$ as input and oracle access to $\pi(x,y,T) = (a(x,y,T), \pi_a, b(x,y,T), \pi_b)$ as above, the verifier invokes the RS-PCPP verifier of [23] on each of $(a(x,y,T), \pi_a)$ and $(b(x,y,T), \pi_b)$. Then it checks that $a(x,y,T) = \psi(x,y,T)(b(x,y,T))$ by sampling...
both $a(P, x, y, T)$ and $b(P, x, y, T)$ at a small number of locations ($1 + Q(P, x, y, T)$ per test). To boost soundness, each of the aforementioned tests is repeated a number of times, using fresh randomness ($\text{SCI}$ uses 14 repetitions to reduce the probability of error to $\text{error} = \frac{1}{2}$). The verifier “accepts” $\tau(P, x, y, T)$ (i.e., proclaims it to be likely true) if and only if $\pi(P, x, y, T)$ passes all these checks; the security analysis guarantees that this verdict is correct with probability $1 - \text{error}$.

**B Algebraic definition of general programs as zero locus of low-degree polynomial system**

Our goal here is to explain how SCI converts programs into succinct algebraic CSP (ACSP) instances. For concreteness this is described for the TinyRAM machine specification [16] — a simple random access machine (RAM) with 16 registers and 16-bit size words that includes opcodes for logical operations, integer arithmetic, conditional jumps and random access memory instructions; the same techniques could be adapted to other machine specifications.

**Algebra preliminaries** Fix a basis $\beta_0, \ldots, \beta_{63}$ for $\mathbb{F}_{2^64}$ over $\mathbb{F}_2$ generated by an irreducible polynomial $h(X)$. Any sequence of $w$ bits $a_0, \ldots, a_{w-1}$ can be naturally mapped to the field element $\sum_{i=0}^{w-1} a_i \beta_i$ as long as $w < 64$ and vice versa, field elements can be converted to sequences of bits; we assume this natural mapping and in particular will often identify the a 16-bit sequence $(a_0, \ldots, a_{15})$ with the field element $\sum_{i=0}^{15} a_i \beta_i$.

**Overview of reduction** The reduction from RAM programs to ACSPs has been described in detail in [13] and further improved in [31]; we follow this route. In particular, instructions that involve the random access memory are verified using affine routing networks as explained in [13] (cf. [31]), although SCI uses an affine graph in which the Beneš network [26] is embedded. Boundary constraints (such as the initial and final state of the machine) are enforced as explained in [13]. A remaining problem of great practical importance that remained from previous works has been how to reduce efficiently the transition function described by a program into a set of low-degree polynomials whose zero-locus corresponds to a valid evolution of the program’s transition function. We describe this below. Our reduction works bottom up and has two main steps. (i) First, we define the input–output relation of each opcode as the zero-locus of a system of low-degree polynomials. (ii) In similar manner we define the transition function of the program as the zero-locus of a (larger) system of polynomials, one that uses the definitions of opcodes in terms of polynomials. The resulting set of polynomials is “glued” into a single large polynomial as described, e.g., in [23, Equation (5.5)] and [13, Section 10].

**B.1 Algebraic definition of opcodes**

Our basic data-unit is called a word, in TinyRAM its size is 16 bits. The atoms of a computer program are opcodes; each opcode has a fixed amount of input...
and output words. For example, XOR receives two words \( A = (a_0, \ldots, a_{15}) \), \( B = (b_0, \ldots, b_{15}) \) and its output is a single word \( C = (c_0, \ldots, c_{15}) \) where \( c_i = a_i \oplus b_i \) and \( \oplus \) denotes exclusive-or; the AND opcode outputs \( c_i = a_i \land b_i \); the ADD opcode performs integer addition, etc. (cf. [16] for details).

An opcode \( \text{op} \) with \( k \) inputs and \( \ell \) outputs defines a relation \( R_{\text{op}} \) that contains all sequences of inputs and outputs that correspond to valid executions of \( \text{op} \). Continuing with the examples above and using \( f \) to denote the flag,

\[
R_{\text{XOR}} = \left\{ (a, b, c) \in \{0, 1\}^{3 \cdot 16} \mid a_i \oplus b_i \oplus c_i = 0 \right\}
\]

\[
R_{\text{AND}} = \left\{ (a, b, c) \in \{0, 1\}^{3 \cdot 16} \mid (a_i \land b_i) \oplus c_i = 0 \right\}
\]

\[
R_{\text{ADD}} = \left\{ (a, b, c) \in \{0, 1\}^{3 \cdot 16}, f \in \{0, 1\} \mid \sum_{i=0}^{15} a_i 2^i + \sum_{i=0}^{15} b_i 2^i - \left( f \cdot 2^{16} + \sum_{i=0}^{15} c_i 2^i \right) = 0 \right\}
\]

An algebraic opcode is an opcode (as defined above) over an alphabet that is a finite field, i.e., \( R_{\text{op}} \subset \mathbb{F}^{k+\ell} \). Any finite set is an algebraic set, meaning it can be described as the zero-locus of a system of polynomials, however, these polynomials may have large degree and/or large arithmetic complexity, which would harm the efficiency of our reduction. To reduce degree and arithmetic complexity we shall allow auxiliary variables and consider algebraic sets \( S \) over \( \mathbb{F}^{k+\ell+m} \) such that \( R_{\text{op}} \) is the projection of \( S \) to the first \( k + \ell \) variables. Formally, an algebraic constraint system \( A_{\text{op}} \) corresponding to an opcode \( \text{op} \) with \( k \) inputs and \( \ell \) outputs is a set of polynomials \( A_{\text{op}} \subset \mathbb{F}[X_1,\ldots,X_k,Y_1,\ldots,Y_\ell,Z_1,\ldots,Z_m] \) such that

\[
R_{\text{op}} = \{ x_1, \ldots, x_k, y_1, \ldots, y_\ell \mid \exists z_1, \ldots, z_m, A_{\text{op}}(x_1, \ldots, x_k, y_1, \ldots, y_\ell, z_1, \ldots, z_m) = 0 \}
\]

(1)

We call \( X_1, \ldots, X_k \) the input variables, \( Y_1, \ldots, Y_\ell \) the output variables and \( Z_1, \ldots, Z_m \) are auxiliary variables. While any relation can be defined without any auxiliary variables, the degree of such \( A_{\text{op}} \) may be very large (e.g., in the case of AND, ADD), therefore, to minimize ACSP degree we shall often use auxiliary variables as shown in the following examples; explanations appear below but notice XOR uses no auxiliary variables and the AND opcode uses 48 of them. We defer the explanation of the more complicated ADD opcode to later on.

\[
A_{\text{XOR}} = \{ X_1 + X_2 + Y_1 \}
\]

(2)

\[
A_{\text{AND}} = \left\{ X_1 + \sum_{i=0}^{15} Z_i \beta_i, X_2 + \sum_{i=0}^{15} Z_{16+i} \beta_i, Y_1 + \sum_{i=0}^{15} Z_{32+i} \beta_i \right\}
\]

\[
\bigcup \{ Z_j \cdot (Z_j + 1) \mid j = 0, \ldots, 47 \}
\]

(3)

\[
\bigcup \{ (Z_1 \cdot Z_{16+i}) + Z_{32+i} \mid i = 0, \ldots, 15 \}
\]

(4)

(5)

Recall that addition in \( \mathbb{F} \) corresponds to exclusive-or, hence XOR has an algebraic constraint system with a single polynomial of degree 1 and no auxiliary
variables, and it satisfies (1). To see that (3)–(5) form an algebraic constraint system for AND we argue as follows. Suppose \((x_1, x_2, y_1, z_0, \ldots, z_{47})\) belongs to the zero-locus of \(A_{\text{AND}}\), i.e., all polynomials in \(A_{\text{AND}}\) vanish on this input. Then by (4) we have \(z_j \in \{0, 1\}\) for \(j = 0, \ldots, 47\). By (3) we see that \(z_{32+i} = z_i \wedge z_{16+i}\) for \(i = 0, \ldots, 15\). Finally, by (3) we see that \(x_1 \) “packs” \(z_0, \ldots, z_{15}\) into a single field elements, meaning \(x_1\) is the field element whose representation in the basis \(\beta_0, \ldots, \beta_{63}\) is the sequence \(z_0, \ldots, z_{15}, 0, 0, \ldots, 0\) and similarly \(x_2 \) “packs” \(z_{16}, \ldots, z_{31}\) and \(y_1 \) “packs” \(z_{32}, \ldots, z_{47}\). Therefore, \(y_1\) is the bitwise and of \(x_1\) and \(x_2\), as required by (1).

The constraints of the ADD opcode correspond to the operation of a full binary adder and appear below ((6)–(10)). In what follows auxiliary variables \(Z_0, \ldots, Z_{15}\) are used to “unpack” \(X_1\), auxiliary variables \(Z_{16}, \ldots, Z_{31}\) “unpack” \(X_2\), auxiliary variables \(Z_{32}, \ldots, Z_{47}\) are the carry bits and \(Z_{48}, \ldots, Z_{63}\) “unpack” the output \(Y_1\); the overflow flag is stored in \(Y_2\). The constraint set (6) “unpacks” both inputs and the output using 16 auxiliary variables each as done in (3) above. The constraint set (7) checks that each auxiliary variable is boolean (as done in (4)) but now we have 16 additional auxiliary variables for the carry bits, reaching a total of 64 auxiliary variables. The set of constraints (8) checks that the carry bits \((Z_{32}, \ldots, Z_{47})\) are computed correctly. In (9) the output is checked to be equal to the exclusive-or of the relevant input and carry bits. Finally, in (10) we check that the least significant carry and output bits are correct, and that the most significant carry bit \((Z_{47})\) equals the overflow flag \((Y_2)\).

\[
A_{\text{ADD}} = \left\{ X_1 + \sum_{i=0}^{15} Z_i \beta_i, X_2 + \sum_{i=0}^{15} Z_{16+i} \beta_i, Y_1 + \sum_{i=0}^{15} Z_{48+i} \beta_i \right\} \bigcup \left\{ Z_j \cdot (Z_i + 1) \mid j = 0, \ldots, 63 \right\} \bigcup \left\{ Z_i \cdot Z_{16+i} + Z_i Z_{31+i} + Z_{16+i} Z_{31+i} + Z_{32+i} \mid i = 1, \ldots, 15 \right\} \bigcup \left\{ Z_i \cdot Z_{16+i} + Z_{32+i} + Z_{48+i} \mid i = 1, \ldots, 15 \right\} \bigcup \left\{ Z_0 \cdot Z_{16} + Z_{32}, Z_0 + Z_{16} + Z_{48}, Z_{63} + Y_2 \right\}
\]

**Complexity of other opcodes** The opcodes described above, applied to \(w\)-bit registers, require \(O(w)\) constraints and auxiliary variables \((R_{\text{XOR}}\) requires \(O(1)\) constraints and auxiliary variables). All other opcodes of the TinyRAM assembly specification [16] can be implemented with \(O(w)\) complexity. For most opcodes this can be verified by inspection. For integer multiplication — i.e., to prove that

\[
\left( \sum_{i=0}^{w-1} a_i 2^i \right) \cdot \left( \sum_{i=0}^{w-1} b_i 2^i \right) = \sum_{i=0}^{2w-2} c_i 2^i, \quad a_i, b_i, c_i \in \{0, 1\}
\]

we fix a generator \(g\) for the multiplicative group of \(\mathbb{F}\) (the order of \(g\) is \(2^{63} - 1\) for our choice of field) and then apply repeated squaring to verify that

\[
\left( g(\sum_i a_i 2^i) \right)^{\sum_i b_i 2^i} = g(\sum_i c_i 2^i)
\]

21
Inspection reveals this solution scales asymptotically like \( O(w) \) and for small values, \( R_{\text{MUL}} \) is twice as costly as \( R_{\text{ADD}} \) in terms of number of constraints and auxiliary variables.

### B.2 Program flow via multi-linear Lagrange polynomials

A program \( P \) of length \( s \) is a sequence of instructions \( I_0, \ldots, I_{s-1} \), each instruction contains an opcode and a list of \( k \) inputs and \( \ell \) outputs, where \( k \) and \( \ell \) should match the number of inputs and outputs consumed and produced by the opcode, respectively. An input is either a constant (also known as immediate) or a register location and outputs are invariably register locations. (Instructions related to random access memory are dealt with separately, below; until then we assume our programs do not access it and use only the 16 registers.) Each instruction also points to the next instruction in the program; by default \( I_j \) points to \( I_{j+1} \) but certain instructions (jumps and conditional jumps) may point to a different instruction, and the pointer may further depend on the value of certain registers. The *program counter* (PC) is a special register that contains the number of the current instruction, and thus takes values in \( \{0, \ldots, s-1\} \).

A *machine state* is a pair \( S = (\text{PC}, R) \) where PC holds the value of the program counter and \( R \) contains the values of all registers. The program \( P \) induces a natural relation \( R_P \) that contains all pairs \( (S = (\text{PC}, R), S' = (\text{PC}', R')) \) of machine states such that a single cycle of the machine in state \( S \) (with program counter being \( \text{PC} \) and registers holding values \( R \)) results in state \( S' \). As done for opcodes in (1), our purpose in this subsection is to define a system of constraints, denoted \( A_P \), that defines \( R_P \) as its zero-locus, projected onto its first few variables. Formally, let \( \text{PC}, \text{PC}', R, R' \) denote variables ranging over \( \mathbb{F} \), and recall \( x, y, z \) denote variables for opcode inputs, outputs and auxiliary variables, respectively. Then

\[
R_P = \left\{ (\text{PC}, R, \text{PC}', R') \mid \exists x, y, z A_P (\text{PC}, R, \text{PC}', R', x, y, z) = 0 \right\} \tag{11}
\]

In words, \( A_P \) is a set of polynomials whose zero-locus, projected to \( \text{PC}, R, \text{PC}', R' \), equals the “program evolution” relation \( R_P \).

To minimize degree complexity, the program counter value is recorded via \( r = \lceil \log s \rceil \) many variables, denoted \( \text{PC}_1, \ldots, \text{PC}_r \), each ranging over \( \{0, 1\} \). For \( \alpha \in \{0, 1\}^r \) let

\[
L_\alpha(\text{PC}_1, \ldots, \text{PC}_r) = \prod_{i=1}^{r} (\text{PC}_i + \alpha_i + 1)
\]

be the Lagrange multi-linear polynomial that evaluates to 1 on \( \alpha \) and evaluates to 0 on \( \{0, 1\}^r \setminus \{\alpha\} \). We multiply the polynomials in the algebraic constraint system appearing in the \( i \)th instruction by \( L_\gamma(\text{PC}_1, \ldots, \text{PC}_r) \) where \( \gamma \in \{0, 1\}^r \) is the binary representation of \( i \). Informally, this has the effect of applying the set of constraints \( A_{\text{op}} \) only when the PC points to an instruction that contains \( \text{op} \). Formally, for each opcode \( \text{op} \) appearing in the program \( P \), let
\( I \) \( \in \) \( P \subseteq \{0, \ldots, s - 1\} \) be the set of program instructions in which \( \text{op} \) is executed. Then define

\[
\hat{A}^{\text{op}}_P = \left\{ P \cdot \sum_{i \in I} L^r_i (PC_1, \ldots, PC_r) \mid P \in A^{\text{op}} \right\} \quad (12)
\]

Inputs and outputs to an opcode are checked in a similar way. In particular, let \( i_{j,1}, \ldots, i_{j,k_i} \) denote the indices of the registers that are the inputs of the opcode in instruction \( i \) and let \( o_{i,1}, \ldots, o_{i,\ell_i} \) be the indices of output registers of that instruction, then we define

\[
\hat{A}^{i/\circ}_i = \left\{ (X_j - R_{i,j}) \cdot L^r_i (PC_1, \ldots, PC_r) \mid j = 1, \ldots, k_i \right\} \cup \left\{ (Y_j - R'_{i,j}) \cdot L^r_i (PC_1, \ldots, PC_r) \mid j = 1, \ldots, \ell_i \right\} \cup \left\{ (R_j - R'_{i,j}) \cdot L^r_i (PC_1, \ldots, PC_r) \mid j \text{ is not an output register of instruction } i \right\} \quad (13)
\]

In similar fashion, updating the program counter during the \( i \)th instruction is defined using a set of polynomials whose zero locus corresponds to the correct update of PC value. Typically, this modification simply increments the value of the PC by 1, and this can be done by multiplying each polynomial in (6)–10) by \( L^r_i (PC_1, \ldots, PC_r) \). Let \( \hat{A}^{\text{pc}}_i \) denote the corresponding set of polynomials. The final set \( A_P \) that defines the “program evolution” relation \( R_P \) is

\[
A_P \triangleq \left\{ \hat{A}^{\text{op}}_P \mid \text{op appears in } P \right\} \bigcup \left\{ \hat{A}^{i/\circ}_i \mid i = 0, \ldots, s - 1 \right\} \bigcup \left\{ \hat{A}^{\text{pc}}_i \mid i = 0, \ldots, s - 1 \right\} \quad (14)
\]

and the discussion above shows that its zero locus \( A_P \), projected to \( \text{PC}, \text{R}, \text{PC}', \text{R}' \), indeed equals \( R_P \).

**C Two programs computing subset-sum**

Code 1 shows a high-level description of the exhaustive subset-sum program, and Code 2 gives an equivalent TinyRAM hand-optimized implementation (cf. Appendix D for discussion of machine compiled assembly). In Code 1, the variable \( k \) is treated as a binary vector that iterates over all the possible combinations of the inputs. The inputs that correspond to each combination are summed up by inspecting whether the least significant bit (LSB) of \( k \) is 1, and then shifting \( k \) rightward. Code 2 uses the \texttt{AND.CMP.E.SHR} TinyRAM instructions for these inspections and shifts. It should be noted that the instruction set that is needed for Code 2 is uncanny, in particular the cost of the \texttt{DIV} instruction would have been about twice higher than \texttt{SHR} in terms of the number of field elements that the prover commits to in a time step.
The total number of time steps $T$ of the ACSP for Code 2 is sufficiently large if the inequality $2^n \cdot (9n + 7) < T$ holds, where $n$ is the size of the input array. With 16-bit TinyRAM architecture, $n \leq 16$ is also required, unless extra logic is added to Code 2. In this inequality, the term $9n$ can be inferred by amortizing the number of TinyRAM instructions that are executed when the LSB of $k$ is either 0 or 1. For example, $T = 2^{20}$ is sufficient for $n = 13$ inputs. For a further demonstration of the dependency between $T$ and $n$, see Figure 2.

The TinyRAM architecture relies on 16 or less registers, in particular Code 2 needs 5 registers in total. This helps with keeping the complexity low, as it implies that a relatively small number of field elements are required per time step. However, this also means that we do not have enough registers to store the entire input array. Since it is preferable to avoid the poly-logarithmic blowup of programs with memory, Code 2 employs a special “read-only memory” (ROM) instruction. The ROM instruction takes a single operand, treats it as an index $J \leq n$, and returns the corresponding array $[J]$ input value. The algebraic constraints of the ROM instruction consist of unpacking the bits of $J$ and using a selector polynomial to force the prover to use the predefined array $[J]$ field element. For example, with $n = 8$, the ROM instruction can be implemented as

$$\bigcup_{k=0}^{2} \{b_k(b_k+1)\} \bigcup \{J + \sum_{k=0}^{2} b_k x^k, \sum_{\alpha,\beta,\gamma \in \{0,1\}} (b_0 + \alpha)(b_1 + \beta)(b_2 + \gamma)(R + C_{\alpha,\beta,\gamma})\},$$

where $R$ is the returned operand and $C_{\alpha,\beta,\gamma}$ are the array input values that the ACSP instance specifies. Thus, the degree of the ROM constraints is bounded by $\lceil \log n \rceil + 1$, and overall the ROM instruction is far less complex than deploying the full read/write memory construction.

Code 3 is a subset-sum program that computes all the partial sums of half of the input numbers, as well as the other half, and then does a linear scan to look for two partial sums that add up to the target value [51]. The partial sums are first stored in memory in a sorted order, which can be done in $O(n)$ time due to the following observation: given a sorted list $S_1, S_2, \ldots, S_{2^k}$ of all the possible sums that can be produced from combinations of certain $k$ numbers, and another number $m$, the sorted list $S_1 + m, S_2 + m, \ldots, S_{2^k} + m$ can be merged into $S_1, S_2, \ldots, S_{2^k}$ to obtain one sorted list of size $2^{k+1}$, in linear time. Hence, Code 3 needs to store $O(\sqrt{2^n})$ elements in memory, where $n$ is the size of the input array.

Code 4 gives a hand-optimized TinyRAM implementation of this high-level pseudocode, in which the dependency between $n$ and the total number of time steps $T$ is $n \approx 2(T - 7)$. Section D discusses the machine compiled code for the same program. As can be seen in Figure 2, Code 4 can thus cope with greater values of $n$ than Code 2, even after the poly-logarithmic blowup in complexity that is due to memory handling is taken into account.

Notice that unlike the high-level description in Code 3, the Code 4 implementation that we benchmark actually outputs a bit-string of the correct combination, if one exists (Code 5 and Code 6 do this as well). This extra work is done for a
fair comparison with Code 2, that does this “for free”. However, since subset-sum
is an NP-complete problem, it makes sense to generate the PCP on unsatisfiable
instances. Thus, this extra work can be regarded as unnecessary in this context.

Code 1  Pseudocode of the exhaustive search subset-sum program

<table>
<thead>
<tr>
<th>line</th>
<th>operation</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>for k = 1 to 2^n - 1 do</td>
<td>k loops over all {0,1}^n \setminus {0^n} combinations</td>
</tr>
<tr>
<td>2:</td>
<td>curr ← k, idx ← 0, sum ← 0</td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>while curr ≠ 0 do</td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>if 1 = (curr bitwise-and 1) then</td>
<td>▶ LSB of curr is 1?</td>
</tr>
<tr>
<td>5:</td>
<td>sum ← sum + array[idx]</td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>curr ← curr/2, idx ← idx + 1</td>
<td></td>
</tr>
<tr>
<td>8:</td>
<td>end while</td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>if sum = target then</td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>return k</td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>end for</td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>return 0</td>
<td></td>
</tr>
</tbody>
</table>

Code 2  TinyRAM assembly code of the exhaustive search subset-sum program

<table>
<thead>
<tr>
<th>line</th>
<th>operation</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>MOV r0, 1</td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>CMP r0, 2^n</td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td>JMP Line#21</td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td>MOV r1, 0</td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td>MOV r2, r0</td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td>MOV r3, 0</td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td>AND r4, r2, 1</td>
<td></td>
</tr>
<tr>
<td>8:</td>
<td>CMP r4, 0</td>
<td></td>
</tr>
<tr>
<td>9:</td>
<td>JMP Line#12</td>
<td></td>
</tr>
<tr>
<td>10:</td>
<td>JMP Line#7</td>
<td></td>
</tr>
<tr>
<td>11:</td>
<td>ADD r1, r1, r4</td>
<td></td>
</tr>
<tr>
<td>12:</td>
<td>SHR r2, r2, 1</td>
<td></td>
</tr>
<tr>
<td>13:</td>
<td>CMP r2, 0</td>
<td></td>
</tr>
<tr>
<td>14:</td>
<td>CMP r2, 0</td>
<td></td>
</tr>
<tr>
<td>15:</td>
<td>ADD r3, r3, 1</td>
<td></td>
</tr>
<tr>
<td>16:</td>
<td>JMP Line#17</td>
<td></td>
</tr>
<tr>
<td>17:</td>
<td>CMP r1, target</td>
<td></td>
</tr>
<tr>
<td>18:</td>
<td>JMP Line#22</td>
<td></td>
</tr>
<tr>
<td>19:</td>
<td>ADD r0, r0, 1</td>
<td></td>
</tr>
<tr>
<td>20:</td>
<td>JMP Line#2</td>
<td></td>
</tr>
<tr>
<td>21:</td>
<td>MOV r0, 0</td>
<td></td>
</tr>
<tr>
<td>22:</td>
<td>ANSW r0</td>
<td></td>
</tr>
</tbody>
</table>

D  Compiling C code to TinyRAM

Our TinyRAM compiler is implemented as a GCC back end, with support for
some optimization techniques. Code 5 shows C source for the memory-based
subset-sum program, and the corresponding compiled code is given as Code 6. As
shown, Code 6 has 21 more instruction than the hand-written assembly of Code 4.
Likewise, the running time of Code 6 is somewhat greater than that of Code 4,
for example with \(n = 14\) it takes 13582 time steps until Code 6 terminates, while
Code 4 terminates in 11231 time steps.
Code 3 Pseudocode of the memory-based subset-sum program

input: \( n = 2^h, \text{array}[n], \text{target} \)
1: \( H_1 \leftarrow \{ \text{array}[0], \text{array}[1], \ldots, \text{array}[h-1] \} \)
2: \( H_2 \leftarrow \{ \text{array}[h], \text{array}[1], \ldots, \text{array}[n-1] \} \)
3: for \( m \in \{1, 2\} \) do
   ▷ sort each half
   4: let \( A_{m,0} \) be an array of size 1 with \( A_{m,0}[0] = 0 \)
   5: \( i \leftarrow 0 \)
   6: for \( x \in H_m \) do
      7: let \( B_{m,i} \) be an array of size \( i \) and \( C_{m,i} \) be an array of size \( 2i \)
      8: for \( k \in \{0, 1, 2, \ldots, 2^i - 1\} \) do
         9: \( B_{m,i}[k] \leftarrow A_{m,i}[k] + x \)
      10: end for
      11: \( C_{m,i} \leftarrow \text{merge}(A_{m,i}, B_{m,i}) \) ▷ note: \( A_{m,i} \) and \( B_{m,i} \) are already sorted
      12: \( A_{m,i+1} \leftarrow C_{m,i} \)
      13: \( i \leftarrow i + 1 \)
   14: end for
5: end for
16: \( i \leftarrow 0 \), \( k \leftarrow 2^h - 1 \)
17: while True do ▷ search for the target
   18: if \( \text{target} = A_{1,h}[i] + A_{2,h}[k] \) then return 1 end if
   19: if \( \text{target} > A_{1,h}[i] + A_{2,h}[k] \) then
      20: if \( i = 2^h - 1 \) then return 0 end if
      21: \( i \leftarrow i + 1 \)
   22: else
      23: if \( k = 0 \) then return 0 end if
      24: \( k \leftarrow k - 1 \)
   25: end if
26: end while
### Code 4  TinyRAM assembly code of the memory-based subset-sum program

**input:** \( n = 2^h \), \( \text{array}[n] \), \( \text{target} \), \( \ell = 2^{h+1} - 2 \)

**constants:** \( \text{INPADDR} = 2^{16} - 2^6 \), \( \text{ADDR1} = 0 \), \( \text{ADDR2} = 2^{14} \), \( \text{OFFSET} = 2^{15} \)

**preprocess:** store \( \text{array}[n] \) at \( \text{INPADDR} \)

```assembly
1: MOV r0, INPADDR
2: MOV r1, ADDR1
3: MOV r9, 0
4: STOR r9, r1
5: ADD r2, r1, OFFSET
6: STOR r9, r2
7: MOV r2, r1
8: ADD r4, r1, 1
9: MOV r5, r4
10: MOV r8, 1
11: ADD r9, h
12: LOAD r3, r0
13: JMP Line#44
14: ADD r0, r0, 1
15: CMPE r9, r0
16: CJMP Line#60
17: LOAD r3, r0
18: SHL r8, r8, 1
19: MOV r5, r4
20: JMP Line#44
21: ADD r7, r4, OFFSET
22: STOR r6, r7
23: ADD r4, r4, 1
24: CMPE r5, r1
25: CJMP Line#36
26: CMPE r5, r2
27: CJMP Line#14
28: LOAD r6, r2
29: ADD r6, r6, r3
30: STOR r6, r4
31: ADD r6, r2, OFFSET
32: LOAD r6, r6
33: XOR r6, r6, r8
34: ADD r2, r2, 1
35: JMP Line#21
36: STOR r5, r2
37: CMPE r4, r2
38: LOAD r6, r2
39: STOR r6, r4
40: ADD r4, r4, 1
41: CMPE r1, ADDR2
42: ADD r1, ADDR2 + \ell
43: JMP Line#21
44: LOAD r6, r1
45: LOAD r7, r2
46: LOAD r6, r1
47: CMPE r6, r7
48: CJMP Line#54
49: STOR r6, r4
50: ADD r6, r1, OFFSET
51: LOAD r6, r6
52: ADD r1, r1, 1
53: JMP Line#21
54: STOR r7, r4
55: ADD r6, r2, OFFSET
56: LOAD r6, r6
57: XOR r6, r6, r8
58: ADD r2, r2, 1
59: JMP Line#21
60: CMPA r1, ADDR2
```

---

Code 5  C source of the memory-based subset-sum program

```c
#define N 7
#define TARGET 123
int input[2*N] = {10,20,30,40,50,60,70,-10,-20,-30,-40,-50,-60,70};
int arr[ 4 * ( (1 << (N+1)) - 1 ) ];

int main(void) {
    register int *inp = &input[0], *last_inp, *p1, *p2, *next, *next_backup, b;
    p1 = p2 = &arr[0]; // phase1: prepare arrays
    for(;;) { // prepare each half array
        next = next_backup = (p1+2);
        *p1 = *(p1+1) = 0; b = 1; last_inp = inp + N;
        for(;;) { // iterate over each input
            for(;;) { // merge
                if(p1 == next_backup) {
                    while(p2 < next_backup) {
                        *(next++) = *(p2++) + *inp;
                        *(next++) = *(p2++) ^ b;
                    }
                    break;
                }
                if(p2 == next_backup) {
                    while(p1 < next_backup) {
                        *(next++) = *(p1++);
                        *(next++) = *(p1++) ^ b;
                    }
                    break;
                }
                if(p1 > p2) {
                    *(next++) = *(p2++) + *inp;
                    *(next++) = *(p2++) ^ b;
                } else {
                    *(next++) = *(p1--);
                    *(next++) = *(p1--);
                }
            }
            if(++inp == last_inp) break;
            b = b << 1;
            next_backup = next;
        }
        if( p1 > &arr[0] + (1 << (N+2)) ) break;
p1 = p2 = next;
    }
p1 = &arr[ 2*((1 << (N+1)) - 1) - 2 ]; // phase2: search
    for(;;) { //
        if(TARGET == *p1 + *p2)
            return *(p1+1) ^ (*(p2+1) << N);
        if(TARGET > *p1 + *p2) {
            if(p2 == &arr[0]) break;
            p2 = p2 + 2;
        } else {
            if(p1 == &arr[0]) break;
p1 = p1 - 2;
        }
    }
    return 0;
}
```

28
input: $n = 2h$, $\text{array}[n]$, target
preprocess: store $\text{array}[n]$ at address 0

\begin{verbatim}
1: MOV r9, 0  38: LOAD r2, r8  75: SHL r14, r14, 1
2: MOV r12, 28 39: ADD r8, r8, 2 76: MOV r13, r4
3: MOV r8, r12 40: STOR r2, r4 77: JMP Line#12
4: ADD r13, r8, r4 41: ADD r4, r4, 2 78: CMPA r8, 1052
5: MOV r4, r13 42: CMPEAE r8, r13 79: CJMP Line#83
6: MOV r2, 0 43: CJMP Line#34 80: MOV r12, r4
7: ADD r0, r8, 2 44: JMP Line#72 81: MOV r8, r4
8: STOR r2, r0 45: LOAD r2, r12 82: JMP Line#4
9: STOR r2, r8 46: LOAD r3, r9 83: MOV r4, 1044
10: MOV r14, 1 47: ADD r3, r2, r3 84: LOAD r3, r4
11: ADD r5, r9, 14 48: LOAD r2, r8 85: LOAD r2, r12
12: CMPE r8, r13 49: CMPG r2, r3 86: ADD r2, r3, r2
13: CJMP Line#30 50: CJMP Line#63 87: CMPE r2, target
14: CMPEAE r12, r13 51: LOAD r5, r12 88: CJMP Line#96
15: CJMP Line#72 52: LOAD r2, r9 89: ADD r0, r12, 2
16: LOAD r3 r12 53: ADD r2, r3, r2 90: LOAD r12, r0
17: LOAD r2, r9 54: ADD r12, r12, 2 91: SHL r12, r12, h
18: ADD r2, r3, r2 55: STOR r2, r4 92: ADD r0, r4, 2
19: ADD r12, r12, 2 56: ADD r4, r4, 2 93: LOAD r4, r0
20: STOR r2, r4 57: LOAD r2, r12 94: XOR r2, r12, r4
21: ADD r4, r4, 2 58: XOR r2, r14, r2 95: JMP Line#110
22: LOAD r2, r12 59: ADD r12, r12, 2 96: LOAD r3, r4
23: XOR r2, r14, r2 60: STOR r2, r4 97: LOAD r2, r12
24: ADD r12, r12, 2 61: ADD r4, r4, 2 98: ADD r2, r3, r2
25: STOR r2, r4 62: JMP Line#12 99: CMPG r2, target-1
26: ADD r4, r4, 2 63: LOAD r2, r8 100: CJMP Line#106
27: CMPEAE r12, r13 64: ADD r8, r8, 2 101: MOV r2, 2064
28: CJMP Line#16 65: STOR r2, r4 102: CMPE r12, r2
29: JMP Line#72 66: ADD r4, r4, 2 103: CJMP Line#109
30: CMPE r12, r13 67: LOAD r2, r8 104: ADD r12, r12, 4
31: CJMP Line#45 68: ADD r8, r8, 2 105: JMP Line#84
32: CMPEAE r8, r13 69: STOR r2, r4 106: CMPE r4, 28
33: CJMP Line#72 70: ADD r4, r4, 2 107: CJMP Line#109
34: LOAD r2, r8 71: JMP Line#12 108: JMP Line#84
35: ADD r8, r8, 2 72: ADD r9, r9, 2 109: MOV r2, 0
36: STOR r2, r4 73: CMPE r9, r5 110: ANSV r2
37: ADD r4, r4, 2 74: CJMP Line#78
\end{verbatim}
Abstract. We study the problem of constructing proof systems that achieve both soundness and zero knowledge unconditionally (without relying on intractability assumptions). Known techniques for this goal are primarily combinatorial, despite the fact that constructions of interactive proofs (IPs) and probabilistically checkable proofs (PCPs) heavily rely on algebraic techniques to achieve their properties.

We present simple and natural modifications of well-known ‘algebraic’ IP and PCP protocols that achieve unconditional (perfect) zero knowledge in recently introduced models, overcoming limitations of known techniques.

- We modify the PCP of Ben-Sasson and Sudan [BS08] to obtain zero knowledge for $\text{NEXP}$ in the model of Interactive Oracle Proofs [BCS16, RRR16], where the verifier, in each round, receives a PCP from the prover.

- We modify the IP of Lund, Fortnow, Karloff, and Nisan [LFKN92] to obtain zero knowledge for $\#\text{P}$ in the model of Interactive PCPs [KR08], where the verifier first receives a PCP from the prover and then interacts with him.

The simulators in our zero knowledge protocols rely on solving a problem that lies at the intersection of coding theory, linear algebra, and computational complexity, which we call the succinct constraint detection problem, and consists of detecting dual constraints with polynomial support size for codes of exponential block length. Our two results rely on solutions to this problem for fundamental classes of linear codes:

† Work conducted while at Stanford.
‡ Work conducted while at Technion.
§ Work conducted while at the University of Toronto.
An algorithm to detect constraints for Reed–Muller codes of exponential length. This algorithm exploits the Raz–Shpilka [RS05] deterministic polynomial identity testing algorithm, and shows, to our knowledge, a first connection of algebraic complexity theory with zero knowledge.

An algorithm to detect constraints for PCPs of Proximity of Reed–Solomon codes [BS08] of exponential degree. This algorithm exploits the recursive structure of the PCPs of Proximity to show that small-support constraints are “locally” spanned by a small number of small-support constraints.

Keywords: probabilistically checkable proofs, interactive proofs, sumcheck, zero knowledge, polynomial identity testing

1 Introduction

The study of interactive proofs (IPs) [BM88, GMR89] that unconditionally achieve zero knowledge [GMR89] has led to a rich theory, with connections well beyond zero knowledge. For example, the class of languages with statistical zero knowledge IPs, which we denote by \( \text{SZK-IP} \), has complete problems that make no reference to either zero knowledge or interaction [SV03, GV99] and is closed under complement [Oka00, Vad99]. Despite the fact that all \( \text{PSPACE} \) languages have IPs [Sha92], \( \text{SZK-IP} \) is contained in \( \text{AM} \cap \text{coAM} \), and thus \( \text{NP} \) is not in \( \text{SZK-IP} \) unless the polynomial hierarchy collapses [BHZ87]; one consequence is that Graph Non-Isomorphism is unlikely to be NP-complete. Moreover, constructing \( \text{SZK-IP} \) for a language is equivalent to constructing instance-dependent commitments for the language [IOS97, OV08], and has connections to other fundamental information-theoretic notions like randomized encodings [AR16, VV15] and secret-sharing schemes [VV15].

Unconditional zero knowledge in other models behaves very differently. Ben-Or, Goldwasser, Kilian, and Wigderson [BGKW88] introduced the model of multi-prover interactive proofs (MIPs) and showed that all such proofs can be made zero knowledge unconditionally. The analogous statement for IPs is equivalent to the existence of one-way functions, as shown by [GMR89, IY87, BGG+88] in one direction and by [Ost91, OW93] in the other (unless \( \text{BPP} = \text{PSPACE} \), in which case the statement is trivial). Subsequent works not only established that all \( \text{NEXP} \) languages have MIPs [BFL91], but also led to formulating probabilistically checkable proofs (PCPs) and proving the celebrated PCP Theorem [FRS88, BFLS91, FGL+96, AS98, ALM+98], as well as constructing statistical zero knowledge PCPs [KPT97] and applying them to black-box cryptography [IMS12, IMSX15].

The theory of zero knowledge for these types of proofs, however, is not as rich as in the case of IPs. Most notably, known techniques to achieve zero knowledge MIPs or PCPs are limited, and come with caveats. Zero knowledge MIPs are obtained via complex generic transformations [BGKW88], assume the full power of the PCP Theorem [DFK+92], or support only languages in \( \text{NP} \) [LS95]. Zero knowledge PCPs are obtained via a construction that incurs polynomial
blowups in proof length and requires the honest verifier to adaptively query the PCP [KPT97]. Alternative approaches are not known, despite attempts to find them. For example, [IWy16] apply PCPs to leakage-resilient circuits, obtaining PCPs for \( \text{NP} \) that do have a non-adaptive honest verifier but are only witness indistinguishable.

Even basic questions such as “are there zero-knowledge PCPs of quasilinear-size?” or “are there zero-knowledge PCPs with non-adaptive honest verifiers?” have remained frustratingly hard to answer, despite the fact the answers to these questions are well understood when removing the requirement of zero knowledge. This state of affairs begs the question of whether a richer theory about zero knowledge MIPs and PCPs could be established.

The current situation is that known techniques to achieve zero knowledge MIPs and PCPs are combinatorial, namely they make black-box use of an underlying MIP or PCP, despite the fact that most MIP and PCP constructions have a rich algebraic structure arising from the use of error correcting codes based on evaluations of low-degree polynomials. This separation is certainly an attractive feature, and perhaps even unsurprising: while error-correcting codes are designed to help recover information, zero knowledge proofs are designed to hide it.

Yet, a recent work by Ben-Sasson, Chiesa, Gabizon, and Virza [BCGV16] brings together linear error correcting codes and zero knowledge using an algebraic technique that we refer to as ‘masking’. The paper introduces a “2-round PCP” for \( \text{NP} \) that unconditionally achieves zero knowledge and, nevertheless, has both quasilinear size and a non-adaptive honest verifier. Their work can be viewed not only as partial progress towards some of the open questions above, but also as studying the power of zero knowledge for a natural extension of PCPs (“multi-round PCPs” as discussed below) with its own motivations and applications [BCS16, RRR16, BCG+17].

The motivation of this work is to understand the power of algebraic tools, such as linear error correcting codes, for achieving zero knowledge unconditionally (without relying on intractability assumptions).

1.1 Results

We present new protocols that unconditionally achieve soundness and zero knowledge in recently suggested models that combine features of PCPs and IPs [KR08, BCS16, RRR16]. Our protocols consist of simple and natural modifications to well-known constructions: the PCP of Ben-Sasson and Sudan [BS08] and the IP for polynomial summation of Lund, Fortnow, Karloff, and Nisan [LFKN92]. By leveraging the linear codes used in these constructions, we reduce the problem of achieving zero knowledge to solving exponentially-large instances of a new linear-algebraic problem that we call constraint detection, which we believe to be of independent interest. We design efficient algorithms for solving this problem for notable linear code families, along the way exploiting connections to algebraic complexity theory and local views of linear codes. We now elaborate on the above by discussing each of our results.
Zero knowledge for non-deterministic exponential time

Two recent works [BCS16, RRR16] independently introduce and study the notion of an interactive oracle proof (IOP), which can be viewed as a “multi-round PCP”. Informally, an IOP is an IP modified so that, whenever the prover sends to the verifier a message, the verifier does not have to read the message in full but may probabilistically query it. Namely, in every round, the verifier sends the prover a message, and the prover replies with a PCP. IOPs enjoy better efficiency compared to PCPs [BCG+17], and have applications to constructing argument systems [BCS16] and IPs [RRR16].

The aforementioned work of [BCGV16] makes a simple modification to the PCP of Ben-Sasson and Sudan [BS08] and obtains a 2-round IOP for NP that is perfect zero knowledge, and yet has quasilinear size and a non-adaptive honest verifier. Our first result consists of extending this prior work to all languages in NEXP, positively answering an open question raised there. We do so by constructing, for each time T and query bound b, a suitable IOP for NTIME(T) that is zero knowledge against query bound b; the result for NEXP follows by setting b to be super-polynomial.

The foregoing notion of zero knowledge for IOPs directly extends that for PCPs, and requires showing the existence of an algorithm that simulates the view of any (malicious and adaptive) verifier interacting with the honest prover and making at most b queries across all oracles; here, ‘view’ consists of the answers to queries across all oracles.\footnote{More precisely, while in a zero knowledge IP or MIP one is required to simulate the entire transcript of interaction (with one or multiple provers), in a zero knowledge IOP or PCP one is merely required to simulate answers to the oracle queries but not the entire oracle.}

Theorem 1 (informal). For every time bound T and query bound b, the complexity class NTIME(T) has 2-round Interactive Oracle Proofs that are perfect zero knowledge against b queries, and where the proof length is \(\tilde{O}(T+b)\) and the (honest verifier’s) query complexity is polylog(T + b).

The prior work of [BCGV16] was “stuck” at NP because their simulator runs in poly(T + b) time so that T, b must be polynomially-bounded. In contrast, we achieve all of NEXP by constructing, for essentially the same simple 2-round IOP, a simulator that runs in time poly(\(\tilde{q} + \log T + \log b\)), where \(\tilde{q}\) is the actual number of queries made by the malicious verifier. This is an exponential improvement in simulation efficiency, and we obtain it by conceptualizing and solving a linear-algebraic problem about Reed–Solomon codes, and their proximity proofs, as discussed in Section 1.1.

In sum, our theorem gives new tradeoffs compared to [KPT97]’s result, which gives statistical zero knowledge PCPs for NTIME(T) with proof length poly(T, b) and an adaptive honest verifier. We obtain perfect zero knowledge for NTIME(T), with quasilinear proof length and a non-adaptive honest verifier, at the price of “2 rounds of PCPs”.\footnote{More precisely, while in a zero knowledge IP or MIP one is required to simulate the entire transcript of interaction (with one or multiple provers), in a zero knowledge IOP or PCP one is merely required to simulate answers to the oracle queries but not the entire oracle.}
Zero knowledge for counting problems Kalai and Raz [KR08] introduce and study the notion of interactive PCPs (IPCPs), which “sits in between” IPs and IOPs: the prover first sends the verifier a PCP, and then the prover and verifier engage in a standard IP. IPCPs also enjoy better efficiency compared to PCPs or IPs alone [KR08].

We show how a natural and simple modification of the sumcheck protocol of Lund, Fortnow, Karloff, and Nisan [LFKN92] achieves perfect zero knowledge in the IPCP model, even with a non-adaptive honest verifier. By running this protocol on the usual arithmetization of the counting problem associated to 3SAT, we obtain our second result, which is IPCPs for \( \#P \) that are perfect zero knowledge against unbounded queries. This means that there exists a polynomial-time algorithm that simulates the view of any (malicious and adaptive) verifier making any polynomial number of queries to the PCP oracle. Here, ‘view’ consists of answers to oracle queries and the transcript of interaction with the prover. (In particular, this notion of zero knowledge is a ‘hybrid’ of corresponding notions for PCPs and IPs.)

**Theorem 2 (informal).** The complexity class \( \#P \) has Interactive PCPs that are perfect zero knowledge against unbounded queries. The PCP proof length is exponential, and the communication complexity of the interaction and the (honest verifier’s) query complexity are polynomial.

Our construction relies on a random self-reducibility property of the sumcheck protocol (see Section 2.2 for a summary) and its completeness and soundness properties are straightforward to establish. As in our previous result, the “magic” lies in the construction of the simulator, which must solve the same type of exponentially-large linear-algebraic problem, except that this time it is about Reed–Muller codes rather than Reed–Solomon codes. The algorithm that we give to solve this task relies on connections to the problem of polynomial identity testing in the area of algebraic complexity theory, as we discuss further below.

Goyal, Ishai, Mahmoody, and Sahai [GIMS10] also study zero knowledge for IPCPs, and show how to obtain IPCPs for \( \text{NP} \) that (i) are statistical zero knowledge against unbounded queries, and yet (ii) each location of the (necessarily) super-polynomial size PCP is polynomial-time computable given the NP witness. They further prove that these two properties are not attainable by zero knowledge PCPs. Their construction consists of replacing the commitment scheme in the zero knowledge IP for 3-colorability of [GMW91] with an information-theoretic analogue in the IPCP model. Our Theorem 2 also achieves zero knowledge against unbounded queries, but targets the complexity class \( \#P \) (rather than \( \text{NP} \)), for which there is no clear analogue of property (ii) above.

Information-theoretic commitments also underlie the construction of zero knowledge PCPs [KPT97]. One could apply the [KPT97] result for \( \text{NEXP} \) to obtain zero knowledge PCPs (thus also IPCPs) for \( \#P \), but this is an indirect and complex route (in particular, it relies on the PCP Theorem) that, moreover, yields an adaptive honest verifier. Our direct construction is simple and natural, and also yields a non-adaptive honest verifier.
We now discuss the common algebraic structure that allowed us to obtain both of the above results. We believe that further progress in understanding these types of algebraic techniques will lead to further progress in understanding the power of unconditional zero knowledge for IOPs and IPCPs, and perhaps also for MIPs and PCPs.

Succinct constraint detection for Reed–Muller and Reed–Solomon codes The constructions underlying both of our theorems achieve zero knowledge by applying a simple modification to well-known protocols: the PCP of Ben-Sasson and Sudan [BS08] underlies our result for $\text{NEXP}$ and the sumcheck protocol of Lund, Fortnow, Karloff, and Nisan [LFKN92] underlies our result for $\#\text{P}$.

In both of these protocols the verifier has access (either via a polynomial-size representation or via a PCP oracle) to an exponentially-large word that allegedly belongs to a certain linear code, and the prover ‘leaks’ hard-to-compute information in the process of convincing the verifier that this word belongs to the linear code. We achieve zero knowledge via a modification that we call masking: the prover sends to the verifier a PCP containing a random codeword in this code, and then convinces the verifier that the sum of these two (the original codeword and this random codeword) is close to the linear code.\footnote{This is reminiscent of the use of a random secret share of 0 to achieve privacy in information-theoretic multi-party protocols [BGW88].}

Intuitively, zero knowledge comes from the fact that the prover now argues about a random shift of the original word.

However, this idea raises a problem: how does the simulator ‘sample’ an exponentially-large random codeword in order to answer the verifier’s queries to the PCP? Solving this problem crucially relies on solving a problem that lies at the intersection of coding theory, linear algebra, and computational complexity, which we call the constraint detection problem. We informally introduce it and state our results about it, and defer to Section 2.2 a more detailed discussion of its connection to zero knowledge.

Detecting constraints in codes. Constraint detection is the problem of determining which linear relations hold across all codewords of a linear code $C \subseteq \mathbb{F}^D$, when considering only a given subdomain $I \subseteq D$ of the code rather than all of the domain $D$. This problem can always be solved in time that is polynomial in $|D|$ (via Gaussian elimination); however, if there is an algorithm that solves this problem in time that is polynomial in the subdomain’s size $|I|$, rather than the domain’s size $|D|$, then we say that the code has succinct constraint detection; in particular, the domain could have exponential size and the algorithm would still run in polynomial time.

Definition 1 (informal). We say that a linear code $C \subseteq \mathbb{F}^D$ has succinct constraint detection if there exists an algorithm that, given a subset $I \subseteq D$, runs in time $\text{poly}(\log |\mathbb{F}| + \log |D| + |I|)$ and outputs $z \in \mathbb{F}^I$ such that $\sum_{i \in I} z(i)w(i) = 0$ for all $w \in C$, or “no” if no such $z$ exists. (In particular, $|D|$ may be exponential.)
We further discuss the problem of constraint detection in Section 2.1, and provide a formal treatment of it in Section 4.1. Beyond this introduction, we shall use (and achieve) a stronger definition of constraint detection: the algorithm is required to output a basis for the space of dual codewords in \( C^\perp \) whose support lies in the subdomain \( I \), i.e., a basis for the space \( \{ z \in D^I : \forall w \in C, \sum_{i \in I} z(i)w(i) = 0 \} \). Note that in our discussion of succinct constraint detection we do not leverage the distance property of the code \( C \), but we do leverage it in our eventual applications.

Our zero knowledge simulators’ strategy includes sampling a “random PCP”: a random codeword \( w \) in a linear code \( C \) with exponentially large domain size \( |D| \) (see Section 2.2 for more on this). Explicitly sampling \( w \) requires time \( \Omega(|D|) \), and so is inefficient. But a verifier makes only polynomially-many queries to \( w \), so the simulator has to only simulate \( w \) when restricted to polynomial-size sets \( I \subseteq D \), leaving open the possibility of doing so in time \( \text{poly}(|I|) \). Achieving such a simulation time is an instance of (efficiently and perfectly) “implementing a huge random object” [GGN10] via a stateful algorithm [BW04]. We observe that if \( C \) has succinct constraint detection then this sampling problem for \( C \) has a solution: the simulator maintains the set \( \{(i, a_i)\}_{i \in I} \) of past query-answer pairs; then, on a new verifier query \( j \in D \), the simulator uses constraint detection to determine if \( w_j \) is linearly dependent on \( w_I \), and answers accordingly (such linear dependencies characterize the required probability distribution, see Lemma 1).

Overall, our paper thus provides an application (namely, obtaining zero knowledge simulators) where the problem of efficient implementation of huge random objects arises naturally.

We now state our results about succinct constraint detection.

(1) Reed–Muller codes, and their partial sums. We prove that the family of linear codes comprised of evaluations of low-degree multivariate polynomials, along with their partial sums, has succinct constraint detection. This family is closely related to the sumcheck protocol [LFKN92], and indeed we use this result to obtain a PZK analogue of the sumcheck protocol (see Section 2.2), which yields Theorem 2 (see Section 2.3).

Recall that the family of Reed–Muller codes, denoted \( \text{RM} \), is indexed by tuples \( n = (F, m, d) \), where \( F \) is a finite field and \( m, d \) are positive integers, and the \( n \)-th code consists of codewords \( w : F_m \to F \) that are the evaluation of an \( m \)-variate polynomial \( Q \) of individual degree less than \( d \) over \( F \). We denote by \( \Sigma \text{RM} \) the family that extends \( \text{RM} \) with evaluations of all partial sums over certain subcubes of a hypercube:

Definition 2 (informal). We denote by \( \Sigma \text{RM} \) the linear code family that is indexed by tuples \( n = (F, m, d, H) \), where \( H \) is a subset of \( F \), and where the \( n \)-th code consists of codewords \( (w_0, \ldots, w_m) \) such that there exists an \( m \)-variate polynomial \( Q \) of individual degree less than \( d \) over \( F \) for which \( w_i : F^{m-i} \to F \) is the evaluation of the \( i \)-th partial sum of \( Q \) over \( H \), i.e, \( w_i(\alpha) = \sum_{\gamma \in H} Q(\alpha, \gamma) \) for every \( \alpha \in F^{m-i} \).
The domain size for codes in \( \Sigma_{RM} \) is \( \Omega(|F|^m) \), but our detector’s running time is exponentially smaller.

**Theorem 3** (informal statement of Thm. 5). The family \( \Sigma_{RM} \) has succinct constraint detection: there is a detector algorithm for \( \Sigma_{RM} \) that runs in time \( \text{poly}(|F| + m + d + |H| + |I|) \).

We provide intuition for the theorem’s proof in Section 2.1 and provide the proof’s details in Section 4.2; the proof leverages tools from algebraic complexity theory. (Our proof also shows that the family \( RM \), which is a restriction of \( \Sigma_{RM} \), has succinct constraint detection.) Our theorem implies perfect and stateful implementation of a random low-degree multivariate polynomial and its partial sums over any hypercube; our proof extends an algorithm of [BW04], which solves this problem in the case of parity queries to boolean functions on subcubes of the boolean hypercube.

(2) Reed–Solomon codes, and their PCPPs. Second, we prove that the family of linear codes comprised of evaluations of low-degree univariate polynomials concatenated with corresponding BS proximity proofs [BS08] has succinct constraint detection. This family is closely related to quasilinear-size PCPs for \( \text{NEXP} \) [BS08], and indeed we use this result to obtain PZK proximity proofs for this family (see Section 2.2), from which we derive Theorem 1 (see Section 2.3).

**Definition 3** (informal). We denote by \( \text{BS-RS} \) the linear code family indexed by tuples \( \mathbf{r} = (F, L, d) \), where \( F \) is an extension field of \( \mathbb{F}_2 \), \( L \) is a linear subspace in \( F \), and \( d \) is a positive integer; the \( \mathbf{r} \)-th code consists of evaluations on \( L \) of univariate polynomials \( Q \) of degree less than \( d \), concatenated with corresponding [BS08] proximity proofs.

The domain size for codes in \( \text{BS-RS} \) is \( \Omega(|L|) \), but our detector’s running time is exponentially smaller.

**Theorem 4** (informal statement of Thm. 6). The family \( \text{BS-RS} \) has succinct constraint detection:

there is a detector algorithm for \( \text{BS-RS} \) that runs in time \( \text{poly}(|F| + \text{dim}(L) + |I|) \).

We provide intuition for the theorem’s proof in Section 2.1 and provide the proof’s details in Section 4.3; the proof leverages combinatorial properties of the recursive construction of BS proximity proofs.

2 Techniques

We informally discuss intuition behind our algorithms for detecting constraints (Section 2.1), their connection to zero knowledge (Section 2.2), and how we derive our results about \( \#P \) and \( \text{NEXP} \) (Section 2.3). Throughout, we provide pointers to the technical sections that contain further details.
2.1 Detecting constraints for exponentially-large codes

As informally introduced in Section 1.1, the constraint detection problem corresponding to a linear code family \( \mathcal{C} = \{ C_n \} \) with domain \( D(\cdot) \) and alphabet \( \mathbb{F}(\cdot) \) is the following: given an index \( n \in \{0, 1\}^* \) and subset \( I \subseteq D(n) \), output a basis for the space \( \{ z \in D(n)^I : \forall w \in C_n, \sum_{i \in I} z(i) w(i) = 0 \} \). In other words, for a given subdomain \( I \), we wish to determine all linear relations that hold for codewords in \( C_n \) restricted to the subdomain \( I \).

If a generating matrix for \( C_n \) can be found in polynomial time, this problem can be solved in \( \text{poly}(|n| + |D(n)|) \) time via Gaussian elimination (such an approach was implicitly taken by \([BCGV16]\) to construct a perfect zero knowledge simulator for an IOP for \( \text{NP} \)). However, in our setting \( |D(n)| \) is exponential in \( |n| \), so the straightforward solution is inefficient. With this in mind, we say that \( \mathcal{C} \) has succinct constraint detection if there exists an algorithm that solves its constraint detection problem in \( \text{poly}(|n| + |I|) \) time, even if \( |D(n)| \) is exponential in \( |n| \).

The formal definition of succinct constraint detection is in Section 4.1. In the rest of this section we provide intuition for two of our theorems: succinct constraint detection for the family \( \Sigma \text{RM} \) and for the family \( \text{BS-RS} \). As will become evident, the techniques that we use to prove the two theorems differ significantly. Perhaps this is because the two codes are quite different: \( \Sigma \text{RM} \) has a simple and well-understood algebraic structure, whereas \( \text{BS-RS} \) is constructed recursively using proof composition.

From algebraic complexity to detecting constraints for Reed–Muller codes and their partial sums The purpose of this section is to provide intuition about the proof of Theorem 3, which states that the family \( \Sigma \text{RM} \) has succinct constraint detection. (Formal definitions, statements, and proofs are in Section 4.2.) We thus outline how to construct an algorithm that detects constraints for the family of linear codes comprised of evaluations of low-degree multivariate polynomials, along with their partial sums. Our construction generalizes the proof of \([BW04]\), which solves the special case of parity queries to boolean functions on subcubes of the boolean hypercube by reducing this problem to a probabilistic identity testing problem that is solvable via an algorithm of \([RS05]\).

Below, we temporarily ignore the partial sums, and focus on constructing an algorithm that detects constraints for the family of Reed–Muller codes \( \text{RM} \), and at the end of the section we indicate how we can also handle partial sums.

Step 1: phrase as linear algebra problem. Consider a codeword \( w : \mathbb{F}^m \rightarrow \mathbb{F} \) that is the evaluation of an \( m \)-variate polynomial \( Q \) of individual degree less than \( d \) over \( \mathbb{F} \). Note that, for every \( \alpha \in \mathbb{F}^m \), \( w(\alpha) \) equals the inner product of \( Q \)'s coefficients with the vector \( \phi_\alpha \) that consists of the evaluation of all \( d^m \) monomials at \( \alpha \). One can argue that constraint detection for \( \text{RM} \) is equivalent to finding the nullspace of \( \{ \phi_\alpha \}_{\alpha \in I} \). However, “writing out” this \( |I| \times d^m \) matrix and performing Gaussian elimination is too expensive, so we must solve this linear algebra problem succinctly.
Step 2: encode vectors as coefficients of polynomials. While each vector $\phi_\alpha$ is long, it has a succinct description; in fact, we can construct an $m$-variate polynomial $\Phi_\alpha$ whose coefficients (after expansion) are the entries of $\phi_\alpha$, but has an arithmetic circuit of only size $O(md)$: namely, $\Phi_\alpha(X) := \prod_{i=1}^m (1 + \alpha_i X_i + \alpha_i^2 X_i^2 + \cdots + \alpha_i^{d-1} X_i^{d-1})$. Computing the nullspace of $\{\Phi_\alpha\}_{\alpha \in I}$ is thus equivalent to computing the nullspace of $\{\phi_\alpha\}_{\alpha \in I}$.

Step 3: computing the nullspace. Computing the nullspace of a set of polynomials is a problem in algebraic complexity theory, and is essentially equivalent to the Polynomial Identity Testing (PIT) problem, and so we leverage tools from that area.\(^9\) While there are simple randomized algorithms to solve this problem (see for example [Kay10, Lemma 8] and [BW04]), these algorithms, due to a nonzero probability of error, suffice to achieve statistical zero knowledge but do not suffice to achieve perfect zero knowledge. To obtain perfect zero knowledge, we need a solution that has no probability of error. Derandomizing PIT for arbitrary algebraic circuits seems to be beyond current techniques (as it implies circuit lower bounds [KI04]), but derandomizations are currently known for some restricted circuit classes. The polynomials that we consider are special: they fall in the well-studied class of “sum of products of univariates”, and for this case we can invoke the deterministic algorithm of [RS05] (see also [Kay10]). (It is interesting that derandomization techniques are ultimately used to obtain a qualitative improvement for an inherently probabilistic task, i.e., perfect sampling of verifier views.)

The above provides an outline for how to detect constraints for $\Sigma \Gamma \Sigma$. The extension to $\Sigma \Gamma \Gamma$, which also includes partial sums, is achieved by considering a more general form of vectors $\phi_\alpha$ as well as corresponding polynomials $\Phi_\alpha$. These polynomials also have the special form required for our derandomization. See Section 4.2 for details.

From recursive code covers to detecting constraints for Reed–Solomon codes and their PCPPs The purpose of this section is to provide intuition about the proof of Theorem 4, which states that the family BS-RS has succinct constraint detection. (Formal definitions, statements, and proofs are in Section 4.3.) We thus outline how to construct an algorithm that detects constraints for the family of linear codes comprised of evaluations of low-degree univariate polynomials concatenated with corresponding BS proximity proofs [BS08].

Our construction leverages the recursive structure of BS proximity proofs: we identify key combinatorial properties of the recursion that enable “local” constraint detection. To define and argue these properties, we introduce two notions that play a central role throughout the proof:

\(^9\) PIT is the following problem: given a polynomial $f$ expressed as an algebraic circuit, is $f$ identically zero? This problem has well-known randomized algorithms [Zip79, Sch80], but deterministic algorithms for all circuits seem to be beyond current techniques [KI04]. PIT is a central problem in algebraic complexity theory, and suffices for solving a number of other algebraic problems. We refer the reader to [SY10] for a survey.
A (local) view of a linear code $C \subseteq \mathbb{F}^D$ is a pair $(\tilde{D}, \tilde{C})$ such that $\tilde{D} \subseteq D$ and $\tilde{C} = C|_D \subseteq \mathbb{F}^{\tilde{D}}$.

A cover of $C$ is a set of local views $S = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ of $C$ such that $D = \cup_j \tilde{D}_j$.

Combinatorial properties of the recursive step. Given a finite field $\mathbb{F}$, domain $D \subseteq \mathbb{F}$, and degree $d$, let $C := \text{RS}[\mathbb{F}, D, d]$ be the Reed–Solomon code consisting of evaluations on $D$ of univariate polynomials of degree less than $d$ over $\mathbb{F}$; for concreteness, say that the domain size is $|D| = 2^n$ and the degree is $d = |D|/2 = 2^{n-1}$.

The first level of [BS08]’s recursion appends to each codeword $f \in C$ an auxiliary function $\pi_1(f) : D' \rightarrow \mathbb{F}$ with domain $D'$ disjoint from $D$. Moreover, the mapping from $f$ to $\pi_1(f)$ is linear over $\mathbb{F}$, so the set $C^1 := \{f \parallel \pi_1(f)\}_{f \in C}$, where $f \parallel \pi_1(f) : D \cup D' \rightarrow \mathbb{F}$ is the function that agrees with $f$ on $D$ and with $\pi_1(f)$ on $D'$, is a linear code over $\mathbb{F}$. The code $C^1$ is the “first-level” code of a BS proximity proof for $f$.

The code $C^1$ has a naturally defined cover $S^1 = \{(\tilde{D}_j, \tilde{C}_j)\}_j$ such that each $\tilde{C}_j$ is a Reed–Solomon code $\text{RS}[\mathbb{F}, \tilde{D}_j, d_j]$ with $2d_j \leq |\tilde{D}_j| = O(\sqrt{d})$, that is, with rate $1/2$ and block length $O(\sqrt{d})$. We prove several combinatorial properties of this cover:

- $S^1$ is 1-intersecting. For all distinct $j, j'$ in $J$, $|\tilde{D}_j \cap \tilde{D}_{j'}| \leq 1$ (namely, the subdomains are almost disjoint).
- $S^1$ is $O(\sqrt{d})$-local. Every partial assignment to $O(\sqrt{d})$ domains $\tilde{D}_j$ in the cover that is locally consistent with the cover can be extended to a globally consistent assignment, i.e., to a codeword of $C^1$. That is, there exists $\kappa = O(\sqrt{d})$ such that every partial assignment $h : \cup_{l=1}^\kappa \tilde{D}_{j_l} \rightarrow \mathbb{F}$ with $h|_{\tilde{D}_{j_l}} \in \tilde{C}_{j_l}$ (for each $l$) equals the restriction to the subdomain $\cup_{l=1}^\kappa \tilde{D}_{j_l}$ of some codeword $f \parallel \pi_1(f)$ in $C^1$.
- $S^1$ is $O(\sqrt{d})$-independent. The ability to extend locally-consistent assignments to “globally-consistent” codewords of $C^1$ holds in a stronger sense: even when the aforementioned partial assignment $h$ is extended arbitrarily to $\kappa$ additional point-value pairs, this new partial assignment still equals the restriction of some codeword $f \parallel \pi_1(f)$ in $C^1$.

The locality property alone already suffices to imply that, given a subdomain $I \subseteq D \cup D'$ of size $|I| < \sqrt{d}$, we can solve the constraint detection problem on $I$ by considering only those constraints that appear in views that intersect $I$. But $C$ has exponential block length so a “quadratic speedup” does not yet imply succinct constraint detection. To obtain it, we also leverage the intersection and independence properties to reduce “locality” as follows.

Further recursive steps. So far we have only considered the first recursive step of a BS proximity proof; we show how to obtain covers with smaller locality (and thereby detect constraints with more efficiency) by considering additional recursive steps. Each code $\tilde{C}_j$ in the cover $S^1$ of $C^1$ is a Reed–Solomon code $\text{RS}[\mathbb{F}, \tilde{D}_j, d_j]$ with $|\tilde{D}_j|, d_j = O(\sqrt{d})$, and the next recursive step appends to
each codeword in $\tilde{C}_j$ a corresponding auxiliary function, yielding a new code $C^2$. In turn, $C^2$ has a cover $S^2$, and another recursive step yields a new code $C^3$, which has its own cover $S^3$, and so on. The crucial technical observation is that the intersection and independence properties, which hold recursively, enable us to deduce that $C^i$ is $1$-intersecting, $O(\sqrt[3]{p/d})$-local, and $O(\sqrt[3]{p/d})$-independent; in particular, for $r = \log \log d + O(1)$, $S^r$ is $1$-intersecting, $O(1)$-local, $O(1)$-independent.

Then, recalling that detecting constraints for local codes requires only the views in the cover that intersect $I$, our constraint detector works by choosing $i \in \{1, \ldots, r\}$ such that the cover $S^i$ is $\text{poly}(|I|)$-local, finding in this cover a $\text{poly}(|I|)$-size set of $\text{poly}(|I|)$-size views that intersect $I$, and computing in $\text{poly}(|I|)$ time a basis for the dual of each of these views — thereby proving Theorem 4.

Remark 1. For the sake of those familiar with BS-RS we remark that the domain $D'$ is the carefully chosen subset of $\mathbb{F} \times \mathbb{F}$ designated by that construction, the code $C^1$ is the code that evaluates bivariate polynomials of degree $O(\sqrt{d})$ on $D \cup D'$ (along the way mapping $D \subseteq \mathbb{F}$ to a subset of $\mathbb{F} \times \mathbb{F}$), the subdomains $D_j$ are the axis-parallel “rows” and “columns” used in that recursive construction, and the codes $\tilde{C}_j$ are Reed–Solomon codes of block length $O(\sqrt{d})$. The $O(\sqrt{d})$-locality and independence follow from basic properties of bivariate Reed–Muller codes; see the full version for more details.

Remark 2. It is interesting to compare the above result with linear lower bounds on query complexity for testing proximity to random low density parity check (LDPC) codes [BHR05, BGK+10]. Those results are proved by obtaining a basis for the dual code such that every small-support constraint is spanned by a small subset of that basis. The same can be observed to hold for BS-RS, even though this latter code is locally testable with polylogarithmic query complexity [BS08, Thm. 2.13]. The difference between the two cases is due to the fact that, for a random LDPC code, an assignment that satisfies all but a single basis-constraint is (with high probability) far from the code, whereas the recursive and 1-intersecting structure of BS-RS implies the existence of words that satisfy all but a single basis constraint, yet are negligibly close to being a codeword.

2.2 From constraint detection to zero knowledge via masking

We provide intuition about the connection between constraint detection and zero knowledge (Section 2.2), and how we leverage this connection to achieve two intermediate results: (i) a sumcheck protocol that is zero knowledge in the Interactive PCP model (Section 2.2); and (ii) proximity proofs for Reed–Solomon codes that are zero knowledge in the Interactive Oracle Proof model (Section 2.2).

Local simulation of random codewords Suppose that the prover and verifier both have oracle access to a codeword $w \in C$, for some linear code $C \subseteq \mathbb{F}^D$ with exponential-size domain $D$, and that they need to engage in some protocol...
that involves $w$. During the protocol, the prover may leak information about $w$ that is hard to compute (e.g., requires exponentially-many queries to $w$), and so would violate zero knowledge (as we see below, this is the case for protocols such as sumcheck).

Rather than directly invoking the protocol, the prover first sends to the verifier a random codeword $r \in C$ (as an oracle since $r$ has exponential size) and the verifier replies with a random field element $\rho \in \mathbb{F}$; then the prover and verifier invoke the protocol on the new codeword $w' := \rho w + r \in C$ rather than $w$. Intuitively, running the protocol on $w'$ now does not leak information about $w$, because $w'$ is random in $C$ (up to resolvable technicalities). This random self-reducibility makes sense for only some protocols, e.g., those where completeness is preserved for any choice of $\rho$ and soundness is broken for only a small fraction of $\rho$; but this will indeed be the case for the settings described below.

The aforementioned masking technique was used by [BCGV16] for codes with polynomial-size domains, but we use it for codes with exponential-size domains, which requires exponentially more efficient simulation techniques. Indeed, to prove (perfect) zero knowledge, a simulator must be able to reproduce, exactly, the view obtained by any malicious verifier that queries entries of $w'$, a uniformly random codeword in $C$; however, it is too expensive for the simulator to explicitly sample a random codeword and answer the verifier’s queries according to it. Instead, the simulator must sample the “local view” that the verifier sees while querying $w'$ at a small number of locations $I \subseteq D$.

But simulating local views of the form $w'|_I$ is reducible to detecting constraints, i.e., codewords in the dual code $C^\perp$ whose support is contained in $I$. Indeed, if no word in $C^\perp$ has support contained in $I$ then $w'|_I$ is uniformly random; otherwise, each additional linearly independent constraint of $C^\perp$ with support contained in $I$ further reduces the entropy of $w'|_I$ in a well-understood manner. (See Lemma 1 for a formal statement.) In sum, succinct constraint detection enables us to “implement” [GGN10, BW04] random codewords of $C$ despite $C$ having exponential size.

Note that in the above discussion we implicitly assumed that the set $I$ is known in advance, i.e., that the verifier chooses its queries in advance. This, of course, need not be the case: a verifier may adaptively make queries based on answers to previous queries and, hence, the set $I$ need not be known a priori. This turns out to not be a problem because, given a constraint detector, it is straightforward to compute the conditional distribution of the view $w'|_I$ given $w'|_J$ for a subset $J$ of $I$. This is expressed precisely in Lemma 1.

We now discuss two concrete protocols for which the aforementioned random self-reducibility applies, and for which we also have constructed suitably-efficient constraint detectors.

Zero knowledge sumchecks The celebrated sumcheck protocol [LFKN92] is not zero knowledge. In the sumcheck protocol, the prover and verifier have oracle access to a low-degree $m$-variate polynomial $F$ over a field $\mathbb{F}$, and the prover wants to convince the verifier that $\sum_{\alpha \in H^m} F(\alpha) = 0$ for a given subset $H$ of...
During the protocol, the prover communicates partial sums of $F$, which are \#P-hard to compute and, as such, violate zero knowledge.

We now explain how to use random self-reducibility to make the sumcheck protocol (perfect) zero knowledge, at the cost of moving from the Interactive Proof model to the Interactive PCP model.

**IPCP sumcheck.** Consider the following tweak to the classical sumcheck protocol: rather than invoking sumcheck on $F$ directly, the prover first sends to the verifier (the evaluation of) a random low-degree polynomial $R$ as an oracle; then, the prover sends the value $z := \sum_{\alpha \in H^m} R(\alpha)$ and the verifier replies with a random field element $\rho$; finally, the two invoke sumcheck on the claim “$\sum_{\alpha \in H^m} Q(\alpha) = z$” where $Q := \rho F + R$.

Completeness is clear because if $\sum_{\alpha \in H^m} F(\alpha) = 0$ and $\sum_{\alpha \in H^m} R(\alpha) = z$ then $\sum_{\alpha \in H^m} (\rho F + R)(\alpha) = z$; soundness is also clear because if $\sum_{\alpha \in H^m} F(\alpha) \neq 0$ then $\sum_{\alpha \in H^m} (\rho F + R)(\alpha) \neq z$ with high probability over $\rho$, regardless of the choice of $R$. (For simplicity, we ignore the fact that the verifier also needs to test that $R$ has low degree.) We are thus left to show (perfect) zero knowledge, which turns out to be a much less straightforward argument.

The simulator. Before we explain how to argue zero knowledge, we first clarify what we mean by it: since the verifier has oracle access to $F$ we cannot hope to ‘hide’ it; nevertheless, we can hope to argue that the verifier, by participating in the protocol, does not learn anything about $F$ beyond what the verifier can directly learn by querying $F$ (and the fact that $F$ sums to zero on $H^m$). What we shall achieve is the following: an algorithm that simulates the verifier’s view by making as many queries to $F$ as the total number of verifier queries to either $F$ or $R$.\(^{10}\)

On the surface, zero knowledge seems easy to argue, because $\rho F + R$ seems random among low-degree $m$-variate polynomials. More precisely, consider the simulator that samples a random low-degree polynomial $Q$ and uses it instead of $\rho F + R$ and answers the verifier queries as follows: (a) whenever the verifier queries $F(\alpha)$, respond by querying $F(\alpha)$ and returning the true value; (b) whenever the verifier queries $R(\alpha)$, respond by querying $F(\alpha)$ and returning $Q(\alpha) - \rho F(\alpha)$. Observe that the number of queries to $F$ made by the simulator equals the number of (mutually) distinct queries to $F$ and $R$ made by the verifier, as desired.

However, the above reasoning, while compelling, is insufficient. First, $\rho F + R$ is not random because a malicious verifier can choose $\rho$ depending on queries to $R$. Second, even if $\rho F + R$ were random (e.g., the verifier does not query $R$ before choosing $\rho$), the simulator must run in polynomial time, both producing correctly-distributed ‘partial sums’ of $\rho F + R$ and answering queries to $R$, but sampling $Q$ alone requires exponential time. In this high level discussion we

\(^{10}\) A subsequent work \cite{CFS17} shows how to bootstrap this IPCP sumcheck protocol into a more complex one that has a stronger zero knowledge guarantee: the simulator can sample the verifier’s view by making as many queries to $F$ as the number of verifier queries (plus one). Nevertheless, the weaker zero knowledge guarantee that we achieve suffices for our purposes.
ignore the first problem (which nonetheless has to be tackled), and focus on the second.

At this point it should be clear from the discussion in Section 2.2 that the simulator does not have to sample \( Q \) explicitly, but only has to perfectly simulate local views of it by leveraging the fact that it can keep state across queries. And doing so requires solving the succinct constraint detection problem for a suitable code \( C \). In this case, it suffices to consider the code \( C = \Sigma \text{RM} \), and our Theorem 3 guarantees the required constraint detector.

We refer the reader to the full version for further details.

Zero knowledge proximity proofs for Reed–Solomon. Testing proximity of a codeword \( w \) to a given linear code \( C \) can be aided by a proximity proof [DR04, BGI+06], which is an auxiliary oracle \( \pi \) that facilitates testing (e.g., \( C \) is not locally testable). For example, testing proximity to the Reed–Solomon code, a crucial step towards achieving short PCPs, is aided via suitable proximity proofs [BS08].

From the perspective of zero knowledge, however, a proximity proof can be ‘dangerous’: a few locations of \( \pi \) can in principle leak a lot of information about the codeword \( w \), and a malicious verifier could potentially learn a lot about \( w \) with only a few queries to \( w \) and \( \pi \). The notion of zero knowledge for proximity proofs requires that this cannot happen: it requires the existence of an algorithm that simulates the verifier’s view by making as many queries to \( w \) as the total number of verifier queries to either \( w \) or \( \pi \) [IW14]; intuitively, this means that any bit of the proximity proof \( \pi \) reveals no more information than one bit of \( w \).

We demonstrate again the use of random self-reducibility and show a general transformation that, under certain conditions, maps a PCP of proximity \( (P, V) \) for a code \( C \) to a corresponding 2-round Interactive Oracle Proof of Proximity (IOPP) for \( C \) that is (perfect) zero knowledge.

IOP of proximity for \( C \). Consider the following IOP of Proximity: the prover and verifier have oracle access to a codeword \( w \), and the prover wants to convince the verifier that \( w \) is close to \( C \); the prover first sends to the verifier a random codeword \( r \) in \( C \), and the verifier replies with a random field element \( \rho \); the prover then sends the proximity proof \( \pi' := P(w') \) that attests that \( w' := \rho w + r \) is close to \( C \). Note that this is a 2-round IOP of Proximity for \( C \), because completeness follows from the fact that \( C \) is linear, while soundness follows because if \( w \) is far from \( C \), then so is \( \rho w + r \) for every \( r \) with high probability over \( \rho \). But is the zero knowledge property satisfied?

The simulator. Without going into details, analogously to Section 2.2, a simulator must be able to sample local views for random codewords from the code \( L := \{ w || P(w) \}_{w \in C} \), so the simulator’s efficiency reduces to the efficiency of constraint detection for \( L \). We indeed prove that if \( L \) has succinct constraint detection then the simulator works out. See the full version for further details.

The case of Reed–Solomon. The above machinery allows us to derive a zero knowledge IOP of Proximity for Reed–Solomon codes, thanks to our Theorem 4,
which states that the family of linear codes comprised of evaluations of low-degree univariate polynomials concatenated with corresponding BS proximity proofs [BS08] has succinct constraint detection; see the full version for details. This is one of the building blocks of our construction of zero knowledge IOPs for NEXP, as described below in Section 2.3.

2.3 Achieving zero knowledge beyond NP

We outline how to derive our results about zero knowledge for #P and NEXP.

Zero knowledge for counting problems We provide intuition for the proof of Theorem 2, which states that the complexity class #P has Interactive PCPs that are perfect zero knowledge.

We first recall the classical (non zero knowledge) Interactive Proof for #P [LFKN92]. The language $\mathcal{L}_{\#3\text{SAT}}$, which consists of pairs $(\phi, N)$ where $\phi$ is a 3-CNF boolean formula and $N$ is the number of satisfying assignments of $\phi$, is #P-complete, and thus it suffices to construct an IP for it. The IP for $\mathcal{L}_{\#3\text{SAT}}$ works as follows: the prover and verifier both arithmetize $\phi$ to obtain a low-degree multivariate polynomial $p_\phi$ and invoke the (non zero knowledge) sumcheck protocol on the claim \[ \sum_{\alpha \in \{0,1\}^n} p_\phi(\alpha) = N, \] where arithmetic is over a large-enough prime field.

Returning to our goal, we obtain a perfect zero knowledge Interactive PCP by simply replacing the (non zero knowledge) IP sumcheck mentioned above with our perfect zero knowledge IPCP sumcheck, described in Section 2.2. In the full version we provide further details, including proving that the zero knowledge guarantees of our sumcheck protocol suffice for this case.

Zero knowledge for nondeterministic time We provide intuition for the proof of Theorem 1, which implies that the complexity class NEXP has Interactive Oracle Proofs that are perfect zero knowledge. Very informally, the proof consists of combining two building blocks: (i) [BCGV16]’s reduction from NEXP to randomizable linear algebraic constraint satisfaction problems, and (ii) our construction of perfect zero knowledge IOPs of Proximity for Reed–Solomon codes, described in Section 2.2. Besides extending [BCGV16]’s result from NP to NEXP, our proof provides a conceptual simplification over [BCGV16] by clarifying how the above two building blocks work together towards the final result. We now discuss this.

Starting point: [BS08]. Many PCP constructions consist of two steps: (1) arithmetize the statement at hand (in our case, membership of an instance in some NEXP-complete language) by reducing it to a “PCP-friendly” problem that looks like a linear-algebraic constraint satisfaction problem (LACSP); (2) design a tester that probabilistically checks witnesses for this LACSP. In this paper, as in [BCGV16], we take [BS08]’s PCPs for NEXP as a starting point, where the
Step 1: sanitize the proximity proof. We first address the problem that the BS proximity proof “leaks”, by simply replacing it with our own perfect zero knowledge analogue. Namely, we replace it with our perfect zero knowledge 2-round IOP of Proximity for Reed–Solomon codes, described in Section 2.2. This modification ensures that there exists an algorithm that perfectly simulates the verifier’s view by making as many queries to the LACSP witness as the total number of verifier queries to either the LACSP witness or other oracles used to facilitate proximity testing. At this point we have obtained a perfect zero knowledge 2-round IOP of Proximity for \( \text{NEXP} \) (analogous to the notion of a zero knowledge PCP of Proximity \([IW14]\)); this part is where, previously, \([BCGV16]\) were restricted to \( \text{NP} \) because their simulator only handled Reed–Solomon codes with polynomial degree while our simulator is efficient even for such codes with exponential degree. But we are not done yet: to obtain our goal, we also need to address the problem that the LACSP witness itself “leaks” when the verifier queries it, which we discuss next.

Step 2: sanitize the witness. Intuitively, we need to inject randomness in the reduction from \( \text{NEXP} \) to LACSP because the prover ultimately sends an LACSP witness to the verifier as an oracle, which the verifier can query. This is precisely what \([BCGV16]\)’s reduction from \( \text{NEXP} \) to randomizable LACSPs enables, and we thus use their reduction to complete our proof. Informally, given an a-priori query bound \( b \) on the verifier’s queries, the reduction outputs a witness \( w \) with the property that one can efficiently sample another witness \( w' \) whose entries are \( b \)-wise independent. We can then simply use the IOP of Proximity from the previous step on this randomized witness. Moreover, since the efficiency of the verifier is polylogarithmic in \( b \), we can set \( b \) to be super-polynomial (e.g., exponential) to preserve zero knowledge against any polynomial number of verifier queries.

The above discussion is only a sketch and we refer the reader to the full version for further details. One aspect that we did not discuss is that an LACSP witness actually consists of two sub-witnesses, where one is a “local” deterministic function of the other, which makes arguing zero knowledge somewhat more delicate.

2.4 Roadmap

Our results are structured as in the table below. For details, see the full version.
§4.2 Theorem 3/5 detecting constraints for $\SigmaRM$ §4.3 Theorem 4/6 detecting constraints for BS-$\SigmaRM$ PZK IPCP for sumcheck PZK IOP of Proximity for BS-$\SigmaRM$ Theorem 2 PZK IPCP for $\#P$ Theorem 1 PZK IOP for $\text{NEXP}$

3 Definitions

3.1 Basic notations

Functions, distributions, fields. We use $f : D \rightarrow R$ to denote a function with domain $D$ and range $R$; given a subset $D'$ of $D$, we use $f|_{D'}$ to denote the restriction of $f$ to $D'$. Given a distribution $D$, we write $x \leftarrow D$ to denote that $x$ is sampled according to $D$. We denote by $\mathbb{F}$ a finite field and by $\mathbb{F}_q$ the field of size $q$; we say $\mathbb{F}$ is a binary field if its characteristic is 2. Arithmetic operations over $\mathbb{F}_q$ cost $\log q$ but we shall consider these to have unit cost (and inspection shows that accounting for their actual polylogarithmic cost does not change any of the stated results).

Distances. A distance measure is a function $\Delta : \Sigma^n \times \Sigma^n \rightarrow [0, 1]$ such that for all $x, y, z \in \Sigma^n$: (i) $\Delta(x, x) = 0$, (ii) $\Delta(x, y) = \Delta(y, x)$, and (iii) $\Delta(x, y) \leq \Delta(x, z) + \Delta(z, y)$. We extend $\Delta$ to distances to sets: given $x \in \Sigma^n$ and $S \subseteq \Sigma^n$, we define $\Delta(x, S) := \min_{y \in S} \Delta(x, y)$ (or 1 if $S$ is empty). We say that a string $x$ is $\epsilon$-close to another string $y$ if $\Delta(x, y) \leq \epsilon$, and $\epsilon$-far from $y$ if $\Delta(x, y) > \epsilon$; similar terminology applies for a string $x$ and a set $S$. Unless noted otherwise, we use the relative Hamming distance over alphabet $\Sigma$ (typically implicit): $\Delta(x, y) := |\{i : x_i \neq y_i\}|/n$.

Languages and relations. We denote by $\mathcal{R}$ a (binary ordered) relation consisting of pairs $(x, w)$, where $x$ is the instance and $w$ is the witness. We denote by $\text{Lan}(\mathcal{R})$ the language corresponding to $\mathcal{R}$, and by $\mathcal{R}_x$ the set of witnesses in $\mathcal{R}$ for $x$ (if $x \not\in \text{Lan}(\mathcal{R})$ then $\mathcal{R}_x := \emptyset$). As always, we assume that $|w|$ is bounded by some computable function of $n := |x|$; in fact, we are mainly interested in relations arising from nondeterministic languages: $\mathcal{R} \in \text{NTIME}(T)$ if there exists a $T(n)$-time machine $M$ such that $M(x, w)$ outputs 1 if and only if $(x, w) \in \mathcal{R}$. Throughout, we assume that $T(n) \geq n$. We say that $\mathcal{R}$ has relative distance $\delta_{\mathcal{R}} : N \rightarrow [0, 1]$ if $\delta_{\mathcal{R}}(n)$ is the minimum relative distance among witnesses in $\mathcal{R}_x$ for all $x$ of size $n$. Throughout, we assume that $\delta_{\mathcal{R}}$ is a constant.

Polynomials. We denote by $\mathbb{F}[X_1, \ldots, X_m]$ the ring of polynomials in $m$ variables over $\mathbb{F}$. Given a polynomial $P$ in $\mathbb{F}[X_1, \ldots, X_m]$, $\text{deg}_{X_i}(P)$ is the degree of $P$ in the variable $X_i$. We denote by $\mathbb{F}[c \cdot d|X_1, \ldots, X_m]$ the subspace consisting of $P \in \mathbb{F}[X_1, \ldots, X_m]$ with $\text{deg}_{X_i}(P) < d$ for every $i \in \{1, \ldots, m\}$.

Random shifts. We later use a folklore claim about distance preservation for random shifts in linear spaces.
Claim. Let $n \in \mathbb{N}$, $F$ a finite field, $S$ an $F$-linear space in $F^n$, and $x, y \in F^n$. If $x$ is $\epsilon$-far from $S$, then $\alpha x + y$ is $\epsilon/2$-far from $S$, with probability $1 - |F|^{-1}$ over a random $\alpha \in F$. (Distances are relative Hamming distances.)

3.2 Single-prover proof systems

We use two types of proof systems that combine aspects of interactive proofs [Bab85, GMR89] and probabilistically checkable proofs [BFLS91, AS98, ALM+98]: interactive PCPs (IPCPs) [KR08] and interactive oracle proofs (IOPs) [BCS16, RRR16]. We first describe IPCPs (Section 3.2) and then IOPs (Section 3.2), which generalize the former.

Interactive probabilistically checkable proofs An IPCP [KR08] is a PCP followed by an IP. Namely, the prover $P$ and verifier $V$ interact as follows: $P$ sends to $V$ a probabilistically checkable proof $\pi$; afterwards, $P$ and $V^\pi$ engage in an interactive proof. Thus, $V$ may read a few bits of $\pi$ but must read subsequent messages from $P$ in full. An IPCP system for a relation $\mathcal{R}$ is thus a pair $(P, V)$, where $P, V$ are probabilistic interactive algorithms working as described, that satisfies naturally-defined notions of perfect completeness and soundness with a given error $\varepsilon(\cdot)$; see [KR08] for details.

We say that an IPCP has $k$ rounds if this “PCP round” is followed by a $(k-1)$-round interactive proof. (That is, we count the PCP round towards round complexity, unlike [KR08].) Beyond round complexity, we also measure how many bits the prover sends and how many the verifier reads: the proof length $l$ is the length of $\pi$ in bits plus the number of bits in all subsequent prover messages; the query complexity $q$ is the number of bits of $\pi$ read by the verifier plus the number of bits in all subsequent prover messages (since the verifier must read all of those bits).

In this work, we do not count the number of bits in the verifier messages, nor the number of random bits used by the verifier; both are bounded from above by the verifier’s running time, which we do consider. Overall, we say that a relation $\mathcal{R}$ belongs to the complexity class $\text{IPCP}[k, l, q, \varepsilon, tp, tv]$ if there is an IPCP system for $\mathcal{R}$ in which: (1) the number of rounds is at most $k(n)$; (2) the proof length is at most $l(n)$; (3) the query complexity is at most $q(n)$; (4) the soundness error is $\varepsilon(n)$; (5) the prover algorithm runs in time $tp(n)$; (6) the verifier algorithm runs in time $tv(n)$.

Interactive oracle proofs An IOP [BCS16, RRR16] is a “multi-round PCP”. That is, an IOP generalizes an interactive proof as follows: whenever the prover sends to the verifier a message, the verifier does not have to read the message in full but may probabilistically query it. In more detail, a $k$-round IOP comprises $k$ rounds of interaction. In the $i$-th round of interaction: the verifier sends a message $m_i$ to the prover; then the prover replies with a message $\pi_i$ to the verifier, which the verifier can query in this and later rounds (via oracle queries). After the $k$ rounds of interaction, the verifier either accepts or rejects.
An IOP system for a relation $R$ with soundness error $\varepsilon$ is thus a pair $(P, V)$, where $P, V$ are probabilistic interactive algorithms working as described, that satisfies the following properties. (See [BCS16] for more details.)

Completeness: For every instance-witness pair $(x, w)$ in the relation $R$, $\Pr[\langle P(x, w), V(x) \rangle = 1] = 1$.

Soundness: For every instance $x$ not in $R$'s language and unbounded malicious prover $P$, $\Pr[\langle P, V(x) \rangle = 1] \leq \varepsilon(n)$.

Beyond round complexity, we also measure how many bits the prover sends and how many the verifier reads: the proof length $l$ is the total number of bits in all of the prover’s messages, and the query complexity $q$ is the total number of bits read by the verifier across all of the prover’s messages. Considering all of these parameters, we say that a relation $R$ belongs to the complexity class $\text{IOP}[k, l, q, \varepsilon, \tau_p, \tau_v]$ if there is an IOP system for $R$ in which: (1) the number of rounds is at most $k(n)$; (2) the proof length is at most $l(n)$; (3) the query complexity is at most $q(n)$; (4) the soundness error is $\varepsilon(n)$; (5) the prover algorithm runs in time $\tau_p(n)$; (6) the verifier algorithm runs in time $\tau_v(n)$.

IOP vs. IPCP. An IPCP (see Section 3.2) is a special case of an IOP because an IPCP verifier must read in full all of the prover’s messages except the first one (while an IOP verifier may query any part of any prover message). The above complexity measures are consistent with those defined for IPCPs.

Restrictions and extensions The definitions below are about IOPs, but IPCPs inherit all of these definitions because they are a special case of IOP.

Adaptivity of queries. An IOP system is non-adaptive if the verifier queries are non-adaptive, i.e., the queried locations depend only on the verifier’s inputs.

Public coins. An IOP system is public coin if each verifier message $m_i$ is chosen uniformly and independently at random, and all of the verifier queries happen after receiving the last prover message.

Proximity. An IOP of proximity extends the definition of an IOP in the same way that a PCP of proximity extends that of a PCP [DR04, BGH+06]. An IOPP system for a relation $R$ with soundness error $\varepsilon$ and proximity parameter $\delta$ is a pair $(P, V)$ that satisfies the following properties.

Completeness: For every instance-witness pair $(x, w)$ in the relation $R$, $\Pr[\langle P(x, w), V^w(x) \rangle = 1] = 1$.

Soundness: For every instance-witness pair $(x, w)$ with $\Delta(w, R|_x) \geq \delta(n)$ and unbounded malicious prover $P$, $\Pr[\langle P, V^w(x) \rangle = 1] \leq \varepsilon(n)$.

Similarly to above, a relation $R$ belongs to the complexity class $\text{IOPP}[k, l, q, \varepsilon, \delta, \tau_p, \tau_v]$ if there is an IOPP system for $R$ with the corresponding parameters. Following [IW14], we call an IOPP exact if $\delta(n) = 0$.

Promise relations. A promise relation is a relation-language pair $(R^\text{YES}, L^\text{NO})$ with $\text{Lang}(R^\text{YES}) \cap L^\text{NO} = \emptyset$. An IOP for a promise relation is the same as an IOP for the (standard) relation $R^\text{YES}$, except that soundness need only hold for $x \in$
An IOPP for a promise relation is the same as an IOPP for the (standard) relation $R^{\text{YES}}$, except that soundness need only hold for $x \in \text{Lan}(R^{\text{YES}}) \cup L^{\text{NO}}$.

Prior constructions In this paper we give new IPCP and IOP constructions that achieve perfect zero knowledge for various settings. Below we summarize known constructions in these two models.

IPCPs. Prior work obtains IPCPs with proof length that depends on the witness size rather than computation size [KR08, GKR08], and IPCPs with statistical zero knowledge [GIMS10] (see Section 3.3 for more details).

IOPs. Prior work obtains IOPs with perfect zero knowledge for NP [BCGV16], IOPs with small proof length and query complexity [BCG+17], and an amortization theorem for “unambiguous” IOPs [RRR16]. Also, [BCS16] show how to compile public-coin IOPs into non-interactive arguments in the random oracle model.

3.3 Zero knowledge

We define the notion of zero knowledge for IOPs and IPCPs achieved by our constructions: unconditional (perfect) zero knowledge via straightline simulators. This notion is quite strong not only because it unconditionally guarantees simulation of the verifier’s view but also because straightline simulation implies desirable properties such as composability. We now provide some context and then give formal definitions.

At a high level, zero knowledge requires that the verifier’s view can be efficiently simulated without the prover. Converting the informal statement into a mathematical one involves many choices, including choosing which verifier class to consider (e.g., the honest verifier? all polynomial-time verifiers?), the quality of the simulation (e.g., is it identically distributed to the view? statistically close to it? computationally close to it?), the simulator’s dependence on the verifier (e.g., is it non-uniform? or is the simulator universal?), and others. The definitions below consider two variants: perfect simulation via universal simulators against either unbounded-query or bounded-query verifiers.

Moreover, in the case of universal simulators, one distinguishes between a non-blackbox use of the verifier, which means that the simulator takes the verifier’s code as input, and a blackbox use of it, which means that the simulator only accesses the verifier via a restricted interface; we consider this latter case. Different models of proof systems call for different interfaces, which grant carefully-chosen “extra powers” to the simulator (in comparison to the prover) so to ensure that efficiency of the simulation does not imply the ability to efficiently decide the language. For example: in ZK IPs, the simulator may rewind the verifier; in ZK PCPs, the simulator may adaptively answer oracle queries. In ZK IPCPs and ZK IOPs (our setting), the natural definition would allow a blackbox simulator to rewind the verifier and also to adaptively answer oracle queries. The definitions below, however, consider only simulators that are straightline [FS89, DS98], that
is they do not rewind the verifier, because our constructions achieve this stronger notion.

We are now ready to define the notion of unconditional (perfect) zero knowledge via straightline simulators. We first discuss the notion for IOPs, then for IOPs of proximity, and finally for IPCPs.

ZK for IOPs We define zero knowledge (via straightline simulators) for IOPs. We begin by defining the view of an IOP verifier.

Definition 4. Let $A, B$ be algorithms and $x, y$ strings. We denote by $\text{View}\langle B(y), A(x) \rangle$ the view of $A(x)$ in an interactive oracle protocol with $B(y)$, i.e., the random variable $(x, r, a_1, \ldots, a_n)$ where $x$ is $A$’s input, $r$ is $A$’s randomness, and $a_1, \ldots, a_n$ are the answers to $A$’s queries into $B$’s messages.

Straightline simulators in the context of IPs were used in [FS89], and later defined in [DS98]. The definition below considers this notion in the context of IOPs, where the simulator also has to answer oracle queries by the verifier. Note that since we consider the notion of unconditional (perfect) zero knowledge, the definition of straightline simulation needs to allow the efficient simulator to work even with inefficient verifiers [GIMS10].

Definition 5. We say that an algorithm $B$ has straightline access to another algorithm $A$ if $B$ interacts with $A$, without rewinding, by exchanging messages with $A$ and also answering any oracle queries along the way. We denote by $B^A$ the concatenation of $A$’s random tape and $B$’s output. (Since $A$’s random tape could be super-polynomially large, $B$ cannot sample it for $A$ and then output it; instead, we restrict $B$ to not see it, and we prepend it to $B$’s output.)

Recall that an algorithm $A$ is $b$-query if, on input $x$, it makes at most $b(|x|)$ queries to any oracles it has access to. We are now ready to define zero knowledge IOPs.

Definition 6. An IOP system $(P, V)$ for a relation $R$ is perfect zero knowledge (via straightline simulators) against unbounded queries (resp., against query bound $b$) if there exists a simulator algorithm $S$ such that for every algorithm (resp., $b$-query algorithm) $\tilde{V}$ and instance-witness pair $(x, w) \in R$, $S^V(x)$ and $\text{View}(P(x, w), \tilde{V}(x))$ are identically distributed. Moreover, $S$ must run in time $\text{poly}(|x| + q_{\tilde{V}}(|x|))$, where $q_{\tilde{V}}(\cdot)$ is $\tilde{V}$’s query complexity.

For zero knowledge against arbitrary polynomial-time adversaries, it suffices for $b$ to be superpolynomial. Note that $S$’s running time need not be polynomial in $b$ (in our constructions it is polylogarithmic in $b$); rather its running time may be polynomial in the input size $|x|$ and the actual number of queries $\tilde{V}$ makes (as a random variable).

We say that a relation $R$ belongs to the complexity class $\text{PZK-IOP}[k, l, q, \varepsilon, tp, tv, b]$ if there is an IOP system for $R$, with the corresponding parameters, that is perfect zero knowledge with query bound $b$; also, it belongs to the complexity class $\text{PZK-IOP}[k, l, q, \varepsilon, tp, tv, \ast]$ if the same is true with unbounded queries.
ZK for IOPs of proximity We define zero knowledge (via straightline simulators) for IOPs of proximity. It is a straightforward extension of the corresponding notion for PCPs of proximity, introduced in [IW14].

Definition 7. An IOPP system \( (P, V) \) for a relation \( \mathcal{R} \) is perfect zero knowledge (via straightline simulators) against unbounded queries (resp., against query bound \( b \)) if there exists a simulator algorithm \( S \) such that for every algorithm \( (\text{resp., } b\text{-query algorithm}) \) \( \hat{V} \) and instance-witness pair \( (x, w) \in \mathcal{R} \), the following two random variables are identically distributed:

\[
\left( S^{V, w}(x), q_S \right) \text{ and } \left( \text{View} \left( P(x, w), \hat{V}^{w}(x) \right), q_{\hat{V}} \right),
\]

where \( q_S \) is the number of queries to \( w \) made by \( S \), and \( q_{\hat{V}} \) is the number of queries to \( w \) or to prover messages made by \( \hat{V} \). Moreover, \( S \) must run in time \( \text{poly}(|x| + q_{\hat{V}}(|x|)) \), where \( q_{\hat{V}}(\cdot) \) is \( \hat{V} \)'s query complexity.

We say that a relation \( \mathcal{R} \) belongs to the complexity class \( \text{PZK-IOPP}[k, l, q, \varepsilon, \delta, t_p, t_v, b] \) if there is an IOPP system for \( \mathcal{R} \), with the corresponding parameters, that is perfect zero knowledge with query bound \( b \); also, it belongs to the complexity class \( \text{PZK-IOPP}[k, l, q, \varepsilon, \delta, t_p, t_v, *] \) if the same is true with unbounded queries.

Remark 3. Analogously to [IW14], our definition of zero knowledge for IOPs of proximity requires that the number of queries to \( w \) by \( S \) equals the total number of queries (to \( w \) or prover messages) by \( \hat{V} \). Stronger notions are possible: “the number of queries to \( w \) by \( S \) equals the number of queries to \( w \) by \( \hat{V} \)” or, even more, “\( S \) and \( \hat{V} \) read the same locations of \( w \)”. The definition above is sufficient for the applications of IOPs of proximity that we consider.

ZK for IPCPs The definition of perfect zero knowledge (via straightline simulators) for IPCPs follows directly from Definition 6 in Section 3.3 because IPCPs are a special case of IOPs. Ditto for IPCPs of proximity, whose perfect zero knowledge definition follows directly from Definition 7 in Section 3.3. (For comparison, [GIMS10] define statistical zero knowledge IPCPs, also with straightline simulators.)

3.4 Codes

An error correcting code \( C \) is a set of functions \( w: D \to \Sigma \), where \( D, \Sigma \) are finite sets known as the domain and alphabet; we write \( C \subseteq \Sigma^D \). The message length of \( C \) is \( k := \log_{|\Sigma|}|C| \), its block length is \( \ell := |D| \), its rate is \( \rho := k/\ell \), its (minimum) distance is \( d := \min\{\Delta(w, z) : w, z \in C, w \neq z\} \) when \( \Delta \) is the (absolute) Hamming distance, and its (minimum) relative distance is \( \tau := d/\ell \). At times we write \( k(C), \ell(C), \rho(C), d(C), \tau(C) \) to make the code under consideration explicit. All the codes we consider are linear codes, discussed next.

Linearity. A code \( C \) is linear if \( \Sigma \) is a finite field and \( C \) is a \( \Sigma \)-linear space in \( \Sigma^D \). The dual code of \( C \) is the set \( C^\perp \) of functions \( z: D \to \Sigma \) such that, for all
w : D \to \Sigma, \langle z, w \rangle := \sum_{i \in D} z(i)w(i) = 0. We denote by dim(C) the dimension of C; it holds that dim(C) + dim(C^⊥) = \ell and dim(C) = k (dimension equals message length).

Code families. A code family \(\mathcal{C} = \{C_n\}_{n \in \{0,1\}}\) has domain \(D(\cdot)\) and alphabet \(\mathbb{F}(\cdot)\) if each code \(C_n\) has domain \(D(n)\) and alphabet \(\mathbb{F}(n)\). Similarly, \(\mathcal{C}\) has message length \(k(\cdot)\), block length \(\ell(\cdot)\), rate \(\rho(\cdot)\), distance \(d(\cdot)\), and relative distance \(\tau(\cdot)\) if each code \(C_n\) has message length \(k(n)\), block length \(\ell(n)\), rate \(\rho(n)\), distance \(d(n)\), and relative distance \(\tau(n)\). We also define \(\rho(\mathcal{C}) := \inf_{n \in \mathbb{N}} \rho(n)\) and \(\tau(\mathcal{C}) := \inf_{n \in \mathbb{N}} \tau(n)\).

Reed–Solomon codes. The Reed–Solomon (RS) code is the code consisting of evaluations of univariate low-degree polynomials: given a field \(F\), subset \(S\) of \(F\), and positive integer \(d\) with \(d \leq |S|\), we denote by \(\text{RS}[F,S,d]\) the linear code consisting of evaluations \(w : S \to F\) over \(S\) of polynomials in \(F^{<d}[X]\). The code’s message length is \(k = d\), block length is \(\ell = |S|\), rate is \(\rho = \frac{d}{|S|}\), and relative distance is \(\tau = 1 - \frac{d-1}{|S|}\).

Reed–Muller codes. The Reed–Muller (RM) code is the code consisting of evaluations of multivariate low-degree polynomials: given a field \(F\), subset \(S\) of \(F\), and positive integers \(m, d\) with \(d \leq |S|\), we denote by \(\text{RM}[F,S,m,d]\) the linear code consisting of evaluations \(w : S^m \to F\) over \(S^m\) of polynomials in \(F^{<d}[X_1, \ldots, X_m]\) (i.e., we bound individual degrees rather than their sum). The code’s message length is \(k = d^m\), block length is \(\ell = |S|^m\), rate is \(\rho = \left(\frac{d}{|S|}\right)^m\), and relative distance is \(\tau = (1 - \frac{d-1}{|S|})^m\).

4 Succinct constraint detection

We introduce the notion of succinct constraint detection for linear codes. This notion plays a crucial role in constructing perfect zero knowledge simulators for super-polynomial complexity classes (such as \#P and \textbf{NEXP}), but we believe that this naturally-defined notion is also of independent interest. Given a linear code \(C \subseteq F^D\) we refer to its dual code \(C^\perp \subseteq F^D\) as the constraint space of \(C\). The constraint detection problem corresponding to a family of linear codes \(\mathcal{C} = \{C_n\}_n\) with domain \(D(\cdot)\) and alphabet \(\mathbb{F}(\cdot)\) is the following:

Given an index \(n\) and subset \(I \subseteq D(n)\), output a basis for \(\{z \in D(n)^I : \forall w \in C_n, \sum_{i \in I} z(i)w(i) = 0\}\).\(^{11}\)

If \(|D(n)|\) is polynomial in \(|n|\) and a generating matrix for \(C_n\) can be found in polynomial time, this problem can be solved in \(\text{poly}(|n| + |I|)\) time via Gaussian elimination; such an approach was implicitly taken by [BCGV16] to construct a perfect zero knowledge simulator for an IOP for \textbf{NP}. However, in our setting,

\(^{11}\)In fact, the following weaker definition suffices for the applications in our paper: given an index \(n\) and subset \(I \subseteq D(n)\), output \(z \in \mathbb{F}(n)^I\) such that \(\sum_{i \in I} z(i)w(i) = 0\) for all \(w \in C_n\), or ‘independent’ if no such \(z\) exists. We achieve the stronger definition, which is also easier to work with.
$|D(n)|$ is exponential in $|n|$ and $|I|$, and the aforementioned generic solution requires exponential time. With this in mind, we say $C$ has succinct constraint detection if there exists an algorithm that solves the constraint detection problem in $\text{poly}(|n| + |I|)$ time when $|D(n)|$ is exponential in $|n|$. After defining succinct constraint detection in Section 4.1, we proceed as follows.

- In Section 4.2, we construct a succinct constraint detector for the family of linear codes comprised of evaluations of partial sums of low-degree polynomials. The construction of the detector exploits derandomization techniques from algebraic complexity theory. We leverage this result to construct a perfect zero knowledge simulator for an IPCP for $#P$; see the full version for details.

- In Section 4.3, we construct a succinct constraint detector for the family of evaluations of univariate polynomials concatenated with corresponding BS proximity proofs [BS08]. The construction of the detector exploits the recursive structure of these proximity proofs. We leverage this result to construct a perfect zero knowledge simulator for an IOP for $\text{NEXP}$; this simulator can be interpreted as an analogue of [BCGV16]'s simulator that runs exponentially faster and thus enables us to “scale up” from $\text{NP}$ to $\text{NEXP}$; see the full version for details.

Throughout this section we assume familiarity with terminology and notation about codes, introduced in Section 3.4. We assume for simplicity that $|n|$, the number of bits used to represent $n$, is at least $\log D(n) + \log F(n)$; if this does not hold, then one can replace $|n|$ with $|n| + \log D(n) + \log F(n)$ throughout the section.

Remark 4 (sparse representation). In this section we make statements about vectors $v$ in $\mathbb{F}^D$ where the cardinality of the domain $D$ may be super-polynomial. When such statements are computational in nature, we assume that $v$ is not represented as a list of $|D|$ field elements (which requires $\Omega(|D| \log |\mathbb{F}|)$ bits) but, instead, assume that $v$ is represented as a list of the elements in $\text{supp}(v)$ (and each element comes with its index in $D$); this sparse representation only requires $\Omega(|\text{supp}(v)| \cdot (\log |D| + \log |\mathbb{F}|))$ bits.

### 4.1 Definition of succinct constraint detection

Formally define the notion of a constraint detector, and the notion of succinct constraint detection.

Definition 8. Let $C = \{C_n\}_n$ be a linear code family with domain $D(\cdot)$ and alphabet $\mathbb{F}(\cdot)$. A constraint detector for $C$ is an algorithm that, on input an index $n$ and subset $I \subseteq D(n)$, outputs a basis for the space

$$\left\{ z \in D(n)^I : \forall w \in C_n, \sum_{i \in I} z(i) w(i) \right\} .$$

We say that $C$ has $T(\cdot, \cdot)$-time constraint detection if there exists a detector for $C$ running in time $T(n, \ell)$; we also say that $C$ has succinct constraint detection if it has $\text{poly}(|n| + \ell)$-time constraint detection.
A constraint detector induces a corresponding probabilistic algorithm for ‘simulating’ answers to queries to a random codeword; this is captured by the following lemma, the proof of which is in the full version. We shall use such probabilistic algorithms in the construction of perfect zero knowledge simulators.

Lemma 1. Let \( \mathcal{C} = \{C_n\}_n \) be a linear code family with domain \( D(\cdot) \) and alphabet \( \mathbb{F}(\cdot) \) that has \( T(\cdot, \cdot) \)-time constraint detection. Then there exists a probabilistic algorithm \( \mathcal{A} \) such that, for every index \( n \), set of pairs \( S = \{(\alpha_1, \beta_1), \ldots, (\alpha_t, \beta_t)\} \subseteq D(n) \times \mathbb{F}(n) \), and pair \( (\alpha, \beta) \in D(n) \times \mathbb{F}(n) \),

\[
\Pr \left[ \mathcal{A}(n, S, \alpha) = \beta \right] = \Pr_{w \leftarrow C_n} \left[ \begin{array}{c} w(\alpha_1) = \beta_1 \\ \vdots \\ w(\alpha_t) = \beta_t \end{array} \right].
\]

Moreover \( \mathcal{A} \) runs in time \( T(n, \ell) + \text{poly}(\log |\mathbb{F}(n)| + \ell) \).

For the purposes of constructing a constraint detector, the sufficient condition given in Lemma 2 below is sometimes easier to work with. To state it we need to introduce two ways of restricting a code, and explain how these restrictions interact with taking duals; the interplay between these is delicate (see Remark 5).

Definition 9. Given a linear code \( C \subseteq \mathbb{F}^D \) and a subset \( I \subseteq D \), we denote by
(i) \( C_{\subseteq I} \) the set consisting of the codewords \( w \in C \) for which \( \text{supp}(w) \subseteq I \), and
(ii) \( C|_I \) the restriction to \( I \) of codewords \( w \in C \).

Note that \( C_{\subseteq I} \) and \( C|_I \) are different notions. Consider for example the 1-dimensional linear code \( C = \{00, 11\} \) in \( \mathbb{F}^{1,2}_2 \) and the subset \( I = \{1\} \): it holds that \( C_{\subseteq I} = \{00\} \) and \( C|_I = \{0, 1\} \). In particular, codewords in \( C_{\subseteq I} \) are defined over \( D \), while codewords in \( C|_I \) are defined over \( I \). Nevertheless, throughout this section, we sometimes compare vectors defined over different domains, with the implicit understanding that the comparison is conducted over the union of the relevant domains, by filling in zeros in the vectors’ undefined coordinates.

For example, we may write \( C_{\subseteq I} \subseteq C|_I \) to mean that \( \{00\} \subseteq \{00, 10\} \) (the set obtained from \( \{0, 1\} \) after filling in the relevant zeros).

Claim. Let \( C \) be a linear code with domain \( D \) and alphabet \( \mathbb{F} \). For every \( I \subseteq D \),

\[
(C|_I)^\perp = (C_{\subseteq I})^\perp,
\]

that is,

\[
\left\{ z \in D(n)^I : \forall w \in C_n, \sum_{i \in I} z(i)w(i) \right\} = \left\{ z \in C_n^\perp : \text{supp}(z) \subseteq I \right\}.
\]

Proof. For the containment \( (C_{\subseteq I})^\perp \subseteq (C|_I)^\perp \): if \( z \in C^\perp \) and \( \text{supp}(z) \subseteq I \) then \( z \) lies in the dual of \( C|_I \) because it suffices to consider the subdomain \( I \) for determining duality. For the reverse containment \( (C_{\subseteq I})^\perp \supseteq (C|_I)^\perp \): if \( z \in (C|_I)^\perp \) then \( \text{supp}(z) \subseteq I \) (by definition) so that \( \langle z, w \rangle = \langle z, w|_I \rangle \) for every \( w \in C \), and
the latter inner product equals 0 because \( z \) is in the dual of \( C_{I} \); in sum \( z \) is dual to (all codewords in) \( C \) and its support is contained in \( I \), so \( z \) belongs to \( (C^\perp)_{\subseteq I} \), as claimed.

Observe that Claim 4.1 tells us the constraint detection is equivalent to determining a basis of \( (C_{n|I})^\perp = (C_{n}^\perp)_{\subseteq I} \). The following lemma asserts that if, given a subset \( I \subseteq D \), we can find a set of constraints \( W \) in \( C^\perp \) that spans \( (C^\perp)_{\subseteq I} \), then we can solve the constraint detection problem for \( C \); see the full version for a proof.

**Lemma 2.** Let \( \mathcal{C} = \{ C_n \}_{n} \) be a linear code family with domain \( D(\cdot) \) and alphabet \( \mathbb{F}(\cdot) \). If there exists an algorithm that, on input an index \( n \) and subset \( I \subseteq D(n) \), outputs in \( \text{poly}(|n| + |I|) \) time a subset \( W \subseteq \mathbb{F}(n)^{D(n)} \) (in sparse representation) with \( (C_n^\perp)_{\subseteq I} \subseteq \text{span}(W) \subseteq C_n^\perp \), then \( \mathcal{C} \) has succinct constraint detection.

**Remark 5.** The following operations do not commute: (i) expanding the domain via zero padding (for the purpose of comparing vectors over different domains), and (ii) taking the dual of the code. Consider for example the code \( C = \{ 0 \} \subseteq \mathbb{F}_2^\{1\} \): its dual code is \( C^\perp = \{ 0, 1 \} \) and, when expanded to \( \mathbb{F}_2^{\{1,2\}} \), the dual code is expanded to \( \{ (0,0), (1,0) \} \); yet, when \( C \) is expanded to \( \mathbb{F}_2^{\{1,2\}} \) it produces the code \( \{ (0,0) \} \) and its dual code is \( \{ (0,0), (1,0), (0,1), (1,1) \} \). To resolve ambiguities (when asserting an equality as in Claim 4.1), we adopt the convention that expansion is done always last (namely, as late as possible without having to compare vectors over different domains).

### 4.2 Partial sums of low-degree polynomials

We show that evaluations of partial sums of low-degree polynomials have succinct constraint detection (see Definition 8). In the following, \( \mathbb{F} \) is a finite field, \( m, d \) are positive integers, and \( H \) is a subset of \( \mathbb{F} \); also, \( \mathbb{F}^{<d}[X_1, \ldots, X_m] \) denotes the subspace of \( \mathbb{F}[X_1, \ldots, X_m] \) consisting of those polynomials with individual degrees less than \( d \). Moreover, given \( Q \in \mathbb{F}^{<d}[X_1, \ldots, X_m] \) and \( \alpha \in \mathbb{F}^{\leq m} \) (vectors over \( \mathbb{F} \) of length at most \( m \)), we define \( Q(\alpha) := \sum_{\gamma \in H^{m-\langle \alpha \rangle}} Q(\alpha, \gamma) \), i.e., the answer to a query that specifies only a suffix of the variables is the sum of the values obtained by letting the remaining variables range over \( H \). We begin by defining the code that we study, which extends the Reed–Muller code (see Section 3.4) with partial sums.

**Definition 10.** We denote by \( \Sigma RM[\mathbb{F},m,d,H] \) the linear code that comprises evaluations of partial sums of polynomials in \( \mathbb{F}^{<d}[X_1, \ldots, X_m] \); more precisely, \( \Sigma RM[\mathbb{F},m,d,H] := \{ w_Q \}_{Q \in \mathbb{F}^{<d}[X_1, \ldots, X_m]} \) where \( w_Q: \mathbb{F}^{\leq m} \to \mathbb{F} \) is the function defined by \( w_Q(\alpha) := \sum_{\gamma \in H^{m-\langle \alpha \rangle}} Q(\alpha, \gamma) \) for each \( \alpha \in \mathbb{F}^{\leq m} \). We denote by \( \Sigma RM[\mathbb{F},m,d,H] \) is indeed linear, for every \( w_{Q_1}, w_{Q_2} \in \Sigma RM[\mathbb{F},m,d,H] \), \( a_1, a_2 \in \mathbb{F} \), and \( \alpha \in \mathbb{F}^{\leq m} \), it holds that \( a_1 w_{Q_1}(\alpha) + a_2 w_{Q_2}(\alpha) = a_1 \sum_{\gamma \in H^{m-\langle \alpha \rangle}} Q_1(\alpha, \gamma) + a_2 \sum_{\gamma \in H^{m-\langle \alpha \rangle}} Q_2(\alpha, \gamma) = \sum_{\gamma \in H^{m-\langle \alpha \rangle}} (a_1 Q_1 + a_2 Q_2)(\alpha, \gamma) = w_{a_1 Q_1 + a_2 Q_2}(\alpha) \). But \( w_{a_1 Q_1 + a_2 Q_2} \in \Sigma RM[\mathbb{F},m,d,H] \), since \( \mathbb{F}^{<d}[X_1, \ldots, X_m] \) is a linear space.
We prove that the linear code family $\Sigma_{RM}$ has succinct constraint detection:

**Theorem 5** (formal statement of 3). $\Sigma_{RM}$ has $\text{poly}(\log |\mathbb{F}| + m + d + |H| + \ell)$-time constraint detection.

Combined with Lemma 1, the above theorem implies that there exists a probabilistic polynomial-time algorithm for answering queries to a codeword sampled at random from $\Sigma_{RM}$, as captured by the following corollary.

**Corollary 1.** There exists a probabilistic algorithm $A$ such that, for every finite field $\mathbb{F}$, positive integers $m, d$, subset $H$ of $\mathbb{F}$, subset $S = \{(\alpha_1, \beta_1), \ldots, (\alpha_\ell, \beta_\ell)\} \subseteq \mathbb{F}^\leq m \times \mathbb{F}$, and $(\alpha, \beta) \in \mathbb{F}^{\leq m} \times \mathbb{F}$,

$$\Pr\left[A(\mathbb{F}, m, d, H, S, \alpha) = \beta\right] = \Pr_{R \leftarrow \mathbb{F}^{< d}[X_1, \ldots, X_m]} \left[R(\alpha) = \beta \begin{bmatrix} R(\alpha_1) = \beta_1 \\ \vdots \\ R(\alpha_\ell) = \beta_\ell \end{bmatrix} \right].$$

Moreover $A$ runs in time $\text{poly}(\log |\mathbb{F}| + m + d + |H| + \ell)$.

We sketch the proof of Theorem 5, for the simpler case where the code is $\Sigma_{RM}[\mathbb{F}, m, d, H]$ (i.e., without partial sums). We can view a polynomial $Q \in \mathbb{F}^{< d}[X_1, \ldots, X_m]$ as a vector over the monomial basis, with an entry for each possible monomial $X_1^{i_1} \cdots X_m^{i_m}$ (with $0 \leq i_1, \ldots, i_m < d$) containing the corresponding coefficient. The evaluation of $Q$ at a point $\alpha \in \mathbb{F}^m$ then equals the inner product of this vector with the vector $\phi_\alpha$, in the same basis, whose entry for $X_1^{i_1} \cdots X_m^{i_m}$ is equal to $\alpha_1^{i_1} \cdots \alpha_m^{i_m}$. Given $\alpha_1, \ldots, \alpha_\ell$, we could use Gaussian elimination on $\phi_\alpha_1, \ldots, \phi_\alpha_\ell$ to check for linear dependencies, which would be equivalent to constraint detection for $\Sigma_{RM}[\mathbb{F}, m, d, H]$.

However, we cannot afford to explicitly write down $\phi_\alpha$, because it has $d^m$ entries. Nevertheless, we can still implicitly check for linear dependencies, and we do so by reducing the problem, by building on and extending ideas of [BW04], to computing the nullspace of a certain set of polynomials, which can be solved via an algorithm of [RS05] (see also [Kay10]). The idea is to encode the entries of these vectors via a succinct description: a polynomial $\Phi_\alpha$, whose coefficients (after expansion) are the entries of $\phi_\alpha$. In our setting this polynomial has the particularly natural form:

$$\Phi_\alpha(X) := \prod_{i=1}^m (1 + \alpha_i X_i + \alpha_i^2 X_i^2 + \cdots + \alpha_i^{d-1} X_i^{d-1});$$

note that the coefficient of each monomial equals its corresponding entry in $\phi_\alpha$. Given this representation we can use standard polynomial identity testing techniques to find linear dependencies between these polynomials, which corresponds...
to linear dependencies between the original vectors. Crucially, we cannot afford any mistake, even with exponentially small probability, when looking for linear dependencies for otherwise we would not achieve perfect simulation; this is why the techniques we leverage rely on derandomization. We now proceed with the full proof.

Proof (Proof of Theorem 5). We first introduce some notation. Define \( \langle < d \rangle := \{0, \ldots, d-1\} \). For vectors \( \alpha \in \mathbb{F}^m \) and \( a \in \langle < d \rangle \), we define \( \alpha^a := \prod_{i=1}^m \alpha_i^a \); similarly, for variables \( X = (X_1, \ldots, X_m) \), we define \( X^a := \prod_{i=1}^m X_i^{a_i} \).

We identify \( \Sigma \mathcal{RM}[\mathbb{F}, m, d, H] \) with \( \mathbb{F}^{\langle < d \rangle \times m} \); a codeword \( w_Q \) then corresponds to a vector \( Q \) whose \( a \)-th entry is the coefficient of the monomial \( X^a \) in \( Q \). For \( \alpha \in \mathbb{F}^{\langle < d \rangle \times m} \), let

\[
\phi_\alpha := \left( \alpha^a \sum_{\gamma \in H^{m-\langle |\alpha| \rangle}} \gamma^b \right)_{a \in \langle < d \rangle \times m, \ b \in \langle < d \rangle^{m-\langle |\alpha| \rangle}} .
\]

We can also view \( \phi_\alpha \) as a vector in \( \mathbb{F}^{\langle < d \rangle \times m} \) by merging the indices, so that, for all \( \alpha \in \mathbb{F}^{\langle < d \rangle \times m} \) and \( w_Q \in \Sigma \mathcal{RM}[\mathbb{F}, m, d, H] \),

\[
w_Q(\alpha) = \sum_{\gamma \in H^{m-\langle |\alpha| \rangle}} \sum_{a \in \langle < d \rangle \times m} \sum_{b \in \langle < d \rangle^{m-\langle |\alpha| \rangle}} Q_{a,b} \cdot \alpha^a \gamma^b = \sum_{a \in \langle < d \rangle \times m} \sum_{b \in \langle < d \rangle^{m-\langle |\alpha| \rangle}} Q_{a,b} \cdot \alpha^a \sum_{\gamma \in H^{m-\langle |\alpha| \rangle}} \gamma^b = \langle Q, \phi_\alpha \rangle .
\]

Hence for every \( \alpha_1, \ldots, \alpha_\ell, \alpha \in \mathbb{F}^{\langle < d \rangle \times m} \) and \( a_1, \ldots, a_\ell \in \mathbb{F} \), the following statements are equivalent (i) \( w(\alpha) = \sum_{i=1}^\ell a_i w(\alpha_i) \) for all \( w \in \Sigma \mathcal{RM}[\mathbb{F}, m, d, H] \); (ii) \( \langle f, \phi_\alpha \rangle = \sum_{i=1}^\ell a_i \langle f, \phi_{\alpha_i} \rangle \) for all \( f \in \mathbb{F}^{\langle < d \rangle \times m} \) (iii) \( \phi_\alpha = \sum_{i=1}^\ell a_i \phi_{\alpha_i} \). We deduce that constraint detection for \( \Sigma \mathcal{RM}[\mathbb{F}, m, d, H] \) is equivalent to the problem of finding \( a_1, \ldots, a_\ell \in \mathbb{F} \) such that \( \phi_\alpha = \sum_{i=1}^\ell a_i \phi_{\alpha_i} \), or returning ‘independent’ if no such \( a_1, \ldots, a_\ell \) exist.

However, the dimension of the latter vectors is \( d^m \), which may be much larger than \( \text{poly}(\log |\mathbb{F}| + m + d + |H| + \ell) \), and so we cannot afford to “explicitly” solve the \( \ell \times d^m \) linear system. Instead, we “succinctly” solve it, by taking advantage of the special structure of the vectors, as we now describe. For \( \alpha \in \mathbb{F}^m \), define the polynomial

\[
\Phi_\alpha(X) := \prod_{i=1}^m \left( 1 + \alpha_i X_i + \alpha_i^2 X_i^2 + \cdots + \alpha_i^{d-1} X_i^{d-1} \right) .
\]

Note that, while the above polynomial is computable via a small arithmetic circuit, its coefficients (once expanded over the monomial basis) correspond to the entries of the vector \( \phi_\alpha \). More generally, for \( \alpha \in \mathbb{F}^{\langle < d \rangle \times m} \), we define the polynomial

\[
\Phi_\alpha(X) := \left( \prod_{i=1}^{\langle |\alpha| \rangle} \left( 1 + \alpha_i X_i + \cdots + \alpha_i^{d-1} X_i^{d-1} \right) \right) \left( \prod_{i=1}^{m-\langle |\alpha| \rangle} \sum_{\gamma \in H} (1 + \gamma X_i) + \cdots + \gamma^{d-1} X_i^{d-1} \right) .
\]
Note that \( \Phi_\alpha \) is a product of univariate polynomials. To see that the above does indeed represent \( \phi_\alpha \), we rearrange the expression as follows:

\[
\Phi_\alpha(X) = \left(\prod_{i=1}^{\lfloor \alpha \rfloor} \left(1 + \alpha_i X_i + \cdots + \alpha_i^{d-1} X_i^{d-1}\right)\right) \left(\sum_{\gamma \in H^{m-|\alpha|}} \prod_{i=1}^{m-|\alpha|} \left(1 + \gamma_i X_{i+|\alpha|} + \cdots + \gamma_i^{d-1} X_{i+|\alpha|}^{d-1}\right)\right)
\]

indeed, the coefficient of \( X^a b \) for \( a \in \langle d \rangle^{\lfloor \alpha \rfloor} \) and \( b \in \langle d \rangle^{m-|\alpha|} \) is \( \alpha^a \sum_{\gamma \in H^{m-|\alpha|}} \gamma^b \), as required.

Thus, to determine whether \( \phi_\alpha \in \text{span(} \phi_{\alpha_1}, \ldots, \phi_{\alpha_\ell} \text{)}, \) it suffices to determine whether \( \Phi_\alpha \in \text{span(} \Phi_{\alpha_1}, \ldots, \Phi_{\alpha_\ell} \text{)} \). In fact, the linear dependencies are in correspondence: for \( a_1, \ldots, a_\ell \in F, \phi_{\alpha} = \sum_{i=1}^{\ell} a_i \phi_{\alpha_i} \) if and only if \( \Phi_\alpha = \sum_{i=1}^{\ell} a_i \Phi_{\alpha_i} \).

Crucially, each \( \Phi_{\alpha_\ell} \) is not only in \( F^{<d}[X_1, \ldots, X_m] \) but is a product of \( m \) univariate polynomials each represented via an \( F \)-arithmetic circuit of size \( \text{poly}(|H| + d) \).

We leverage this special structure and solve the above problem by relying on an algorithm of [RS05] that computes the nullspace for such polynomials (see also [Kay10]), as captured by the lemma below;\(^{13}\) for completeness, we provide an elementary proof of the lemma in the full version.

**Lemma 3.** There exists a deterministic algorithm \( D \) such that, on input a vector of \( m \)-variate polynomials \( Q = (Q_1, \ldots, Q_\ell) \) over \( F \) where each polynomial has the form \( Q_k(X) = \prod_{i=1}^{m} Q_{k,i}(X_i) \) and each \( Q_{k,i} \) is univariate of degree less than \( d \) with \( d \leq |F| \) and represented via an \( F \)-arithmetic circuit of size \( s \), outputs a basis for the linear space \( Q^\perp := \{ (a_1, \ldots, a_\ell) \in F^\ell : \sum_{k=1}^{\ell} a_k Q_k \equiv 0 \} \). Moreover, \( D \) runs in \( \text{poly}(\log |F| + m + d + s + \ell) \) time.

The above lemma immediately provides a way to construct a constraint detector for \( \Sigma \text{RM} \): given as input an index \( n = (F, m, d, H) \) and a subset \( I \subseteq D(n) \), we construct the arithmetic circuit \( \Phi_\alpha \) for each \( \alpha \in I \), and then run the algorithm \( D \) on vector of circuits \( \langle \Phi_\alpha \rangle_{\alpha \in I} \), and directly output \( D \)'s result. The lemma follows.

### 4.3 Univariate polynomials with BS proximity proofs

We show that evaluations of univariate polynomials concatenated with corresponding BS proximity proofs [BS08] have succinct constraint detection (see Definition 8). Recall that the Reed–Solomon code (see Section 3.4) is not locally testable, but one can test proximity to it with the aid of the quasilinear-size

---

\(^{13}\) One could use polynomial identity testing to solve the above problem in probabilistic polynomial time; see [Kay10, Lemma 8]. However, due to a nonzero probability of error, this suffices only to achieve statistical zero knowledge, but does not suffice to achieve perfect zero knowledge.
proximity proofs of Ben-Sasson and Sudan [BS08]. These latter apply when low-degree univariate polynomials are evaluated over linear spaces, so from now on we restrict our attention to Reed-Solomon codes of this form. More precisely, we consider Reed–Solomon codes \( \text{RS}[F, L, d] \) where \( F \) is an extension field of a base field \( \mathbb{K} \), \( L \) is a \( \mathbb{K} \)-linear subspace in \( F \), and \( d = |L| \cdot |\mathbb{K}|^{-\mu} \) for some \( \mu \in \mathbb{N}^+ \). We then denote by \( \text{BS-RS}[\mathbb{K}, F, L, \mu, k] \) the code obtained by concatenating codewords in \( \text{RS}[F, L, [L] \cdot |\mathbb{K}|^{-\mu}] \) with corresponding BS proximity proofs whose recursion terminates at “base dimension” \( k \in \{1, \ldots, \dim(L)\} \) (for a formal definition of these, see the full version); typically \( \mathbb{K}, \mu, k \) are fixed to certain constants (e.g., [BS08] fixes them to \( \mathbb{F}_2, 3, 1 \), respectively) but below we state our result about constraint detection in full generality. The linear code family BS-RS is indexed by tuples \( n = (\mathbb{K}, F, L, \mu, k) \) and the \( n \)-th code is \( \text{BS-RS}[\mathbb{K}, F, L, \mu, k] \), and our result about BS-RS is the following:

**Theorem 6** (formal statement of 4). BS-RS has \( \text{poly}(|F| + \dim(L) + |\mathbb{K}|^\mu + \ell) \)-time constraint detection.

The proof of the above theorem is technically involved, and we refer the reader to the full version for details.

The role of code covers. We are interested in succinct constraint detection: solving the constraint detection problem for certain code families with exponentially-large domains (such as BS-RS). We now build some intuition about how code covers can, in some cases, facilitate this.

Consider the simple case where the code \( C \subseteq F^D \) is a direct sum of many small codes: there exists \( S = \{(\hat{D}_j, \hat{C}_j)\}_j \) such that \( D = \bigcup_j \hat{D}_j \) and \( C = \bigoplus_j \hat{C}_j \) where, for each \( j \), \( \hat{C}_j \) is a linear code in \( F^{D_j} \) and the subdomain \( \hat{D}_j \) is small and disjoint from other subdomains. The detection problem for this case can be solved efficiently: use the generic approach of Gaussian elimination independently on each subdomain \( \hat{D}_j \).

Next consider a more general case where the subdomains are not necessarily disjoint: there exists \( S = \{(D_j, C_j)\}_j \) as above but we do not require that the \( \hat{D}_j \) form a partition of \( D \); we say that each \( (\hat{D}_j, \hat{C}_j) \) is a local view of \( C \) because \( \hat{D}_j \subseteq D \) and \( \hat{C}_j = C|\hat{D}_j \), and we say that \( S \) is a code cover of \( C \). Now suppose that for each \( j \) there exists an efficient constraint detector for \( \hat{C}_j \) (which is defined on \( \hat{D}_j \)); in this case, the detection problem can be solved efficiently at least for those subsets \( I \) that are contained in \( \hat{D}_j \) for some \( j \). Generalizing further, we see that we can efficiently solve constraint detection for a code \( C \) if there is a cover \( S = \{(D_j, C_j)\}_j \) such that, given a subset \( I \subseteq D \), (i) \( I \) is contained in some subdomain \( D_j \), and (ii) constraint detection for \( C_j \) can be solved efficiently.

We build on the above ideas to derive analogous statements for recursive code covers, which arise naturally in the case of BS-RS. But note that recursive constructions are common in the PCP literature, and we believe that our cover-based techniques are of independent interest as, e.g., they are applicable to other PCPs, including [BFLS91, AS98].
Acknowledgements

Work of E. Ben-Sasson, A. Gabizon, and M. Riabzev was supported by the Israel Science Foundation (grant 1501/14). Work of A. Chiesa and N. Spooner was partially supported in part by the UC Berkeley Center for Long-Term Cybersecurity. Work of M. A. Forbes was supported by the NSF, including NSF CCF-1617580, and the DARPA Safeware program; it was also partially completed when the author was at Princeton University, supported by the Princeton Center for Theoretical Computer Science.

References


Zero Knowledge Protocols from Succinct Constraint Detection


Interactive Oracle Proofs with Constant Rate and Query Complexity∗†

Eli Ben-Sasson†, Alessandro Chiesa2, Ariel Gabizon‡, Michael Riabzev4, and Nicholas Spooner5

1 Technion, Haifa, Israel
eli@cs.technion.ac.il
2 University of California, Berkeley, CA, USA
alexch@berkeley.edu
3 Zerocoin Electronic Coin Company (Zcash), Boulder, CO, USA
ariel@z.cash
4 Technion, Haifa, Israel
mriabzev@cs.technion.ac.il
5 University of Toronto, Toronto, Canada
spooner@cs.toronto.edu

Abstract

We study interactive oracle proofs (IOPs) [7, 43], which combine aspects of probabilistically checkable proofs (PCPs) and interactive proofs (IPs). We present IOP constructions and techniques that let us achieve tradeoffs in proof length versus query complexity that are not known to be achievable via PCPs or IPs alone. Our main results are:

1. Circuit satisfiability has 3-round IOPs with linear proof length (counted in bits) and constant query complexity.
2. Reed–Solomon codes have 2-round IOPs of proximity with linear proof length and constant query complexity.
3. Tensor product codes have 1-round IOPs of proximity with sublinear proof length and constant query complexity. (A familiar example of a tensor product code is the Reed–Muller code with a bound on individual degrees.)

For all the above, known PCP constructions give quasilinear proof length and constant query complexity [12, 16]. Also, for circuit satisfiability, [10] obtain PCPs with linear proof length but sublinear (and super-constant) query complexity. As in [10], we rely on algebraic-geometry codes to obtain our first result; but, unlike that work, our use of such codes is much “lighter” because we do not rely on any automorphisms of the code.

We obtain our results by building “IOP-analogues” of tools underlying numerous IPs and PCPs:

Interactive proof composition. Proof composition [3] is used to reduce the query complexity of PCP verifiers, at the cost of increasing proof length by an additive factor that is exponential in the verifier’s randomness complexity. We prove a composition theorem for IOPs where this additive factor is linear.

Sublinear sumcheck. The sumcheck protocol [34, 46] is an IP that enables the verifier to check the sum of values of a low-degree multi-variate polynomial on an exponentially-large hypercube, but the verifier’s running time depends linearly on the bound on individual degrees. We prove a sumcheck protocol for IOPs where this dependence is sublinear (e.g., polylogarithmic).

∗A full version of the paper is available at https://eprint.iacr.org/2016/324.
†This work has received funding from the Israel Science Foundation (grant 1501/14), the UC Berkeley Center for Long-Term Cybersecurity, and the United States – Israel Binational Science Foundation (grant 2021036).
‡This work was done while Ariel Gabizon was at Technion and visiting UC Berkeley.
Our work demonstrates that even constant-round IOPs are more efficient than known PCPs and IPs.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases probabilistically checkable proofs, interactive proofs, proof composition, sumcheck

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.40

1 Introduction

We study Interactive Oracle Proofs (also known as Probabilistically Checkable Interactive Proofs) [7, 43], which combine aspects of probabilistically checkable proofs (PCPs) and interactive proofs (IPs). We present IOP constructions and general techniques that enable us to obtain tradeoffs in proof length versus query complexity that are not known to be achievable by either PCPs or IPs alone. For some applications (e.g., constructing non-interactive arguments in the random oracle model [7]) considering such general types of proof systems suffices (as opposed to focusing only on PCPs or IPs) and thus these applications inherit the efficiency improvements over PCPs.

1.1 Motivation

Probabilistically checkable proofs (PCPs) were introduced by [22, 5, 20, 3, 2]: in a PCP, a probabilistic polynomial-time verifier has oracle access to the proof string. The complexity class $\text{PCP}[r, q]$ denotes those languages for which the verifier uses at most $r$ random bits and queries at most $q$ proof locations; the proof length is then at most $2^r$. The PCP Theorem [3, 2] states that $\text{NP} = \text{PCP}[O(\log n), O(1)]$: every NP statement has a proof of polynomial length that can be verified via a constant number of queries (say, with soundness error $1/2$).

A fundamental question is how long a PCP needs to be, compared to the corresponding “standard” NP proof. Given $T: \mathbb{N} \to \mathbb{N}$, the PCP Theorem states that every language $L$ in $\text{NTIME}(T)$ has a proof of length $\text{poly}(T(n))$ that can be verified with $O(1)$ queries. A sequence of works [42, 30, 24, 13, 9, 12, 16] gradually reduced the proof length to quasilinear, i.e., $T(n) \cdot \text{polylog}(T(n))$; much of this progress was accompanied by progress on efficient reductions from $\text{NTIME}$ to “PCP-friendly” problems, as well as efficient constructions of PCPs of proximity (PCPPs) for key classes of linear codes. Despite much progress, the following question remains open: are there PCPs with linear proof length and constant query complexity?

Ben-Sasson et al. [10] make progress in this direction by proving that there is $a > 0$ such that for every $\epsilon > 0$ there is a PCP for circuit satisfiability with proof length $2^{a/\epsilon} n$ and query complexity $n^\epsilon$. Beyond the sublinear query complexity, [10]'s result comes with other caveats not affecting most prior constructions: the verifier is non-uniform, namely it requires a polynomial-size advice string for every circuit size; and the verifier is not succinct, namely it cannot run in time that is sublinear in the circuit size even if the circuit comes from a uniform circuit family. (Recent constructions of high-rate locally testable codes with sub-polynomial query complexity [32] are not yet known to be convertible to PCPs with similar parameters.)

In this paper, we continue the study of the tradeoff between proof length and query complexity, but we do so for a natural extension of the PCP model (sufficient for some useful
applications, e.g., [7]) that can be thought of as a "multi-round PCP", described below. Also, from this point onwards, we switch to using relations instead of languages. We denote by $\mathcal{R}$ a relation consisting of pairs $(x, w)$, where $x$ is the instance and $w$ is the witness; we think of $\mathcal{R}$ naturally induced by a non-deterministic language $\mathcal{L}$. We denote by $\mathcal{R}|_x$ the (possibly empty) set of witnesses for a given instance $x$, and by $n$ the size of $x$.

### 1.2 A more general model: interactive oracle proofs

**Interactive Oracle Proofs (IOPs)** are a type of proof system introduced in [7, 43] that combines aspects of IPs [4, 26] and PCPs [5, 3, 2], and generalizes interactive PCPs [31]. IOPs naturally extend the notion of a PCP to multiple rounds or, viewed from another angle, they naturally extend the notion of an IP by allowing probabilistic checking. Prior work shows that IOPs can be used to construct non-interactive proofs in the random oracle model [7], that IOPs efficiently achieve unconditional zero knowledge [6], and that IOPs can be used to obtain doubly-efficient constant-round IPs for polynomial-time bounded-space computations [43].

Informally, an IOP extends an IP as follows: whenever the prover sends to the verifier a message, the verifier does not have to read the message in full but may probabilistically query it. In more detail, a $k$-round IOP comprises $k$ rounds of interaction. In the $i$-th round of interaction: the verifier sends a message $m_i$ to the prover; then the prover replies with a message $f_i$ to the verifier, which the verifier can query in this and all later rounds (by having oracle access to it). After the $k$ rounds of interaction, the verifier either accepts or rejects.

An **IOP system** for a relation $\mathcal{R}$ with round complexity $k$ and soundness $\varepsilon$ is a pair $(P, V)$, where $P, V$ are probabilistic algorithms, that satisfies natural notions of completeness and soundness: for every instance-witness pair $(x, w)$ in $\mathcal{R}$, $V(x)$ always accepts after $k(n)$ rounds of interaction with $P(x, w)$; and, for every instance $x$ with $\mathcal{R}|_x = \emptyset$ and unbounded prover $\tilde{P}$, $V(x)$ accepts with probability at most $\varepsilon(n)$ after $k(n)$ rounds of interaction with $\tilde{P}$.

Like the IP model, one efficiency measure is the round complexity $k$. Like the PCP model, two additional efficiency measures are the **proof length** $l$, which is the total number of alphabet symbols in all of the prover’s messages, and the **query complexity** $q$, which is the total number of locations queried by the verifier across all of the prover’s messages. Considering all of these parameters, we say that a relation $\mathcal{R}$ belongs to the complexity class $\text{IOP}[k, a, l, r, q, \varepsilon]$ if there is an IOP system for $\mathcal{R}$ in which on instances of size $n$:

1. the number of rounds is $k(n)$;
2. the prover messages are over the alphabet $a(n)$;
3. the proof length over this alphabet is $l(n)$;
4. the verifier uses $r(n)$ random bits;
5. the verifier queries the prover messages in $q(n)$ locations;
6. the soundness error is $\varepsilon(n)$.

Many other definitions for IPs and PCPs carry over naturally. An IOP is **public coin** if $m_i$ is a random string and the verifier postpones any oracle queries until after receiving all the oracles from the prover (i.e., after the $k$-th round of interaction). An IOP is **non-adaptive** if the query locations do not depend on answers to any previous queries.

**Prior work on IOPs.** In prior work, [7] prove that public-coin IOPs can be compiled into non-interactive proofs in the random oracle model; their compiler is as a generalization of the Fiat–Shamir paradigm for public-coin IPs [21, 41], and of the “CS proof” constructions of Micali [37] and Valiant [49] for PCPs. Also, [6] construct 2-round IOPs (called “duplex
PCPs’ there) with unconditional zero knowledge and quasilinear proof length; in comparison, short PCPs with unconditional zero knowledge are not known. Also, [43] use IOPs to obtain doubly-efficient constant-round IPs for polynomial-time bounded-space computations. In this paper, we do not study compilers for cryptographic proofs, nor zero knowledge, nor applications to interactive proofs; instead, we focus on tradeoffs of proof length versus query complexity for IOPs.

Prior work on interactive PCPs. An interactive PCP [31] is a PCP followed by a standard IP; in particular, it is an IOP where the verifier sends an empty first message and may query only the first prover message (but must read any other prover messages in full). Prior work on interactive PCPs obtains proof length that depends on the witness size rather than computation size [31, 25], as well as unconditional zero knowledge [28]. In this paper we also study proof length but our results do not seem to extend to the more restricted setting of interactive PCPs.

1.3 Proximity and robustness

To facilitate upcoming technical discussions we briefly introduce two notions that strengthen a PCP.

- **PCPs of proximity (PCPPs)** [17, 9]. On the one hand, a PCP verifier has oracle access to a candidate proof $\pi$ and only decides if $R|_x \neq \emptyset (x \in L)$ or $R|_x = \emptyset (x \notin L)$. On the other hand, a PCPP verifier has oracle access to a candidate witness $w$ and proof $\pi$ and decides if $w \in R|_x$ or $w$ is far from $R|_x$ (in particular, if $R|_x = \emptyset$, then $w$ is far from $R|_x$). A quantity $\delta$ known as the proximity parameter specifies what “far” means: if $w$ is $\delta$-far from $R|_x$ then the PCPP verifier accepts with probability at most $\epsilon$, where $\epsilon$ is the soundness error.

- **Robust PCPs** [9]. When $R|_x = \emptyset$, the answers to the verifier’s queries are, with high probability, far from any answers that make the verifier accept. A quantity $\rho$ known as the robustness parameter specifies what “far” means: if $R|_x = \emptyset$ then, with probability at least $1 - \epsilon$, the answers are $\rho$-far from accepting ones.

The two above notions can also be combined, yielding the definition of a robust PCP of proximity.

Extension to IOPs. The notions of proximity and robustness naturally extend to IOPs; see the full version for details. For example, we say that an IOP has proximity parameter $\delta$ if the analogous property for PCPs of proximity holds; we can then correspondingly define the complexity class $IOPP[k,a,l,r,q,\epsilon,\delta]$.

2 Results

We obtain several IOP constructions with proof length and query complexity that are not known to be achievable either via PCPs or IPs alone (or even via interactive PCPs [31]). First, we show that for circuit satisfaction we can obtain IOPs with linear proof length and constant query complexity; constant round complexity and public coins suffice.

➤ **Theorem 1 (informal).** Let $R$ be the relation consisting of instance-witness pairs $(\phi, w)$ where $\phi$ is a boolean circuit (of two-input NAND gates) and $w$ is a binary input that satisfies $\phi$; we use $n$ to denote the number of gates in $\phi$. There exists $a > 0$ and a public-coin IOP...
system that puts $R$ in the complexity class

$$\begin{array}{|l|c|}
\hline
\text{rounds} & k(n) = 3 \\
\text{answer alphabet} & a(n) = \mathbb{F}_2 \\
\text{proof length} & l(n) = a \cdot n \\
\text{query complexity} & q(n) = a \\
\text{soundness error} & \varepsilon(n) = 1/2 \\
\hline
\end{array}$$

In particular, via [40]'s reduction from Turing machines to circuits, we deduce that

$$\begin{array}{|l|c|}
\hline
\text{rounds} & k(T) = 3 \\
\text{answer alphabet} & a(T) = \mathbb{F}_2 \\
\text{proof length} & l(T) = a \cdot T \log T \\
\text{query complexity} & q(T) = a \\
\text{soundness error} & \varepsilon(T) = 1/2 \\
\hline
\end{array}$$

The main points of comparison of the above theorem with prior work are the following.

- For PCPs with constant query complexity, prior work achieved only quasilinear proof length [12, 16], with the “quasilinear” hiding several logarithmic factors. In comparison, we achieve linear proof length for circuit satisfiability, and $O(T \log T)$ proof length for nondeterministic $T$-time relations.

- Ben-Sasson et al. [10] show that there is $a > 0$ such that for every $\epsilon > 0$ there is a non-uniform PCP for circuit satisfiability with proof length $2^{\epsilon/\epsilon n}$ and query complexity $n^\epsilon$; the non-uniformity comes from the use of algebraic-geometry (AG) codes with transitive automorphism groups, for which uniform families are not known. In comparison, we simultaneously achieve linear proof length and constant query complexity; moreover, we make a much “lighter” use of AG codes, which also allows us to avoid non-uniformity. Namely, we rely only on the multiplication properties of AG codes [14, 36], and do not rely on any code automorphisms. Looking ahead, this is because we do not route circuits on Cayley graphs induced by the automorphisms of the underlying code, unlike [10].

Second, we show that Reed–Solomon codes over binary fields (fields of characteristic 2) have 2-round IOPs of proximity with linear proof length and constant query complexity. Such codes are a key ingredient for constructing PCPs with quasilinear proof length [12]. Recall that a word $w: D \rightarrow \mathbb{F}$ is represented via $|w| = |D| \cdot \log |\mathbb{F}|$ bits.

**Theorem 2 (informal).** Given a “fractional degree” $\alpha \in (0, 1)$, define $R$ to be the relation consisting of instance-witness pairs $(\mathbb{F}_{\alpha^2}, d, w)$ where $d \leq \alpha 2^\lambda$ and $w: \mathbb{F}_{\alpha^2} \rightarrow \mathbb{F}_{\alpha^2}$ is the evaluation of a polynomial of degree less than $d$; we define the instance size to be $\lambda$, and note that $w$ has $|w| = 2^\lambda \cdot \lambda$ bits. For every $\alpha \in (0, \frac{1}{3} (1 - \alpha))$ there exist $a > 0$ and a public-coin IOP of proximity $(P, V)$ that puts $R$ in the complexity class

$$\begin{array}{|l|c|}
\hline
\text{rounds} & k(\alpha) = 2 \\
\text{answer alphabet} & a(\alpha) = \mathbb{F}_2 \\
\text{proof length} & l(\alpha) = a \cdot 2^\lambda \cdot \lambda \\
\text{query complexity} & q(\alpha) = a \\
\text{soundness error} & \varepsilon(\alpha) = 1/2 \\
\hline
\end{array}$$

More generally, our result concerns additive Reed–Solomon codes, where the domain of a codeword is a $\lambda$-dimensional affine subspace $S$ of a potentially larger binary field $\mathbb{F}$; in such cases the above statement involves more parameters but achieves the same asymptotics. The
Interactive Oracle Proofs with Constant Rate and Query Complexity

The main point of comparison of the above theorem with prior work is \cite{12, 16}, who achieve PCPs of proximity with the same parameters but superlinear proof length: \( a \cdot 2^\lambda \cdot \lambda \cdot \text{poly}(\lambda) \).

Third, we show that tensor product codes have 1-round IOPs of proximity with sublinear proof length and constant query complexity. Given a positive integer \( m \) and linear code \( C \) with domain \( D \) and alphabet \( \mathbb{F} \), the tensor product code \( C^{\otimes m} \) is the linear code that comprises all functions \( w: D^m \rightarrow \mathbb{F} \) whose restriction to any axis-parallel line is in \( C \); the message length, block length, and distance of \( C^{\otimes m} \) are each the \( m \)-th power of the corresponding parameters of \( C \). Tensor product codes are a large family, and they include Reed–Muller codes (at least when considering the definition that bounds the variables’ individual degrees, which we do, as opposed to the one that bounds their sum).

\textbf{Theorem 3 (informal).} Let \( m \geq 3 \) and \( C \) be a linear code with domain \( D \), alphabet \( \mathbb{F} \), and relative distance \( \tau \); let \( \ell := |D| \) be the block length. Define \( \mathcal{R} \) to be the relation of instance-witness pairs \((C,m,w)\) such that \( w \in C^{\otimes m} \); note that \( w \) has \( |w| = \ell^m \cdot \log |\mathbb{F}| \) bits. For every \( \delta \in (0, \frac{1}{2^m}) \) there exist \( a > 0 \) and a public-coin IOPP system \((P,V)\) that puts \( \mathcal{R} \) in the complexity class

\[
\begin{array}{c|c}
\text{rounds} & k(\ell^m) = 1 \\
\text{answer alphabet} & a(\ell^m) = \mathbb{F}_2 \\
\text{proof length} & l(\ell^m) = o(\ell^m \cdot \log |\mathbb{F}|) \\
\text{query complexity} & q(\ell^m) = a \\
\text{soundness error} & \varepsilon(\ell^m) = 1/2 \\
\text{proximity parameter} & \delta(\ell^m) = \delta \\
\end{array}
\]

The main points of comparison of the above theorem with prior work are the following.

- Ben-Sasson and Sudan \cite{11} and Viderman \cite{51} give local testers for all tensor product codes with query complexity \( q(\ell^m) = \ell^2 \); Dinur et al. \cite{18} give local testers with \( q(\ell^m) = \ell \) for certain tensor product codes. In contrast, we achieve constant query complexity, with only sublinear proof length, for all tensor product codes. Moreover, given additional mild conditions, we obtain constant soundness error even for non-constant \( m \).

- The work of \cite{12, 16} implies PCPs of proximity for tensor product codes with superlinear proof length and constant query complexity. In contrast, we obtain sublinear proof length, with a single round of interaction.

Analogously to \cite{51}, we can invoke Theorem 3 on different choices of linear codes so to derive different code families that have good properties and an IOP tester (instead of a local tester as in \cite{51}). For example, we can choose a family of linear codes with arbitrarily high rate, constant relative distance, linear-time encoding, and linear-time decoding from a constant fraction of errors \cite{48, 29, 44}; our theorem implies a code with the same properties that also has a 1-round IOP of proximity with sublinear proof length and constant query complexity (cf. \cite[Section 3.1]{51}).

Similar statements hold for list-decodable codes with good parameters \cite{27} (cf. \cite[Section 3.2]{51}); and also for locally correctable and, more generally, locally decodable codes with good parameters \cite{52, 50, 19, 33, 32} (cf. \cite[Section 3.3]{51}). In each of these cases, the tensor product operation preserves the “key” properties of the choice of underlying code \( C \), while endowing the resulting code with an IOP of proximity.

We obtain the above results via techniques of independent interest: we prove that, in the IOP model, there are more efficient analogues of tools that are fundamental to constructing PCPs and IPs. We now discuss these techniques.
3 Techniques

Recall that IOPs generalize both IPs, by treating the prover’s messages as oracle strings, and PCPs, by allowing for multiple rounds of interaction; they also generalize interactive PCPs [31]. We prove that IOPs can express two fundamental techniques in a more efficient way than in these prior models:

(i) in interactive proof composition, the prover is more efficient than in PCP proof composition; and

(ii) in sublinear sumchecks, the verifier is more efficient than in IP sumcheck protocols.

We now discuss both of our new tools, and then how we use them.

3.1 Interactive proof composition

Proof composition [3] is used to reduce PCP query complexity, cf. [2, 30, 9]; it involves two PCPs: an outer one and an inner one. One should think of the outer proof system as having short proofs but large query complexity, while the inner proof system has long proofs but small query complexity.

The composed prover uses the outer prover to send a PCP to the composed verifier, who does not run the outer verifier but, instead, uses the inner verifier to check that the outer verifier would have accepted had it made its queries to the PCP. The composed verifier also needs an auxiliary sub-PCP for the claim that the outer verifier would have accepted; in fact, he needs one sub-PCP for each possible random string of the outer verifier. Hence, the composed prover also sends all of these sub-PCPs along with the first PCP. The benefit is that the query complexity of the composed verifier equals that of the inner verifier, which is typically verifying a much smaller statement than the outer verifier.

Beyond query complexity, most other parameters of the composed proof system are simply the sum of corresponding parameters of the outer and inner proof systems. An exception is the proof length $l$: it does not simply equal the sum $l_{\text{out}} + l_{\text{in}}$, but instead equals $l_{\text{out}} + 2^{\text{out}} \cdot l_{\text{in}}$, because the composed prover uses the inner proof system to generate a proof for each choice of randomness of the outer proof system. (The same is true for prover running time.)

We prove an Interactive Proof Composition Theorem that avoids the above limitations. The outer proof system is a robust PCP $(P_{\text{out}}, V_{\text{out}})$ for a relation $R$, while the inner one is a $k$-round IOP $(P_{\text{in}}, V_{\text{in}})$ for $V_{\text{out}}$’s relation; the composed proof system is a $(k + 1)$-round IOP $(P, V)$ for $R$. The parameters of the composed proof system are exactly as before, except that now the new proof length is much smaller: $l_{\text{out}} + l_{\text{in}}$. (Ditto for the prover running time.) The crucial observation is that, after the prover sends the outer proof to the verifier, soundness is not harmed if the verifier tells the prover his choice of outer randomness; hence, the prover does not have to invest work for all randomness choices but can simply invest work only for the outer randomness that was chosen, which he now knows.
Interactive Oracle Proofs with Constant Rate and Query Complexity

Theorem 4 (Interactive Proof Composition – informal). Suppose that the relation \( \mathcal{R} \) satisfies the following:

(1) there exists a robust PCPP system \((P_{\text{out}}, V_{\text{out}})\) that puts \( \mathcal{R} \) in the complexity class

- \( \text{proof length} \) \( l_{\text{out}} \)
- \( \text{randomness} \) \( r_{\text{out}} \)
- \( \text{query complexity} \) \( q_{\text{out}} \)
- \( \text{soundness error} \) \( \varepsilon_{\text{out}} \)
- \( \text{robustness parameter} \) \( \rho_{\text{out}} \)

(2) for every \( \mathcal{X} \) there exists an IOPP system \((P_{\text{in}}, V_{\text{in}})\) that puts \( \mathcal{R}_{\text{out}} \)’s relation in the complexity class

- \( \text{rounds} \) \( k_{\text{in}} \)
- \( \text{proof length} \) \( l_{\text{in}} \)
- \( \text{randomness} \) \( r_{\text{in}} \)
- \( \text{query complexity} \) \( q_{\text{in}} \)
- \( \text{soundness error} \) \( \varepsilon_{\text{in}} \)
- \( \text{proximity parameter} \) \( \delta_{\text{in}} \)

If \( \delta_{\text{in}} \leq \rho_{\text{out}} \) then there exists an IOPP system \((P, V)\) that puts \( \mathcal{R} \) in the complexity class

- \( \text{rounds} \) \( k \)
- \( \text{proof length} \) \( l = l_{\text{out}} + l_{\text{in}} \)
- \( \text{randomness} \) \( r = r_{\text{out}} + r_{\text{in}} \)
- \( \text{query complexity} \) \( q = q_{\text{out}} + q_{\text{in}} \)
- \( \text{soundness error} \) \( \varepsilon = \varepsilon_{\text{out}} + \varepsilon_{\text{in}} \)
- \( \text{proximity parameter} \) \( \delta = \delta_{\text{out}} \)

The above discussion and informal theorem statement omit many technical details that already arise in non-interactive proof composition (e.g., see lengthy discussions in [9, 8]), and we also need to deal with. For instance, one has to clarify the size of the sub-claim on which the inner proof system is invoked; also, one has to carefully define the notion of a verifier to allow for the composed verifier’s running time to be smaller than the outer verifier’s query complexity. For more details, see the full version.

3.2 Sublinear sumcheck

The sumcheck protocol [34, 46] is an interactive proof for the claim \( \sum_{\mathbf{\alpha} \in H} w(\mathbf{a}) = 0 \), where \( w \) is the evaluation on \( F^m \) of an \( m \)-variate polynomial of individual degree \( d \) and \( H \) is a subset of \( F \). More generally, \( w \) may be a codeword in the tensor product code \( C^\otimes_m \), for a given linear code \( C \) with domain \( D \) and alphabet \( F \), and \( H \) is a subset of \( D \) [36]. The prover receives \( H \) and \( w \) as input, while the verifier receives \( H \) as input and \( w \) as an oracle. The protocol has \( m \) rounds and, if \( C \) has relative distance \( \tau \), the protocol has soundness error \( 1 - \tau^m \); also, the prover runs in time \( \text{poly}(\ell^m) \), and the verifier in time \( \text{poly}(\ell + m) \), where \( \ell := |D| \) is \( C \)’s block length.

In each round, the verifier receives a codeword \( w_i \) in \( C \) and checks that \( \sum_{\mathbf{\alpha} \in H} w_i(\mathbf{\alpha}) \) equals a certain value \( \gamma_{i-1} \) determined in the previous round; in particular, the verifier reads \( \Omega(\ell) \) bits. We show that the verifier complexity can be sublinear in \( \ell \), if the prover and verifier engage in an IOP instead of an IP. The intuition to “go sublinear” is simple: instead of doing these checks explicitly, the verifier uses proximity testers for doing so. Thus, in each round, the prover sends to the verifier two oracles: the codeword in \( w_i \), and a proximity proof attesting that \( w_i \in C \) and that \( \sum_{\mathbf{\alpha} \in H} w_i(\mathbf{\alpha}) = \gamma_{i-1} \). The use of proximity proofs complicates the soundness analysis because the verifier only sees noisy codewords, but the backbone of the proof follows that of the standard sumcheck protocol. Overall, we obtain a sumcheck IOP protocol that enables a verifier to efficiently check sumchecks for codes of much larger blocklength than what he can afford in the standard sumcheck protocol.

We state our Sublinear Sumcheck Theorem below as a reduction: given a PCP of proximity \((P_{SC}, V_{SC})\) for subcodes of the form \( C_{H,\gamma} := \{ w \in C \text{ s.t. } \sum_{\mathbf{\alpha} \in H} w(\mathbf{\alpha}) = \gamma \} \), we...
construct an IOP of proximity \((P, V)\) for sumchecks over \(H^m\) for \(C^\otimes m\). The complexity of the PCPP verifier \(V_{SC}\) determines the complexity of the resulting IOPP verifier \(V\); e.g., if the former is sublinear in \(C\)'s block length \(\ell\), so is the latter.

**Theorem 5 (Sublinear Sumcheck – informal).** Let \(m\) be a positive integer, and \(C\) a linear code with relative distance \(\tau\) and block length \(\ell\). Suppose that there is a PCP of proximity for subcodes of the form \(C|_H, \gamma := \{w \in C \text{ s.t. } \sum_{\alpha \in H} w(\alpha) = \gamma\}\) with proof length \(l_{SC}\), query complexity \(q_{SC}\), soundness error \(\varepsilon_{SC}\), proximity parameter \(\delta_{SC}\), prover running time \(t_{pSC}\), and verifier running time \(t_{vSC}\). Then there is a public-coin IOP for sumchecks over \(H^m\) for \(C^\otimes m\) with the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rounds</td>
<td>(m)</td>
</tr>
<tr>
<td>proof length</td>
<td>(m \cdot l_{SC} + m \cdot \ell)</td>
</tr>
<tr>
<td>query complexity</td>
<td>(m \cdot q_{SC} + m + 1)</td>
</tr>
<tr>
<td>soundness error</td>
<td>(1 - \tau^m + (\varepsilon_{SC} + m \cdot \delta_{SC}))</td>
</tr>
<tr>
<td>prover time</td>
<td>(m \cdot t_{pSC} + m \cdot \ell^m)</td>
</tr>
<tr>
<td>verifier time</td>
<td>(m \cdot t_{vSC} + O(m))</td>
</tr>
</tbody>
</table>

In later sections, it is more natural to state the theorem without assuming that \(w\) is a codeword in \(C^\otimes m\), so the reduction also takes as input a PCP of proximity \((P_{\otimes}, V_{\otimes})\) for \(C^\otimes m\) that is invoked on \(w\); this introduces additional terms in the parameters. More generally, both of the PCPs of proximity \((P_{SC}, V_{SC})\) and \((P_{\otimes}, V_{\otimes})\) can in fact be IOPs of proximity, and we state our theorem for this more general case, which we need. For more details, see the full version.

### 3.3 Applying the new tools

We now sketch how we use the above new tools to derive the results of Section 2. We begin by discussing our results on proximity testing to codes (stated later); we then turn to circuit satisfiability (stated earlier) because its proof requires one of these results on proximity testing.

**Intuition behind Theorem 2.** The construction of linear-size IOPs of proximity for Reed–Solomon codes over binary fields follows from one invocation of our Interactive Proof Composition Theorem with [12]’s robust PCPs of proximity for Reed–Solomon codes as the outer proof system, and [38]’s PCPs of proximity for nondeterministic languages as the inner proof system. Informally, in the first round, the prover sends to the verifier a [12] PCP of proximity, which reduces proximity testing of codewords over \(F_{2^\lambda}\) to proximity testing of sub-codewords over \(F_{2^{\lambda/2} + O(1)}\) with only constant overheads; in the second round, the verifier sends his choice of outer randomness, and the prover replies with a [38] PCP of proximity for the sub-codeword. The proof length of this latter component is quasilinear, but is applied to a claim of “square-root size” only, so we obtain linear proof length.

**Intuition behind Theorem 3.** The construction of sublinear-size IOPs of proximity for tensor product codes follows from one invocation of our Interactive Proof Composition Theorem with [11, 51]’s robust local tester for tensor product codes as the outer proof system, and [38]’s PCPs of proximity for nondeterministic languages as the inner proof system. Unlike before, we now use one round, because the outer proof system only relies on a local tester rather than a PCP of proximity. The verifier thus simply sends his choice of outer randomness, and the prover replies with a [38] PCP of proximity for a suitable sublinear-size
Interactive Oracle Proofs with Constant Rate and Query Complexity

sub-codeword. Since the proof length of this latter component is quasilinear but is applied to a sublinear-size claim, we obtain sublinear proof length.

**A summary:** overall, we can summarize the above sketches via the following diagram of implications.

<table>
<thead>
<tr>
<th>Theorem 2</th>
<th>Theorem 4</th>
<th>[12]</th>
<th>[38]</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear-size IOPP for Reed-Solomon codes</td>
<td>interactive proof composition</td>
<td>robust PCPs of proximity for Reed-Solomon codes</td>
<td>PCP of proximity for NTIME</td>
</tr>
<tr>
<td>Theorem 3</td>
<td>Theorem 4</td>
<td>[11, 51]</td>
<td>[38]</td>
</tr>
<tr>
<td>sublinear-size IOPP for tensor product codes</td>
<td>interactive proof composition</td>
<td>robust local testing for tensor product codes</td>
<td>PCP of proximity for NTIME</td>
</tr>
</tbody>
</table>

**Intuition behind Theorem 1.** We now turn to how to construct 3-round IOPs for circuit satisfiability with linear proof length and constant query complexity.

The first step of many PCP constructions is to arithmetize the NP statement at hand (in our case, the satisfiability of a boolean circuit) by reducing it to a “PCP-friendly” problem that looks like a constraint satisfaction problem over a well-chosen graph and whose assignments involve codewords in a well-chosen linear code \( C \). Meir observes in [35, 36] that key features of \( C \) are good relative distance and, moreover, a *multiplication property*: coordinate-wise multiplication of codewords yields codewords in a code whose relative distance is still good [14, 36]. Moreover, to obtain short PCPs, the aforementioned graph is typically chosen so to behave like a routing network [42]; for example, [12] use De Bruijn graphs, while [10] use hypercubes. To support such graphs, the automorphism group of \( C \) has to be rich enough. This typically holds for Reed–Solomon codes [12] which have a doubly-transitive automorphism group, but is a significantly harder condition to fulfill for AG codes [10], for which obtaining a transitive automorphism group is quite involved and, currently, can only be achieved non-uniformly.

The aforementioned first step would be problematic in our setting, because known routing techniques introduce either logarithmic overheads (as in [12]) or large query complexity (as in [10]), so it is not clear how we could use them. Departing from these prior works, we do not rely on any routing, and instead immediately leverage one round of interaction to directly reduce circuit satisfiability to a sumcheck instance over a given linear code \( C \). Also, we only assume that \( C \) has good relative distance and a multiplication property [14], but we do not rely on any automorphisms.

Informally, the prover first sends three codewords \( w_1, w_2, w_3 \) over a field \( \mathbb{F} \); the first codeword encodes values of the left wires of all gates, the second encodes values for the right wires of all gates, and the third encodes values for the output wires of all gates. (When a gate has fan-out greater than 1 we still consider 1 output wire.) The verifier now must check several things. First, that wire values are boolean and the output gate wire equals 0. Second, that the wire values are “locally consistent” with each gate: for every \( i \in [n] \), \( w_1(i) \) is the NAND of \( w_1(i) \) and \( w_2(i) \). Third, that the three encodings of wire values are consistent with the circuit topology: namely, if \( \ell(i) \) represents the left wire used to compute \( i \), and \( r(i) \) represents the right wire used to compute \( i \), the topology requires that \( w_3(\ell(i)) = w_1(i) \) and \( w_3(r(i)) = w_2(i) \) for every \( i \). The verifier cannot directly conduct these checks (as doing so would incur linear query complexity); instead, the verifier sends some randomness to the prover so to “bundle” the checks into one sumcheck.

But how should the verifier sample randomness to achieve this bundling? One option is to sample a random element in \( \mathbb{F} \) per check so to construct a random subset sum, which can be viewed as an \( n \)-variate polynomial of total degree 1, whose coefficients are the checks,
evaluated at a random point. If not all checks are satisfied, the polynomial is non-zero, and its random evaluation cannot attain any value with too large probability. However, constructing a random subset sum is inefficient because the verifier samples and sends to the prover $\Omega(n)$ random bits, in order to describe the random point. Nevertheless, the verifier may hope to do better by using a different low-degree polynomial for the bundling. In general, if the polynomial has $m$ variables each of degree at most $d$, the verifier must sample and send $m$ field elements; this preserves soundness provided that $|\mathbb{F}| = \Omega(md)$ (for a constant probability of avoiding any particular output value by the Schwartz–Zippel Lemma \cite{Schwartz80,Zippel79}) and $d^m = \Omega(n)$ (to bundle all checks). For example, the univariate case of $m = 1$ was considered in \cite{BCHS17} when reducing to a sumcheck problem; the multivariate case of $m = \log n$ or $m = \frac{\log n}{\log \log n}$ was considered in later works. Unfortunately, either setting does not work for constant-size fields, which we ultimately use to obtain linear proof length.

Taking a step back from polynomials, we see that all we need is an evading set $S$ for $\mathbb{F}^n$, which is a small set such that for any non-zero $v \in \mathbb{F}^n$ the inner product $\langle r, v \rangle$, for random $r \in S$, does not attain any particular value $a \in \mathbb{F}$ with too high probability. Good constructions of evading sets are known: they relax a well-studied notion called $\epsilon$-biased sets \cite{Raz97}. In particular, results of \cite{GR08} imply that, for any $\epsilon$, $\mathbb{F}^n$ has an evading set $S$ of size $\text{poly}(\frac{1}{\epsilon})$ and the aforementioned probability is $\gamma := \epsilon + \frac{1}{|\mathbb{F}|}$; in particular, such a construction is suitable for constant-size fields.

Below we informally state the reduction (see the full version for details), using the following notion: we say that a linear code $C'$ is a degree $d$-closure of $C$ if, for every $w_1, \ldots, w_m \in C$ and $m$-variate polynomial $P$ of total degree at most $d$, it holds that $w' \in C'$ where the $i$-th entry of $w'$ is the evaluation of $P$ on the $i$-th coordinates of $w_1, \ldots, w_m$.

> **Lemma 6** (Circuit SAT to Sumcheck – informal). Let $n$ be a positive integer, $C \subseteq \mathbb{F}^D$ an $n$-systematic linear code, $\phi$ an $n$-gate boolean circuit (of two-input NAND gates), and $S$ an evading set for $\mathbb{F}^n$. There is a 1-round IOP that reduces satisfiability of $\phi$ to proximity testing to $C$ and a sumcheck over any degree-3 closure of $C$. Moreover, the IOP introduces only constant overheads in all relevant parameters, including proof length and query complexity.

After reducing circuit satisfiability to sumcheck over the given code $C$, we are left to choose $C$ so to ensure that the sumcheck can be carried out with 2 additional rounds, linear proof length, and constant query complexity.

For this, our starting point is \cite{BenSassonCGRS17}'s efficient construction of a code family with constant rate, relative distance, and alphabet size. Note that since these codes are AG codes, they have a naturally-defined degree-3 closure. Also, their construction is uniform, and thus represents a much “lighter” use of AG codes as compared to in \cite{BHT02}.

If we simply choose $C$ to be a code from this AG code family, then it is not clear how to efficiently conduct the sumcheck. However, what does work is to take $C$ to be the tensor product of $O(1)$ copies of this AG code. Informally, in this way, we can invoke our Sublinear Sumcheck Theorem (Theorem 5) on the tensor product code $C$ and we can test proximity to it by Theorem 3. See the full version for details.

Overall, we can summarize the above sketch via the following diagram of implications.

\[ \begin{align*}
\text{Lemma 6} & \quad \text{from circuit SAT to sumcheck} \\
\text{Theorem 5} + \quad \text{sublinear sumcheck} \\
\text{Theorem 3} + \quad \text{sublinear-size IOP for tensor product codes} \\
& \quad \text{efficient construction of AG codes} \\
\text{Theorem 1} & \quad \text{linear-size IOP for circuit SAT}
\end{align*} \]
Open questions

The question of whether there exist PCPs with linear proof length and constant query complexity remains open. Nevertheless, our work suggests additional questions that may be stepping stones in this and other intriguing directions:

1. Is there a one-round IOP for circuit satisfiability with linear proof length and query complexity? (Our IOP for circuit satisfiability requires 3 rounds.)
2. Is there an IOP for $\text{NTIME}(T)$ with linear proof length and query complexity, for some number of rounds? (Our results, like [10], only imply proof length $O(T \log T)$.)
3. Is there an IOP for succinct circuit satisfiability with linear proof length and query complexity? (Our results, like [10], “stop” at $\text{NP}$ but do not extend to $\text{NEXP}$.)

Finally, while “positive” applications of IOPs are known (e.g., non-interactive proofs in the random oracle model [7]), “negative” ones are not: do IOP constructions with good parameters imply inapproximability results that are not known to be implied by known PCP constructions?

References

E. Ben-Sasson, A. Chiesa, A. Gabizon, M. Riabzev, and N. Spooner


Interactive Oracle Proofs with Constant Rate and Query Complexity


Fast Reed-Solomon Interactive Oracle Proofs of Proximity

Eli Ben-Sasson
Technion, Haifa, Israel
elis@cs.technion.ac.il
https://orcid.org/0000-0002-0708-0483

Iddo Bentov
Cornell University, Ithaca, NY, USA
iddo333@gmail.com

Yinon Horesh
Technion – Israel Institute of Technology, Haifa, Israel
ynon980@gmail.com

Michael Riabzev
Technion – Israel Institute of Technology, Haifa, Israel
riabzevmichael@gmail.com

Abstract

The family of Reed-Solomon (RS) codes plays a prominent role in the construction of quasilinear probabilistically checkable proofs (PCPs) and interactive oracle proofs (IOPs) with perfect zero knowledge and polylogarithmic verifiers. The large concrete computational complexity required to prove membership in RS codes is one of the biggest obstacles to deploying such PCP/IOP systems in practice.

To advance on this problem we present a new interactive oracle proof of proximity (IOPP) for RS codes; we call it the Fast RS IOPP (FRI) because (i) it resembles the ubiquitous Fast Fourier Transform (FFT) and (ii) the arithmetic complexity of its prover is strictly linear and that of the verifier is strictly logarithmic (in comparison, FFT arithmetic complexity is quasi-linear but not strictly linear). Prior RS IOPPs and PCPs of proximity (PCPPs) required super-linear proving time even for polynomially large query complexity.

For codes of block-length $N$, the arithmetic complexity of the (interactive) FRI prover is less than $6 \cdot N$, while the (interactive) FRI verifier has arithmetic complexity $\leq 21 \cdot \log N$, query complexity $2 \cdot \log N$ and constant soundness – words that are $\delta$-far from the code are rejected with probability $\min\{\delta \cdot (1 - o(1)), \delta_0\}$ where $\delta_0$ is a positive constant that depends mainly on the code rate. The particular combination of query complexity and soundness obtained by FRI is better than that of the quasilinear PCPP of [Ben-Sasson and Sudan, SICOMP 2008], even with the tighter soundness analysis of [Ben-Sasson et al., STOC 2013; ECCC 2016]; consequently, FRI is likely to facilitate better concretely efficient zero knowledge proof and argument systems.

Previous concretely efficient PCPPs and IOPPs suffered a constant multiplicative factor loss in soundness with each round of “proof composition” and thus used at most $O(\log \log N)$ rounds. We show that when $\delta$ is smaller than the unique decoding radius of the code, FRI suffers only a negligible additive loss in soundness. This observation allows us to increase the number of “proof composition” rounds to $O(\log N)$ and thereby reduce prover and verifier running time for fixed soundness.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Interactive proofs, low degree testing, Reed Solomon codes, proximity testing
1 Introduction

The family of Reed-Solomon (RS) codes is a fundamental object of study in algebraic coding theory and theoretical computer science [56]. For an evaluation set $S$ of $N$ elements in a finite field $\mathbb{F}$ and a rate parameter $\rho \in (0, 1]$, the code $\text{RS}[\mathbb{F}, S, \rho]$ is the space of functions $f : S \to \mathbb{F}$ that are evaluations of polynomials of degree $d < \rho N$ [56]. The RS proximity problem assumes a verifier has oracle access to $f : S \to \mathbb{F}$, and asks that verifier to distinguish, with “large” confidence and “small” query complexity, between the case that $f$ is a codeword of $\text{RS}[\mathbb{F}, S, \rho]$ and the case that $f$ is $\delta$-far in relative Hamming distance from all codewords. This problem has been addressed in several different computational models (surveyed next and summarized in Table 1), and is also the focus of this paper.

**RS proximity testing:** When no additional data is provided to the verifier, the RS proximity problem is commonly called a testing problem, and has been first defined and addressed by Rubinfeld and Sudan in [58] (cf. [32]). In this case, one can see that $d + 1$ queries are necessary and sufficient to solve the problem: codewords are accepted by their tester with probability 1 whereas functions that are $\delta$-far from the code are rejected with probability $\geq \delta$. Since no additional information is provided to the verifier in this model, we may say that a prover attempting to convince the verifier that $f \in \text{RS}[\mathbb{F}, S, \rho]$ spends zero computational effort, zero rounds of interaction and produces a proof of length zero.

**RS proximity verification – PCPP model:** Probabilistically checkable proofs of proximity (PCPP) [21, 30] relax the testing problem to a setting in which the verifier is given oracle access also to an auxiliary proof, called a PCPP and denoted $\pi$. This PCPP is produced by the prover, which is given $f \in \text{RS}[\mathbb{F}, S, \rho]$ as input. The time required to produce $\pi$ is the prover complexity and $|\pi|$ is called the proof length\(^1\); similarly, verifier complexity is the total time required to generate queries and check query-answers. The techniques used to prove the celebrated PCP Theorem [2, 3] also show that the proximity problem can be solved with constant query complexity and proof length and prover complexity $N^{O(1)}$, or with proof length $N^{1+\epsilon}$ and query complexity $(\log N)^{O(1/\epsilon)}$ [5]. The current state of the art in the PCPP model gives proofs of length $\tilde{O}(N) \equiv N \cdot \log^{O(1)} N$ with constant query complexity [23, 28] and prover complexity $\tilde{O}(N)$ [16]; verifier complexity is $\text{poly} \log N$ [20, 50].

**RS proximity verification – IOPP model:** Interactive oracle proofs of proximity (IOPP), formally introduced in [13] and, independently, in [57] (under the name “probabilistically checkable interactive proofs of proximity”), generalize IPs, PCPs and interactive PCPs (IPCP) [42]. As in an IP and IPCP, several rounds of interaction are used in which the prover sends messages $\pi_1, \pi_2, \ldots, \pi_r$ in response to successive verifier messages. As in a PCP and

\(^1\) Typically $\pi$ is a sequence of elements in $\mathbb{F}$. Therefore, proof length is measured over the alphabet $\mathbb{F}$. 

Table 1 Comparison of RS proximity protocols. For concreteness, all results are stated for binary additive RS codes with rate $\rho = 1/8$ evaluated over a sufficiently large set $S, |S| = N$ satisfying $N/F < 0.001$ with proximity parameter $\delta < \delta_0$ (cf. Theorem 2) and soundness at least 0.996; i.e., the rejection probability of $\delta$-far words is at least 0.996 for $\delta < \delta_0$ (in particular, smaller $\delta$ leads to smaller soundness). Exponents for the 4th row taken from [16]; the various exponents $c$ in the 5th and 6th row have not been estimated in prior works but are greater than the respective exponents in the 4th row.

<table>
<thead>
<tr>
<th></th>
<th>prover comp.</th>
<th>proof length</th>
<th>verifier comp.</th>
<th>query comp.</th>
<th>round comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Testing [58]</td>
<td>0</td>
<td>0</td>
<td>$O(\rho N)$</td>
<td>$\rho N$</td>
<td>0</td>
</tr>
<tr>
<td>2. PCP [2, 3]</td>
<td>$N^{O(1)}$</td>
<td>$N^{O(1)}$</td>
<td>$N^{O(1)}$</td>
<td>$O \left( \frac{1}{\delta} \right)$</td>
<td>1</td>
</tr>
<tr>
<td>3. PCP [6, 5]</td>
<td>$N^{1+c}$</td>
<td>$N^{1+c}$</td>
<td>$\frac{1}{2} \log^{O(1/c)} N$</td>
<td>$\frac{1}{2} \log^{O(1/c)} N$</td>
<td>1</td>
</tr>
<tr>
<td>4. PCPP [23, 21, 16]</td>
<td>$\geq N \log^{1+c} N$</td>
<td>$\geq N \log^{1+c} N$</td>
<td>$\geq \frac{1}{2} \log^{5.8} N$</td>
<td>$\frac{1}{2} \log^{5.8} N$</td>
<td>1</td>
</tr>
<tr>
<td>5. PCPP [24, 50]</td>
<td>$N \log^{c} N$</td>
<td>$N \log^{c} N$</td>
<td>$\frac{1}{2} \log^{c} N$</td>
<td>$O \left( \frac{1}{\delta} \right)$</td>
<td>1</td>
</tr>
<tr>
<td>6. IOPP [12, 9]</td>
<td>$N \log^{2} N$</td>
<td>$\geq 4 \cdot N$</td>
<td>$\frac{1}{2} \log^{2} N$</td>
<td>$O \left( \frac{1}{\delta} \right)$</td>
<td>$\log \log N$</td>
</tr>
<tr>
<td>7. This work</td>
<td>$&lt; 6 \cdot N$</td>
<td>$&lt; \frac{1}{c}$</td>
<td>$\leq 21 \cdot \log N$</td>
<td>$2 \log N$</td>
<td>$\frac{\log N}{c}$</td>
</tr>
</tbody>
</table>

IOPCP, the verifier is not required to read prover messages in entirety but rather may query them at random locations (in an IPCP, verifier must read the full messages $\pi_2, \ldots$ but may query $\pi_1$ randomly); the query complexity is the total number of entries read from $f$ and $\pi_1, \pi_2, \ldots, \pi_r$. The prover is provided with $f \in \mathbb{R}[F, S, \rho]$ as input and prover complexity is the total time required to produce all (prover) messages, while proof length is generalized from the PCPP setting to the IOPP setting and defined as $|\pi_1| + \ldots + |\pi_r|$. IOPPs can be used to “replace” PCPP proof composition with more rounds of interaction, and thereby reduce proof length and prover complexity without compromising soundness (see Section 1.3). In particular, the IOPP version of the aforementioned PCPP constructions reduces proof length to $O(N)$ with no change to soundness and/or query complexity [8, 13]. In spite of the shorter proof length, prover complexity in prior works was $\Theta(N \text{poly} \log N)$ due to a limitation on the number of proof-composition rounds, explained in Section 2.1.

1.1 Main results

We present a new IOPP for RS codes, called the Fast RS IOPP (FRI) because of its resemblance to the Fast Fourier Transform (FFT) [26]; its analysis relies on the quasi-linear RS-PCPP [23] (see Section 2.1). FRI is the first RS-IOPP to have (i) strictly linear arithmetic complexity for the prover with (ii) strictly logarithmic arithmetic complexity for the verifier and (iii) constant soundness. We start by recalling IOPP systems as described in [12, Section 3.2], after informally summarizing the main complexity parameters of IOPPs (introduced and discussed thoroughly in [19]).

1.1.1 IOP

An Interactive Oracle Proof (IOP) system $S$ is defined by a pair of interactive randomized algorithms $S = (P, V)$, where $P$ denotes the prover and $V$ the verifier. On input $x$ of length $N$, the number of rounds of interaction is denoted by $r(N)$ and called the round complexity of the system. During a single round the prover sends a message to which the verifier is

---

2 Notice that prover complexity does not include the time needed to produce $f$. 
given oracle access, and the verifier responds with a message to the prover. The proof length, denoted $\ell(N)$, is the sum of lengths of all messages sent by the prover. The query complexity of the protocol, denoted $q(N)$, is the number of entries read by $V$ from the various prover messages; since the verifier has oracle access to those messages, typically $q(N) \ll \ell(N)$. (For the FRI system $q(N) = O(\log \ell(N))$). We denote by $(P \leftrightarrow V)(x)$ the output of $V$ after interacting with $P$ on input $x$; this output is either accept or reject. An IOP is said to be transparent (or have public randomness) if all messages sent from the verifier are public random coins and all queries are determined by public coins, which are broadcast to the prover (such protocols are also known as Arthur-Merlin protocols [4]).

### 1.1.2 IOPP

As its name suggests, an IOP of proximity (IOPP) is the natural generalization of a PCP of Proximity (PCPP) to the IOP model. An IOPP for a family of codes\(^3\) $C$ is a pair $(P, V)$ of randomized algorithms, called prover and verifier, respectively. Both parties receive as common input a specification of a code $C \in C$ which we view as a set of functions $C = \{f : S \to \Sigma\}$ for a finite set $S$ and alphabet $\Sigma$. We also assume that the verifier has oracle access to a function $f^{(0)} : S \to \Sigma$ and that the prover receives the same function as explicit input. The number of rounds of interaction, or round complexity, is denoted by $r$, query complexity is denoted by $q$.

\begin{definition}[Interactive Oracle Proof of Proximity (IOPP) [12]] An $r$-round Interactive Oracle Proof of Proximity (IOPP) $S = (P, V)$ is a $(r+1)$-round IOP. We say $S$ is an $(r$-round) IOPP for the error correcting code $C = \{f : S \to \Sigma\}$ with soundness $s^− : \{0, 1\} \to [0, 1]$ with respect to distance measure $\Delta$, if the following conditions hold:

- **First message format**: the first prover message, denoted $f^{(0)}$, is a purported codeword of $C$, i.e., $f^{(0)} : S \to \Sigma$
- **Completeness**: $\Pr[(P \leftrightarrow V) = \text{accept} | \Delta(f^{(0)}, C) = 0] = 1$; this means that for every $f^{(0)} \in C$ the protocol terminates in acceptance.
- **Soundness**: For any $P^*$, $\Pr[(P^* \leftrightarrow V) = \text{reject} | \Delta(f^{(0)}, C) = \delta] \geq s^−(\delta)$

The sum of lengths of all prover messages, except for $f^{(0)}$, is the IOPP proof length; the time required to generate all messages except for $f^{(0)}$ is the prover complexity. The IOPP query complexity is the total number of queries to all messages, including $f^{(0)}$ and the decision complexity is the computational complexity (see following remark) required by the verifier to reach its verdict, once the queries and query answers are provided as inputs.

\begin{remark}[Computational model for decision complexity] The computational model in which decision complexity is computed is left undefined. A natural default is to use boolean circuit complexity. However, later we study families of linear codes in which each IOPP query is answered by a field element. The natural computational model in this case is that of arithmetic complexity, i.e., for a linear code $C$ over a finite field $\mathbb{F}$, it is the number of arithmetic operations over $\mathbb{F}$ made by the verifier to reach its decision.

### 1.1.3 Main Theorem

The finite field of size $q$ is denoted here by $\mathbb{F}_q$; when $q$ is clear from context we omit it. A field is called binary if $q = 2^m$, $m \in \mathbb{N}$. A subset $S$ of a binary field is an additive coset if it is a coset of a subgroup of the additive group $\mathbb{F}_q^+$, i.e., if $S$ is an additive shift of an

\[^3\] The definition of an IOPP can be generalized to arbitrary languages; we study an IOPP for a specific family of codes so prefer to limit the scope of our definition accordingly.
The binary additive RS code family is the collection of codes $RS[\mathbb{F}, S, \rho]$ where $\mathbb{F}$ is a binary field and $S$ an additive coset. This family of codes is one for which quasilinear PCPP were defined in [23], and our main theorem is stated for it (see Table 1).

**Theorem 2 (Main – FRI properties).** The binary additive RS code family of rate $\rho = 2^{-R}, R \geq 2, R \in \mathbb{N}$ has an IOPP (FRI) with the following properties, where $N = |S|$ denotes block-length (which equals the prover-side input length for a fixed $RS[\mathbb{F}, S, \rho]$ code) and $\rho N > 16$:

- **Prover** Complexity is less than $6N$ arithmetic operations in $\mathbb{F}$; proof length is less than $N/3$ field elements and round complexity is at most $\log N$;
- **Verifier complexity** Query complexity is $2 \log N$; the verifier decision is computed using at most $21 \log N$ arithmetic operations over $\mathbb{F}$;
- **Soundness:** There exists $\delta_0 \geq \frac{1}{4} (1 - 3\rho) - \frac{3N}{|\mathbb{F}|}$ such that every $f$ that is $\delta$-far in relative Hamming distance from the code, is rejected with probability at least $\min(\delta, \delta_0) - 3N/|\mathbb{F}|$;
- **Parallelization:** Each prover-message can be computed in $O(1)$ time on a Parallel Random Access Machine (PRAM) with common read and exclusive write (CREW), assuming a single $\mathbb{F}$ arithmetic operation takes unit time.

**Remark (Space complexity).** Given the $i$th prover message as input, each symbol of the $(i + 1)$th prover message can be computed with space complexity $O(\log |\mathbb{F}|)$, i.e., the space required to hold a constant number of field elements.

This follows immediately from the fact that each prover message is computed in $O(1)$ arithmetic operations on a parallel machine.

Generalizing Theorem 2 to arbitrary rate $\rho \in (0, 1]$ can be done as described in [23, Proposition 6.13] (cf. remark 6.2 there); this leads to slightly larger constants in the prover and verifier complexity. For practical applications like ZK-IOPs [14, 12], rates of the form stated in the theorem above suffice.

**Remark (FRI for “smooth codes”).** We call a multiplicative group $H \subset \mathbb{F}_q$ smooth if its order ($|H|$) is $2^k$ for $k \in \mathbb{N}$. The family of smooth RS codes of rate $\rho$ is the set of $RS[\mathbb{F}_q, H, \rho]$ codes in which $H$ is a smooth multiplicative group. Theorem 2 holds also with respect to the family of smooth RS codes, with somewhat smaller constants than 6 and 21 for the prover and verifier arithmetic complexity (see full version of this paper [10]); see Section 2.1 for a high-level overview of the smooth case and full version of this paper [10] for more details on modifying the protocol to this case. The protocol can be further generalized to groups of order $c^k$ for constant $c$ (perhaps with different arithmetic complexity constants), details omitted.

The soundness bound of Theorem 2 is nearly tight for $\delta \leq \delta_0$. We conjecture that a similar bound holds for all $\delta$. See full version of this paper [10] for a more detailed version of the conjecture that implies it, and a discussion of Equation (1).

**Conjecture 3.** The soundness limit $\delta_0$ of Theorem 2 approaches $1 - \rho$. Specifically, for all $\delta \leq 1 - \rho$, the rejection probability of any $f$ that is $\delta$-far from the RS-code of rate $\rho$ and block-length $N$ over $\mathbb{F}$, is at least

$$\delta - \frac{2 \log N}{\sqrt{|\mathbb{F}|}}.$$  \(1\)
1.2 Applications to transparent zero knowledge implementations

Prover-efficient IOPPs of the kind presented here are crucially needed to facilitate practical ZK argument systems that are (i) transparent (public randomness), (ii) universal – apply to any computation – and (iii) (doubly) scalable – have quasi-linear proving time and poly-logarithmic verification time, simultaneously. In a follow-up paper we use the FRI protocol (among other things) to realize in code the first ZK system (called a ZK-STARK there) that achieves the three properties listed above [11]. The concrete efficiency of that protocol, which relies to a large degree on the efficiency of the FRI protocol presented here, allows one to construct ZK arguments of knowledge (ZK-ARKs) for computational statements, where verifying the computational integrity of the statement using the ZK-STARK verifier is strictly faster than naïve verification via re-execution, and the communication complexity is strictly smaller than the size of the non-deterministic witness supporting the claim. The ZK-STARK prover is $\approx 50 \times$ faster than the previous state-of-the-art transparent system, code-named SCI [7] (that system does not have ZK), and $\approx 10 \times$ faster than state-of-the-art ZK-SNARKs [17] (which are not transparent); see [11, Figure 5] for details.

In the remainder of this section we explain, briefly, how our system could be incorporated in a larger practical ZK system (like the ZK-STARK mentioned earlier). In Section 1.3 we discuss the range of block-lengths that are relevant in applications, and the resulting communication complexity arising from their use.

The seminal works of Babai et al. [6, 5] showed that verifying the correctness of an arbitrary nondeterministic computation running for $T(N)$ steps can be achieved by a verifier running in time $\text{poly}(N, \log T(N))$ in the PCP model. Kilian’s construction transforms such PCPs into a 4-round ZK argument in which the total communication complexity and verifier running time are bounded by $\text{poly} \log T(N)$ [43] (cf. [44, 40, 41]), assuming a family of collision-resistant hash functions. Micali further compressed this system into a non-interactive computationally sound (CS) proof system, assuming both prover and verifier share access to the same random function [48]; this is typically realized in practice using a hash function like SHA2 and relying on the Fiat-Shamir heuristic [31]. No implementation of these marvelous techniques has appeared during the quarter century that has passed since they were first published. This is explained, in part, by concerns about the efficiency of these constructions for concrete programs and run-times. Among the numerous components involved in building these systems, a significant computational bottleneck is that of computing solutions to the Reed-Muller (RM) proximity problem, also known as “low degree testing” of multivariate polynomials.

Quasilinear PCPs based on RS codes have prover complexity that is asymptotically more efficient than RM codes which lead to PCPs with super-quasi-linear length, and a number of works have explored the concrete efficiency of these RS-based protocols [16, 8]. Recently, Ben-Sasson et al. suggested an IOP with perfect zero knowledge (PZK) for NP [14], later extended to NEXP [12], in which prover complexity is quasilinear and verifier complexity is $\text{poly}(N, \log T(N))$; this PZK-IOP can be compiled, using Kilian’s technique, into an interactive ZK argument with succinct\(^4\) communication complexity, or, using Micali’s technique (cf. [61]), into a non-interactive random oracle proof (NIROP) as defined in [19]. In light of this, the practicality of Kilian- and Micali-type ZK argument systems with polylogarithmic verifiers should be reconsidered.

To add motivation, a number of interesting practical succinct argument systems (with and without zero-knowledge) have been reported recently (see [62] for an excellent updated survey)

\(^4\) Here, as in past works, “succinct” is synonymous to “polylogarithmic”.
of the subject and [7] for a comparison of PCP/IOP-based solutions to other approaches). A particular system based on the quadratic span programs (QSP) of Gennaro et al. [33] (cf. [17]) has been used by Ben-Sasson et al. to build a decentralized anonymous payment (DAP) system called “Zerocash” [15], later deployed as a practical commercial crypto-currency called “Zcash” [52, 38]. However, the QSP based ZK system used in Zerocash/Zcash, called a “preprocessing SNARK” [24], requires a setup phase that involves private randomness; additionally, it relies on rather strong cryptographic “knowledge of exponent” assumptions, and quantum computers can create pseudo-proofs of falsities in polynomial time for such systems [60] (cf. [54]). In contrast, the aforementioned succinct interactive and non-interactive (NIROP) systems based on quasilinear PZK-IOPs require only public randomness for their setup, and the only cryptographic assumption required to realize them\(^5\) is the existence of a family of collision resistant hash functions [43], in particular, they are not known to be breakable by quantum computers in polynomial time. Therefore, there is great interest in understanding whether succinct (interactive and non-interactive) ZK argument systems which require only public randomness (and resistant to known polynomial time quantum algorithms) can be practically built and used, say, by Zcash. Ben-Sasson et al. [7] describe such an implemented system, called “succinct computational integrity (SCI)” which is not ZK and has comparatively large communication complexity\(^6\). As mentioned above, the RS proximity solution described in Theorem 2 is already used within an implemented ZK system [11].

1.3 Concrete degree, communication, and round complexity

In this section we briefly discuss the “size” of RS codes that would be needed for various practical applications and the effect of logarithmic round complexity on security. Due to space limitations, and because the focus of this paper is theoretical (within the information theoretic IOP model), we omit implementation details and point the interested reader to full version of this paper [10]; cf. [7, 14].

The message length of RS codes of degree \(d = \rho \cdot N - 1\) is precisely \(d\), so we start by recounting the range of degrees (message sizes) that seem practically relevant. Later we calculate the communication complexity arising from using the FRI protocol to argue proximity to codes of practically relevant block-lengths, and end by discussing the practical implications of an IOPP with \(\log d\) rounds. Throughout this section \(\rho = 1/8\) (\(N = 8 \cdot d\)) because this setting is used in prior [7] and future [11] works.

1.3.1 RS block-length of systems realized in code

The recently realized IOP-based argument system called SCI (“Scalable Computational Integrity”) reduces computational statements, like “the output of program \(P\) on input \(x\) equals \(y\) after \(T\) steps” to a pair of RS-proximity testing problems. SCI uses an IOP version of the quasilinear PCP of [23], which could be replaced with FRI. Programs bench-marked by SCI were executed on a simple MIPS-like virtual machine called TinyRAM [18]. Generally speaking, RS degree increases in size with the number of TinyRAM machine cycles \(T\).

\(^5\) To reach a (non-interactive) computationally sound (CS) proof [49], the “random oracle” is assumed, and realized in practice by relying on the Fiat-Shamir heuristic. In particular, this approach as well is not known to be breakable by quantum computers in polynomial time.

\(^6\) Communication complexity in SCI is on the order of tens of megabytes long, compared with QSP based zk-SNARKs that are shorter than 300 Bytes.
Figure 1 A. Degree of RS code arising from the exhaustive subset sum program [7, Appendix C], as a function of degree, using $\lambda = 160$ bits, field size $2^{64}$, soundness error $\epsilon = 2^{-80}$, and maximal proximity parameter $\delta = 1 - \rho$. The higher (red) graph corresponds to proven soundness (see full version of this paper [10]) and the lower (blue) corresponds to conjectured soundness (Conjecture 3). Both plots use code rate $\rho = 1/8$.

Figure 1A plots the degree $d$ as a function of $T$ for a specific simple program, showing that $d \approx T \cdot 2^{21}$. For crypto-currency applications requiring zero knowledge, block-length will be dominated by the type of cryptographic primitives required, and the number of times they are invoked within a computational statement. For instance, ZK contingent payments [47] require a single hash, and Zerocash’s Pour circuit [15] uses 64 hash invocations, leading in that work to RS codewords (over a prime field) with degree (=number of gates) approximately $2^{22}$. Our new work in progress shows that a single hash invocation requires RS block-length between $2^{12} = 4096$ (for a Davies–Meyer hash based on AES128) to $2^{19}$ (for SHA2), meaning that degrees in the range $d \in [2^{12}, 2^{26}]$ are relevant for existing crypto-currency (ZK) applications [11].

1.3.2 Estimated communication complexity and argument length

The practical realization of interactive proof systems (see Section 1.2) into interactive argument systems [43] and CS proofs [49] can be extended to the IOP model, in which multiple rounds of interaction are used [19]. Using Kilian’s scheme [43], during the $i$th round the prover sends the root $\text{root}^{(i)}$ of a Merkle hash tree $\text{Tree}^{(i)}$ whose leaves are labeled by entries of $f^{(i)}$, and the verifier replies with randomness. Using Micali’s scheme [49], the (non-interactive) prover queries the random oracle with $\text{root}^{(i)}$ to “simulate” the verifier’s $i$th message. When verifier queries to $f^{(i)}$ are answered by the prover, each answer is accompanied by an authentication path (AP) that shows the query answer is consistent with $\text{root}^{(i)}$. Let $CC_{\delta,\epsilon}(N)$ denote the prover-side communication complexity (in bits) of an argument/CS proof realized by applying the Kilian/Micali scheme to FRI, where $\delta$ is the proximity parameter and $\epsilon$ is the error bound, i.e., words that are $\delta$-far from the RS code are rejected with probability $< \epsilon$. Then

$$CC_{\delta,\epsilon}(N) = q_{\delta,\epsilon} \cdot \log |F| + AP_{\delta,\epsilon} \cdot \lambda$$  \hspace{1cm} (2)

where $q_{\delta,\epsilon}$ denotes total query complexity in the IOP model to reach soundness $\geq 1 - \epsilon$ for proximity parameter $\delta$, $AP_{\delta,\epsilon}$ is the number of nodes in the sub-forest of the Merkle trees $\text{Tree}^{(0)}, \ldots, \text{Tree}^{(i)}$ induced by all authentication paths, and $\lambda$ is the number of output bits of
the hash function used to construct the Merkle trees. In our preliminary results [11] we use \(\lambda = 160, \epsilon = 2^{-80}, |F| = 2^{64}\) and \(\rho = 1/8\). Figure 1.B plots the communication complexity for this setting under the proven soundness of Theorem 2 and the (better) soundness of Conjecture 3. In both cases we use maximally large distance \(\delta = 1 - \rho = 7/8\) to show the concrete difference in communication complexity between the proven and conjectured soundness. This plot also motivates the quest for improving the soundness analysis of Theorem 2.

### 1.3.3 Round complexity considerations

Assuming that a crypto-currency block-chain serves as a time-stamping service for public messages and a public beacon of randomness, one may use block-chains to simulate verifier messages. Several block-chains (including Zcash) generate blocks every 2.5 minutes, which means that a FRI proof for \(d = 2^k\) will take roughly \(k \cdot \frac{5}{4}\) minutes to complete, or less than 1 hour\(^7\) for \(d < 2^{40}\).

For fixed \(d\), the round complexity stated in Theorem 2 is \(\frac{1}{2} \log d\), but the more refined version (see [10]) gives a trade-off between query (\(q\)) and round (\(r\)) complexity, of the form \(r = \log d / \log q\), allowing further reduction in round complexity in exchange for larger communication complexity.

Finally, the Random Oracle model used by Micali to “compress” interactive argument systems (like Kilian’s) into CS proofs applies equally to multi-round IOPs like FRI, with negligible impact on argument length; see [19, Remark 1.6] for a detailed discussion. Practically speaking, those who treat hash functions like SHA2 as realizations of the RO model (a position taken by Bitcoin and other crypto-currency miners), might feel comfortable compiling IOP protocols like FRI into succinct non-interactive arguments, as described in [19].

### 1.4 Related works

**High-rate LTCs** Locally testable codes (LTCs) are error correcting codes for which – by definition – prover complexity and proof length equal 0 (as stated for the case of RS codes by Rubinfeld and Sudan [58]); in other words, when focusing solely on prover complexity, LTCs offer an optimal solution (zero complexity). Nevertheless, as discussed in Section 1.2, the specific question of small prover complexity for RS codes is highly relevant because of the its applications to practical ZK-IOPs.

Classical “direct” constructions of LTCs, such as the Hadamard code studied by Blum, Luby and Rubinfeld [25] and the \(\log N\)-variate RM codes used in early PCP constructions [1, 5] have sub-constant rate, thus lead to long proofs and large PCP prover complexity.

More recently, there has been remarkable progress on constructing locally testable codes (LTCs) with small query complexity and large soundness. Kopparty et al. obtained such codes with rate approaching 1 [45] and Gopi et al. presented LTCs that reach the Gilbert Varshamov bound [36]. These LTCs have super-polylogarithmic query complexity. Additionally, in contrast to RS codes, we are not aware of PCP constructions with similar parameters nor do we know how to convert these LTCs into PCPs.

**PCPs and IOPs:** A number of recent works have considered PCP constructions with small proof length and query complexity. In addition to the aforementioned works on quasi-linear PCPs, Moshkovitz and Raz constructed PCPs with optimally small query complexity

\(^7\) Compare this with Bitcoin’s “best practice” of waiting 1 hour for confirmations, or 3 days required to clear standard checks.
(measured in bits) and proofs of length $N^{1+o(1)}$ [51], where $N$ denotes the length of the NP statement (like a 3CNF) for which the PCP is constructed, achieving better soundness than Hastad’s result [37]. A different line of works attempts to optimize the bit-length of PCP proofs; the state of the art, due to Ben-Sasson et al., achieves PCPs of bit-length $O(N)$ and query complexity $N^*$ [22]. In the IOP model, which generalizes PCPs by allowing more rounds of interaction, Ben-Sasson et al. presented a 2-round IOP with bit-length $O(N)$, constant query complexity (measured in bits) and constant soundness [13]. (Prover arithmetic complexity in all of these systems is super-linear.)

Soundness amplification: A number of results in the PCP literature have suggested techniques for improving soundness of general PCP constructions, including the parallel repetition theorem of Raz [55], the gap amplification technique of Dinur [28] and direct-product testing, introduced by Goldreich and Safra [34] (cf. [29, 39]). These techniques lead to excellent soundness bounds with small query complexity. The concrete prover complexity of PCPs and PCPPs associated with these methods has not been studied in prior works but prover complexity is at least super-linear, and often polynomially large.

Doubly-efficient “proofs for muggles”: A recent line of works, initiated by Goldwasser, Kalai and Rothblum [35], revisits the IP model which is equivalent to PSPACE [46, 59], focusing on doubly efficient systems in which the prover runs in polynomial time (as opposed to polynomial space, as in the aforementioned results) and verifier runs in nearly linear time. The state of the art along this line is due to Reingold et al. [57], they construct doubly-efficient IP protocols with a constant number of rounds for a family of languages in P. Prover complexity in this line of works is at least super-linear, and typically polynomially large and verifier complexity is super-polylogarithmic, and often super-linear as well (cf. [27, 57]).

2 Overview of the FRI IOPP and its soundness

In this section we consider the task of building an IOPP for a “smooth” RS code (defined below). We start in Section 2.1 by considering the completeness case, where we describe the interaction between the verifier and an honest prover attempting to prove membership in the RS code of a valid codeword $f^{(0)}$. The IOPP protocol is explained in similarity to the Inverse Fast Fourier Transform (IFFT) [26]. Then, in Section 2.2, we consider the soundness case, where we assume $f^{(0)}$ is far in relative Hamming distance from the code and need to prove lower bounds on the verifier’s rejection probability. Soundness analysis is the most challenging aspect of our work (as it is for all prior PCPP/IOPP works). Our analysis uses the soundness analysis of the quasilinear RS-PCPP [23] for the case of “large” Hamming distance (beyond the unique decoding radius of the code), and presents a novel, tighter, analysis for “small” Hamming distance (below that radius).

2.1 FRI overview and similarity to the Fast Fourier Transform (FFT)

We start by describing the protocol in similarity to the IFFT algorithm; that algorithm is also related to the quasi-linear PCPP for RS codes of [23], and towards the end of this section we explain the connection between FRI and that quasi-linear PCPP.

Let $\omega^{(0)}$ generate a smooth multiplicative group of order $N = 2^n$ (see Remark 1.1.3), denoted $L^{(0)}$, that is contained in a field $F$; in signal processing applications $\omega^{(0)}$ is a complex root of unity of order $2^n$ and $F$ is the field of complex numbers (we shall use a different setting). Assume the prover claims that $f^{(0)} : L^{(0)} \to F$ is a member of $\text{RS}[F, L^{(0)}, \rho]$, i.e., $f^{(0)}$ is the evaluation of an unknown polynomial $P^{(0)}(X) \in F[X], \deg(P) < \rho 2^n$; for simplicity we assume $\rho = 2^{-R}$ and $R$ is a positive integer. The task of the verifier is to distinguish between
low-degreeness \( f^{(0)} \equiv P^{(0)} \) for some low degree polynomials \( P^{(0)} \). Recalling the IFFT, if \( f^{(0)} \equiv P^{(0)} \) there exist polynomials \( P^{(1)}_0, P^{(1)}_1 \in \mathbb{F}[Y] \) such that \( \text{max} \{ \deg P^{(1)}_0, \deg P^{(1)}_1 \} < \frac{1}{2} \rho 2^n \) and

\[
\forall x \in L^{(0)} \quad f^{(0)}(0) = P^{(1)}_0(x^2) + x \cdot P^{(1)}_1(x^2),
\]

or, letting \( Q^{(1)}(X,Y) \triangleq P^{(1)}_0(Y) + X \cdot P^{(1)}_1(Y) \) and defining \( q^{(0)}(X) \triangleq X^2 \), we have

\[
P^{(0)}(X) \equiv Q^{(1)}(X,Y) \mod Y - q^{(0)}(X)
\]

(3)

where \( \deg X (Q^{(1)}) < 2 \) and \( \deg Y (Q^{(1)}) < \frac{1}{2} \rho 2^n \). The map \( x \mapsto q^{(0)}(x) \) is 2-to-1 on \( L^{(0)} \) because \( q^{(0)}(x) = q^{(0)}(-x) \), and the output of this map is the multiplicative group generated by \( \omega^{(1)} = (\omega^{(0)})^2 \), this group has order \( 2^{n-1} \), denote it by \( L^{(1)} \). Moreover, for every \( x^{(0)} \in \mathbb{F} \) and \( y \in L^{(1)} \), the value of \( Q^{(1)}(x^{(0)}, y) \) can be computed by querying two entries of \( f^{(0)} \) because \( \deg X (Q^{(1)}) < 2 \) (the two entries are the two roots of the polynomial \( y - q^{(0)}(X) \)).

Our verifier thus samples \( x^{(0)} \in \mathbb{F} \) uniformly at random and requests the prover to send as its first oracle a function \( f^{(1)} : L^{(1)} \rightarrow \mathbb{F} \) that is supposedly the evaluation of \( Q^{(1)}(x^{(1)}, Y) \) on \( L^{(1)} \). Assuming \( f^{(0)} \in \text{RS}[\mathbb{F}, L^{(0)}] \), the discussion above shows that \( f^{(1)} \in \text{RS}[\mathbb{F}, L^{(1)}] \).

Notice that there exists a 3-query test for the consistency of \( f^{(0)} \) and \( f^{(1)} \), we call it the round consistency test:

1. sample a pair of distinct elements \( s_0, s_1 \in L^{(0)} \) such that \( s_0^2 = s_1^2 = y \); in other words, sample a uniform \( y \in L^{(1)} \) and let \( s_0, s_1 \) be the two roots of the polynomial \( y - X^2 \);
2. query \( f^{(0)}(s_0), f^{(0)}(s_1) \) and \( f^{(1)}(y) \), denote the query answers by \( \alpha_0, \alpha_1 \) and \( \beta \), respectively;
3. interpolate the “line” through \((s_0, \alpha_0)\) and \((s_1, \alpha_1)\), i.e., find the polynomial \( p(X) \) of degree at most 1 that satisfies \( p(s_0) = \alpha_0 \) and \( p(s_1) = \alpha_1 \); notice \( p \) is unique and well-defined because \( s_0 \neq s_1 \);
4. accept if and only if \( p(x^{(0)}) = \beta \) and otherwise reject;

Tallying the costs of the first round, the verifier sends a single field element \( x^{(0)} \) and the prover responds with a message (oracle) \( f^{(1)} : L^{(1)} \rightarrow \mathbb{F} \) evaluated on a domain that is half the size of \( L^{(0)} \); testing the consistency of \( f^{(0)} \) and \( f^{(1)} \) requires three field elements per test (repeating the test boosts soundness). We thus reduced a single proximity problem of size \( 2^n \) and rate \( \rho \) to a single analogous problem of size \( 2^{n-1} \) and same rate. Repeating the process for \( r = n \) leads to a function \( f^{(0)} \) that is supposedly of constant degree and evaluated on a domain of constant size \( 2^R \), so at this point the prover sends the single constant that describes the function, and the verifier uses it as \( f^{(r)} \) in the last round consistency test, the one that tests consistency of \( f^{(r-1)} \) and \( f^{(0)} \).

Applying inductive analysis to all \( r \) rounds, if \( f^{(0)} \in \text{RS}[\mathbb{F}, L^{(0)}, \rho] \) (and the prover is honest) then all \( r \) round consistency tests pass with probability 1 and \( f^{(0)} \) is indeed a constant function. In other words, the protocol we described has perfect completeness.

\textbf{Remark} (FRI as a “biased” version of quasi-linear RS-PCPP). The quasi-linear PCPP of [23] is quite similar to FRI, including the degree-reduction (from \( P^{(0)} \) to \( P^{(1)} \)) obtained by requesting the prover to evaluate a bivariate polynomial \( Q(X,Y) \) on a collection of axis-parallel lines. There are two main differences between FRI and that PCPP:

1. the quasi-linear PCPP is non-interactive, and thus the prover evaluates \( Q^{(1)}(X,Y) \) on a large subset of \( \mathbb{F} \times \mathbb{F} \), whereas the FRI protocol uses interaction to reduce proving time, by requesting the prover to apply recursion only to the axis-parallel lines selected by the verifier.
2. the polynomial \( q^{(0)}(X) \) used in [23] has degree \( \approx \sqrt{\deg(P^{(0)})} \) and thus \( Q^{(1)}(X,Y) \) has degree \( \approx \sqrt{\deg(P^{(0)})} \) in each of its variables. In contrast, the polynomial \( q^{(0)} \) used by FRI has constant degree, and so the degrees of \( Q^{(1)} \) are very biased (constant degree in \( X \) vs. \( \deg(P^{(0)}) / 2 \) in \( Y \)). This leads to larger recursion depth for FRI but also avoids the necessity to apply recursive low-degree testing to each of the axes (the \( X \)-axis) because of its constant degree.

### 2.1.1 Differences between informal and actual protocol

The differences between the informal and formal protocols are mostly technical; we list them now. The field \( F \) is finite and binary, i.e., of characteristic 2; nevertheless the construction and analysis can be immediately applied to RS codes evaluated over smooth multiplicative groups (of order \( 2^n \)), as explained informally above (cf. Remark 1.1.3). In binary fields, the natural evaluation domains (like \( L^{(0)}, L^{(1)} \) above) are cosets of additive groups (not multiplicative ones), i.e., \( L^{(i)} \) is an affine shift of a linear space over \( F_2 \). The map \( q^{(0)}(X) = X^2 \) is not 2-to-1 on \( L^{(0)} \) (in binary fields it is a 1-to-1 map, a Frobenius automorphism of \( F \) over \( F_2 \)) so we use a different polynomial \( q^{(0)}(X) \) that is many-to-one on \( L^{(0)} \) and such that the set \( L^{(1)} = \{ y = q^{(0)}(x) \mid x \in L^{(0)} \} \) is a coset of an additive group, like \( L^{(0)} \), but of smaller size \((|L^{(1)}| \ll |L^{(0)}|)\); the polynomial \( q^{(0)} \) is known as an affine subspace polynomial, belonging to the class of linearized polynomials. We use \( q^{(0)} \) of degree 4 instead of 2 because this reduces the number of rounds from \( n \) to \( n/2 \) with no increase in total query complexity; notice that a similar reduction could be applied in the multiplicative setting by using \( q^{(0)} = X^4 \) (but we preferred simplicity to efficiency in the informal exposition above). Finally, the actual protocol performs all queries only after the prover has sent all of \( f^{(1)}, \ldots, f^{(r)} \). Thus, we construct a protocol with two phases. The first phase, called the COMMIT phase, involves \( r \) rounds. At the beginning of the \( i \)-th round the prover has sent oracles \( f^{(0)}, \ldots, f^{(i-1)} \), and during this \( (i) \)-th round the verifier samples and sends \( x^{(i)} \) and the prover responds by sending the next oracle \( f^{(i)} \). During the second phase, called the QUERY phase, the verifier applies the round consistency test to all \( r \) rounds. To save query complexity and boost soundness, the query made to \( L^{(i)} \) is used to test both consistency of \( f^{(i-1)} \) vs. \( f^{(i)} \) and consistency of \( f^{(i)} \) vs. \( f^{(i+1)} \).

### 2.2 Soundness analysis – overview

*Proof composition* is a technique introduced by Arora and Safra [3] in the context of PCPs, adapted to PCPPs in [21, 30] and optimized for the special case of the RS code in [23]. Informally, it reduces proximity testing problems over a large domain to similar proximity testing problems over significantly smaller domains. The process reducing \( f^{(0)} \) to \( f^{(1)} \) above is a special case of proof composition, and each invocation of it incurs two costs on behalf of the verifier. The first is the query complexity needed to check consistency of \( f^{(0)} \) and \( f^{(1)} \) (the “round consistency test”) and the second is the reduction in distance, which affects the soundness of the protocol. Assuming \( f^{(0)} \) is \( \delta^{(0)} \)-far from all codewords in relative Hamming distance, for proof composition to work one should prove that with high probability \( f^{(1)} \) is \( \delta^{(1)} \)-far from all codewords where \( \delta^{(1)} \) depends on \( \delta^{(0)} \), larger values of \( \delta^{(1)} \) imply higher (better) soundness and smaller communication complexity. A benefit of the FRI protocol is that with high probability \( \delta^{(1)} \geq (1 - o(1))\delta^{(0)} \), i.e., the reduction in distance in our protocol is negligible. In contrast, prior RS proximity PCPP and IOPP solutions follow the construction and analysis of [23] which in turn is based on the bivariate testing Theorem of
Polischuk and Spielman [53] and incur a constant multiplicative loss in distance per round of proof composition ($\delta^{(1)} \leq \delta^{(0)}/2$). This loss limited the number of proof composition rounds to $\leq \log N$ and thus required replacing $q^{(0)}(X) = X^2$ with a higher degree polynomial, like $q^{(0)}(X) = X^{2^n/2}$. The higher degree of $q^{(0)}$ results in $Q^{(1)}(X,Y)$ having balanced $X$- and $Y$-degrees, namely

$$\deg_X(Q^{(1)}) \approx \deg_Y(Q^{(1)}) \approx 2^{n/2}.$$ 

Moving to $q^{(0)}(X)$ of constant degree as in FRI gives a biased RS-IOPP (because $\deg_X(Q^{(1)}) \ll \deg_Y(Q^{(1)})$). The main benefit of this bias is that one side of the recursive process (that of $X$) terminates immediately and consequently removes the constant multiplicative soundness loss incurred in prior works, replacing it with a negligible additive loss. More to the point, we show that for $\delta^{(0)}$ less than the unique decoding radius of the code ($\delta^{(0)} < (1 - \rho)/2$), with high probability (namely, $1 - O(1)|F|$) the sum of (i) the round consistency error and (ii) the “new” distance $\delta^{(1)}$ is at least as large as the “old” distance $\delta^{(0)}$. This statement is relatively straightforward to prove in case the prover is honest, i.e., when $f^{(1)}(y) = Q^{(1)}(x^{(0)}, y)$ for all $y \in L^{(1)}$ (in this case there is no round consistency error). The challenging part of the proof is to show this also holds for non-honest provers and arbitrary $f^{(1)}$; see full version of this paper [10] for more details.

References


Scalable, transparent, and post-quantum secure computational integrity

Eli Ben-Sasson*  Iddo Bentov†  Yinon Horesh*  Michael Riabzev*

March 6, 2018

Abstract

Human dignity demands that personal information, like medical and forensic data, be hidden from the public. But veils of secrecy designed to preserve privacy may also be abused to cover up lies and deceit by institutions entrusted with Data, unjustly harming citizens and eroding trust in central institutions.

Zero knowledge (ZK) proof systems are an ingenious cryptographic solution to this tension between the ideals of personal privacy and institutional integrity, enforcing the latter in a way that does not compromise the former. Public trust demands transparency from ZK systems, meaning they be set up with no reliance on any trusted party, and have no trapdoors that could be exploited by powerful parties to bear false witness. For ZK systems to be used with Big Data, it is imperative that the public verification process scale sublinearly in data size. Transparent ZK proofs that can be verified exponentially faster than data size were first described in the 1990s but early constructions were impractical, and no ZK system realized thus far in code (including that used by crypto-currencies like Zcash™) has achieved both transparency and exponential verification speedup, simultaneously, for general computations.

Here we report the first realization of a transparent ZK system (ZK-STARK) in which verification scales exponentially faster than database size, and moreover, this exponential speedup in verification is observed concretely for meaningful and sequential computations, described next. Our system uses several recent advances on interactive oracle proofs (IOP), such as a “fast” (linear time) IOP system for error correcting codes.

Our proof-of-concept system allows the Police to prove to the public that the DNA profile of a Presidential Candidate does not appear in the forensic DNA profile database maintained by the Police. The proof, which is generated by the Police, relies on no external trusted party, and reveals no further information about the contents of the database, nor about the candidate’s profile. In particular, no DNA information is disclosed to any party outside the Police. The proof is shorter than the size of the DNA database, and verified faster than the time needed to examine that database naively.

*Technion — Israel Institute of Technology, Haifa, Israel; supported by the Israel Science Foundation (grant # 1501/14), the US–Israel Binational Science Foundation, and the European Union’s Horizon 2020 Research And Innovation Programme under grant agreement no. 693423.
†Cornell University, Ithaca, NY, USA.
## Contents

1 Introduction 4
   1.1 Main contribution 6
   1.2 Discussion — Applications to decentralized societal functions 8
   1.3 Comparison to other realized universal ZK systems 9
      1.3.1 Theoretical discussion 9
      1.3.2 Concrete performance 11

2 Methods 14
   2.1 Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI3rd) 14
   2.2 Arithmetization I — Algebraic Intermediate Representation (AIR) 14
   2.3 Arithmetization II — Algebraic Linking Interactive Oracle Proof (ALI) 15
   2.4 Low degree extension and composition degree 17
   2.5 Minimizing Authentication Path Complexity (APC) and Communication Complexity (CC) 17
   2.6 Organization of the remaining sections 18

3 On STIKs and STARKs — formal definitions and prior constructions 19
   3.1 Scalable Transparent IOP of Knowledge (STIK) 19
   3.2 Main Theorems 21
   3.3 Scalable Transparent ARgument of Knowledge (STARK) as a realization of STIK 22
   3.4 Prior STIK and STARK constructions 23

A Measurements of the ZK-STARK for the DNA profile match 31
   A.0.1 Prover 31
   A.0.2 Verifier 32

B From AIR to ZK-STARK 33
   B.1 Preliminaries and notation 34
   B.2 Algebraic Intermediate Representation (AIR) 35
   B.3 Algebraic placement and routing (APR) 39
   B.4 APR reduction 40
   B.5 Algebraic linking IOP (ALI) 42
      B.5.1 On the soundness error of the ALI protocol 44
   B.6 Fast Reed-Solomon (RS) IOP of Proximity (IOPP) (FRI) 45
   B.7 Proof of main theorems 45
      B.7.1 Proof of Main Theorem 3.4 46
      B.7.2 Proof of Main Lemma B.6 46
      B.7.3 Proof of Lemma B.7 48
      B.7.4 Proof of ZK-STIK Theorem 3.5 50
   B.8 Realization considerations 51

C Algebraic placement and routing (APR) reduction 52
   C.1 The APR reduction for space bounded computation 52
      C.1.1 Common definitions 52
      C.1.2 Instance reduction 52
1 Introduction

Scalable verification of computational integrity over confidential datasets The problem addressed here is best illustrated by a hypothetical example: Suppose the Police (P), that is in charge of the national forensic DNA profile database (D), claims that the DNA profile (p) of a soon-to-be-appointed and alleged-to-be-corrupt Presidential Candidate, does not appear in D. Can cryptographic protocols convince the doubtful public to believe this claim, without compromising D or p, without relying on any external trusted party (e.g., the Chief Justice), and with “reasonable” computational resources?

The DNA profile match (DPM) example is a special case of a more general problem. A party (P) executing a computation (C) on a dataset (D) may have incentive to misreport the correct output (C(D)), raising the problem of computational integrity (CI)\(^1\) — ensuring that P indeed reports C(D) rather than an output more favorable to P. When the dataset D is public, any party (V) interested in verifying CI can naïvely re-execute C on D and compare its output to that reported by P, as a customer might inspect a restaurant bill, or as a new Bitcoin node will verify its blockchain [86]. This naïve solution does not scale because the time spent by the verifier (T\(_V\)) is as large as the time required to execute the program (T\(_C\)) and V must read the full dataset D. Commitment schemes based on cryptographic hash functions [33] are commonly used to compute a short immutable “fingerprint” cm\(_t\) for the state at time t of a large dataset D\(_t\) [33]. Typically cm\(_t\) is negligible in length\(^2\) compared to D\(_t\), and may be easily posted on a block-chain to serve as a public notice\(^3\). Thus, the CI solution we seek should have scalable verification, one in which verification time and communication complexity scale roughly like \(\log T_C\) and \(|cm_t|\) (the bit-length of cm\(_t\)), rather than like \(T_C\) and \(|D_t|\); at the very least verification time/communication should be strictly less than \(T_C\) and \(|D_t|\).

When the dataset D contains confidential data, the naïve solution can no longer be implemented and the party P in charge of D may conceal violations of computational integrity under the veil of secrecy. Prevailing methods for enforcing CI over confidential data rely on a “trusted party”, like an auditor or accountant to naïvely verify the computation on behalf of the public. This solution still offers no scaling, much like when the data is public. Worse still, it requires the public to trust a third party, which creates a potential single point of failure in the protocol, as this third party — to the extent it can be agreed upon — can be breached, bribed, or coerced by malicious parties.

Zero knowledge (ZK) proof and argument systems are automated protocols that replace human auditors as a means of guaranteeing computational integrity over confidential data for any efficient computation\(^4\), eliminating corruptibility and reducing costs [59]. A ZK system S for a computation C is a pair of randomized algorithms, \(S = (P, V)\); the prover P is the algorithm used to prove computational integrity and the verifier V checks such proofs. The completeness and soundness of S imply that P can efficiently prove all truisms but will fail to convince V of any falsities (with all but negligible probability). The very first theoretical constructions of ZK systems with scalable verifiers for general computations\(^5\), discussed in the early 1990s, were based on Probabilistically Checkable Proofs (PCP). (See Section 1.3 for recent alternative ZK constructions.) The celebrated PCP Theorem [7, 6, 3, 2] offered a surprising trade-off between the running time spent by the prover constructing the proof (T\(_P\)) and the running time consumed by the verifier

---

\(^1\)This problem is also known as delegation of computation [58], certified computation [41] and verifiable computation [52].

\(^2\)Commonly, cm\(_t\) is the SHA2 hash of D\(_t\), which is 256 bits long for any dataset length.

\(^3\)A recent report by the World Economic Forum mentions several use cases, among them monitoring blood diamonds and curbing human trafficking [68].

\(^4\)In the interactive oracle proof model that we consider, as in the model of multi-prover interactive proofs, ZK proof systems exist for any language in nondeterministic exponential time (NEXP) [11, 15].

\(^5\)Special cases for ZK, like proving membership/non-membership in a hidden-and-committed set — the “ZK-set” problem — are efficiently solved by other cryptographic means [84].
checking it ($T_V$); this trade-off means proving time increases \textit{polynomially} compared to naïve computation time ($T_P = T_C^{O(1)}$) whereas verification time decreases \textit{exponentially} with respect to it ($T_V = \log^{O(1)} T_C$).

A ZK system based on the PCP Theorem (ZK-PCP) \cite{74, 85, 49, 75, 71} has three additional advantages that are essential for ongoing public trust in computational integrity. First, the assumptions on which the security of these constructions is founded — the existence of collision-resistant hash functions \cite{74} for interactive solutions, and common access to a random function\footnote{Even though the random oracle model, per se, is unattainable, it’s use is prevalent in cryptography and the theoretical justification for it discussed, e.g., in \cite{10} and following works.} (the “random oracle model” \cite{50}) for non-interactive ones \cite{85} — are not known to be susceptible to attacks by large-scale quantum computers; we call such solutions \textit{post-quantum secure}. The anticipated increase in scale of quantum computers \cite{43} and the call for post-quantum cryptographic protocols, e.g., by the USA National Institute of Standards and Technology (NIST) \cite{37}, highlight the importance of a post-quantum secure ZK solution.

Second, ZK-PCPs are \textit{proof of knowledge} (POK) systems, or, when realized as described above, \textit{argument of knowledge} (ARK) systems \cite{33, 9}. Informally, in the context of the DPM example, a ZK-ARK is a proof that convinces the public that the Police has used “the true” dataset $D_t$ and Presidential Candidate DNA profile $p$ whose commitments were previously announced (see Definition 3.3).

Third, and most important, ZK-PCPs are \textit{transparent} (or “public randomness”\footnote{Transparent systems are also known as \textit{Arthur-Merlin} protocols \cite{4}.}), which means that the randomness\footnote{Randomness is necessary for ZK proof systems for non-trivial computations \cite[Section 3.2]{57}.} used by the verifier is \textit{public}; in particular, setting up a ZK-PCP requires no external trusted setup phase, in contrast to newer ZK solutions, including the one used by the Zcash™ cryptocurrency (see Section 1.3). Transparency is essential for ongoing public trust because it severely limits the ability of even the most powerful of parties $P$ to abuse the system, and thus transparent systems are ones which the public may reliably trust as long as there exists something unpredictable in the observable universe.

Summarizing, ZK-PCPs are an excellent method for ensuring public trust in CI over confidential data, and possess six core virtues: (i) \textit{transparency}, (ii) \textit{universality} — apply to any efficient computation $C$, even if it requires auxiliary (and possibly confidential) input like $D_t$ above, (iii) \textit{confidentiality (ZK)} — do not compromise auxiliary inputs like $D_t$, (iv) \textit{post-quantum security}, (v) \textit{proof of argument of knowledge} and (vi) \textit{scalable verification}. Although ZK-PCPs have been known since the mid-1990’s, none have been realized in code thus far because, in the words of a recent survey \cite{110}, “the proofs arising from the PCP theorem (despite asymptotic improvements) were so long and complicated that it would have taken thousands of years to generate and check them, and would have needed more storage bits than there are atoms in the universe.” Consequently, recent realization efforts of ZK systems for general computations (surveyed in Section 1.3) focused on alternative techniques that do not achieve all of (i)–(vi), though some are extremely efficient in practice for concrete circuit sizes and for amortized computations.

**Interactive Oracle Proofs (IOP) with scalable proofs** To improve prover scalability without sacrificing properties (i)–(vi), a new model was recently suggested \cite{22, 94}, called an \textit{interactive oracle proof (IOP)}\footnote{Reingold et al. \cite{94} use the name “Probabilistically Checkable Interactive Proofs” (PCIP).}, a common generalization of the IP, PCP, and interactive PCP (IPCP) models \cite{72}. As in the PCP setting, the IOP verifier need not read prover messages in entirety but rather may query them at random locations; as in the IP setting, prover and verifier interact over several rounds. As was the case for ZK-PCPs, a ZK-IOP system can be converted into an interactive ARK assuming a family of collision-resistant hash functions, and can be turned into a non-interactive argument in the random oracle model \cite{22}, which is typically realized using a standard hash function. As a strict generalization of IP/PCP/IPCP, the IOP model offers several
advantages. Most relevant to this work is the improved *prover scalability* of IOPs, described below; this advantage holds both asymptotically — as input size $n \to \infty$ (cf. [15, 16]) — and for concrete input lengths that arise in practice. Based on this efficiency, a proof-of-concept implementation of an IOP, codenamed SCI, was recently reported\(^\text{10}\) [13]; however, SCI does not have ZK and it’s concrete argument length and proving time are still quite large. IOPs with (perfect) ZK and scalable verifiers were recently described, first for NP [17], then for NEXP [15]. In both works, prover running time ($T_P$) is bounded by $T_C \cdot \log^{O(1)} T_C$; we refer to this as *scalable proving time* (also known as *quasi-linear* proving time).

Henceforth, we shall call a (universal) ZK system (vi’) *fully scalable*, or, simply *scalable*, if both prover and verifier running times are scalable; this is justified because both running times are nearly-optimal, up to poly-logarithmic factors. A ZK-IOP system satisfying properties (i)–(v) and full scalability (vi’) will be called a *Scalable Transparent IOP of Knowledge (ZK-STIK)*; see Section 3 for formal definitions. Summarizing, *theoretical* constructions of ZK-STIK systems were recently presented, but their concrete efficiency and applicability to “practical” computations have not been demonstrated thus far.

### 1.1 Main contribution

We present a new construction of a (doubly) scalable and transparent ZK system in the IOP model (a ZK-STIK); see Theorems 3.4 and 3.5 for details. We realize this system as a ZK-STARK and apply it to a proof-of-concept “meaningful” computation that is highly sequential in nature — the DPM problem presented earlier. Our realization achieves (i) verification time that is strictly smaller than naive running time ($T_V < T_C$) and (ii) communication complexity that is strictly smaller than witness size. The core innovation and main source of improved performance in this system is the extended reliance on the IOP model, including the *Fast Reed-Solomon (RS) IOP of Proximity (IOPP)* (FRI) protocol discussed in Section 2 (cf. [14]) and a new arithmetization procedure (see Section 2.3). We stress that the exponential speedup in verification time and witness-size described next (and displayed in Figure 1) apply to any computation that is defined for arbitrarily large witness size, though the particular point at which this speedup materializes depends on the complexity of the computation (as defined in Section 2.2)\(^\text{11}\).

**DNA profile match computation**  As a proof-of-concept “meaningful” computation we construct a ZK-STARK for the *DNA profile match (DPM)* problem, which we describe informally next (see Appendix E for details). This computation addresses the following hypothetical scenario: Suppose that the Police (acting as the prover $P$) is in charge of the national forensic DNA profile database ($D$), and at previous time $t$ has posted (say, on a block-chain) a hiding commitment $c_m$ to the state $D_t$ of the database at that point in time. The Police now claims that the DNA profile $p$ of the soon-to-be-appointed and alleged-to-be-corrupt Presidential Candidate, does not appear in $D_t$ and thus wishes to create, in a scalable manner, a proof that will convince the public that the DPM computation was carried out correctly, and the output reported by the Police is correct (with respect to $p$ and $D_t$).

The prevailing standard for DNA profiles, used in over 50 countries, is the *Combined DNA Index System (CODIS)* format; according to this standard an individual is represented by the Short Tandem Repeat (STR) count of his/her DNA, measured for a set of 20 “core loci” [87] (the number of core loci increased from 13 to 20 starting January 2017). The commitment $c_{m_t}$ to the state $D_t$ of a CODIS database is assumed to be public information (say, published at time $t$ on a blockchain), as is a commitment $c_{m_p}$ to the profile

\(^\text{10}\)https://github.com/elibensasson/SCI-POC

\(^\text{11}\)In particular, a computation with parameters similar to the last row of Figure 4 will behave similarly to the DPM computation displayed on Figure 1.
p of the Presidential Candidate; we assume p was extracted by an independent laboratory that handed it (confidentially) to the Police while publishing cm_p publicly. Assume that the Police declares

“The value α is the result of the match search for the profile with commitment cm_p in the database with commitment cm_t.”

The answer α is one of three possibilities: “no match”, “partial match”, or “full match”. The public (open source) computation C is the one that would have been executed by a trusted third party verifying the claim above. This computation requires three public inputs — cm_t, cm_p, and A — and two confidential inputs: (i) a DNA profile database D’ and individual DNA profile p’. The computation C terminates successfully if and only if the public inputs (cm_t, cm_p, A) and the confidential ones (D’, p’) satisfy three conditions: (i) the commitment cm’ of the confidential input D’ equals the public input cm_t; (ii) the commitment cm_p of the confidential input p’ equals the public input cm_p; and (iii) the output of the match search for the confidential input p’ in the confidential dataset D’ leads to the publicly announced outcome α; see Appendix E.5 for details.

Let |D(n)| denote the bit-length of a dataset D(n) that contains n profiles (each profile is 40 bytes long); let CC(n) denote the communication complexity of the ZK-STARK for D(n), i.e., the total number of bits communicated between prover and verifier; similarly, let T_C(n) denote the time needed to naively verify C by executing it on D with n entries, and let T_V(n) denote the time required by V to verify it. (both measured on a fixed physical computer.)

Realizing time and witness-size compression Consider a computation C which requires auxiliary confidential input D that varies in size, like the DPM example. Any ZK-system S = (P, V) for C induces a pair of rate measures for time and witness-size, respectively:

\[
\rho_{time}(n) = \frac{T_V(n)}{T_C(n)}; \quad \rho_{size}(n) = \frac{CC(n)}{|D(n)|}
\]

The rate measures (and thresholds defined next) depend on C and the system S, so the notation \(\rho_{time}^{(S,C)}\) would be more precise, but we prefer notational simplicity and assume C and S are known.

A rate value smaller than 1 indicates compression, meaning verification in S is more efficient than naive verification. In fully scalable ZK systems verifier complexity is poly-logarithmic in prover complexity. Therefore eventually, for large enough n, the system achieves compression. Our main claim here is that we exhibit, for the first time, time and witness-size compression for a ZK-STARK for a large-scale sequential computation. Define the compression threshold to be the smallest value n_0 such that for all n ≥ n_0 the rate is less than 1,

\[
\theta_{time} = \min\{n_0 \mid \forall n \geq n_0 \ \rho_{time}(n) < 1\}; \quad \theta_{size} = \min\{n_0 \mid \forall n \geq n_0 \ \rho_{size}(n) < 1\}
\]

Figure 1 shows the rate measures for the DPM problem on a double logarithmic scale. The time compression threshold is at \(\theta_{time} = 2.8 \times 10^5\) and the witness-size threshold is \(\theta_{size} = 9 \times 10^3\). The largest database for which we could generate a proof during our tests is \(n_{max} = 2^{20} \approx 1.14 \times 10^6\) DNA profiles; larger databases require more disk space and RAM than was available to us. Each profile occupies 40 bytes so |D|_{max} \approx 43 megabytes. The time-rate for \(n_{max}\) is \(\rho_{time}(n_{max}) = 1/6\) and the witness-size rate is \(\rho_{size}(n_{max}) = 1/100\). This figure also demonstrates that compression will improve if supported by stronger hardware than that on which our tests were executed. (see Appendix A for more measurements.)
1.2 Discussion — Applications to decentralized societal functions

Cryptocurrencies, led by Bitcoin, are disrupting established financial systems by suggesting a fully decentralized monetary system to replace fiat currency. Money is but one of the societal functions that could be decentralized, and legal contracts are already being replaced by automated smart contracts [103] in the Ethereum blockchain. We end this section by discussing the two expected impacts of ZK-STARK systems on decentralized public ledgers.

**Scalability** A heated discussion is taking place in blockchains today, surrounding the proper way to scale the transaction throughput without over-taxing the time and space of nodes participating in the network. As first pointed out by one of the co-authors [12] and embraced recently by several crypto-currency initiatives [66, 34, 76], fully scalable proof systems (even without zero-knowledge) could solve the scalability problem by exponentially decreasing verification time. In more detail, a single “prover node” can generate in quasilinear time a proof that will convince all other nodes to accept the validity of the current state of the ledger, without requiring those nodes to naïvely re-execute the computation, nor to store the entire blockchain’s state, which would be required for such a naïve verification.

**Privacy** The confidentiality of ZK proofs is already being used to enhance coin fungibility and financial privacy in cryptocurrencies. The Zerocash protocol [18] — recently implemented in the Zcash™ cryptocurrency [89, 67] — uses a particular kind of ZK proofs called Succinct Non-interactive ARguments of Knowledge (ZK-SNARK) based on cryptographic knowledge of exponent (KOE) assumptions [53, 21] to maintain with integrity a decentralized registry whose entries are hiding commitments of unspent funds. These ZK-SNARKs are non-transparent as they require a “setup phase” which uses non-public randomness that, if compromised, could be used to compromise the system’s security (see Section 1.3). Looking forward, ZK-STARKs could replace ZK-SNARKs and achieve the fungibility and confidentiality of Zcash™, transparently. Currently, ZK-SNARKs are roughly $1000\times$ shorter than ZK-STARK proofs so replacing ZK-SNARKs with STARKs calls for more research to either shorten proof length, or aggregate and compress
several ZK-STARK proofs using incrementally verifiable computation [105] (cf. [29]).

1.3 Comparison to other realized universal ZK systems

Recent years have seen a dramatic effort to realize in code zero knowledge proof systems using various theoretical approaches that differ from that of our ZK-STARK. Many of these systems outperform our ZK-STARK for sufficiently small-size computations, for low-depth parallel computations, and/or for batched and amortized computations; all of these cases are extremely useful in practice. But for large scale computations, especially sequential ones, the improved full scalability of our IOP-based approach is, eventually, noticeable.

Next, we briefly survey the different implemented approaches that are universal, i.e., apply to general computations and languages in NP; the interested reader is referred to [110] and [13] for more information on computational integrity solutions, including ones that are non-universal and/or without zero-knowledge. We start by an “asymptotic” discussion in Section 1.3.1 and continue with a comparison of concrete parameters for published and realized systems (Section 1.3).

1.3.1 Theoretical discussion

Within the vast (and growing) literature on realizations of ZK systems, we must limit the scope of our discussion and do so somewhat arbitrarily, by considering only systems that are ZK, Turing complete, and which have been realized in code. We compare these for the most general class of computational integrity statements (see Definition 3.1 for a formal definition) and consider four properties: asymptotic (i) prover scalability (quasilinear running time), (ii) asymptotic verifier scalability (poly-logarithmic verification), (iii) transparency, and (iv) post-quantum security. The first three terms are formally defined in Definition 3.3, and the last one is informal, but could be replaced with the property of reliance only on collision resistant hash functions. Figure 2 summarizes our discussion, and we provide details next.

- **Homomorphic public-key cryptography (hPKC):** This approach, initiated by Ishai et al. [69] (for the “designated verifier” case) and Groth [60] (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 [53] (see [64, 52, 81, 30, 62, 63] 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 [88], with subsequent implementations including libSNARK [21, 96] (discussed in the next section) which is used in the Zerocash and Zcash™ implementations; additional implementations appear in [98, 101, 100, 99, 24, 108, 46].

The theoretical differences between hPKC and ZK-STARK are that of transparency and post-quantum security — hPKC lacks both. Verification time in hPKC is scalable (i.e., poly-logarithmic in $T_C$) only for computations that are repeated many times, because the hPKC “setup phase” requires time $\geq T_C$.

- **Discrete logarithm problem (DLP):** An approach initiated by Groth [61] (cf. [97]) and implemented in [31], 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 $\geq T_C$ hence it is non-scalable (according

---

12 This assumption covers only the interactive setting; see discussion in Section 3.3.
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 [31], and a recent improved version is called BulletProofs [35].

- **Interactive Proofs (IP) based**: IP protocols can be performed with zero knowledge [11] 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. [58, 94]. This led to a line of realizations in code, early works lacked ZK [42, 41, 104, 107], but the state-of-the-art ones, like [112] and Hyrax [109], do have it.

Like ZK-STAR{K}, these recent IP-based proofs are transparent and have a scalable prover, but are quantum-susceptible and their verifier is not scalable, as it scales linearly with computation time for “standard” (i.e., sequential) computations (like other approaches, it is quite efficient for batched and amortized computations and for small circuits).

- **Secure multi-party computation (MPC)**: This approach, suggested by Ishai et al. [70] and implemented first in the ZKBoo [55] 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-STAR{K}, the MPC-based proofs are transparent, post-quantum secure and have scalable (quasilinear) proving time. However, MPC based systems have a non-scalable verifier, one that runs in time \( \geq T_C \) and communication complexity is non-scalable, it is \( \sqrt{T_C} \) in the state of the art system [1]; for concrete circuits and amortized computations it is, nevertheless, extremely efficient.

- **Incrementally Verifiable Computation (IVC)**: This approach, suggested by Valiant [105] (cf. [39, 29]) 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-STAR{K}, but thus far has been realized only for a single hPKC system [23].

Compared with ZK-STAR{K}, 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.

<table>
<thead>
<tr>
<th></th>
<th>prover scalability (quasilinear time)</th>
<th>verifier scalability (polylogarithmic time)</th>
<th>Transparency (public randomness)</th>
<th>Post-quantum security</th>
</tr>
</thead>
<tbody>
<tr>
<td>hPKC</td>
<td>Yes</td>
<td>Only repeated computation</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DLP</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IP</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MPC</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IVC+hPKC</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ZK-STAR{K}</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 2: Theoretical comparison of universal (NP complete) realized ZK systems.
1.3.2 Concrete performance

Different ZK proof systems are based on different cryptographic assumptions and are designed for different computational problems. Their realizations are written in different programming languages and tested on varied hardware. Therefore, exact “apples to apples” comparisons are difficult, if not impossible, to perform. Having said that, in this section we attempt to qualitatively compare the realized proof systems reported in the previous section in terms of verifier, prover, and communication complexity for computational problems that are similar in nature to the DPM.

Arithmetic circuit complexity as standard measuring yard  All realized proof systems surveyed here (including our ZK-STARK) use arithmetization to reduce computational integrity (CI) statements to statements about systems of low-degree polynomials over finite fields (see Section 2). All other surveyed systems use prime fields $\mathbb{F}_p$, though some (like MPC- and IP-based) could operate also over binary fields, like ZK-STARK; we stress that ZK-STARK could also operate over prime fields\(^\text{13}\) but we have not realized this in code. Most systems (ZK-STARK not included) reduce CI statements to arithmetic circuits, i.e., ones that correspond to constraints that are quadratic polynomials; ZK-STARK reduces to systems of higher-degree polynomials, e.g., for our DPM benchmark this degree is 8.

Arithmetic circuit complexity is a a reasonable metric to use in order to compare various proof-systems. The main parameters that influence proof-system complexity (and are mentioned in prior works) are circuit depth, circuit width (number of gates in each “level” of the circuit), nondeterministic witness size, and multiplication complexity, i.e., the number of multiplication gates. (Addition complexity is also relevant, but most proof systems are less affected by it.)

Our DPM computation corresponds to an arithmetic circuit with the following parameters, when applied to a database with $n$ entries (see Figure 4):

- circuit depth is $\text{depth}_n = 62 \cdot n$;
- circuit width is $w = 81$;
- witness complexity is $\text{wit}_n = 40 \cdot n$ bytes;
- multiplication complexity is $\text{mult}_n = 1467 \cdot 62 \cdot n = 90954 \cdot n \approx 2^{16.4} \cdot n$.

As discussed at length in Appendix A, we measured the full ZK-STARK system (prover+verifier) with 60 bits of security for $n = 2^k$, $k = 1, \ldots, 20$, i.e., for arithmetic circuits with depth up to $\text{depth}_n \approx 2^{25.9}$, and with up to $\approx 2^{36.4}$ multiplication gates over $\mathbb{F}_{2^{64}}$; the (nonadaptive) verifier alone was measured even for larger inputs, up to $n = 2^{36}$ (see Appendix A).

To attempt an “apples to apples” comparison with other systems, we ran several of them on a single machine — one that is different than that used to measure the DPM code\(^\text{14}\) — using the same benchmark computation based on the “exhaustive subset-sum” computation measured in prior work [13]; the results are summarized by Figure 3. The systems measured thus far this way are

- \textbf{libSNARK} (commit dc78fd, September 7, 2017) with 80-bit security
- \textbf{SCI} (same measurements used in [13]) with 80-bit security

\(^{13}\text{the FRI system requires } p \text{ to contain a sufficiently large multiplicative subgroup of order } 2^{2+O(1)}; \text{ such prime fields abound, as implied by Linnik’s Theorem [80].}\)

\(^{14}\text{The machine used to measure the DPM code was kindly offered to us by Intel™ for a limited time, whereas for the “apples-to-apple” comparison we needed to provide other teams with access to a machine, for long periods of time.}\)
• BCCGP with logarithmic communication complexity and 128-bit security, single threaded (same system used in [31])

• Ligero with 60 bits of security (same system as reported in [1]);

• our ZK-STARK, with 60-bit security ZK-STARK; we estimate prover prover time for 80 bits to be at most 5% longer; cf. Appendix B.5.1.

We now briefly discuss the performance of these (and other prior reported works), focusing on the following complexity parameters: prover time, verifier time, and communication complexity.

**Prover complexity**  Nearly all systems surveyed earlier have prover complexity that scales either linearly or nearly-linearly in computation size. As shown in Figure 3, our ZK-STARK prover is at least $10 \times$ faster than the other measured systems across the full range of compared computations (all systems were tested up to maximal proving time of 12 hours). We hope to perform similar “apples-to-apples” comparisons (i.e., same machine, circuit depth, width and size) with other systems like Hyrax and BulletProofs in future work.

![Figure 3: An “apples-to-apples” comparison of different realized proof systems as function of computation size, measured by number of multiplication gates. 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 [13, Section 3]. The compared systems are SCI (purple x-marks), which lacks ZK, libSNARK (blue triangles), BCCGP (cyan +-marks), executed in single-thread mode, Ligero (red squares) and ZK-STARK (green circles). From left to right, we measure prover time, verifier time and communication complexity. For libSNARK, the hollow marks in the middle and right plots measure only post-processing verification time and CC, respectively; the full marks measure total verification time and CC, and this includes the (non-transparent) key-generation phase. Server specification: 32 AMD cores at clock speed of 3.2GHz, with 512GB of DDR3 RAM. (Each pair of cores shares memory; this roughly corresponds to a machine with 16 cores and hyper-threading.)](image)

**Verifier complexity**  Different proof systems excel on different circuit topologies. For example, Ligero achieves best performance for circuits of size $s$ that are iterated $s$ times (i.e., when depth $\approx w \approx \sqrt{\text{mult}}$), and Hyrax works best on small depth, massively parallel, circuits (depth $= O(1)$ and $w, \text{mult} \gg \text{depth}$). The concrete performance of IOP-based systems on such circuit topologies is an interesting question, left for future work.

For “deep” and “narrow” circuits, like the ones arising from the DPM, verifier arithmetic complexity of prior works scales at least like $\sqrt{\text{mult}}$ (and, often, like mult), whereas our ZK-STARK scales like $w + \log \text{mult}$ (see Theorems 3.4 and 3.5). Consequently, for medium- and large-scale sequential computations our ZK-STARK verifier time is better than other solutions, as shown by the middle plot of Figure 3. We...
expect the comparison with other works, like Hyrax and BulletProofs, to behave similarly; in particular, the Hyrax prover reaches $\approx 10$ seconds for a circuit with $\approx 2^{28}$ gates (but measured on a different machine than ours); BCCGP and BulletProofs require even greater running time [109, Figure 4.(i)]. For comparison, the ZK-STARK verifier for the DPM computation requires less than 50 ms (on a different machine), even for huge circuits\footnote{We stress that current hardware does not support generating proofs for such large instances, as discussed later.}, with $n = 2^{36}$ entries (profiles), $\text{wit}_n \approx 2.5$-terabyte size witnesses and arithmetic circuits with $\text{mulp}_n = 2^{52}$ multiplication gates and depth $\approx 2^{40}$ (cf. Figure 7).

The hPKC systems like Pinocchio and libSNARK, and IVC+hPKC systems like that of [23], are different in this respect. They have a pre-processing phase 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. For the IVC+hPKC system, pre-processing time is constant and does not depend on circuit size; however, this constant is quite large compared to our verifier time, it is $\approx 10$ seconds for a computation similar to our DPM.

**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 system also enjoys extremely short CC and, because its 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, and that of BCCGP is less than 7KB [31] (see also Figure 3). 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 a computation like our DPM.

For Ligero, communication complexity scales like $70\sqrt{\text{mulp}_n}$ field elements [1, Section 5.3], and for Hyrax it scales like $\text{wit}^{1/k} + 10 \cdot \text{depth} \cdot \log w$ field elements for arbitrary integer $k$ [109, Section 1]; increasing $k$ decreases CC but also increases verification time (which is at least $\text{wit}/(\text{wit}^{1/k})$). Using the estimate for Hyrax, a quick calculation shows that for a circuit arising from our DPM computation with, say $n = 2^{13}$ profiles, the CC of Hyrax would reach several megabytes, compared with ZK-STARK CC that is less than 1 megabyte even for $n = 2^{36}$ profiles.

**Summary** Among all ZK systems tested in the “apples-to-apples” manner described above, our ZK-STARK has the fastest prover for all circuit-sizes we were able to measure; 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 computations our ZK-STARK has verification time and communication complexity outperform 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 that are practically relevant, like the DPM, and suggest that our type of system is potentially useful for constructing scalability solutions, e.g., for decentralized crypto-currencies (as discussed in Section 1.2).
2 Methods

This section highlights the main innovative components that underlie the (double) scalability and concrete efficiency of our ZK-STARK; the exposition is short and informal. Later, in Section 3 we shall formally define the theoretical model which our ZK-STARK uses, and state the main theorems for this model (Theorems 3.4 and 3.5); then, in Appendix B we formally present the steps of the reduction (proved in subsequent sections).

Overview Many ZK systems (including ours) use arithmetization, a technique first\(^{16}\) used to prove circuit lower bounds [93, 102], then adopted to interactive proof systems [5, 82]. 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

\[
\alpha \text{ is the result of executing } C \text{ for } T \text{ steps on (public) input } x
\]

Notice the DPM statement (*) is a special case of (**). For our ZK-STARK, and for related prior systems [27, 25, 13], the end point of arithmetization is a pair of Reed-Solomon (RS) proximity testing (RPT) problems\(^{17}\), and the scalability of our ZK-STARK relies on a new solution to the RPT problem, discussed first; later we explain the arithmetization process in more detail.

2.1 Fast Reed-Solomon Interactive Oracle Proof of Proximity (FRI3rd)

For \(S \subset \mathbb{F}\) and rate parameter \(\rho \in (0, 1)\), the Reed-Solomon code \(RS[\mathbb{F}, S, \rho]\) is the family of functions \(f : S \to \mathbb{F}\) that are evaluations of polynomials of degree \(< \rho|S|\). The RPT problem assumes a verifier is given oracle access to \(f\), and to auxiliary information like a probabilistically checkable proof of proximity (PCPP) [25, 48] or an interactive oracle proof of proximity (IOPP) [22, 94, 15]; the verifier’s task is to distinguish with high probability and with a small number of queries to \(f\) and the auxiliary PCPP/IOPP oracle(s), between the case that \(f \in RS[\mathbb{F}, S, \rho]\) and the case that \(f\) is \(0.1\)-far from (all members of) \(RS[\mathbb{F}, S, \rho]\) in relative Hamming distance. Finding solutions to the RPT problem (a special case of the “low-degree testing” problem) is a major bottleneck for transparent systems.

Our ZK-STARK uses a new protocol to solve RPT, called the Fast RS IOPP (FRI). FRI is the first RPT solution to achieve prover arithmetic complexity that is strictly linear — \(6 \cdot |S|\) arithmetic operations in \(\mathbb{F}\) — and verifier arithmetic complexity that is strictly logarithmic: \(21 \cdot \log |S|\) arithmetic operations; additionally, the proof can be constructed in \(\log |S|\) cycles on a parallel machine, and is structured as to lead to short arguments (see Section 2.5). FRI improves significantly, both asymptotically and concretely, on the previous RPT solutions which required quasilinear prover arithmetic complexity (\(\theta(|S| \cdot \log^{O(1)} |S|)\)). See [14] for a detailed description.

2.2 Arithmetization I — Algebraic Intermediate Representation (AIR)

Having discussed its end point, we return to describe the innovative components of our ZK-STARK within the arithmetization process itself. The arithmetization is comprised of several phases that are similar to other

---

\(^{16}\)Earlier reductions, such as the one used in Gödel’s Incompleteness Theorem, involved infinite algebraic domains, in particular the natural numbers [56].

\(^{17}\)The other solutions described in Section 1.3 have different end points.
program and circuit compilation processes, so we borrow terminology used there and adapt it to our process.

The first phase of arithmetization is that of constructing an *algebraic intermediate representation (AIR)* of the program C. Informally, the AIR is a set

\[ \mathcal{P} = \{ P_1(\vec{X}, \vec{Y}), \ldots, P_s(\vec{X}, \vec{Y}) \} \]

of low degree polynomials with coefficients in \( \mathbb{F} \) over a pair of variable sets \( \vec{X} = (X_1, \ldots, X_w) \) and \( \vec{Y} = (Y_1, \ldots, Y_w) \) that represent respectively the current and next state of the computation\(^{18}\) (see Appendix C and Definition B.3 for more details). The AIR defines the *transition relation* of the computation C in the sense that a pair \((\vec{x}, \vec{y}) \in \mathbb{F}^w \times \mathbb{F}^w\) corresponds to a single valid transition (or “cycle”) of C if and only if

\[ P_1(\vec{x}, \vec{y}) = \ldots = P_s(\vec{x}, \vec{y}) = 0, \]

i.e., if and only if \((\vec{x}, \vec{y})\) is a common solution of the AIR system \(\mathcal{P}\). The following parameters of \(\mathcal{P}\) determine prover and verifier complexity, so minimizing them is a major goal of this phase. The *degree* of the AIR is \(\deg(\mathcal{P}) = \max_{i=1}^s \deg(P_i)\); the *state* width is the number of variables \(w\) needed to represent a state; the (AIR) size is the number of constraints \(s\), and the *cycle count* is the number of machine cycles needed to execute\(^{19}\) C; when the program processes a large number \(n\) of data elements, as is the case for the DPM benchmark, we are interested in the number of cycles per element, denoted \(c\); the total cycle count for \(n\) elements is \(c \cdot n\). If the computation is “expanded” to a circuit (as commonly done in the other solutions described in Section 1.3), the cycle count is a lower bound on circuit depth; for the sake of comparison with those other systems, we compute in the rightmost column of Figure 4 the total number of multiplication gates for this expanded circuit, as this measure along with circuit depth, are the complexity measures that dictate prover and verifier complexity.

A major contributor to prover complexity in our benchmarks is the cost of proving computational integrity of repeated invocations of a cryptographic hash function; other computations are negligible compared to this cost. Thus, choice of the particular hash function (H) is of great importance, as is its definition in terms of \(\mathcal{P}\). Our ZK-STARK uses the binary (characteristic 2) field \(\mathbb{F}_{2^64}\) because (i) it has efficient arithmetic operations (e.g., addition is equivalent to exclusive-or) and (ii) its algebraic structure is needed for the FRI3rd protocol. Therefore, the cryptographic hash function we seek is one that is “binary field friendly”, meaning, informally, its AIR has small complexity parameters when defined over binary fields. Figure 4 summarizes the main AIR complexity parameters for the DPM benchmark described in Section 1 and for three hash functions: the Secure Hash Algorithm 2 (SHA2) family \([92]\) and the Davies–Meyer \([111]\) hash based on the Rijndael block cipher \([44]\) with 128 bits (AES128+DM) and with 160 bits (Rij160+DM). See Appendices E and F for details.

### 2.3 Arithmetization II — 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*; we discuss both in Section 2.4. 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 on a dataset of size \(n\) denote this degree by \(d^{\text{max}}(n)\). Prior state-of-the-art \([27, 20, 38, 13]\) gave

\[ d^{\text{max}}_{\text{old}}(n) = n \cdot c \cdot w \cdot d + n \cdot c \cdot s. \]  

\(^{18}\)This informal description omits, for simplicity, the *boundary conditions*, like public inputs and outputs of the computation.

\(^{19}\)In general, this number may depend arbitrarily on the particular input, however, in all our benchmarks it depends linearly on the size \((n)\) of the input dataset.
which leads to concretely large values (see first column of Figure 5). Our ZK-STARK reduces $d^{\text{max}}$ to

$$d_{\text{ZK-STARK}}(n) = n \cdot c \cdot d$$  \hspace{1cm} (4)

which results in a multiplicative savings factor of $6.5 \times 10^4$–$1.8 \times 10^5$ over prior works (see the last two columns of Figure 5). The improved efficiency of our ZK-STARK is due to two reasons, explained next. The first one completely removes the second summand of (3) and the second one removes $w$ from its first summand.

**Algebraic linking IOP (ALI)** The second summand of (3) arises because our prover needs to apply a “local map” induced by the AIR system $P$ (see [17] for a discussion of “local maps”). Prior state-of-the-art systems, like [13], 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 $P_1, \ldots, P_s$. This round of interaction completely removes the second summand of (3).

**Register-based encoding** The naive computation performed by the prover can be recorded by an execution trace, a two-dimensional array with $c \cdot n$ rows and $w$ columns, in which each row represents the state of the computation at a single point in time and each column corresponds to an algebraic register tracked over all $c \cdot n$ cycles. Prior systems, like [13], encoded the full execution trace by a single Reed-Solomon codeword, leading to degree $n \cdot c \cdot w$; this degree is then multiplied by $d$ to account for application of the aforementioned “local map” to the codeword, resulting in the first summand of (3). Our ZK-STARK uses a separate Reed-Solomon codeword for each register, leading to $w$ many codewords, each of lower degree $n \cdot c$. At first glance this tradeoff may seem wasteful, because we now have to solve an RPT problem for each of these $w$ 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 $w$ codewords. The use of a single codeword per register also helps with reducing communication complexity, as explained in Section 2.5 below.

Figure 5 compares the $d^{\text{max}}$ value of our ZK-STARK to that of the prior state of the art [27, 20, 38, 13] and shows a multiplicative reduction factor of $6.5 \times 10^4$–$1.8 \times 10^5$ for the computations discussed in Section 2.2 and Figure 4.

---

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, see Appendix C.3.
<table>
<thead>
<tr>
<th>Function</th>
<th>(d_{\text{max}}^{\text{old}}(1))</th>
<th>(d_{\text{max}}^{\text{ZK-STARK}}(1))</th>
<th>(d_{\text{max}}^{\text{old}} / d_{\text{max}}^{\text{ZK-STARK}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA2</td>
<td>6323922</td>
<td>41382</td>
<td>18491</td>
</tr>
<tr>
<td>AES128+DM</td>
<td>39504</td>
<td>384</td>
<td>6584</td>
</tr>
<tr>
<td>Rij160+DM</td>
<td>49996</td>
<td>464</td>
<td>6896</td>
</tr>
<tr>
<td>DPM</td>
<td>78988</td>
<td>496</td>
<td>10192</td>
</tr>
</tbody>
</table>

Figure 5: The maximal degree (\(d_{\text{max}}\)) as given by formulas (3) and (4), respectively, for the computations discussed in Section 2.2. The last column gives the multiplicative improvement factor of \(d_{\text{max}}^{\text{ZK-STARK}}\) over the prior state of the art.

2.4 Low degree extension and composition degree

Decreasing \(d_{\text{max}}\) affords our ZK-STARK greater scalability. But, eventually, as the input size \(n\) grows, so does \(d_{\text{max}}\). The main bottleneck for prover time and space is the computation of the low degree extension (LDE) of the execution trace, defined next.

**Definition 2.1** (Low degree extension (LDE)). Given finite subsets \(S, S'\) of a field \(\mathbb{F}\) satisfying \(|S'| > |S|\) and a function \(f : S \rightarrow \mathbb{F}\), the low degree extension (LDE) of \(f\) to \(S'\) is the function \(f' : S' \rightarrow \mathbb{F}\) that has the same interpolating polynomial as that of \(f\).

The LDE is typically computed by polynomial interpolation, followed by a polynomial multi-point evaluation step. State of the art algorithms for interpolating and evaluating polynomials over binary fields are known as additive FFT algorithms because they resemble the fast Fourier transform (FFT) [40]. To improve prover scalability, our ZK-STARK uses the recent and novel additive FFT of Lin et al. [79], inspection of this algorithm shows it leads to arithmetic complexity of \(3 \cdot |S'| \log |S'|\) for the sets \(S'\) that our system requires (see Theorem B.2). Prior additive FFTs [36, 106, 51] required \(\Theta(N \log N \log \log N)\) operations; moreover, the memory-access pattern of this encoding algorithm leads to favorable running times compared with prior implementations [13].

Using this additive FFT, which has strictly quasi-linear arithmetic complexity\(^{21}\), we also obtain the first ZK-STARK in which prover complexity is strictly quasi-linear, and verifier complexity is strictly logarithmic, in the size of the execution trace \(c \cdot n \cdot w\); see Lemma B.6.

2.5 Minimizing Authentication Path Complexity (APC) and Communication Complexity (CC)

The largest contributor to communication complexity, and to verifier time and space complexity in ZK-STARK (and prior related works [27, 20, 38, 13]) is the cost of realizing the IOP model via Merkle trees. We now discuss the way our ZK-STARK reduces this cost.

The commit–reveal scheme of Kilian [74] (which uses the “cut-and-choose” method of Brassard et al. [33]) has the prover commit to each oracle by sending the root of a Merkle tree whose leaves are labeled by oracle entries. Recall an IOP involves several oracles, hence also several Merkle trees and several roots/commitments. After the prover has committed to all oracles, the verifier queries these oracles at randomly chosen positions. When the prover reveals the oracle answers to these queries, each answer must be appended with an authentication path proving the query answers are consistent with previously committed Merkle tree roots. Let \(\lambda\) denote the number of output bits of the cryptographic hash function used to construct a Merkle tree in our system; let \(\text{AP}_{\text{total}}\) denote the total number of authentication path nodes in all

\(^{21}\) A function \(g(n)\) is called strictly quasi-linear if \(g(n) = O(n \log n)\), and called strictly logarithmic if \(g(n) = O(\log n)\).
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

$$CC = q_{total} \cdot \log |\mathbb{F}| + AP_{total} \cdot \lambda$$  \hspace{1cm} (5)

In addition to reducing the first summand above by improved soundness analysis, our ZK-STARK also reduces the second summand in two separate ways:

1. The ZK-STARK verifier queries rows of the (LDE of the) execution trace, each row comprised of $w$ field elements that represent the state at some point in the computation (or its LDE). 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 $w$ many paths in prior solutions).

2. The QUERY phase of the FRI protocol queries affine cosets of a fixed subspace. Accordingly, the ZK-STARK prover places each such coset 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).

Finally, to improve running time and further reduce communication complexity, we use the Davies–Meyer hash composed with AES as the hash function for our ZK-STARK commit-reveal scheme (recall AES is part of the instruction set of many modern processors).

2.6 Organization of the remaining sections

The following sections give a full and formal description of our construction. Section 3 formally defines the notion of a ZK-STIK and its realization as a ZK-STARK, and presents the main asymptotic results (Section 3.2); along the way we recall the formal definitions of the IOP model. Appendix B describes the main components used in our construction, and uses these to prove our main results (Appendix B.7). The components are then described in more detail in the remaining sections.
3 On STIKs and STARkS — formal definitions and prior constructions

In this section we state the main theorems that our ZK-STARK realizes (Section 3.2). Along the way we explain what constitutes a ZK-STARK (Section 3.3) and point to earlier relevant works that are variants of it (Section 3.4). We assume familiarity with standard definitions of zero knowledge (ZK) interactive proof (IP) and argument systems [59, 33], probabilistically checkable proofs (PCP) [2] and PCPs of proximity (PCPP) [26, 48], as well as interactive oracle proofs (IOP) [22, 94] and interactive oracle proofs of proximity (IOPP) [16].

A nondeterministic machine $M$ that decides a language $L \in \text{NTIME}(T(n))$ in time $T(n)$ ($n$ denotes instance size) induces a binary relation $R_M$ consisting of all pairs $(\pi, w)$ where $\pi \in L$ and $w$ is a sequence of nondeterministic choices of $M(\pi)$ that lead to an accepting state. In this case we say $R = R_M$ is induced by $L$ and implicitly assume $M$ is fixed and known. The language that we shall be most interested in, is the NEXP-complete computational integrity language\textsuperscript{22}.

**Definition 3.1** (Computational Integrity). The binary relation $R_{CI}$ is the set of pairs $(\pi, w)$ where

- $\pi = (M, x, y, T, S)$ with $M$ a nondeterministic Turing Machine, $x$ and $y$ denote input and output, and $T \geq S$ are integers in binary notation, indicating time and space bounds, respectively
- $w$ is a description of the nondeterministic choices of $M$ on input $x$ that cause it to reach output $y$ within $\leq T$ steps, using a memory tape of size at most $S$ (not including the read-only input tape on which $\pi$ is written).

The computational integrity (CI) language $L_{CI}$ is the projection of the binary relation $R_{CI}$ onto its first coordinate; alternatively, $L_{CI} \triangleq \{\pi = (M, x, y, T, S) \mid \exists w (\pi, w) \in R_{CI}\}$.

The space bound $S$ in the definition above is unneeded, because $S \leq T$. However, IOP systems for space bounded computations ($S = o(T)$) are simpler and, often, concretely more efficient (this holds for our DPM computation). Thus, we treat space-bounded computations separately and dedicate a theorem to it (Theorem 3.4) before treating the more general (and complicated) case of general computational integrity (Theorem 3.5).

### 3.1 Scalable Transparent IOP of Knowledge (STIK)

A STARK, defined later, is a realization of a scalable and transparent IOP of knowledge (STIK), discussed next. We start by recalling the IOP model as defined in [22].

**Definition 3.2** (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 $\epsilon$ is a pair of interactive randomized algorithms $S = (P, V)$ that satisfy the properties below; $P$ is the prover and $V$ is the verifier.

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

\textsuperscript{22}This language is called the “Computationally Sound” language in [85] and the “universal language” in [8]; we choose the name used in [13].
• **completeness** If \((x, w) \in R\) then \(\Pr[\langle P(x, w) \leftrightarrow V(x) \rangle = \text{accept}] = 1\)

• **soundness** If \(x \notin L\) then for any \(P^*\), \(\Pr[\langle P^* \leftrightarrow V(x) \rangle = \text{accept}] \leq \epsilon\)

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

Next, we formally define a ZK-STIK. Most of the work described in later sections is related to constructing a new, and more efficient, ZK-STIK; similarly, Sections 2.1–2.4 describe a ZK-STIK and only Section 2.5 discusses a ZK-STARK. A (ZK-)STIK can be proven to be unconditionally sound, even against computationally unbounded provers; ZK-STARK systems have only computational soundness, against bounded provers, thus require additional cryptographic assumptions, discussed later.

**Definition 3.3** (Scalable Transparent IOP of Knowledge (STIK)). Let \(R\) be a binary relation induced by a nondeterministic language \(L \in \text{NTIME}(T(n))\) for \(T(n) \geq n\) and let \(S = (P, V)\) 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.

• (fully, or doubly) **scalable** if for every instance \(x\) of length \(n\), both of the following hold:
  1. scalable verifier: \(tv(n) = \text{poly}(n, \log T(n), \log 1/\epsilon(n))\)
  2. scalable prover: \(tp(n) = T(n) \cdot \text{poly}(n, \log T(n), \log 1/\epsilon(n))\)

• **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 \(x\) with probability \(p(n) > \epsilon'(n)\), outputs in expected time \(\text{poly}\left(\frac{T(n)}{p(n) - \epsilon'(n)}\right)\) a witness \(w\) such that \((x, w) \in R\).

• **privacy preservation** if there exists a randomized simulator \(\text{Sim}\) that samples (perfectly) the distribution on transcripts of interactions between \(V\) and \(P\), and runs in time \(\text{poly}(T(n))\).

A (fully) scalable and transparent IOP of knowledge will be denoted by STIK. For \(T(n) = \text{poly}(n)\), a privacy-preserving STIK has perfect\(^{23}\) zero knowledge (ZK-STIK) but for \(T(n) = n^{\omega(1)}\) it implies only the weaker notion of a witness indistinguishable proof system (wi-STIK).

A few remarks regarding the definition above:

• **transparency**: Interactive proofs in which the verifier sends only public random coins are known as Arthur Merlin type protocols. The term transparent proof was introduced in [6] and is synonymous to PCP. We choose this term because it adequately reflects the beneficial effect of public randomness on the transparency of the proof system and. Our terminology does not contradict the earlier definition of the term because transparent proofs (and PCPs) are also transparent according to the definition above.

• **scalability**: Scalable provers are called “quasi-linear” in a number of prior works and scalable verifiers are often called “succinct”. We identify both terms into a single one that reflects the beneficial effect of quasi-linear and succinct proof systems.

---

\(^{23}\)We omit the discussion of statistical zero knowledge because all known ZK-STIK systems have perfect zero knowledge.
3.2 Main Theorems

We now state the two main theorems regarding IOP systems, that our ZK-STARK system realizes; the proof of both theorems appears in Appendix B.7. Since our IOP constructions use finite fields, 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 \( \pi \) (see Remark B.1); we use \( tp^F \) and \( tp^F \) to denote arithmetic complexity, assuming the field \( F \) is understood from context.

Our first theorem is quite efficient when applied to space bounded computations, like our DPM, and indeed our implementation realizes the IOP described in the theorem (cf. Appendix B.8). Although we use asymptotic notation below, the algebraic version of this theorem (Lemma B.6), involves only explicit constants, which may be useful in the future for estimating explicit proof parameters and for asymptotic comparisons with other works.

Recall that \( \text{NTIMESpace}(T(n), S(n)) \) is the class of nondeterministic languages that are decidable in simultaneous time \( T(n) \) and space \( S(n) \).

**Theorem 3.4** (ZK-STIK for space bounded computations). Let \( L \) be a language in \( \text{NTIMESpace}(T(n), S(n)) \), \( T(n) \geq n \) 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 function \( \text{err}(n) = 2^{-\lambda(n)} \)

- perfect completeness and soundness error at most \( \text{err}(n) \) for instances of size \( n \)
- knowledge error bound \( \text{err}'(n) = O(\text{err}(n)) \)
- round complexity \( r(n) = \frac{\log T(n)}{2} + O(1) \)
- query complexity \( q(n) = 36(\lambda + 2) \cdot (\log T(n) + S(n) + O(1)) \), each query is an element of a binary field \( F \), \( |F| = 2^n \) for \( n = \lambda + \log T(n) + O(1) \)
- verifier arithmetic complexity \( tv^F(n) = 2n^2 + O(\lambda \cdot S(n) + \log T(n)) \)
- prover arithmetic complexity \( tp^F(n) = O(S(n) \cdot T(n) \cdot \log T(n)) \)
- proof length \( O(T(n) \cdot S(n)) \), measured in field elements.

In particular, for \( S(n) = \text{poly} \log T(n) \), this IOP is fully scalable, i.e., the system is a wi-STIK; if, additionally, \( T(n) = \text{poly}(n) \), then 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 fully scalable for any nondeterministic language, i.e., it can be said to be a universal wi-STIK.

**Theorem 3.5** (wi-STIK for \( \text{NEXP} \)). Let \( L \in \text{NTIME}(T(n)) \), \( T(n) \geq n \) and \( R \) be induced by \( L \). Then \( R \) has a witness-indistinguishable, fully scalable, and transparent IOP of knowledge (wi-STIK) with the following parameters, stated for soundness error function \( \text{err}(n) = 2^{\lambda(n)} \)

- perfect completeness and soundness error \( \text{err}(n) \leq 2^{-\lambda(n)} \) for instances of size \( n \)
- knowledge extraction bound \( \text{err}'(n) = O(\text{err}(n)) \)
- round complexity \( r(n) = \frac{\log T(n)}{2} + O(1) \)
• query complexity $O(\lambda(n) \cdot \log T(n))$, each query is an element of a binary field $F$, $|F| = 2^n$ for $n = \lambda(n) + \log T(n) + \log \log T(n) + O(1)$.

• verifier arithmetic complexity $tv^F(n) = O(\lambda(n) \cdot T(n))$.

• prover arithmetic complexity $tp^F(n) = O(T(n) \log^2 T(n))$.

• proof length $O(T(n) \log T(n))$, measured in field elements.

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

3.3 Scalable Transparent ARgument of Knowledge (STARK) as a realization of STIK

Definition 3.3 refers to the IOP model, in which results can be proved with no cryptographic assumptions. Indeed, most of our contributions, described in following sections (like the FRI protocol), are stated and studied in this “clean” IOP model; and a majority of our engineering work was dedicated to implementing IOP-based algorithms of a STIK system. However, we are not aware of any unconditionally secure IOP realization that is scalable, and theoretical works show that such constructions are unlikely to emerge [54].

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 [74] 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 [85] and Valiant [105] 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. In certain settings the public may view certain blockchains (like Bitcoin’s) as a realization of both (i) an immutable public time-stamping service and (ii) a public beacon of randomness. Under this view, the blockchain can be used to emulate the verifier on an iSTARK system, resulting in smaller communication complexity under better (more standard) cryptographic 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.
3.4 Prior STIK and STARK constructions

The acronyms STIK and (ZK-)STARK may be new to this work, but IOP systems obtaining the properties that define them have been described in the past, as discussed next.

STIK  PCP systems are, by definition, transparent (1-round) IOP systems. The first such system with a scalable verifier was given in the works\(^ {24}\) of Babai et al. [7, 6] and the first fully scalable PCP, i.e., the first STIK construction, appears in the works\(^ {25}\) of Ben-Sasson et al. [25, 20]. The first ZK-STIK for NP appears in the work of Ben-Sasson et al. [17], later extended to a ZK-STIK for NEXP [15].

STARK  The first realization of a STIK system, i.e., the first STARK, appears in the recent work of Ben-Sasson et al. [13]; our current publication describes the first realization of a ZK-STIK and is therefore the first ZK-STARK construction. (See Section 1.3 for other ZK solutions).

Acknowledgements

We thank Eli Eliezer and Tal Shoenwald for providing access to the server used to measure prover performance described in Appendix A.0.1. 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 [31, 1] for assistance in obtaining the data reported in Figure 3. We thank Shmuel (Muli) Ben-Sasson, Yuval Ishai and Uri Kolodny for commenting on an earlier draft.

\(^{24}\)The first work [7] shows this for NEXP and the second [6] scales it down to NP.

\(^{25}\)The first work [25] presents a PCP with scalable verification and quasi-linear proof length, the second work [20] bounds the prover running time and also proves the proof of knowledge property.
References


A Measurements of the ZK-STARK for the DNA profile match

In this section we provide additional raw measurements for proving complexity and verification complexity for the DPM discussed in Section 1. We used two separate machines to measure performance: a strong server for the prover, and a “standard” laptop for the verifier.

A.0.1 Prover

Figure 6 shows the running time required to generate the ZK-STARK proofs, both in absolute terms (left) and as a multiplicative overhead in running time over naïve computation (middle). On the right we plot the size occupied by the all IOP oracles and Merkle trees used by the prover. Due to space limitations (768GB of RAM, henceforth called the “RAM threshold”), the actual space used by the prover was lower than plotted there but saving space required larger running time. Specifically, notice that at \( n = 2^{14} \approx 16,000 \) the total space (plotted on right) passes the RAM threshold, corresponding to (and explaining) the jump in proving time which is noticed on the middle plot at the same value of \( n \). This jump is due to re-computing parts of the proof oracles on demand, which is required to operate within RAM limits.

The code of the prover (written in C++) has been optimized for large instances and running times, hence it uses multi-threading (MT) intensively. Our use of MT seems empirically quite efficient; in particular, disabling MT incurs a slowdown factor close to \( \times 2 \). The relatively large multiplicative overhead noticeable on the middle plot for small instance sizes is likely explained by the overhead that the use of MT introduces. However, since prover execution time is measured in fractions of a second for these short executions we leave further optimizations of it to future work.

Figure 6: On the left we plot, on a double-logarithmic scale, prover running time as a function of the number of entries (\( n \)) in the DNA profile database. On the middle we plot the ratio between proving time to naïve execution time; the horizontal axis is logarithmic (\( \log n \)) and the vertical one measures ratio. On the right we plot, on a double-logarithmic scale, the total size of all oracles and their commitment trees (Merkle trees), generated by the prover during execution. See text inline for an explanation for the “phase transitions” seen in the middle plot.

Prover machine specifications

- CPU (2 units): Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz (24 cores, 2 threads per core)
- RAM: 768GB DDR4 (32GB \( \times 24 \), Speed: 2666 MHz)
- SW AP: 3.2TB NVMe (1.6TB \( \times 2 \))
- Operating System: Red Hat Enterprise Linux 7 (3.10.0-693.5.2.el7.x86_64)
A.0.2 Verifier

Figure 7 gives the ZK-STARK verifier running time ($T_V$) on the left, and communication complexity on the right. The verifier is non-adaptive, which means its complexity can be measured even for databases sizes $n$ that are too large to generate a proof for. The values for which an actual proof was generated are indicated by full circles in both plots.

The ZK-STARK verifier is comprised of two sub-verifiers. The first is the ZK-STIK verifier that verifies proofs in the “pure” but unrealistic IOP model. The second is the sub-verifier that checks (only) consistency of values residing in various Merkle trees with previously committed Merkle tree roots (see Section 2.5). In both plots of Figure 7 the bottom line indicates the complexity of the ZK-STIK verifier, both for time (left) and communication complexity (right). As evident from the Figure, the ZK-STIK complexity is small relative to overall complexity. Moreover, as oracle size increases, the ratio of STIK/STARK complexity grows smaller. This is because as oracles grow larger, the relevant Merkle trees grow deeper and hence there are more authentication paths, and each is of larger length.

We executed the verifier in single thread mode; the tests run by it are amenable to parallelization and faster execution time. However, since verification time is already quite small we leave these further optimizations to future work. Similarly, we point out that the additional memory consumption required by the verifier is negligible, compared to the communication-complexity. In particular, when the verifier is executed on weaker machines than the one reported here (see specification below), verification complexity does not increase significantly.

We executed the verifier in single thread mode; the tests run by it are amenable to parallelization and faster execution time. However, since verification time is already quite small we leave these further optimizations to future work. Similarly, we point out that the additional memory consumption required by the verifier is negligible, compared to the communication-complexity. In particular, when the verifier is executed on weaker machines than the one reported here (see specification below), verification complexity does not increase significantly.

![Figure 7: Verifier execution time (left) and communication complexity (right) as a function of the number of entries ($n$) in the DNA database size; for both plots the horizontal axis is logarithmic and the vertical one is linear. In both plots, the lower line measure the complexity of the ZK-STIK verifier, and the upper line measures that of the ZK-STARK verifier. We stress that verifier measurements are performed for values of $n$ that are (significantly) larger than those for which a proof can be generated; this is possible because our verifier is non-adaptive, thus, we tested it with randomly generated “proofs”. Full circles indicate values of $n$ for which proofs were generated.](image)

Verifier machine specifications

- **Model**: Lenovo ThinkPad W530 Laptop
- **CPU**: Intel(R) Core(TM) i7-3740QM CPU @ 2.70GHz (4 cores, 2 threads per core)
- **RAM**: 32GB DDR3 (8GB × 4, Speed: 1600 MT/s)
- **Operating System**: Arch Linux (4.13.12-1-ARCH)
The purpose of this section is to specify in detail our IOP constructions, expanding on Section 2. The existence of a ZK-STIK system for NEXP was already established in [17, 15]; thus, our focus is on concrete efficiency and on a detailed description of the construction realized in code.

For the purposes of the current discussion, an instance of a computational integrity statement, denoted $x$, is specified by (i) a transition relation over a space of machines states and (ii) a set of boundary constraints (like inputs and outputs). A witness to the integrity of $x$ is a valid execution of the computation, given by an execution trace — a sequence of machine states that adheres to both boundary and transition constraints of the computation. Casting CI statements like (*) in this format is a straightforward application of the Cook-Levin Theorem.

![Figure 8: The reduction from an AIR instance to a pair of RPT problems, solved using the FRI protocol.](image)

**Overview**  
Our process has 4 parts (see Figure 8):

1. The starting point is a natural algebraic intermediate representation (AIR) of $x$ and $w$, denoted $x_{AIR}, w_{AIR}$. By “algebraic” we mean that states of the execution trace are represented by sequences of elements in a finite field $F$, and the transition relation is specified by a set of polynomials over variables representing the “current” and “next” step of the execution trace.

2. 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 (see Appendix B.3). The process of “placing” machine states on nodes of a circuit resembles 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 the transformation as the algebraic placement and routing (APR) reduction, and the resulting representation is an APR instance/witness pair ($x_{APR}, w_{APR}$). This reduction 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. (The prover uses randomness to obtain zero-knowledge; however, this use does not affect completeness, nor soundness.)

3. The APR representation is used to produce, via a 1-round IOP, a pair of instances of the Reed-Solomon proximity testing (RPT) problem. The instances are defined now by the parameters of the RS code. The two codes resulting from the reduction are over the same field $F$ but have different evaluation domains ($L, L_{cmp}$) and different code rates ($\rho, \rho_{cmp}$). The witness in this case is a pair of purported codewords $(f^{(0)}, g^{(0)})$. The verifier’s randomness in the 1-round IOP is used (among other things) to “link” the numerous constraints of the transition relation into a single (random) one. We thus refer to this step as the algebraic linking IOP (ALI) protocol.
Finally, for each of the two functions (oracles) \( f^{(0)}, g^{(0)} \), we invoke the fast RS IOP of proximity (FRI) protocol from [14], and this completes our reduction.

**Section organization** After setting up notation (Appendix B.1) we formally define our starting points — the algebraic intermediate representation (AIR) of CI statements (Appendix B.2); this section also includes the main technical lemmas (Lemmas B.6 and B.9) that prove our main Theorems 3.4 and 3.5, as well as the lemma on which our ZK-STARK realization relies (Lemma B.7). In Appendix B.3 we formally define the representation of CI statements that is the output of the algebraic placement and routing (APR) reduction (Definition B.10) and the APR-reduction (leftmost arrow of Figure 8) appears in Appendix B.4. Appendix B.5 describes the next step of the reduction (middle arrow of Figure 8) starting with Definition B.14 of the RS proximity testing (RPT) problem and followed by the algebraic linking IOP (ALI) stated in Theorem B.15 and discussed later there. Appendix B.6 states the main properties of the FRI protocol that solves the RPT problem (rightmost arrow of Figure 8). Appendix B.7 uses the components defined earlier — AIR, APR, ALI, and FRI— to prove our main Theorems 3.4 and 3.5. Appendix B.8 concludes with a discussion of the particular setting of parameters that our ZK-STARK uses for the DPM computation.

**B.1 Preliminaries and notation**

**Density** The density of a subset \( S' \) of a finite set \( S \) is \( \mu(S'/S) \triangleq |S'|/|S| \).

**Sets and functions** We denote by 0 the constant zero function. For \( S \) a set and \( f \) a function with domain \( S \) let \( f(S) = \{ f(x) \mid x \in S \} \) and for \( F \) a set of functions with domain \( S \) let \( F(S) = \cup_{f \in F} f(S) \).

**Binary fields** A finite field is denoted by \( \mathbb{F} \) and \( \mathbb{F}_q \) is the field of size \( q \). A binary field is a finite field of characteristic 2; all fields considered in this paper are binary\(^{26}\). When we use the term affine space for a subset \( H \subset \mathbb{F} \) that is an additive coset of an \( \mathbb{F}_2 \)-linear space, meaning \( H = \alpha + V \triangleq \{ \alpha + v \mid v \in V \} \) for some fixed \( \alpha \in \mathbb{F} \) and \( V \subseteq \mathbb{F} \) a linear space over the two-element field \( \mathbb{F}_2 \).

**Remark B.1** (Canonical representations of binary fields). We assume canonical representation for binary fields \( \mathbb{F} \), given by an irreducible polynomial and a primitive element \( g \in \mathbb{F} \) for it (i.e., \( g \) generates \( \mathbb{F}^* \)). We use the standard basis \( \{ 1, g, g^2, \ldots, g^{n-1} \} \) to represent \( \mathbb{F}_{2^n} \) over \( \mathbb{F}_2 \). Similarly, for a fixed \( \mathbb{F}_2 \)-subspace of dimension \( k \), the (non-zero) polynomial of degree \( 2^k \) that vanishes on \( S \) will assumed to be known to the verifier; recall this polynomial is linearized (see [78, Section 3.4]) and has at most \( k \) non-zero coefficients.

In particular, when discussing arithmetic complexity of prover and verifier, we assume the representations of all relevant fields, primitive elements and subspace polynomials are known to both parties. As will become evident, these depend only on the parameters of the computation (like degree and running time). This assumption can be made with no significant loss in generality, because the verifier could request the prover to present all such representations, along with a suitable “proof of primitivity” for \( g \).

**Codes** We view codewords in a linear error correcting code \( C \) over \( \mathbb{F} \) of blocklength \( n \) as functions with domain \( S, |S| = n \) and range \( \mathbb{F} \), i.e., \( C \subseteq \mathbb{F}^S \); for \( x \in S \), the \( x \)-entry of a codeword \( w \in C \) is denoted by \( w(x) \). For \( v \in \mathbb{F}^S \) let \( w_v \in C \) be a codeword that is closest to \( v \) in relative hamming distance, breaking

\(^{26}\text{Many of the results here apply, with minimal modifications, to arbitrary fields of small characteristic. For the sake of simplicity we avoid dealing with this generalization.}\)
ties arbitrarily (say, by ordering the elements of \( C \) arbitrarily and setting \( w_v \) to be the smallest \( w \in C \) with minimal distance to \( v \)).

**Restrictions** For \( S' \subseteq S \) we denote by \( w|_{S'} \) the restriction of \( w \) to sub-domain \( S' \) and similarly define \( C|_{S'} = \{ w|_{S'} \mid w \in C \} \), noticing \( C|_{S'} \subseteq \mathbb{F}^S \).

**Interpolants, evaluations and low degree extensions** The interpolant of \( w \in \mathbb{F}^S \), denoted \( \text{interpolant}^w \), is the unique polynomial \( P, \deg(P) < |S| \) satisfying \( \forall x \in S, P(x) = w(x) \). The multi-point evaluation of \( P(X) \) on domain \( S \) is the function \( f : S \to \mathbb{F} \) satisfying \( f(x) = P(x) \) for all \( x \in S \); when \( |S| > \deg(P) \), the interpolant of this evaluation is \( P \).

Recall the definition of a low degree extension (LDE) from Definition 2.1. We use the following efficient LDE algorithm of Lin et al. [79].

**Theorem B.2 (LDE arithmetic complexity[79]).** For \( \mathbb{F} \) a binary field, \( S \subset \mathbb{F} \) an \( \mathbb{F}_2 \)-affine space and \( c_1, \ldots, c_n \in \mathbb{F} \) there exists an arithmetic circuit computing an advice string \( A \) that requires \( (n+1)\text{polylog}|S| \) arithmetic operations in \( \mathbb{F} \), and another arithmetic circuit that uses the advice \( A \) and computes the LDE of a function \( f : S \to \mathbb{F} \) to domain \( \bigcup c_i S + c_i \). This latter circuit has arithmetic complexity \( 3(n+1)(|S| \log |S|) \) over \( \mathbb{F} \).

For simplicity we shall ignore the complexity of advice computation, which in negligible compared to \( 3(n+1)(|S| \log |S|) \).

**Arithmetic complexity** For an arithmetic circuit \( C \) with gates of fan-in \( \leq 2 \) over the set of gates \( \{+ , \times, \div\} \), we denote by \( T_{\text{arith}}(C) \) the arithmetic complexity of \( C \), defined as the total number of gates in the arithmetic circuit. The multiplication complexity is the number of \( \times \)-gates and the addition complexity is similarly defined. For a function \( f : \mathbb{F}^n \to \mathbb{F}^m \) whose “canonical circuit” \( C_f \) is implicitly known, we abuse notation and define \( T_{\text{arith}}(f) = T_{\text{arith}}(C_f) \).

Given a set of functions \( \mathcal{F} = \{ f_1, f_2, \ldots, f_m \} \) mapping elements of \( \mathbb{F}^n \) to \( \mathbb{F} \), we denote by \( T_{\text{arith}}(\mathcal{F}) \) the (total) arithmetic complexity of \( \mathcal{F} \) and define it as the the arithmetic complexity of a circuit \( C : \mathbb{F}^n \to \mathbb{F}^m \) such that for any \( \bar{v} \in \mathbb{F}^n \) and any \( 1 \leq i \leq m \), \( C(\bar{v})_i = f_i(\bar{v}) \).

**Functions evaluated on sets of inputs** For \( f : S \to \Sigma \) a function and \( S' \subset S \) let \( f(S') = \{ f(x) \mid x \in S' \} \); when \( f(S') \) is a singleton set \( \{ \alpha \} \) we shall simplify notation and write \( \alpha = f(S') \).

**B.2 Algebraic Intermediate Representation (AIR)**

An algebraic execution trace of a program running for \( T \) steps is represented by a \( w \times T \) array in which each entry is an element of a finite field. A single row describes the state of the computation at a certain point in time, and a single column tracks a “register” and its contents over time. It is straightforward to reduce (instance,witness) pairs for a relation that defines a language \( L \in \text{NTIME} \text{Space}(\hat{T}(n), \hat{w}(n)) \) to AIR instances with algebraic execution traces of size \( T(n) = O(\hat{T}(n)) \) and width \( w(n) = O(\hat{w}(n)) \), see, e.g., [90] for examples of such reductions.

**Definition B.3 (Algebraic internal representation (AIR)).** The relation \( R_{\text{AIR}} \) is the set of pairs \( (\bar{z}, \bar{w}) = (z_{\text{AIR}}, w_{\text{AIR}}) \) satisfying

1. **Instance Format:** the instance \( \bar{z} \) is a tuple \( \bar{z} = (F, T, w, P, C, B) \) where
• $\mathbb{F}$ is a finite field
• $T$ is an integer representing a bound on running time
• $w$ is an integer representing state width
• $\mathcal{P} = \{P_1, \ldots, P_3\} \subset \mathbb{F}[X_1, \ldots, X_w, Y_1, \ldots, Y_w]$ is a set of constraints. The degree of $\mathcal{P}$ is $\deg(\mathcal{P}) \triangleq \max_{P \in \mathcal{P}} \deg(P)$
• $C$ is a monotone boolean circuit over variables $Z_1, \ldots, Z_s$ with multi-input AND and OR gates. The size of the circuit is the number of gates in it, and its degree, denoted $\deg(C)$ is defined inductively thus:
  - the degree of the input variable $Z_i$ is $\deg(Z_i) = \deg(P_i)$, where $P_i \in \mathcal{P}$ is defined above;
  - the degree of an AND gate $g_j$ with input gates $g_1, \ldots, g_t$ is $\deg(g_j) = \max \{\deg(g_1), \ldots, \deg(g_t)\}$
  - the degree of an OR gate $g_j$ with input gates $g_1, \ldots, g_t$ is $\deg(g_j) = \deg(g_1) + \ldots + \deg(g_t)$
  - finally, $\deg(C)$ is the degree of its (single) output gate.
• $B$ is a set of boundary constraints, where each boundary constraint is a tuple $(i, j, \alpha)$ for $i \in [T], j \in [w], \alpha \in \mathbb{F}$

2. Witness Format: The witness $\mathfrak{w}$ is a set of functions $w_1, \ldots, w_w : [T] \to \mathbb{F}$; we say $\mathfrak{w}$ satisfies the instance if and only if

(a) For all boundary constraints $(i, j, \alpha)$ we have $w_j(i) = \alpha$

(b) For all $t \in [T - 1]$ and we have $C(\text{IsZero}_w(\mathcal{P}(w[t], w[t + 1]))) = \text{TRUE}$ where
  - $w[t] = (w_1(t), \ldots, w_w(t))$
  - $\text{IsZero} : \mathbb{F} \to \{\text{TRUE}, \text{FALSE}\}$ is the mapping that sends $0 \in \mathbb{F}$ to $\text{TRUE}$ and all non-zero elements to $\text{FALSE}$, and $\text{IsZero}_k : \mathbb{F}^k \to \{\text{TRUE}, \text{FALSE}\}$ is the natural extension of $\text{IsZero}$ to multiple inputs and outputs, namely, $\text{IsZero}_k(\alpha_1, \ldots, \alpha_k) : = (\text{IsZero}(\alpha_1), \ldots, \text{IsZero}(\alpha_k))$

Finally, $R_{\text{AIR}}$ is the set of all pairs $(\mathfrak{x}, \mathfrak{w})$ such that $\mathfrak{w}$ satisfies $\mathfrak{x}$ and $L_{\text{AIR}} \triangleq \{\mathfrak{x} \mid \exists \mathfrak{w}, (\mathfrak{x}, \mathfrak{w}) \in R_{\text{AIR}}\}$.

Remark B.4 (Generalization to non-monotone circuits). The reason that we restrict the definition above to monotone circuits, as opposed to general circuit which are more expressive, is that adding negation gates to the boolean circuit may destroy perfect completeness, reduce efficiency, or require more rounds of interaction (depending on the way negation gates are arithmetized). Often (as is the case with our benchmarks here), it suffices to use De Morgan’s laws to effectively push negation gates to the inputs of $C$ and incorporate them into the system of polynomials $\mathcal{P}$.

Our reductions will use $\text{AIRs}$ in which $\mathbb{F}$ is invariably a binary field, and we shall further assume the witness size and degree are “binary friendly” according to the following definition.

Definition B.5 ($\text{BAIR}$). A binary $\text{AIR}$ ($\text{BAIR}$) instance is an $\text{AIR}$ instance $\mathfrak{x} = (\mathbb{F}, T, w, \mathcal{P}, C, B)$ satisfying for some $n, t, d \in \mathbb{N}^+$ (i) $|\mathbb{F}| = 2^n$; (ii) $T = 2^t - 1$; and (iii) $\deg(C) \leq 2^d$. We call $\mathfrak{x}$ an $(n, t, d)$-BAIR instance. The relations $R_{\text{BAIR}}$ and language $L_{\text{BAIR}}$ are the natural restriction of $R_{\text{AIR}}$ and $L_{\text{AIR}}$ to binary $\text{AIR}$ instances.

The following lemma implies Theorem 3.4, as shown in Appendix B.7. Its statement is “information theoretic”, i.e., relies on no unproven cryptographic assumptions and holds against computationally unbounded provers. In contrast to prior $\text{ZK-IOP}$ protocols [22, 15], it fully specifies all constants and uses no asymptotic notation.
Lemma B.6 (wi-IOP of knowledge for R_{BAIR}). For soundness error parameter $\lambda : \mathbb{N}^+ \rightarrow \mathbb{N}^+$, rate integer $R \geq 2$, and ZK parameter $k \geq 1$, let $\delta = (1 - 2^{-R} (1 + 2^{-d}))$. Let relation $R_{AIR}^{(\lambda, R, k)}$ denote the restriction of $R_{AIR}$ to $(n, t, d)$-BAIR instances that satisfy (i) $n > \lambda + t + d + R + k + 2$ and (ii) $k + t \geq 6 + \log \lambda + \log \log \frac{1}{1 - \frac{1}{10}}$. Then $R_{AIR}^{(\lambda, R)}$ has a ZK-IOP of knowledge with the following parameters for $(n, t, d)$-BAIR instances:

- soundness error at most $err = 2^{-\lambda}$
- knowledge error bound at most $\epsilon' = 4 \cdot err$
- prover arithmetic complexity
  \[
  tp^F \leq (9w(t + d + R + 3) + |P| + w + T_{arith}(P) + 12) \cdot 2^{t+d+R+3},
  \]
- verifier arithmetic complexity
  \[
  tv^F \leq 2\left(|B|^2 + \left\lfloor \frac{\lambda + 2}{\log \frac{1}{1 - \frac{1}{10}}} \right\rfloor \cdot (8w + |\Phi| + T_{arith}(P) + |B|) + 21(k + R + t + d + 2)\right),
  \]
- round complexity $r = \frac{t+d}{2} + 2$,
- query complexity $q \leq 8\left\lfloor \frac{\lambda + 2}{\log \frac{1}{1 - \frac{1}{10}}} \right\rfloor \cdot (2w + k + t + d + 2)$

We conjecture that the soundness (of two separate components) in the IOP above is not tight. Consequently, we conjecture that the same soundness as above can be obtained with fewer queries, as stated next.

Lemma B.7 (wi-IOP for R_{BAIR} with improved conjectured soundness). Assuming Conjectures B.17 and B.19, the following holds. For soundness error parameter $\lambda : \mathbb{N}^+ \rightarrow \mathbb{N}^+$, rate integer $R \geq 3$, and ZK parameter $k \geq 1$, let relation $R_{AIR}^{(\lambda, R, k)}$ denote the restriction of $R_{AIR}$ to $(n, t, d)$-BAIR instances that satisfy (i) $n > \max\{58, \lambda, t + d + R + k\} + 2$ and (ii) $k + t \geq 5 + \log \left\lfloor \frac{\lambda + 2}{R} \right\rfloor$. Then $R_{AIR}^{(\lambda, R)}$ has a ZK-IOP of knowledge with the following parameters for $(n, t, d)$-BAIR instances:

- soundness error and round complexity as stated in Lemma B.6,
- verifier arithmetic complexity
  \[
  tv^F \leq 2|B|^2 + \left\lfloor \frac{\lambda + 2}{R} \right\rfloor \cdot (16w + |\Phi| + T_{arith}(P) + 42 \cdot (k + R + t + d + 2))
  \]
- query complexity $q \leq 4 \cdot \left(\left\lfloor \frac{\lambda + 2}{R + d} \right\rfloor \cdot (w + 3 + k + t + d) + \left\lfloor \frac{\lambda + 2}{R} \right\rfloor \cdot (3w + 3 + k + t + d)\right)$
- prover arithmetic complexity
  \[
  tp^F \leq \left(9w(t + d + R + 3) + |P| + w + T_{arith}(P) + 12\left\lfloor \frac{\lambda + 2}{n - (10 + 2 \log n)} \right\rfloor\right) \cdot 2^{t+d+R+3}
  \]
Definition B.3 assumes that the full machine state is captured by \( w \) field elements. Certain computations require as much space complexity as time complexity \( (w = \Omega(T)) \), leading to AIR witnesses of total size \( \Omega(w \cdot T) = \Omega(T^2) \). To reach witnesses of size \( O(T \log T) \), irrespective of memory consumption, we require the following definition. It follows the approach of [19, 20] and uses a pair of AIR constraints — one for verifying the validity of consecutive time steps and another for verifying memory consistency; the latter set of constraints is applied to a permutation of the steps of the execution trace, and part of the witness is a specification of this permutation. We call this relation the permuted algebraic intermediate representation \( (R_{PAIR}) \).

**Definition B.8 (Permuted algebraic intermediate representation (PAIR)).** The relation \( R_{PAIR} \) is the set of pairs \( (x, w) = (x_{PAIR}, w_{PAIR}) \) satisfying

1. **Instance Format:** the instance \( x \) is a tuple \( x = (\mathcal{F}, T, w, \mathcal{P}_T, \mathcal{P}_\pi, \mathcal{B}) \) where
   - \( \mathcal{F} \) is a finite field
   - \( T \) is an integer representing a bound on running time
   - \( w \) is an integer representing state width
   - \( \mathcal{P}_T, \mathcal{P}_\pi \subset \mathbb{F}[X_1, \ldots, X_w, Y_1, \ldots, Y_w] \) are two sets of constraints.
   - \( \mathcal{B} \) is a set of boundary constraints, where each boundary constraint is a tuple \( (i, j, \alpha) \) for \( i \in [T], j \in [w], \alpha \in \mathbb{F} \)

2. **Witness Format:** The witness \( w \) is a pair \((\hat{w}, \pi)\) where \( \hat{w} \) is a sequence of \( w \) functions \( w_1, \ldots, w_w: [T] \to \mathcal{F} \), and \( \pi: [T] \to [T] \) is a permutation; we say \( w \) satisfies the instance if and only if
   
   (a) For all boundary constraints \( (i, j, \alpha) \) we have \( w_j(i) = \alpha \)
   
   (b) For all \( t \in [T - 1] \) and for all \( P \in \mathcal{P}_T \) we have \( P(w[t], w[t + 1]) = 0 \)
   
   (c) For all \( t \in [T - 1] \) and for all \( P \in \mathcal{P}_\pi \) we have \( P(w[t], w[\pi(t)]) = 0 \)

Finally, \( R_{PAIR} \) is the set of all pairs \( (x, w) \) such that \( w \) satisfies \( x \), and \( PAIR \triangleq \{ x \ | \ \exists w, (x, w) \in R_{AIRwRAM} \} \).

We define the sub-relation of binary PAIR \( (BPAIR) \) analogously to the definition of binary AIR in Definition B.5.

\( L_{PAIR} \) is NEXP-complete, as stated next. We omit the proof, which appears, e.g., in [90].

**Lemma B.9 (wi-STIK for BPAIR).** For every language \( L \in \text{NTIME}(T(n)), T(n) \geq n \) there exists a deterministic polynomial time reduction from \( L \) to \( L_{PAIR} \), mapping an instance \( x \) of \( L \) to an instance \( x = (\mathcal{F}, T, w, \mathcal{P}_T, \mathcal{P}_\pi, \mathcal{B}) \) of \( L_{PAIR} \) that satisfies

- \( T = O(T(n)) \),
- \( w = O(\log T(n)) \),
- \( \deg(\mathcal{P}_T \cup \mathcal{P}_\pi) = O(1), |\mathcal{P}_T \cup \mathcal{P}_\pi| = O(\log T(n)), \) and \( T_{\text{arith}}(\mathcal{P}) = O(\log T(n)) \),
- \( |\mathcal{B}| = O(n) \).

Furthermore, a nondeterministic witness \( w \) for \( x \) can be computed in time \( O(T(n) \log T(n)) + n^{O(1)} \), given \( x \) and a nondeterministic witness \( w \) for the membership of \( x \) in \( L \).
B.3 Algebraic placement and routing (APR)

The phase in circuit design known as placement and routing deals with laying out states of a computation as to optimize certain physical constraints. Our next phase does something similar, placing and routing information about the states of a computation in a way that optimizes algebraic, rather than physical, constraints; hence we call it the algebraic placement and routing (APR) binary relation.

Recall that $\text{Aff}_1(\mathbb{F})$ denotes the 1-dimensional affine group over $\mathbb{F}$, its set of elements is isomorphic to $\{aX + b \mid a \in \mathbb{F}^*, b \in \mathbb{F}\}$. An affine graph generated by a set $\mathcal{N} = \{N_1, \ldots, N_s\} \subset \text{Aff}_1(\mathbb{F})$ is the directed graph with vertex set $\mathbb{F}$ and edge set $\{(x, N(x)) \mid x \in \mathbb{F}, N \in \mathcal{N}\}$.

In the following definition, the set $\Phi$ captures both boundary, and transition, constraints of the computation. The affine graph is added to the instance description via its generating set $\mathcal{N}$. Due to the algebraic topology, each column of the algebraic execution trace (corresponding to a register, tracked over time) is now a Reed Solomon codeword of rate $\rho$ and the sequence of rates (one rate per register/column) is also part of the instance description. As explained in Section 2.3, our construction differs from prior works [27, 20, 38, 13] by using a separate codeword for each algebraic register.

**Definition B.10 (Algebraic placement and routing (APR) problem).** The relation $R_{\text{APR}}$ contains all pairs $(x, \hat{w})$ satisfying the following requirements:

1. **Instance Format:** the instance $x$ is a tuple $x = (\mathbb{F}, \mathcal{T}, \mathcal{N}, \Phi, L, L_{\text{cmp}}, \tilde{\rho}, \rho_{\text{cmp}})$ where
   - $\mathbb{F}$ is a finite field of characteristic 2.
   - $\mathcal{T}$ is a set of indices called the algebraic-register indices.
   - $\mathcal{N} \subseteq \mathcal{T} \times \text{Aff}_1(\mathbb{F})$ is a set of pairs called neighbors, each pair contains an algebraic-register index, and a member of the affine group over $\mathbb{F}$ (a polynomial of degree exactly 1). Given a set $S \subset \mathbb{F}$ we denote by $\mathcal{N}(S)$ the set $\mathcal{N}(S) := \{N(x) \mid x \in S, (\tau, N) \in \mathcal{N} \text{ for some } \tau \in \mathcal{T}\}$.
   - $\Phi \subseteq (\mathbb{F} \times \mathbb{F}^N)^\mathcal{T}$ is a set of mappings over the variables $\mathcal{V} := \{X_{\text{loc}}\} \cup \{X_N\}_{N \in \mathcal{N}}$. Assignments to any $\phi \in \Phi$ expressed by mappings $\alpha : \mathcal{V} \to \mathbb{F}$. Given a set $S \subset \mathbb{F}$ and a sequence of functions indexed by elements of $\mathcal{T}$, $f \in (\mathbb{F}^N(S))^{\mathcal{T}}$, denote by $\alpha_{f,S} : S \to \mathbb{F}^\mathcal{V}$ the mapping defined by $\alpha_{f,S}^{X_{\text{loc}}}(x) := x$, and for every $(\tau, N) \in \mathcal{N}$, $\alpha_{f,S}^{X_N}(x) := f_{\tau}(N(x))$. Given $\phi \in \Phi$ we denote by $\phi_{f,S}^{N} : f \in \mathbb{F}^S$ the vector defined by $\phi_{f,S}^{N} : x := \phi(\alpha_{f,S}^{X_N}(x))$ for every $x \in S$.
   - $L$ and $L_{\text{cmp}}$ are two $\mathbb{F}^2$-affine subspaces of $\mathbb{F}$ called the witness evaluation space and the composition evaluation space respectively;
   - $\tilde{\rho} \in (0,1)^\mathcal{T}$ is a sequence of rates called the witness rates, and $\rho_{\text{cmp}} \in (0,1)$, called the composition rate.

2. **Witness format and satisfiability:** A witness $\hat{w}$ is a sequence of functions indexed by elements from $\mathcal{T}$; formally, $\hat{w} \in (L^\mathcal{T})^\mathcal{T}$. We say $\hat{w}$ satisfies $x$ if both of the following hold:
   - assignment code membership: $\forall \tau \in \mathcal{T} : w_{\tau} \in \text{RS}[\mathbb{F}, L, \rho_{\tau}]$
   - constraint code membership: $\forall \phi \in \Phi, \phi_{N}^{N} \hat{w} \in \text{RS}[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}]$

Let $R_{\text{APR}}$ be the binary relation containing all pairs $(x, \hat{w})$ such that $x$ is an instance as defined above and $\hat{w}$ satisfies $x$, and let $\text{APR}$ be the nondeterministic language induced by $R_{\text{APR}}$.

$$\text{APR} = \{x \mid \exists \hat{w}, (x, \hat{w}) \in R_{\text{APR}}\}.$$  

(6)
The three properties defined next are used later to prove zero knowledge and to provide better soundness bounds.

**Definition B.11 (APR properties).** We say an instance \( \zeta := (\mathbb{F}, \mathcal{T}, \Phi, L, L_{\text{cmp}}, \rho, \hat{\rho}_{\text{cmp}}) \in \text{APR} \)

1. is \( \kappa \)-independent if there is a sequence of sets \( \{V_\tau\}_{\tau \in \mathcal{T}} \) such that each \( V_\tau \subset \mathbb{F}^L \) is \( \kappa |L| \)-independent, and for any \( \hat{w} \) such that \( \hat{w}_\tau \in V_\tau \), it holds that \( \hat{w} \) satisfies \( \zeta \). Let \( T_{\text{Sampling}}(\cdot) \) denote the arithmetic complexity of the circuit, that on input \( \hat{w}_\tau \in V_\tau \) (and random field elements), samples a uniformly random \( w \in \{V_\tau\}_{\tau \in \mathcal{T}} \).

2. has \( \delta \)-distance if:
   - \( \delta \leq 1 - \rho_{\text{cmp}} \).
   - \( \delta \leq 1 - \rho_{\tau} \) for any \( \tau \in \mathcal{T} \).
   - There exists some linear code \( C \subset \mathbb{F}^{L_{\text{cmp}}} \) of relative distance at least \( \delta \), containing the code \( RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \), such that for any sequence of mappings \( f \in (\mathbb{F}^L)^T \) where \( f_\tau \in RS[\mathbb{F}, L, \rho_{\tau}] \) for all \( \tau \in \mathcal{T} \), it holds \( \forall \phi \in \Phi, \phi_N[\hat{w}] \in C \).

3. is \( \Theta \)-overlapping if for any \( y \in L \) the size of the set \( S_y := \{ x \in L_{\text{cmp}} \mid y \in \mathcal{N}(\{x\}) \} \) is at most \( \Theta \).

**B.4 APR reduction**

The following pair of theorems describe the first step of our reduction, in which a BAIR, or BPAIR, instance-witness pair, is reduced to an APR instance-witness pair. The reductions are deterministic and have no error associated with them. Since the instance-side reduction is carried out by the verifier, and the witness-side reduction by the prover, we denote the two reductions of Theorem B.12 by \( V^{\text{BAIR}\rightarrow\text{APR}}, p^{\text{BAIR}\rightarrow\text{APR}} \), respectively, and those of Theorem B.13 are denoted \( V^{\text{BPAIR}\rightarrow\text{APR}}, p^{\text{BPAIR}\rightarrow\text{APR}} \).

The ideas used in both reduction are based on ideas from [91, 25, 27, 19, 38] where affine graphs are used to encode and verify program traces. The current construction is most similar to the one used in [13], where a cyclic graph is used to embed known to verifier order of the trace, usually used to verify two consecutive configurations are consistent, and a back-to-back De Bruijn graph to verify a permutation not known to the verifier, usually used to verify the memory consistency. The novelty in the construction in [13] in as opposed to previous construction is (i) the usage of a cyclic graph for consecutive configurations consistency testing, (ii) using back-to-back butterfly De Bruijn of to represent permutations instead of a ‘straight De Bruijn graph which is \( \times 2.5 \) bigger, and (iii) selectively routing on the De Bruijn graph only a small part of the configuration. All those reduce the size of the witness, effectively reducing the degrees of polynomials handled by the proof system, and the concrete efficiency of the prover. The novel technique in this work uses the techniques in [13], and additionally splits the witness to several polynomials, each of degree much smaller then what needed to represent the whole witness, effectively reducing further more the degree of polynomials handled by the proof system. As described in Appendix D.1, the proof-system protocol uses interactivity to eliminate the need to provide separate proof of RS-proximity to each witness polynomial. The reductions use the ZK-IOP construction introduced [17], preparing the APR instance and witness for the zero-knowledge protocol described in Appendix D.1.

**Theorem B.12 (Algebraic placement and routing).** There are two algorithms \( V^{\text{BAIR}\rightarrow\text{APR}}, p^{\text{BAIR}\rightarrow\text{APR}} \) such that for any \( k, \kappa, t, d \in \mathbb{N}^+ \) and BAIR instance \( \zeta = (\mathbb{F}, T, w, P, C, B) \) with \( T = 2^t - 1 > 4, \deg P \leq 2^d, \) and \( |\mathbb{F}| = 2^n \) with \( n > 2 + k + \kappa + t + d \):
1. Perfect completeness: \((x, w) \in R_{AIR} \Rightarrow (V^{BAIR\rightarrow APR}(x, k, R), P^{BAIR\rightarrow APR}(x, w, k, R)) \in R_{APR}\)

2. Perfect soundness: \(x \in AIR \Leftrightarrow V^{BAIR\rightarrow APR}(x, k, R) \in APR\)

3. Knowledge extraction: There is an efficient knowledge extractor \(E^{BAIR\rightarrow APR}\) such that for every \((V^{BAIR\rightarrow APR}(x, k, R), \omega_{APR}) \in R_{APR}\), the extractor outputs a witness \(w \leftarrow E^{BAIR\rightarrow APR}(x, k, R, \omega_{APR})\) such that \((x, w) \in R_{AIR}\).

4. Let \(\varepsilon_{APR} = (F, T, N, \Phi, L, L_{cmp}, \rho_{cmp}) = V^{BAIR\rightarrow APR}(x, k, R);\)

   (a) \(|T| = w\)
   (b) \(|N| = 3w\), with same 3 neighboring polynomials for each witness index \(\tau \in T\)
   (c) \(|\Phi| \leq 2|P|\) and \(T_{arith} (\Phi) \leq 2T_{arith} (P) + 3|P| + O (w \cdot |P| \cdot |B|)\)
   (d) \(\log_2 |L| = 2 + k + R + t + d\)
   (e) \(\log_2 |L_{cmp}| = k + R + t + d\)
   (f) The maximal rate \(\rho_{max}\) in \(\rho\) satisfies \(\rho_{max} \leq 2^{-(2+R+d)}\)
   (g) \(\rho_{max} \leq 2^{-R}\)
   (h) \(\varepsilon_{APR}\) is \((1 - 2^{-R}(1 + 2^{-d}))\)-distance, 1-overlapping, and \((2^{-(2+R+d)}(1 - 2^{-k}))\)-independent with \(T_{Sampling} (\varepsilon_{APR}) \leq O (|L|)\)

5. Arithmetic complexity over \(F\) (cf. Remark B.1):

   (a) **Verifier:** The arithmetic complexity of \(V^{BAIR\rightarrow APR}(x, k, R)\) is that which is required to compute the polynomials \(Z_{B,j}, \varepsilon_{B,j} \in F[x]\) for all \(1 \leq j \leq w\) (Appendix C.1.1), and in particular, it is at most \(2|B|^2\) arithmetic operations.\(^{27}\)

   (b) **Prover:** The arithmetic complexity of \(P^{BAIR\rightarrow APR}(x, w, k, R)\) is at most the accumulated arithmetic complexity of (i) the arithmetic complexity of \(V^{BAIR\rightarrow APR}(x, k, R)\), (ii) \(2^t\) multiplications and additions over \(F\), (iii) \(3w\) low-degree extension (LDE) computations (cf. Definition 2.1), each over an affine space of dimension at most \(k + t\), with evaluation over at most \(2R + d + 2\) shifts; in particular, each LDE can be computed in arithmetic complexity at most \(3 \cdot (k + t) \cdot 2^{k + t + R + d + 2}\) (see Theorem B.2).

The following theorem is the analog of the previous one, stated for the \(R_{BPAIR}\) rather than for \(R_{BAIR}\).

**Theorem B.13** (Algebraic placement and routing – permuted version). There are two algorithms, denoted \(V^{BPAIR\rightarrow APR}, P^{BPAIR\rightarrow APR}\), such that for any \(k, R \in \mathbb{N}\), with \(R > 1\), and \(BPAIR\) instance \(x = (F, T, w, P_T, P_{\pi}, B)\) with \(T = 2^t - 1 > 4\), \(\max \{\deg P_T, \deg P_{\pi}, 2\} \leq 2^d\), and \([F : F_2] > 2 + k + R + t + \lceil \log (t + 1) \rceil + d\):

1. **Perfect completeness:** \((x, w) \in R_{BPAIR} \Rightarrow (V^{BPAIR\rightarrow APR}(x, k, R), P^{BPAIR\rightarrow APR}(x, w, k, R)) \in R_{APR}\)

2. **Perfect soundness:** \(x \in BPAIR \Leftrightarrow V^{BPAIR\rightarrow APR}(x, k, R) \in APR\)

\(^{27}\)For large sets of boundary constraints, one may modify the construction so that \(Z_{B,j}\) is a linearized polynomial, in which case the arithmetic complexity decreases to \(O(|B| \log |B|)\); details omitted because typically \(|B|\) is small.
3. Knowledge extraction: There is an efficient knowledge extractor \( E_{\text{PAIR} \rightarrow \text{APR}} \), such that for every \( (\text{PAIR} \rightarrow \text{APR}(x, k, R), w_{\text{APR}}) \in R_{\text{APR}} \), the extractor outputs a witness \( w \leftarrow E_{\text{PAIR} \rightarrow \text{APR}}(x, k, R, w_{\text{APR}}) \) such that \( (x, w) \in R_{\text{PAIR}} \).

4. Let \( n_{\text{APR}} = (F, T, N, \Phi, L, L_{\text{cmp}}, \vec{p}, p_{\text{cmp}}) = V_{\text{PAIR} \rightarrow \text{APR}}(x, k, R) \):

   (a) \( |T| = 2 (w + 1) \)
   (b) \( |N| \leq 11 |T| \), with same 11 neighbor polynomials for each witness index \( \tau \in T \)
   (c) \( |\Phi| \leq 2 |P_T| + |P_x| + 8w + 11 \) and \( T_{\text{arith}}(\Phi) \leq 2 T_{\text{arith}}(P_T) + T_{\text{arith}}(P_x) + O(w (|P_T| + |P_x|) + t) \)
   (d) \( \log_2 |L| = 2 + k + R + t + \lceil \log (t + 1) \rceil + d \)
   (e) \( \log_2 |L_{\text{cmp}}| = k + R + t + \lceil \log (t + 1) \rceil + d \)
   (f) The maximal rate \( \rho_{\text{max}} \) in \( \vec{p} \) satisfies \( \rho_{\text{max}} \leq 2^{-(2+R+d)} \)
   (g) \( p_{\text{cmp}} \leq 2^{-R} \)
   (h) \( n_{\text{APR}} \) is \( (1 - 2^{-R} (1 + 2^{-d})) \)-distance, \( 5 \)-overlapping, and \( (2^{-(2+R+d)}(1 - 2^{-k})) \)-independent with \( T_{\text{Sampling}}(n_{\text{APR}}) \leq \tilde{O}(|L|) \)

5. Arithmetic complexity over \( F \) (cf. Remark B.1):

   (a) **Verifier:** The arithmetic complexity of \( V_{\text{PAIR} \rightarrow \text{APR}}(x, k, R) \) is that which is required to compute the polynomials \( Z_{B,j}, E_{B,j} \in F[x] \) for all \( 1 \leq j \leq w \) (Appendix C.1.1), and in particular, it is at most \( 2 |B|^{2} \) arithmetic operations.
   (b) **Prover:** The arithmetic complexity of \( P_{\text{PAIR} \rightarrow \text{APR}}(x, w, k, R) \) is at most the accumulated arithmetic complexity of (i) the arithmetic complexity of \( V_{\text{PAIR} \rightarrow \text{APR}}(x, k, R) \), (ii) the complexity of routing a back-to-back De Bruijn butterfly network of dimension \( t \) (Theorem G.3) (iii) \( 2^t + 2^{\lceil \log(t+1) \rceil} \) multiplications and additions over \( F \), (iv) \( 5(w + 1) \) low degree extension (LDE) computations (cf. Definition 2.1), each over an affine space of dimension at most \( k + t + \lceil \log (t + 1) \rceil \), with evaluation over at most \( 2^{R+d+2} \) shifts; in particular, each LDE can be computed in arithmetic complexity at most \( 3 \cdot (k + t + \lceil \log (t + 1) \rceil) \cdot 2^{k+t+\lceil \log(t+1) \rceil + R+d+2} \) (see Theorem B.2).

### B.5 Algebraic linking IOP (ALI)

The output of the reductions described in the previous section are pairs in the relation \( R_{\text{APR}} \). The next phase in our reduction uses a 1-round IOP, called the *algebraic linking IOP* (ALI), to reduce the problem to a pair of *binary RS proximity testing* (BRPT) problems. The following definition formally defines the relation underlying the problem, adding to the informal discussion in Section 2.1).

**Definition B.14** (Binary RS proximity testing (RPT)). *Instances of the RS proximity testing problem (RPT) are triples \( x_{\text{RS}} = (F, S, \rho) \) where \( S \subseteq F \) and \( \rho \in [0, 1] \). A witness \( w_{\text{RS}} \) for \( x_{\text{RS}} \) is a function \( w_{\text{RS}} : S \rightarrow F \), and we say it satisfies \( x_{\text{RS}} \) iff and only if \( w_{\text{RS}} \) is a function \( w_{\text{RS}} : S \rightarrow F \), and we say it satisfies \( x_{\text{RS}} \) iff and only if \( w_{\text{RS}} \in \text{RS}([F, S, \rho]) \). The relation \( R_{\text{RPT}} \) is the set of pairs \( (x_{\text{RS}}, w_{\text{RS}}) \) such that \( w_{\text{RS}} \) satisfies \( x_{\text{RS}} \).

The binary RPT relation \( (R_{\text{BRPT}}) \) is one in which the instance satisfies:

- \( F \) is a binary field
• \( S \) is an affine coset of an \( \mathbb{F}_2 \)-linear subspace of \( \mathbb{F} \), defined by a coset shift \( a_0 \) and a basis \((a_1, \ldots, a_k)\) such that \( S = \left\{ a_0 + \sum_{i=1}^k \alpha_i a_i \mid \alpha_1, \ldots, \alpha_k \in \mathbb{F}_2 \right\} \)

• \( \rho = 2^{-\mathcal{R}} \) for \( \mathcal{R} \in \mathbb{N}^+ \).

The following theorem describes the main properties of the ALI protocol. This protocol, described in Appendix D.1, takes an instance-witness pair of \( R_{\text{APR}} \) and uses one round of interaction to reduce it to two instance-witness pairs of the \( R_{\text{BRPT}} \) relation.

**Theorem B.15 (Algebraic linking IOP (ALI) properties).** Let \( \mathcal{z} := (\mathbb{F}, \mathcal{T}, \mathcal{N}, \Phi, L, L_{\text{cmp}}, \bar{\rho}, \rho_{\text{cmp}}) \) be a \( \Theta \)-overlapping and \( \delta \)-distance APR instance with \( \delta > 0 \). The ALI protocol, applied to \( \mathcal{z} \), satisfies the following properties:

1. **Protocol Schedule**
   
   (a) **Prover:** On input \((\mathcal{z}, \mathcal{w})\), the output of \( \mathcal{P}^{\text{ALI}}(\mathcal{z}, \mathcal{w}) \) is a single oracle \( \mathcal{O}_{\text{assignment}} \) comprised of \(|\mathcal{T}| + 1\) functions in \( \mathbb{F}^L \) and a single function in \( \mathbb{F}^{L_{\text{cmp}}} \).
   
   (b) **Verifier:** On input \( \mathcal{z} \) the output of \( \mathcal{V}^{\text{ALI}}(\mathcal{z}) \) is a single message \( \mathcal{R} \) comprised of \( 2|\mathcal{T}| + |\Phi| \) random field elements and a pair \( \mathcal{z}_{\text{RS}} = (\mathbb{F}, L, \rho_{\text{max}}), \mathcal{z}_{\text{RS}}' = (\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}) \) of instances of the \( \mathcal{R}_{\text{BRPT}} \) relation.
   
   (c) **Induced output:** The oracle \( \mathcal{O}_{\text{assignment}} \) and verifier randomness define a pair of functions \( f^{(0)} : L \to \mathbb{F} \) and \( g^{(0)} : L_{\text{cmp}} \to \mathbb{F} \), such that each entry of \( f^{(0)} \) depends on \(|\mathcal{T}| + 1\) entries of \( \mathcal{O}_{\text{assignment}} \) and each entry of \( g^{(0)} \) depends on \(|\mathcal{N}| + 1\) entries of \( \mathcal{O}_{\text{assignment}} \).

2. **Completeness** If \( \mathcal{z} \) is satisfied by \( \mathcal{w} \) then for any randomness \( \mathcal{R} \) we have \( f^{(0)} \in \mathcal{R} [\mathbb{F}, L, \rho_{\text{max}}] \) and \( g^{(0)} \in \mathcal{R} [\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \).

3. **Soundness** Let

   \[
   \zeta \triangleq \max \left\{ 4, 1 + \Theta \frac{|L|}{|L_{\text{cmp}}|} \right\}.
   \]

   If \( \mathcal{z} \notin \mathcal{APR} \) then for every \( \mathcal{O}_{\text{assignment}} \) at least one of the following holds,
   
   (a) \( \Pr_R \left[ \Delta_H (f^{(0)}, \mathcal{R} [\mathbb{F}, L, \rho_{\text{max}}]) < \frac{\delta}{2^\zeta} \right] \leq \frac{1}{|\mathcal{T}|} \)
   
   (b) \( \Pr_R \left[ \Delta_H (g^{(0)}, \mathcal{R} [\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}]) < \frac{\delta}{2^\zeta} \right] \leq \frac{1}{|\mathcal{N}|} \)

4. **Knowledge extraction** Assume \( \rho_{\text{max}} \leq \frac{1}{4} \). Then there exists a Las Vegas (randomized) polynomial time algorithm \( \mathcal{E} \) for which the following holds. If \( \mathcal{w}^* \) does not satisfy both conditions claimed in Items 3a and 3b above, then the output of \( \mathcal{E} \) on input \((\mathcal{z}, \mathcal{w}^*)\) is a witness \( \mathcal{w}' \) that satisfies \( \mathcal{z} \).

5. **Perfect Zero Knowledge** If the instance \( \mathcal{z} \) is \( \kappa \)-independent, then the ALI protocol has perfect zero knowledge against any verifier that makes at most \( \kappa|L| \) queries to each \( \mathcal{w}_\tau \). This holds for any choice of \( RS_{\text{IOPP}} \) sub-protocols used in step 2c of the protocol (cf. Appendix D.1).

6. **Arithmetic complexity**
   
   (a) **Prover:** Assuming prover has \( \mathcal{w}' \) satisfying \( \mathcal{z} \), prover arithmetic complexity is at most

   \[
   |\mathcal{T}| \cdot |L| \cdot (3 \log |L| + 5) + 2 |L_{\text{cmp}}| \cdot (T_{\text{arith}}(\Phi) + |\Phi| + 1)
   \]

   where \( T_{\text{arith}}(\Phi) \) denotes the sum of the arithmetic complexities of the constraint set \( \Phi \).

43
Verifier: \( \mathcal{V}^{\text{ALI}} \) requires no arithmetic complexity, and involves only sampling \( 2|\mathcal{T}| + |\Phi| \) uniformly random field elements from \( \mathbb{F} \).

### B.5.1 On the soundness error of the ALI protocol

The probability that the “bad event” in which neither of Items 3a and 3b holds, is \( \frac{1}{|\mathbb{F}|} \), translating to a lower bound on soundness error of at least \( 1/|\mathbb{F}| \). Repeating Step 1b for a number \( k \) of times, each with independent randomness \( R_1, \ldots, R_k \), reduces soundness error to \( \frac{1}{|\mathbb{F}|^k} \). We stress that for constant \( k \), the increase in prover arithmetic complexity is negligible. Prover complexity is dominated by the computation of Item 1a which is executed only once, and requires \( O(|L| \cdot \log |L|) \) operations (first summand of Equation (8)); subsequent steps are linear in \( |L| \) (second summand of Equation (8)).

We conjecture the soundness error of the ALI protocol is lower than stated Item 3 above. In particular, we believe the denominator \( 2\zeta \) on the left hand side of Items 3a and 3b is not tight. Our ZK-STARK uses the FRI protocol as the RS-IOPP in the ALI protocol, thus, we state our next conjecture regarding ALI soundness in a manner that will be amenable to the soundness analysis of the FRI protocol.

**Blockwise distance — notation**

Recall the notion of the blockwise distance measure used by FRI; cf. [14, Section 3]. In particular, for \( L \subseteq \mathbb{F} \) an affine space and \( L_0 \subseteq L \) a subspace of size \( |L_0| = 2^n \), we define the \( L_0 \)-blockwise distance between \( f, g : L \rightarrow \mathbb{F} \) be the fraction of cosets \( S \) of \( L_0 \) in \( L \) on which \( f|_S \neq g|_S \), and let \( \mathcal{S}(L_0, L) \) denote the set of cosets of \( L_0 \) that are contained in \( L \).

With this notation in mind, we say that \( L_0 \) and \( L_0' \) are good for an instance \( x = (\mathbb{F}, \mathcal{T}, \mathcal{N}, \Phi, L, L_{\text{cmp}}, \bar{\rho}, \rho_{\text{cmp}}) \), if (i) \( L_0 \subseteq L \) and \( L_0' \subseteq L_{\text{cmp}} \), and (ii) for every neighbor \( N \in \mathcal{N} \) and subset \( \hat{S} \subseteq \mathcal{S}(L_0, L_{\text{cmp}}) \), it holds that

\[
\left| \left\{ S \in \mathcal{S}(L_0, L) \mid S \cap N \left( \bigcup_{\hat{S} \in \hat{S}} \hat{S} \right) \right\} \right| \geq \left| \bigcup_{\hat{S} \in \hat{S}} \hat{S} \right|.
\]

Informally, this condition means that attempting to change the all entries of the union of cosets in \( \hat{S} \) by changing entries of \( \mathbf{w}^* \) in the set \( N \left( \bigcup_{\hat{S} \in \hat{S}} \hat{S} \right) \) will modify at least as many cosets of \( \mathcal{S}(L_0, L) \) as there are elements in \( \bigcup_{\hat{S} \in \hat{S}} \hat{S} \).

**Conjecture B.16.** Suppose \( x = (\mathbb{F}, \mathcal{T}, \mathcal{N}, \Phi, L, L_{\text{cmp}}, \bar{\rho}, \rho_{\text{cmp}}) \) is an unsatisfiable instance of ALI with \( \delta \)-distance, and let \( L_0, L_0' \) be good for \( x \) and satisfy \( |L_0| = |L_0'| = 2^n \). Let \( \delta(0) (f(0)) \) denote the distance of \( f(0) \) from RS[\( \mathbb{F}, L, \rho_{\text{max}} \)] using the blockwise distance measure induced by \( L_0 \). Similarly, let \( \delta(0) (g(0)) \) denote the distance of \( g(0) \) from RS[\( \mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}} \)] using the blockwise distance measure induced by \( L_0' \).

Notice \( \delta(0) (f(0)) \) and \( \delta(0) (g(0)) \) are random variables that depend on the verifier randomness \( R \).

\[
\Pr[R] \left[ \delta(0) (f(0)) < \min \left\{ 1 - \rho_{\text{max}}, \frac{2^n |L_{\text{cmp}}|}{|L|} \cdot \left( \delta - \delta(0) (g(0)) \right) \right\} \right] \leq \frac{1}{|\mathbb{F}|}. \tag{9}
\]

For practical security purposes, weaker conjecture like the one below may suffice. In what follows, a pseudo-prover \( P^* \) is a polynomially-bounded randomized machine that acts as a prover in the ALI protocol above.

**Conjecture B.17.** For every pseudo-prover \( P^* \) and sufficiently large instance \( x = (\mathbb{F}, \mathcal{T}, \mathcal{N}, \Phi, L, L_{\text{cmp}}, \bar{\rho}, \rho_{\text{cmp}}) \), we have

\[
\Pr \left[ \delta(0) (f(0)) < 1 - \rho_{\text{max}} \land \delta(0) (g(0)) < 1 - \rho_{\text{cmp}} \right] \leq \frac{1}{|\mathbb{F}|}. \tag{10}
\]

The probability above is over the randomness of both \( P^* \) and \( V \).
See Appendix D.3 for a discussion of these conjectures.

**B.6 Fast Reed-Solomon (RS) IOP of Proximity (IOPP) (FRI)**

We now recall the main results of the FRI interactive oracle proof of proximity (IOPP) protocol, as stated in [14]. We assume familiarity with the definition of IOPP (see, e.g., [14, Definition 1.1]).

**Theorem B.18 (FRI properties).** The RS code family of rate $\rho = 2^{-R}$, $R \geq 2$, $R \in \mathbb{N}$ has an IOPP (FRI) with the following properties, where $N = |L|$ denotes blocklength (which equals Prover’s input length for a fixed RS[\F, L, \rho] code):

- **Schedule and rounds:** The protocol has two phases; during the first phase (COMMIT), verifier sends public randomness and prover sends oracles; round complexity is $\lceil \log N - R \rceil$. During the second phase (QUERY), verifier queries the oracles and reaches a decision (prover does not participate in this phase).
- **Prover:** prover complexity is less than $6N$ arithmetic operations in $\F$; proof length is less than $N/3$ field elements and round complexity is at most $\log N$.
- **Verifier:** for query-repetition parameter $\ell \in \mathbb{N}^+$, query complexity is $2\ell \log N$; the verifier decision is computed using at most $\ell \cdot 21 \log N$ arithmetic operations over $\F$.
- **Soundness:** There exists $\delta_0 \geq \frac{1}{4} (1 - 3\rho) - \frac{1}{\sqrt{N}}$ such that every $f$ that is $\delta$-far in relative Hamming distance from the code, is accepted with probability at most

\[
\text{err}_{\text{FRI}}(\delta) \triangleq \frac{3N}{|\F|} + (1 - \min \{\delta, \delta_0\})^\ell.
\]  

where $\ell \in \mathbb{N}^+$ is the repetition parameter defined above.

- **Parallelization:** Each prover-message can be computed in $O(1)$ time on a Parallel Random Access Machine (PRAM) with common read and exclusive write (CREW), assuming a single $\F$ arithmetic operation takes unit time.

We conjecture that soundness is nearly equal to the blockwise distance measure $\delta(0)$, for any distance value $\delta(0)$, even when $\delta(0) \approx 1 - \rho$. The following conjecture formalizes this; see [14] for a discussion of it.

**Conjecture B.19.** Using the notation of Theorem B.18, for any $f : L \to \F$ of blockwise distance $\delta(0)$ from RS[\F, L, \rho], $|L| = 2^k$ and $\epsilon \in [0, \delta(0)]$, with probability at least

\[
1 - \frac{(k)^2 \cdot 2^\eta}{\epsilon \cdot \eta^2 \cdot |\F|}
\]

over the randomness of the verifier during the COMMIT phase, the probability of rejection during the QUERY phase is at least $1 - \delta(0) (1 - \epsilon)$.

**B.7 Proof of main theorems**

In this section we prove Theorems 3.4 and 3.5 and Lemma B.6.
B.7.1 Proof of Main Theorem 3.4

We start with the following lemma. Its proof follows immediately from the proof of Cook–Levin Theorem, hence we omit it. Often the reduction from a general computations to the relevant BAIR instance may benefit from the ambient field structure, and lead to reductions that are concretely more efficient than a general-purpose reduction. Indeed the BAIR that corresponds to our DPM example is constructed in this way (cf. Appendix E).

Lemma B.20 (Reduction to BAIR). Given \( L \in \text{NTIME}(n) \), there exists a reduction mapping an instance \( x \) of \( L \) of size \( n \) to an \((n, t, d)\)-BAIR instance \( x_{\text{BAIR}} = (\mathcal{F}, T, w, P, C, B) \) in time in time \( \text{poly}(n, n) \) (see Remark B.1). The resulting \( x_{\text{BAIR}} \) satisfies the conditions of Lemma B.6, and furthermore

\[
w = S(n) + O(1), \quad t = \log T(n) + O(1), \quad |B| = n, \quad T_{\text{arith}}(\mathcal{P}) = O(S(n)) \quad \text{and} \quad d = O(1).
\]

The asymptotic constants above are independent of \( n \).

Given \( x, x_{\text{BAIR}} \) as above, and a witness \( w \) such that \((x, w) \in R_{\text{BAIR}}\), a BAIR witness \( w_{\text{BAIR}} \) \((x_{\text{BAIR}}, w_{\text{BAIR}}) \in R_{\text{BAIR}}\) can be computed in time \( \text{poly}(T(n) \cdot S(n), n) \). Vice versa, given \( x, x_{\text{BAIR}} \), \( w_{\text{BAIR}} \) such that \((x_{\text{BAIR}}, w_{\text{BAIR}}) \in R_{\text{BAIR}}\), a witness \( w \) such that \((x, w) \in R_L\) can be computed (or extracted) in time \( \text{poly}(T(n) \cdot S(n), n) \).

Proof of Theorem 3.4. Given instance \( x \) of \( L \) of size \( n \), denote \( T = T(n), S = S(n) \) and \( \lambda = \lambda(n) \). Set \( R = 3 \) and let \( \delta = (1 - 2^{-R}(1 + 2^{-d})) \), for \( d \) given by Lemma B.20; notice \( \delta \) is independent of \( n \) (which we have yet to specify). Let \( k \) be the smallest positive integer such that \( t + k > 6 + \log \lambda + \log \log \frac{1}{1 - \delta} \), noticing \( k \leq \lambda + O(1) \). Now let \( n = \lambda + t + d + R + k + 3 \), noticing \( n \leq 2\lambda + t + O(1) \). Apply Lemma B.20 and let \( x_{\text{BAIR}} \) be the resulting \((n, t, d)\) instance. By construction, this instance satisfies the assumptions of Lemma B.6, so we apply the ZK-IOP of knowledge specified there to \( x_{\text{BAIR}} \). Soundness, knowledge extraction and round complexity follow directly from Lemmas B.6 and B.20. Query, verifier, and arithmetic complexity also follow directly from these two lemmas, by using the bounds stated in Equation (13) above. This completes the proof. \( \square \)

B.7.2 Proof of Main Lemma B.6

Proof of Lemma B.6. We are given an instance-witness pair \((x_{\text{BAIR}}, w_{\text{BAIR}})\) for \( R_{\text{BAIR}} \) where \( x_{\text{BAIR}} = (\mathcal{F}, T, w, P, C, B) \) satisfies the assumptions of the the lemma.

Protocol description Prover and verifier apply the following reductions:

1. the deterministic reduction of Theorem B.12: the prover executes \( p_{\text{BAIR} \rightarrow \text{APR}}(x_{\text{BAIR}}, w_{\text{BAIR}}) \), leading to the witness \( w_{\text{APR}} \); the verifier executes \( v_{\text{BAIR} \rightarrow \text{APR}}(x_{\text{BAIR}}) \). By Theorem B.12 the resulting instance \( x_{\text{APR}} = (\mathcal{F}, T, N, \Phi, L, L_{\text{cmp}}, \rho, \rho_{\text{cmp}}) \) has parameters as stated in Item 4 there; additionally, \( x_{\text{APR}} \) is 1-overlapping, \( 2^{-(2+R+d)}(1 - 2^{-k}) \)-independent and has \( \delta \)-distance for \( \delta = (1 - 2^R(1 + 2^{-d})) \);

2. the 1-round ALI protocol on \((x_{\text{APR}}, w_{\text{APR}})\): Prover sends the oracle \( \overline{\Phi} \text{assignment} \), then verifier sends public randomness that, with \( \overline{\Phi} \text{assignment} \), induces a a pair of functions \( f^{(0)} : L \rightarrow \mathcal{F}, g^{(0)} : L_{\text{cmp}} \rightarrow \mathcal{F} \) that we denote by \( \bar{w}_{\text{RS}}, \bar{w}_{\text{RS}} \), respectively. Notice that the parameter \( \zeta \) defined in Equation (7) equals 5 because \( x_{\text{APR}} \) is 1-overlapping and \( \frac{|L|}{|L_{\text{cmp}}|} = 4 \).
the FRI protocol is now applied to each the two witnesses; Verifier uses instances (i) \( \mathbf{x}_{\text{RS}} = (\mathbb{F}, L, \rho_{\text{max}}) \) for \( w_{\text{RS}} \), and (ii) \( \mathbf{x}'_{\text{RS}} = (\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}) \) for \( w'_{\text{RS}} \). The repetition parameter in both cases is set to

\[
\ell = \left\lceil \frac{\lambda + 2}{\log \frac{1}{1 - \frac{\delta}{10}}} \right\rceil. \tag{14}
\]

**Completeness** Perfect completeness follows directly from the perfect completeness of the various reductions: If \( (\mathbf{x}_{\text{BAIR}}, \mathbf{w}_{\text{BAIR}}) \in R_{\text{BAIR}} \) then by the completeness part of Theorem B.12 the pair \( (\mathbf{x}_{\text{APR}}, \mathbf{w}_{\text{APR}}) \in R_{\text{APR}} \); so, by the completeness of Theorem B.15 \( (\mathbf{x}_{\text{RS}}, \mathbf{w}_{\text{RS}}) \) and \( (\mathbf{x}'_{\text{RS}}, \mathbf{w}'_{\text{RS}}) \) belong to BRPT, hence, by the completeness of Theorem B.18 the verifier accepts both with probability 1.

**Soundness** Suppose \( \mathbf{x}_{\text{BAIR}} \notin L_{\text{BAIR}} \). Then by the soundness of Theorem B.12 we also have \( \mathbf{x}_{\text{APR}} \notin L_{\text{APR}} \). Hence, by the soundness of Theorem B.15 we have with all but probability \( \epsilon_{\text{ALI}} = 1/|\mathbb{F}| \leq 2^{-(\lambda + 3)} \) have that at least one of \( f^{(0)}, g^{(0)} \) is \( \delta^{(0)} \)-far from the relevant RS code, where \( \delta^{(0)} = \frac{\delta}{2^\kappa} = \frac{\delta}{10} \); assume \( f^{(0)} \) is it (the analysis for \( g^{(0)} \) is identical). Notice that \( \delta^{(0)} < \delta_0 \) for \( \delta_0 \) as defined in Theorem B.18 because \( \mathcal{R} \geq 2 \). Therefore, by Theorem B.18, but for probability \( \text{err}_{\text{COMMIT}} = \frac{3|L|}{|\mathbb{F}|} < 2^{-(\lambda + 3)} \) over the randomness of the COMMIT phase, the probability of accepting \( f^{(0)} \) during the QUERY phase, conducted with repetition parameter \( \ell \), is at most

\[
\text{err}_{\text{QUERY}} = \left( 1 - \frac{\delta}{10} \right)^\ell \leq 2^{-(\lambda + 2)}.
\tag{15}
\]

Summing up, the total probability of error is at most \( \text{err}_{\text{ALI}} + \text{err}_{\text{COMMIT}} + \text{err}_{\text{QUERY}} \leq \text{err} \), as claimed.

**Knowledge extraction** Let \( P^* \) be a (not necessarily honest) prover that interacts with the verifier specified by the protocol above, and which leads that verifier to accept with probability \( p > 4 \cdot \text{err} \). Therefore, with probability at least \( O(1/(p - \text{err})) \), the oracle \( g^-_{\text{assignment}} \) provided by the Prover at the end of step 1 above, leads the verifier to accept with probability \( > 2\text{err} \). Fix such an \( g^-_{\text{assignment}} \): By the soundness of the FRI protocol as stated in Theorem B.18 and our setting of parameters, we conclude that that with probability strictly greater than \( 1/|\mathbb{F}| \) both \( \Delta_H(f^{(0)}, RS[\mathbb{F}, L, \rho]) \leq \frac{\delta}{2^\kappa} \) and \( \Delta_H(g^{(0)}, RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}]) \leq \frac{\delta}{2^\kappa} \). Noticing \( \rho_{\text{max}} = 2^{-\mathcal{R}} \leq 1/4 \), the knowledge extractor of Theorem B.15 outputs with high probability a witness \( \mathbf{w}_{\text{APR}} \) that satisfies \( \mathbf{x}_{\text{APR}} \). Hence, by the knowledge extraction property of Theorem B.12, Item 3, we conclude that a witness \( \mathbf{w}_{\text{AIR}} \) can be extracted from \( P^* \) in time \( \text{poly} \left( \frac{T}{p - \text{err}} \right) \) as claimed.

**Query complexity** The query complexity of the verifier is the sum of queries to the FRI oracles; we use the protocol and set \( \eta = 2 \) in [14, Theorem 3.3]), leading to round complexity \( t = \log |\mathbb{F}| \cdot \mathcal{R} \) and \( 2^\eta = 4 \) queries to each oracle per test (there are \( \ell \) many tests). Each query to \( f^{(0)} \) is simulated by making a single query to each of the \( w \) members of \( w_{\text{APR}} \) and a single query to \( f_{\text{mask}} \), and each query to \( g^{(0)} \) is simulated by making 3 queries to each member of \( w_{\text{APR}} \) and a single query to \( g_{\text{mask}} \). Using the value of \( \ell \) above, and the equalities \( \dim(L) = 2 + \dim(L_{\text{cmp}}) = k + \mathcal{R} + t + d + 2 \), from Theorem B.12, total query complexity is thus

\[
q = 2^n \cdot \ell \cdot (4w + 2 + \dim(L) - \mathcal{R} + \dim(L_{\text{cmp}}) - \mathcal{R})
= 2^{\eta+1} \cdot \ell \cdot (2w + k + t + d + 2)
= 8\left\lceil \frac{\lambda + 2}{\log \frac{1}{1 - \frac{\delta}{10}}} \right\rceil \cdot (2w + k + t + d + 2) \tag{16}
\]
Verifier arithmetic complexity  The arithmetic complexity of the verifier is the sum of four sub-components: (i) the computation of the polynomials arising from the boundary constraints described in Item 4h, at a cost of at most $2|B|^2$ arithmetic operations (see Footnote 27); (ii) the arithmetic complexity of the FRI verifier, at a cost of $21 \cdot \ell \cdot \log |L|$, (iii) the cost of computing an entry of $f^{(0)}$ from $w$, which costs $4(w + 1)$ operations per query (each test makes $2^n = 4$ such queries), and (vi) the computation of $g^{(0)}$, which costs $4(|\Phi| + T_{\text{arith}}(\Phi) + 1)$, where $T_{\text{arith}}(\Phi) = T_{\text{arith}}(P) + |B|$; recall that here we also make $4$ queries per test. Summing up, total verifier arithmetic complexity is

$$t_v \leq 2|B|^2 + \ell \cdot (16(w + |\Phi| + T_{\text{arith}}(\Phi)) + 2 \cdot 21(k + R + t + d + 2)) = 2 \left(|B|^2 + \left\lceil \frac{\lambda + 2}{\log \frac{1}{1-\delta/10}} \right\rceil \cdot (8(w + |\Phi| + T_{\text{arith}}(P) + |B|) + 21(k + R + t + d + 2)) \right)$$

(17)

Prover arithmetic complexity  Inspection of the prover arithmetic complexity shows that the components that dominate it are the $3 \cdot w$ low degree extension computations, each over a space of size at most $2^{k+t+R+d+2}$. In fact, this is the only phase of the prover that requires arithmetic complexity that is super-linear in $|L|$. The computation of $f^{(0)}$ and $g^{(0)}$ require $|L| \cdot (4|\Phi| + 4w + T_{\text{arith}}(P) + |B|)$ arithmetic operations and the two invocations of the FRI prover cost $2 \cdot 6 \cdot |L|$ operations. This completes the analysis of prover complexity.

Zero knowledge  By Theorem B.12, Item 4h, the instance $z_{\text{APR}}$ is $\kappa$-independent for $\kappa = 2^{-(2 + R + d)} (1 - 2^{-k})$ and $|L| = 2^{2 + k + R + t + d}$. By Item 5 of Theorem B.15, if $\kappa|L|$ is greater than the number of queries made into each member $w_\tau$ of $O_{\text{assignment}}$ then the protocol has perfect zero knowledge. We have

$$\kappa|L| = 2^{2 + k + R + t + d - (2 + R + d)} (1 - 2^{-k}) = 2^{k + t} (1 - 2^{-k}) \geq 2^{k + t - 1}.$$  

(18)

The last equality holds because $k \geq 1$.

A single query to $f^{(0)}$ requires a single query to $f^{(0)}_\tau$ and a single query to $g^{(0)}$ requires $3$ queries to each $f^{(0)}_\tau$. Each test of the FRI verifier makes $2^n = 4$ queries to either $f^{(0)}$ or $g^{(0)}$ and we use the repetition parameter $\ell$ defined above. Therefore, the total number of queries into each $w_\tau$ is bounded $16 \cdot \ell$. Combining this number with Equation (18) shows that for $t \geq 4 + \log \ell$ the protocol has perfect zero knowledge, thus, it holds for

$$k + t \geq 6 + \log \lambda + \log \log \frac{1}{1 - \delta/10}.$$

\[ \square \]

B.7.3 Proof of Lemma B.7

Proof of Lemma B.7. The proof is essentially the same as that of Lemma B.6, with the following changes to the protocol description. We use a smaller field, as specified in the statement of Lemma B.7. In Step 3 of the protocol above, we repeat the FRI-COMMIT phase more than once on $f^{(0)}$, $g^{(0)}$, leading to several sets of oracles; let $s$ denote the number of COMMIT repetitions (to be specified below). Additionally, we use different repetition parameters for $f^{(0)}$ and $g^{(0)}$, and both are smaller than specified above. Finally, each QUERY invocation is applied to a randomly selected COMMIT oracle-set, out of $s$.

Denoting by $\ell_{f^{(0)}}$, $\ell_{g^{(0)}}$ the repetition parameters for the FRI-QUERY phase for $f^{(0)}$, $g^{(0)}$, respectively, we set them (and $s$) thus

$$s = \left\lceil \frac{\lambda + 2}{n - (10 + 2 \log n)} \right\rceil; \quad \ell_{f^{(0)}} = \left\lceil \frac{\lambda + 2}{R + d} \right\rceil; \quad \ell_{g^{(0)}} = \left\lceil \frac{\lambda + 2}{R} \right\rceil$$

(19)
Since the modified protocol follows that of Lemma B.6, completeness and round complexity do not change. The reduced query complexity allows for a smaller (i.e., better) bound on the knowledge parameter $k$. Recalling the analysis in the proof of Lemma B.6, to achieve zero knowledge it suffices to ensure $2^{k+t-1}$ is greater than the number of queries to an individual member of $w$, i.e., greater than

$$2^n \cdot \left( \ell_{f^{(0)}} + 3 \ell_{g^{(0)}} \right) = 4 \cdot \left( \left\lceil \frac{\lambda + 2}{R + d} \right\rceil + 3 \left\lceil \frac{\lambda + 2}{R} \right\rceil \right) \leq 16 \cdot \left\lceil \frac{\lambda + 2}{R} \right\rceil.$$  

Thus, this inequality holds when $k + t > 5 + \log \left( \frac{\lambda + 2}{R} \right)$.

The computation of query and verifier follow directly from the definition of $\ell_{f^{(0)}}, \ell_{g^{(0)}}$ above, and gives

$$q = 2^n \left( \ell_{f^{(0)}} \cdot (w + 3 + k + t + d) + \ell_{g^{(0)}} \cdot (3w + 3 + k + t + d) \right)$$

$$= 4 \cdot \left( \left\lceil \frac{\lambda + 2}{R + d} \right\rceil \cdot (w + 3 + k + t + d) + \left\lceil \frac{\lambda + 2}{R} \right\rceil \cdot (3w + 3 + k + t + d) \right)$$

$$\leq 8 \left\lceil \frac{\lambda + 2}{R} \right\rceil (2w + 3 + k + t + d)$$

$$t v^T \leq 2 |B|^2 + \left\lceil \frac{\lambda + 2}{R} \right\rceil \cdot (16(w + |\Phi| + T_{arith}(P)) + 42 \cdot (k + R + t + d + 2))$$

Prover complexity accounts for the $s$ repetitions of the FRI-COMMIT phase, and is

$$t p^F \leq \left( 9w(t + d + R + 3) + |P| + w + T_{arith}(P) + 12 \left( \frac{\lambda + 2}{n - (10 + 2 \log n)} \right) \right) \cdot 2^{t + d + R + 3}$$

**Soundness error** Under Conjecture B.17, with all but probability $err_{_{\text{ALL}}} = 1/|F| < 2^{-(\lambda + 2)}$ we have that either $f^{(0)}$ has distance at least $\delta_{f^{(0)}} = 1 - 2^{-\left( R + d \right)}$ from the code $RS[F, L, \rho_{\text{max}}]$, or $g^{(0)}$ has distance at least $\delta_{g^{(0)}} = 1 - 2^{-R}$ from $RS[F^{\prime}, L_{\text{cmp}}, \rho_{\text{cmp}}]$; both $\delta_{f^{(0)}}, \delta_{g^{(0)}}$ are block-wise distances. Assume $\delta_{f^{(0)}} = 1 - 2^{-R}$ (the case of $\delta_{f^{(0)}} \geq 1 - \rho_{\text{max}}$ is argued similarly). Fixing $\epsilon = 2^{-10}$ (somewhat arbitrarily) and recalling $\eta = 2$, Conjecture B.19 implies that with all but probability $err_{\text{COMMIT}} = \frac{2^{10} \lambda^2}{|P|}$ over the randomness of the verifier during the COMMIT phases, the ensuing QUERY phase, repeated $\ell_{g^{(0)}}$ times, accepts with probability at most $\delta_{f^{(0)}} \cdot \delta_{g^{(0)}}$. Assuming $\delta_{g^{(0)}} \geq 1 - 2^{-R}$ and simplifying notation to $p = err_{\text{COMMIT}}, \delta = \delta_{f^{(0)}}$ and $\ell = \ell_{g^{(0)}}$, by the law of conditional probability the error probability is at most

$$\sum_{i=0}^{s} \left( \begin{array}{c} s \\ i \end{array} \right) p^i (1-p)^{s-i} \cdot \left( \frac{i}{s} + \frac{\delta (s - i)}{s} \right)^\ell \leq \sum_{i=0}^{s} \left( sp \right)^i \cdot \left( \frac{i + \delta (s - i)}{s} \right)^\ell \tag{20}$$

We claim that each of the $s$ terms on the right hand side above is bounded by $\delta^\ell / s$, which is equivalent to claiming

$$s \cdot (s \cdot p)^i \leq \left( \frac{\delta s}{i + (s - i) \delta} \right)^\ell .$$

By the assumption $\delta \in [1/2, 1]$, the base of the exponent on the right hand side is at least $1/2$, so it suffices to show

$$s \cdot (s \cdot p)^i \leq 2^{-\ell}$$
taking logarithms, it suffices to show $i \log \frac{1}{p} - (i + 1) \log s \geq \ell$, which we shall do even for $i = 1$. We have $\log \frac{1}{p} \geq n - (10 + 2 \log n)$ because $k \leq n$, and $\log s \leq 1 + \log \lambda \leq \log n$ thus, assuming

$$n - (12 + 4 \log n) \geq \lceil \frac{\lambda + 2}{R} \rceil$$

which holds for all $n \geq 60$ because $n > \lambda$, we conclude that indeed each summand of Equation (20) is bounded by $\delta^\ell/s$, hence $\mathsf{err}_{\text{FRI}} \leq \delta^\ell \leq 2^{-(\lambda+2)}$. The total error probability of the protocol is thus at most $\mathsf{err}_{\text{ALI}} + \mathsf{err}_{\text{FRI}} < 2^{-\lambda}$, as claimed. This completes the proof.

B.7.4 Proof of ZK-STIK Theorem 3.5

The proof below follows that of Theorem B.13, with minor modifications in parameters; all are due to the different starting point — a pair $(x_{\text{BPAIR}}, w_{\text{BPAIR}})$ — and the slightly different parameters of the end point of that particular reduction. We thus point out the parameter settings that differ from those appearing in the proof of Lemma B.6. Then we argue that asymptotic verifier complexity is strictly logarithmic, and prover complexity is strictly $O(T(n) \log^2 T(n))$.

Proof of Theorem 3.5. The proof follows that of Lemma B.6 with the following different parameter choices:

- In Item 1 of the protocol description in the proof of Lemma B.6, both parties apply the deterministic reduction of Theorem B.13: Prover executes $P_{\text{BPAIR}} \rightarrow APR(x_{\text{BPAIR}}, w_{\text{BPAIR}})$ leading to the witness $w_{\text{APR}}$ and verifier executes $V_{\text{BPAIR}} \rightarrow APR(x_{\text{BPAIR}})$, leading to $x_{\text{APR}}$.

- The repetition parameter is set differently than in the proof of Lemma B.6, to account for the fact that $x_{\text{APR}}$ is 5-overlapping (not 1-overlapping, as above) and thus $\zeta = 21$ (cf. Equation (7)) so $\delta^{(0)} = \frac{\delta}{21}$. This increases the repetition parameter $\ell$ defined in Equation (14) to

$$\ell = \lceil \frac{\lambda + 2}{\log \frac{1}{1-\frac{\delta}{21}}} \rceil$$

so that $\mathsf{err}_{\text{FRI}}(\delta^{(0)})$ will be bounded by $\mathsf{err}/4$ as stated in Equation (15).

- We also have $\dim(L) = 2 + k + R + t + \lceil \log (t + 1) \rceil + d$, i.e., the dimension of $L$ is greater than that used in the proof of Lemma B.6 by an additive factor of $\lceil \log (t + 1) \rceil$.

- Each query to $f^{(0)}$ requires now $|T| = 2(w + 1) + 1$ queries to $O_{\text{assignment}}$ (compared with $w + 1$ in the proof above) and each query to $g^{(0)}$ requires $11|T| = 22(w + 1) + 11$ queries to $O_{\text{assignment}}$ rather than $3|T|$ as before.

Query complexity Using these parameters in the computation of query complexity requires increasing the outer most factor from 36 to 156, to account for the factor $4\frac{1}{3} = \frac{40}{9}$ increase in repetition parameter, and replacing $w$ with $25w + 36$ in (16). This gives

$$q = 156 \cdot (\lambda + 2) \cdot (25w + t + d + \lceil \log (t + 1) \rceil + 38) = O(\lambda \cdot t).$$

The last equality uses $w = O(t)$ as stated in Lemma B.9.
**Verifier complexity**  The arithmetic complexity of the verifier is dominated by the arithmetic complexity of the FRI verifier and the computation of entries of \( f^{(0)} \) and \( g^{(0)} \), which should be repeated \( O(\ell) = O(\lambda) \) many times. Consequently, and using the bounds on \( t, d = O(1) \), \( |\mathcal{P} \cup \mathcal{P}_\pi| \), \( T_{\text{arith}}(\mathcal{P}) = O(\log T(n)) \), and \( |B| = O(n)|B| \), we conclude verifier arithmetic complexity is \( O(\lambda \cdot t) \) as well.

**Prover complexity**  Asymptotic prover complexity is similar to that discussed in the proof of Lemma B.6, with two important differences. First, our prover needs to compute a routing on a back-to-back De Bruijn network, costing \( O(T(n) \log T(n)) \) operations (see Theorem G.3); second, the dimension of \( L \) (and of \( L_{\text{cmp}} \)) is larger by an additive factor of \( \lceil \log (t + 1) \rceil \). Therefore, the LDE computations are now over spaces of size \( O(T(n) \log T(n)) \) and total arithmetic complexity of the prover is \( O(T(n) \log^2 T(n)) \).

**B.8 Realization considerations**

Our code realization for the DPM described in Figure 1 (cf. Appendix A) uses the verifier-side parameter settings of Lemma B.7. The IOP is converted to a transparent argument system with computational zero knowledge, via the Kilian/Micali cryptographic compiler (cf. Section 2.5). Our code for the prover realizes both reductions described in Theorems 3.4 and 3.5. Our focus here is on the DPM program that requires small space, hence the former system is more efficient in this case.

For the hash function used to construct Merkle trees as commitments to oracles, we use the Davies–Meyer construction instantiated with AES128. This gives an estimated collision resistance parameter of 64 bits.

Using the notation in the proof of Lemma B.7, we set \( R = 3 \) and \( \lambda = 60 \), leading to security error of at most \( \text{err} \leq 2^{-60} \). The binary field we use has \( |\mathbb{F}| = 2^{64} \), i.e., \( n = 64 \). The degree of our constraint system is 8 thus \( d = 3 \). We fix the ZK parameter to \( k = 1 \); this ensures zero knowledge for \( t \geq 8 \), and this is obtained for dataset sizes \( n \geq 2^3 \) because \( t = \lceil n \cdot 62 \rceil \). For all smaller datasets, setting \( k = 3 \) would suffice for ZK but since our focus is on large datasets we did not implement this.

The repetition parameter for the number of FRI-COMMIT is \( s = 2 \), and for the FRI – QUERY we have \( \ell_{f^{(0)}} = 9 \) and \( \ell_{g^{(0)}} = 22 \), as follows from Equation (19).
C Algebraic placement and routing (APR) reduction

In this section we prove Theorems B.12 and B.13, sequentially. The proof of Theorem B.12 starts with a description of the reduction in Appendix C.1, followed by a proof that it satisfies the conditions and parameters of Theorem B.12, given in Appendix C.2. Similarly, the proof of Theorem B.13 starts with a description of the reduction in Appendix C.3, followed by the proof of Theorem B.13 in Appendix C.4.

Notation In this section we view $\mathbb{F}$ as $\mathbb{F}_2[g]/h(g)$ for a primitive polynomial $h$ (see Remark B.1). For $a, b \in \mathbb{F}$ viewed as polynomials over $\mathbb{F}_2$, the notation $a \% b$ represents the remainder of $a$ divided by $b$ in the ring of polynomials $\mathbb{F}_2[g]$.

C.1 The APR reduction for space bounded computation

C.1.1 Common definitions

We first start by defining a few objects used in the construction:

- $\zeta \in \mathbb{F}_2[g]$ is a primitive polynomial of degree $t$, notice $\zeta$ is a member of $\mathbb{F}$ because $t < n$
- $H \subset \mathbb{F}$ is the space spanned by $\{g^k \mid 0 \leq k < t\}$
- $H_0 \subset H$ is the subspace spanned by $\{g^k \mid 0 \leq k < t - 1\}$
- $H_1 \subset H$ is the affine space $H_1 \triangleq H_0 + g^{t-1}$
- Given an affine space $S \subset \mathbb{F}$ we define the vanishing polynomial $Z_S \in \mathbb{F}[x]$ to be the monic polynomial of degree $|S|$ such that $\forall s \in S, Z_S(s) = 0$
- For every $j \in [w]$ we define $Z_{B,j}, \varepsilon_{B,j} \in \mathbb{F}[x]$ by $Z_{B,j}(x) \triangleq \prod_{(i,j,\alpha) \in B} (x - (g^i \% \zeta(g)))$ and $\varepsilon_{B,j}$ is the polynomial of minimal degree such that for every $(i,j,\alpha) \in B$, $\varepsilon_{B,j}(g^i \% \zeta(g)) = \alpha$
- Given a neighbor $(\tau, N) \in N$ we denote by $(\tau, N)$ the expression $(\tau, N) \triangleq X_{(\tau, N)} \cdot Z_{B,\tau} (N (X_{loc})) + \varepsilon_{B,\tau} (N (X_{loc})).$ We extend this notation further more, and denote by $(\tau_1, \tau_2, \ldots, \tau_k, N)$ the sequence $(\tau_1, N), (\tau_2, N), \ldots, (\tau_k, N)$.

C.1.2 Instance reduction

We now describe the (deterministic) verifier-side reduction $V^{BAIR\rightarrow APR}$:

1. $T \triangleq [w]$
2. $N \triangleq \{ (\tau, n_{id}^i), (\tau, n_{cy}^0), (\tau, n_{cy}^1) \mid \tau \in T \}$ where (i) $n_{id}^i(x) \triangleq x$, (ii) $n_{cy}^0(x) \triangleq gx + b\zeta$ for $b \in \{0, 1\}$.
3. For every $j \in [w]$ denote by $q_j(x, y) \triangleq x Z_{B,j}(y) + \varepsilon_{B,j}(y)$. Define for each $P \in P$ the set $\Phi_P$ by:

$$\Phi_P \triangleq \left\{ \frac{X_{loc}(X_{loc}-1)}{Z_{H_0}(X_{loc})} \cdot P \left( \frac{1, \ldots, w, n_{id}}{X_{loc}}, \frac{1, \ldots, w, n_{cy}^0}{\varepsilon_{B,j}(y)} \right) \right\} \cup \left\{ \frac{1}{Z_{H_1}(X_{loc})} \cdot P \left( \frac{1, \ldots, w, n_{id}}{X_{loc}}, \frac{1, \ldots, w, n_{cy}^1}{\varepsilon_{B,j}(y)} \right) \right\}$$
Finally, we define \(\Phi \triangleq \bigcup_{P \in \mathcal{P}} \Phi_P\).

4. Define \(L\) to be the affine space \(L \triangleq \text{span}\{g^{1+k+R+t+d} \cdot (1 + g) \cup \{g^i \mid 0 \leq i < 1 + k + R + t + d\}\}\) + \(g^{1+k+R+t+d}\).

5. Define \(L_{cmp}\) to be the affine space \(L_{cmp} \triangleq \text{span}\{g^i \mid 0 \leq i < k + R + t + d\} + g^{1+k+R+t+d}\).

6. For every \(j \in [w]\) define \(\rho_j\) to be \(\rho_j \triangleq \frac{2^{k+t} - \text{deg}(Z_{B,j})}{|L|}\).

7. Define \(\rho_{cmp} \triangleq \frac{1 + 2^{k+t+d}}{|L_{cmp}|}\).

C.1.3 Witness reduction

Finally, we describe the prover-side (randomized) reduction \(^{\text{PBAIR} \rightarrow \text{APR}}\), denoting the (input) \(\text{AIR}\) witness members by \(\{w_{j,\text{AIR}}\}_j\) and the (output) \(\text{APR}\) members by \(\{w_{j,\text{APR}}\}_j\). We show the construction of \(w_{j,\text{APR}}\) for every \(j \in [w]\) independently. Draw uniformly random a polynomial \(Q_j : \mathbb{F} \rightarrow \mathbb{F}\) of degree less than \(2^{k+t}\) such that for every \(i \in [T]\) it satisfies \(Q_j (g^i \zeta) = w_{j,\text{AIR}}[i]\). We define the \(\text{APR}\) witness by \(w_{j,\text{APR}}(x) \triangleq Q_j(x) - E_{B,j}(x)\) for every \(x \in L\). Notice this division is well defined, as all the roots of \(Z_{B,j}\) are in \(H\) by definition, and \(H \cap L = \emptyset\) by construction.

C.2 Proof of Theorem B.12 for space bounded computation

C.2.1 Proof of completeness (Item 1)

**Proof.** We show that: (i) for all \(j \in [w]\), \(w_{j,\text{APR}} \in \text{RS}[\mathbb{F}, L, \rho_j]\), and (ii) \(\forall \phi \in \Phi, \phi^\prime \in \Phi, w_{\phi}\) \(\in \text{RS}[\mathbb{F}, L_{cmp}, \rho_{cmp}]\).

**Assignment code membership:** For every \(j \in [w]\), \(\text{deg}(Q_j - E_{B,j}) < 2^{k+t}\), and by definition of satisfaction of \(\text{AIR} Z_{B,j} \mid (Q_j - E_{B,j})\), thus \(Q_j - E_{B,j}\) is a polynomial of degree less then \(2^{k+t} - \text{deg}(Z_{B,j})\), showing \(w_{j,\text{APR}} \in \text{RS}[\mathbb{F}, L, \rho_j]\) as required.

**Constraint code membership:** We notice first that for any \(j \in [w]\) and any neighbor \((j, N) \in \mathcal{N}\), \(\hat{(j, N)} = Q_j \circ N\) denoting by \(f \circ g\) the function \((f \circ g)(x) \triangleq f(g(x))\) thus it is sufficient to show that for any \(P \in \mathcal{P}\) both rational functions:

\[
\frac{x(x-1)}{Z_{H_0}(x)} \cdot P(Q_1(x), \ldots, Q_w(x), Q_1(gx), \ldots, Q_w(gx)) \tag{21}
\]

\[
\frac{1}{Z_{H_1}(x)} \cdot P(Q_1(x), \ldots, Q_w(x), Q_1(gx - \zeta), \ldots, Q_w(gx - \zeta)) \tag{22}
\]

are polynomials of degree less than \(2^{k+t+d}\). We notice it is sufficient to show the denominator indeed divides the enumerator in each of those rational functions. For that we use the observations: (i) \(H = \{g^i \zeta \mid 0 < i < T - 1\} \cup \{0, 1\}\), and (ii) \(0, 1 \in H_0\).
For Equation (21) it is sufficient to show that
\[ P(elQ_1(x), \ldots, Q_w(x), Q_1(gx), \ldots, Q_w(gx)) = 0 \]
for every \( g^i \in H_0 \setminus \{0, 1\} \). The claim follows by the fact that for any \( g^i \in H_0 \), we have \( g^{i+1}\%\zeta = g^{i+1} \), thus by construction
\[
P(Q_1(g^i), \ldots, Q_w(g^i), Q_1(g^{i+1}), \ldots, Q_w(g^{i+1}))
\]
\[ = P(w_1^{\text{AIR}}[i], \ldots, w_w^{\text{AIR}}[i], w_1^{\text{AIR}}[i+1], \ldots, w_w^{\text{AIR}}[i+1]) = 0 \]
(23)
(24)

Where the last equation follows by definition of satisfaction in AIR.

For Equation (22) the claim follows similarly, by noticing \( gx - \zeta = gx\%\zeta \) for all \( x \in H_1 \).

\( \square \)

C.2.2 Proof of soundness (Item 2)

Proof. Assume \( V^{\text{BAIR} \rightarrow \text{APR}}(\epsilon, k, R) \in \text{APR} \) for \( \epsilon = (\mathbb{F}, T, w, P, C, B) \), and let \( \hat{\omega}^{\text{APR}} \) be a witness for it. We show \( \epsilon \in \text{BAIR} \) by constructing a witness \( \hat{\omega}^{\text{BAIR}} \) for it. We define for every \( j \in [w] \) the mapping \( w_j^{\text{BAIR}} : [T] \rightarrow \mathbb{F} \) by
\[
w_j^{\text{BAIR}}[i] \triangleq w_j^{\text{APR}}(g^i\%\zeta) \cdot Z_{B,j}(g^i\%\zeta) + \epsilon_{B,j}(g^i\%\zeta) \]
(25)
denoting by \( w_j^{\text{APR}} \) as the low degree extension of the actual witness. We show \( \hat{\omega}^{\text{BAIR}} \) satisfies \( \epsilon \). Denote by \( q_j(x, y) \triangleq x \cdot Z_{B,j}(x) + \epsilon_{B,j}(x) \).

Boundary constraints: \( \hat{\omega}^{\text{BAIR}} \) satisfies the boundary constraints \( B \) by construction, as having \((i, j, \alpha) \in B \) implies \( Z_{B,j}(g^i\%\zeta) = 0 \) and \( \epsilon_{B,j}(g^i\%\zeta) = \alpha \), thus \( w_j^{\text{BAIR}}[i] = \alpha \) as required.

Consistency with \( P \): We show that for any \( i \in [T - 1] \) and for any \( P \in P \):
\[
P(w_1^{\text{BAIR}}[i], \ldots, w_w^{\text{BAIR}}[i], w_1^{\text{BAIR}}[i+1], \ldots, w_w^{\text{BAIR}}[i+1]) = 0
\]
Assume this is not the case, thus there is some \( i \) and some \( P \in P \) for which the above does not hold, we show in such case \( \hat{\omega}^{\text{APR}} \) can not be a witness of \( V^{\text{BAIR} \rightarrow \text{APR}}(\epsilon, k, R) \). Denote by \((\mathbb{F}, T, N, \Phi, L, L_{\text{cmp}}, \rho_{\text{cmp}}) = V^{\text{BAIR} \rightarrow \text{APR}}(\epsilon, k, R) \). Assume \( \hat{\omega}^{\text{APR}} \in \text{RS}[\mathbb{F}, L, \rho_j] \), and we show \( \exists \phi \in \Phi \) such that \( \phi_N[\hat{\omega}^{\text{APR}}] \notin \text{RS}[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \). Without loss of generality assume \( g^i\%\zeta \in H_0 \setminus \{0, 1\} \), we show the following constraint is not a member of \( \text{RS}[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \):
\[
\phi(x) = \frac{x(x-1)}{Z_{H_0}(x)} \cdot P\left(q_1^{\text{APR}}(x), q_w^{\text{APR}}(x), \ldots, q_w^{\text{APR}}(gx), \ldots, q_w^{\text{APR}}(gx), g^i\%\zeta, g^i\%\zeta\right)
\]
Otherwise
\[
\phi(x) \cdot Z_{H_0}(x) = x(x-1) \cdot P\left(q_1^{\text{APR}}(x), q_w^{\text{APR}}(x), \ldots, q_w^{\text{APR}}(gx), \ldots, q_w^{\text{APR}}(gx), g^i\%\zeta, g^i\%\zeta\right)
\]
(26)
for every \( x \in L_{\text{cmp}} \), where both sides, as codewords over \( L_{\text{cmp}} \), are of rate at most
\[
\rho_{\text{cmp}} + \frac{|H_0|}{|L_{\text{cmp}}|} \leq \frac{2^{k+t+d} + 2^t}{2^k + R + t + d} \leq 2^{-R} \left(1 + 2^{-k-d}\right) < 1
\]
54
But they don’t agree over the entire field \( \mathbb{F} \), as for \( g^i \% \zeta \) the left hand of the equation vanishes as \( Z_{H_0}(g^i \% \zeta) = 0 \), while the right hand does not vanish as

\[
P \left( q_1 \left( w_{i+1}^{\text{APR}}(g^i), g^i \right), \ldots, q_w \left( w_{i+1}^{\text{APR}}(g^i), g^i \right) \right) = 0 \quad (27)
\]

\[
P \left( w_{i+1}^{\text{BAIR}}[i], \ldots, w_{i+1}^{\text{BAIR}}[i] \right) \neq 0 \quad (28)
\]

The contradiction follows by showing both sides are polynomials of degree less then \( \deg (\phi \cdot Z_{H_0}) \) is low enough by assumption. We assumed \( w_j^{\text{APR}} \in \mathbb{R}[\mathbb{F}, L, \rho_j] \), thus

\[
\deg \left( q_j \left( w_j^{\text{APR}}(x), x \right) \right) < 2^{k+t}
\]

thus the right hand of Equation (26) is of degree at most

\[
\deg < 2^{k+t-d} + 2 < 2^{k+t-d} + \left| H_0 \right| = \rho_{\text{cmp}} |L_{\text{cmp}}| + |H_0|
\]

concluding the proof.

\[\square\]

C.2.3 Knowledge extraction (Item 3)

**Proof.** The knowledge extractor is described in Equation (25), and the soundness proof in Appendix C.2.2 shows it is indeed an extractor.

\[\square\]

C.2.4 Instance properties (Item 4h)

**Arithmetic complexity of \( \Phi \):** While most of the summands in Item 4c are straightforward, the only non trivial dependency is the dependency in \( t \). It is a known fact that for every linear space \( V \), the polynomial \( Z_V \) vanishing over \( V \) is constructible in time \( \text{poly} \left( |V| \right) \), and has exactly \( |V| \) nonzero coefficients, although its degree is \( |V| \). Moreover, for every constant \( c \in \mathbb{F} \), the vanishing polynomial over \( V + c \) is exactly \( Z_{V+c} = Z_V + Z_V(c) \). Given all this, we conclude that the values of \( Z_{H_0}(X_{\text{loc}}), Z_{H_1}(X_{\text{loc}}) \) can be all computed using a single evaluation of \( Z_{H_0}(X_{\text{loc}}) \) and addition of a single constant value to compute \( Z_{H_1}(X_{\text{loc}}) \). In total, this single computation requires \( 3t \) multiplications and additions in the field. Additional optimization is possible when representing \( Z_{H_0} \) by a binary matrix, eliminating completely multiplications over the field from the evaluation process.

\( z_{\text{APR}} \) is \( \left( 2^{-2(R+d)}(1 - 2^{-d}) \right) \)-independent:

**Proof.** The witness reduction samples each \( Q_j \) from a \( \left( 2^{k+t} - 2^t \right) \) independent space, so by construction we notice \( w_j^{\text{APR}} \) is sampled from a \( \left( 2^{k+t} - 2^t \right) \) independent space as well. The claim follows by noticing \( \left( 2^{-2(R+d)}(1 - 2^{-d}) \right) |L| = \left( 2^{k+t} - 2^t \right) \).

\[\square\]

\( z_{\text{APR}} \) has \( \left( 1 - 2^{-R} \left( 1 + 2^{-d} \right) \right) \)-distance:

**Proof.** \( \rho_{\text{cmp}} \leq 2^{-R} < 2^{-R} \left( 1 + 2^{-d} \right) \), and \( \rho_{\text{max}} \leq 2^{-2(R+d)} < 2^{-R} \left( 1 + 2^{-d} \right) \). It is sufficient to show the existence of the code \( C \) from Item 2. We define \( C \) to be the set of mappings \( w : L \rightarrow \mathbb{F} \) which are rational functions of the form \( w(x) = \frac{p(x)}{q(x)} \) where \( p(x) \) is a polynomial of degree less than \( 2^{t+d} \) and \( q(x) \triangleq \prod_{\alpha \in H \setminus \{0,1\}} (x - \alpha) \). \( w \) is well defined over \( L \), as none of the zeros of \( q(x) \) are in \( L \), and the distance of the
code $C$ is the same as the distance of $RS \left[ \mathbb{F}, L, \frac{2^{t+d}}{|H|} \right] = RS \left[ \mathbb{F}, L, 2^{-2(k+R)} \right]$, which is $1 - 2^{-2(k+R)}$, and in particular grater than $1 - 2^{-R} \left( 1 + 2^{-d} \right)$. We conclude the proof by showing $RS \left[ \mathbb{F}, L, \rho_{\text{max}} \right] \subset C$ by noticing it is exactly the sub-code where $p(x)$ is restricted to be of the form $p(x) = q(x) \cdot v(x)$, where $v$ is a polynomial of degree less than $2^{k+t}$.

$\exists_{\text{APR}}$ is 1-overlapping:

**Proof.** Define the following 3 disjoint affine subspaces of $L$:

- $S_x \triangleq L_{\text{cmp}} = \text{span} \left\{ g^i \mid 0 \leq i < k + R + t + d \right\} + g^{1+k+R+t+d}$
- $S_{gx} \triangleq \text{span} \left\{ g^{i+1} \mid 0 \leq i < k + R + t + d \right\} + g^{2+k+R+t+d}$
- $S_{gx-\zeta} \triangleq \text{span} \left\{ g^{i+1} \mid 0 \leq i < k + R + t + d \right\} + (1 + g^{2+k+R+t+d})$

The claims follows by the observation that for any $N(x) \in \{ x, gx, gx - \zeta \}$, and for any $z \in L_{\text{cmp}}$, $N(z) \in S_{\tilde{N}(x)}$. □

C.3 The APR reduction for general computation

C.3.1 Common definitions

We expand the definition in Appendix C.1.1 ($\zeta, H$). For the sake of completeness we provide those definitions again, after providing the definitions introduced first in this section. The definitions of $Z_{B,\tau}$, $E_{B,\tau}$ similar to those defined in Appendix C.1.1, providing the same semantic purpose, but syntactically differ, fitting the construction described in this section. We first start by defining a few objects used in the construction:

- $\xi \in \mathbb{F}_2[g]$ is a primitive polynomial of degree $\lceil \log (t+1) \rceil$, notice $\xi$ is a member of $\mathbb{F}$
- $W \subset \mathbb{F}$ is the space spanned by $\{ g^{t+k} \mid 0 \leq k < \lceil \log (t+1) \rceil \}$
  - $W_0 \subset W$ is the subspace spanned by $\{ g^{t+k} \mid 0 \leq k < \lceil \log (t+1) \rceil - 1 \}$
  - $W_1 \subset W$ is the affine space $W_1 \triangleq W_0 + g^{t+\lceil \log (t+1) \rceil - 1}$
  - we denote by $s$ the element $s \triangleq g^{t+1} \in W$
- Given a neighbor $(\tau, N) \in \mathcal{N}$ we denote by $(\widehat{\tau}, \widehat{N})$ the expression $(\tau, N) \triangleq X_{(\tau, N)} \cdot Z_{B,\tau} \left( N \left( X_{\text{loc}} \right) \right) + E_{B,\tau} \left( N \left( X_{\text{loc}} \right) \right)$. We extend this notation further more, and denote by $(\tau_1, \tau_2, \ldots, \tau_k, N)$ the sequence $(\tau_1, N), (\tau_2, N), \ldots (\tau_k, N)$.

Definition identical or similar to Appendix C.1.1:

- $\zeta \in \mathbb{F}_2[g]$ is a primitive polynomial of degree $t$, notice $\zeta$ is a member of $\mathbb{F}$
- $H \subset \mathbb{F}$ is the space spanned by $\{ g^k \mid 0 \leq k < t \}$
  - $H_0 \subset H$ is the subspace spanned by $\{ g^k \mid 0 \leq k < t - 1 \}$
\(-H_1 \subset H\) is the affine space \(H_1 \triangleq H_0 + g^{-1}\)

- For every \(\tau \in \mathcal{T}\) we define \(Z_{B,\tau}, \mathcal{E}_{B,\tau} \in \mathbb{F}[x]\) by:
- if \(\tau = (j,0)\) for \(j \in [w]\) then \(Z_{B,\tau}(x) \triangleq \prod_{(i,j,\alpha) \in B} (x - \left(g^{i\bmod{\alpha}} \cdot s\right))\) and \(\mathcal{E}_{B,\tau}\) is the polynomial of minimal degree such that for every \((i, j, \alpha) \in B, \mathcal{E}_{B,\tau}\left(\left(g^{i\bmod{\alpha}}\right) + s\right) = \alpha\)
- otherwise \(Z_{B,\tau} = 1\) and \(\mathcal{E}_{B,\tau} = 0\)

### C.3.2 Instance reduction

We now describe \(\Phi^{\text{PAIR} \rightarrow \text{APR}}\):

1. \(\mathcal{T} \triangleq ([w] \cup \{\text{ctrl}\}) \times \{0, 1\}\)

2. \(\mathcal{N} \triangleq \mathcal{N}_P \cup \mathcal{N}_{\text{routing}}\), with \(\mathcal{N}_P, \mathcal{N}_{\text{routing}}\) defined as follows:

   **Definition of \(\mathcal{N}_P\):** \(\mathcal{N}_P \triangleq \{(\tau, n^{\text{id}}), (\tau, n^{\text{cyc}}), (\tau, n_1^{\text{cyc}}) | \tau \in \mathcal{T}\}\) where (i) \(n^{\text{id}}(x) \triangleq x\), (ii) \(n_0^{\text{cyc}}(x) \triangleq g(x - g^{t+1}) + b\zeta + g^{t+1}\) for \(b \in \{0, 1\}\).

   **Definition of \(\mathcal{N}_{\text{routing}}\):** \(\mathcal{N}_{\text{routing}} \triangleq \{(\tau, n^{\text{routing}}) | \tau \in \mathcal{T}; b, r, c \in \{0, 1\}\}\) where \(n^{\text{routing}}(x) \triangleq g \cdot x + r \cdot g^t + c \cdot (g^t \cdot \xi) + b\)

3. \(\Phi\) is defined as \(\Phi \triangleq \Phi_P \cup \Phi_{P_{\pi}} \cup \Phi_{\text{routing}}\) where \(\Phi_P, \Phi_{P_{\pi}}, \Phi_{\text{routing}}\) defined as follows:

   - **\(\Phi_P\) definition:** Define for every \(P \in \mathcal{P}_T\) define
     \[
     \Phi_P \triangleq \left\{ \frac{(X_{\text{loc}} - s)(X_{\text{loc}} - s - 1)}{Z_{H_1 + s}(X_{\text{loc}})} \cdot P\left( ((1, 0), \ldots, (w, 0), n^{\text{id}}), ((1, 0), \ldots, (w, 0), n_1^{\text{cyc}}) \right) \right\}
     \]

     Define \(\Phi_P \triangleq \bigcup_{P \in \mathcal{P}_T} \Phi_P\).

   - **\(\Phi_{P_{\pi}}\) definition:** Define for every \(P \in \mathcal{P}_{\pi}\) define
     \[
     \Phi_P \triangleq \left\{ \frac{(X_{\text{loc}} - s)(X_{\text{loc}} - s - 1)}{Z_{H_1 + s}(X_{\text{loc}})} \cdot P\left( ((1, 0), \ldots, (w, 0), n^{\text{id}}), ((1, 1), \ldots, (w, 1), n^{\text{id}}) \right) \right\}
     \]

     Define \(\Phi_{P_{\pi}} \triangleq \bigcup_{P \in \mathcal{P}_{\pi}} \Phi_P\).
4. **Φrouting definition**: We define the following polynomials used by our construction:

\[ q_{eql}(x, y) \triangleq x - y \]  
\[ q_{mv}(x, y, z) \triangleq q_{eql}(x, y)q_{eql}(x, z) \]  
\[ q_{cp}(x, x', y, y', z, z') \triangleq q_{eql}(x', z')q_{eql}(x, y) + q_{eql}(x', y')q_{eql}(x, z) \]  
\[ S_{0,0}(x) \triangleq \frac{1}{Z_{H_0 + \text{span}(g^t)}(x)} \]  
\[ S_{1,0}(x) \triangleq \frac{1}{Z_{H_1 + \text{span}(g^t)}(x)} \]  
\[ S_{0,1}(x) \triangleq \frac{1}{Z_{H_0 + W_1}(x)} \]  
\[ S_{1,1}(x) \triangleq \frac{1}{Z_{H_1 + W_1}(x)} \]

and define the control constraints system by:

\[
\Phi_{ctrl} \triangleq \left\{ \frac{1}{X_{loc} - q_{eql}} \left( X_{\{\text{ctrl}, 0\}, \text{r}, \text{id}} \right), \frac{1}{Z_{H_0 + \text{span}(g^t)}(X_{loc})} \left( X_{\{\text{ctrl}, 0\}, \text{r}, \text{id}} \right), \frac{1}{Z_{H_1 + \text{span}(g^t)}(X_{loc})} \left( X_{\{\text{ctrl}, 1\}, \text{r}, \text{id}} \right) \right\}
\cup \left\{ S_{r,c}(X_{loc}) \cdot q_{mv}(X_{\{\text{ctrl}, l\}, \text{r}, \text{id}}) \left| i \in [\mathcal{W}]; l, r, c \in \{0, 1\} \right\} \right\}
\]

The network flow copy constraints defined by:

\[
\Phi_{cp} \triangleq \left\{ S_{r,c}(X_{loc}) \cdot q_{cp}(X_{\{\text{ctrl}, l\}, \text{r}, \text{id}}) \left| i \in [\mathcal{W}]; l, r, c \in \{0, 1\} \right\} \right\}
\]

We define \( \Phi_{routing} \triangleq \Phi_{ctrl} \cup \Phi_{cp} \).

5. Define \( L \) to be the affine space

\[
L \triangleq \text{span} \left( \left\{ g^{i + 1 + k + R + t + [\log(t + 1)] + d} \cdot (1 + g^i) \right\} \cup \left\{ g^i \mid 0 \leq i < 1 + k + R + t + [\log(t + 1)] + d \right\} \right) + g^{1 + k + R + t + [\log(t + 1)] + d}
\]

6. Define \( L_{cmp} \) to be the affine space

\[
L_{cmp} \triangleq \text{span} \left( \left\{ g^i \mid 0 \leq i < k + R + t + [\log(t + 1)] + d \right\} \cup g^{1 + k + R + t + [\log(t + 1)] + d} \right)
\]

7. For every \( 1 \leq j \leq w \) define \( \rho(j, 0) \) to be \( \rho(j, 0) \triangleq \frac{g^{2k + t + [\log(t + 1)] - \deg(Z_{B,j})}}{\deg(Z_{B,j})} \); for any other \( \tau \in \mathcal{T} \) we define \( \rho_{\tau} \) to be

\[
\rho_{\tau} \triangleq \frac{g^{2k + t + [\log(t + 1)] + d}}{|L|} = 2^{-2(2 + R + d)}
\]

8. Define \( \rho_{cmp} \triangleq \frac{1 + 2g^{2k + t + [\log(t + 1)] + d}}{|L_{cmp}|} \)
C.3.3 Witness reduction

General overview  Let \( w = (\hat{w}, \pi) \) be the witness. We construct all functions in
\[
\mathcal{W}_{\text{APR}} = (\mathcal{W}_{w(1,0)}, \mathcal{W}_{w(1,1)}, \ldots, \mathcal{W}_{w(w,0)}, \mathcal{W}_{w(w,1)}, \mathcal{W}_{\text{ctrl}(0)}, \mathcal{W}_{\text{ctrl}(1)})
\]
as follows:

1. For every \( \tau \in T \) we define a mapping \( Q_\tau: H + W \to \mathbb{F} \)
2. For every \( \tau \in T \) we define a mapping \( \hat{Q}_\tau: \mathbb{F} \to \mathbb{F} \) as the low degree extension of \( Q_\tau \)
3. For every \( \tau \in T \) we draw a random polynomial \( R_\tau: \mathbb{F} \to \mathbb{F} \) of degree at most \( 2^{k+t+\lceil \log(t+1) \rceil} \)
   vanishing on \( H + W \) and define \( \hat{Q}_\tau \triangleq \hat{Q}_\tau + R_\tau \)
4. the function \( \mathcal{W}_\tau: L \to \mathbb{F} \) is the evaluation of \( \frac{\hat{Q}_\tau - \xi_{B^r}}{\bar{Z}_{B^r}} \) over \( L \)

Given the above description, it is sufficient to describe the construction of \( Q_\tau \) for every \( \tau \in T \).

Construction of \( Q_{(\text{ctrl},0)}, Q_{(\text{ctrl},1)} \)  Given \( \pi: [T] \rightarrow [T] \) we define the permutation \( \pi': \mathbb{F}[g]/\zeta \rightarrow \mathbb{F}[g]/\zeta \) by:
\[
\pi'(0) = 0
\]
\[
\forall 0 \leq i < T : \pi'(g^i \% \zeta) = g^{\pi(i) \% \zeta} \tag{39}
\]
\[
Q_{(\text{ctrl},0)}, Q_{(\text{ctrl},1)} \text{ are an affine embedding of back-to-back De Bruijn routing (Appendix G.2) of } \pi'
\text{ where } Q_{(\text{ctrl},0)}(x + s) = x \text{ for every } x \in H. \tag{40}
\]

Construction of \( Q_{(i,b)} \) mappings  Any other pair \( Q_{(i,0)}, Q_{(i,1)} \) is intuitively an embedding of the same permutation \( \pi' \) as well, and uses exactly the same routing as \( Q_{(\text{ctrl},0)}, Q_{(\text{ctrl},1)} \). The sole difference is that the \( Q_{(i,0)} \) over \( H + s \) is not required to contain only distinct values, but instead satisfies \( Q_{(i,0)}((g^j \% \zeta) + s) = w_i[j] \).

Further optimizations  The reduction above can be further optimized for concrete settings in the following manner, implemented in our ZK-STARk realization. Often, not all \( w \) algebraic registers are needed to verify memory validity; rather, a small number \( l \ll w \) suffices. In this case one may save by routing on the rearrangeable network only the \( l \) needed registers (cf. [90] for more details).

C.4 Proof of Theorem B.13 for general computation

The construction of the reduction from BPAIR to APR (Appendix C.3) is very similar to the construction of reduction from BAIR to APR (Appendix C), with the main difference of using an embedding of back-to-back De Bruijn routing (Appendix G.2) to represent the permutation in the BPAIR witness. For the sake of completeness we provide below a full proof for this construction.

C.4.1 Proof of completeness (Item 1)

Proof. We show that: (i) for all \( \tau \in T \), \( \mathcal{W}_{\tau}^{\text{APR}} \in \text{RS}[\mathbb{F}, L, \rho_\tau] \), and (ii) \( \forall \phi \in \Phi, \phi_N \left[ \mathcal{W}_{\phi_{\Phi}}^{\text{APR}} \right] \in \text{RS}[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \).
Assignment code membership: For every $\tau \in \mathcal{T}$, $\deg (\bar{Q}_\tau - \mathcal{E}_{\mathcal{B},\tau}) < 2^{k+t+\lceil \log(t+1) \rceil}$, and by definition of satisfaction of BPAIR, we have $Z_{\mathcal{B},\tau} | (\bar{Q}_\tau - \mathcal{E}_{\mathcal{B},\tau})$, thus $\frac{\bar{Q}_\tau - \mathcal{E}_{\mathcal{B},\tau}}{Z_{\mathcal{B},\tau}}$ is a polynomial of degree less then $2^{k+t+\lceil \log(t+1) \rceil} - \deg (Z_{\mathcal{B},\tau})$, showing $w_{\tau}^{\text{APR}} \in RS [\mathbb{F}, L, \rho_\tau]$ as required.

Constraint code membership: We notice first that for any $\tau \in \mathcal{T}$ and any neighbor $(\tau, N) \in \mathcal{N}$, $(\tau, N) = \bar{Q}_\tau \circ N$ is of degree less then $2^{k+t+\lceil \log(t+1) \rceil}$. It is sufficient to show that every rational function $\phi \in \Phi$ is a polynomial of degree at most $2^{k+t+\lceil \log(t+1) \rceil} + d$. We notice that $\phi = \frac{P}{\tilde{S}}$ for some multivariate polynomial $p$, and some set $S$. For simplicity we denote by $g \in \mathbb{F}[x]$ the low degree extension of $p_N [\hat{w}^{\text{APR}}]$. It is sufficient to show $g$ is (i) of degree less then $2^{k+t+\lceil \log(t+1) \rceil} + d + |S|$, and (ii) vanishes over $S$, thus in particular dividable by $Z_S$.

The set $\Phi_{P_\tau}$: for every $P \in P_\tau$ denote by

$$P_0 (x) \triangleq P \left( \bar{Q}_{(1,0)} \circ n^{id} (x), \ldots, \bar{Q}_{(w,0)} \circ n^{id} (x), \bar{Q}_{(0,1)} \circ n^cyc_0 (x), \ldots, \bar{Q}_{(0,w)} \circ n^cyc_0 (x) \right)$$

$$P_1 (x) \triangleq P \left( \bar{Q}_{(1,0)} \circ n^{id} (x), \ldots, \bar{Q}_{(w,0)} \circ n^{id} (x), \bar{Q}_{(1,0)} \circ n^cyc_1 (x), \ldots, \bar{Q}_{(w,0)} \circ n^cyc_1 (x) \right)$$

$$P'_0 (x) = \frac{(x-s)(x-s-1)}{Z_{H_0+s}(x)} \cdot P_0 (x)$$

$$P'_1 (x) = \frac{1}{Z_{H_1+s}(x)} \cdot P_1 (x)$$

We notice $\deg P_0, \deg P_1 < 2^{k+t+\lceil \log(t+1) \rceil} + d$, thus it is sufficient to show that whenever the denominator vanishes, the enumerator vanishes as well. Assume $Z_{H_0+s}(x)$ or $Z_{H_1+s}(x)$ is zero, then $x = y + s$ for some $y \in H$. If $y \in \{0, 1\}$ then $x \in H_0 + s$, thus only $Z_{H_0+s}(x) = 0$, and $(x-s)(x-s-1) = 0$ as well. Otherwise, there is some $j \in [T-1]$ such that $y = g^j%\zeta$, and by the witness construction (Appendix C.3.3) $\bar{Q}_{(j,0)} \left( (g^j%\zeta) + s \right) = w_i[j]$. In the case $(g^j%\zeta) + s \in H_0 + s$ we notice $n^cyc_0 \left( (g^j%\zeta) + s \right) = (g^{j+1}%\zeta) + s$. Similarly, in the case $(g^j%\zeta) + s \in H_1 + s$ it holds $n^cyc_1 \left( (g^j%\zeta) + s \right) = (g^{j+1}%\zeta) + s$. Finally, the claim follows by the assumption the BPAIR witness satisfies the instance, and in particular $P (w_1[j], \ldots, w_\mathcal{W}[j], w_1[j+1], \ldots, w_\mathcal{W}[j+1]) = 0$ for all $j \in [T-1]$.

The set $\Phi_{P_\pi}$: Let $P \in P_\pi$ be a polynomial, denote by

$$P'(x) \triangleq P \left( \bar{Q}_{(1,0)}(x), \ldots, \bar{Q}_{(w,0)}(x), \bar{Q}_{(1,1)}(x), \ldots, \bar{Q}_{(1,w)}(x) \right)$$

$$P''(x) = \frac{(x-s)(x-s-1)}{Z_{H+s}(x)} \cdot P_0 (x)$$

We notice $\deg P' < 2^{k+t+\lceil \log(t+1) \rceil} + d$, thus it is sufficient to show that whenever the denominator vanishes, the enumerator vanishes as well. Let $x$ be a member of $H + s$. If $x \in \{s, 1+s\}$ then $(x-s)(x-s-1) = 0$, otherwise there is some $j \in [T-1]$ such that $x = (g^j%\zeta) + s$, thus by construction $P'(x) = P (w_1[j], \ldots, w_\mathcal{W}[j], w_1[\pi(j)], \ldots, w_\mathcal{W}[\pi(j)])$, and in particular vanishes by definition of satisfaction in BPAIR.
The set $\Phi_{\text{ctrl}}$:

- For $\phi(x) = 1$, we have $\deg\left(\q_{\text{eq}(x)}\left(\bar{Q}(\text{ctrl},0)(x), x - s\right)\right) < 2k + t + \lceil \log(t+1) \rceil$ and for every $x \in H + s$, it holds by construction (Appendix C.3) $\bar{Q}(\text{ctrl},0)(x) = x - s$.

- For $\phi(x) = \frac{1}{z_{H+\epsilon}(x)}\q_{\text{eq}(x)}\left(\bar{Q}(\text{ctrl},0)(x), \bar{Q}(\text{ctrl},1)(x)\right)$ we have $\deg\left(\q_{\text{eq}(x)}\left(\bar{Q}(\text{ctrl},0)(x), \bar{Q}(\text{ctrl},1)(x)\right)\right) < 2k + t + \lceil \log(t+1) \rceil$, and $\bar{Q}(\text{ctrl},0)(s) = \bar{Q}(\text{ctrl},1)(s)$ because $\pi'(0) = 0$.

- For $\phi$ of the form $S_{r,c}(x) \cdot \q_{\text{eqv}}(\bar{Q}(\text{ctrl},0)(x), \bar{Q}(\text{ctrl},1))$, we have $\deg\left(\q_{\text{eqv}}(\bar{Q}(\text{ctrl},0)(x), \bar{Q}(\text{ctrl},1))\right) < 2k + t + \lceil \log(t+1) \rceil$, and for every $x \in H + g^t$ it holds $\bar{Q}(\text{ctrl},0)(x) = \bar{Q}(\text{ctrl},1)(x)$, by the property of the back-to-back De Bruijn routing (Appendix G.2).

- For $\phi$ of the form $S_{r,c}(x) \cdot \q_{\text{eqv}}(\bar{Q}(\text{ctrl},0)(x), \bar{Q}(\text{ctrl},1))$, we have $\deg\left(\q_{\text{eqv}}(\bar{Q}(\text{ctrl},0)(x), \bar{Q}(\text{ctrl},1))\right) < 2k + t + \lceil \log(t+1) \rceil$, and for every $x \in H + g^t$ it holds $\bar{Q}(\text{ctrl},0)(x) = \bar{Q}(\text{ctrl},1)(x)$, by the property of the back-to-back De Bruijn routing (Appendix G.2).

The set $\Phi_{\text{cmp}}$: The low degree of every $\phi \in \Phi_{\text{cmp}}$ follows by construction, as all networks $\bar{Q}(j,0), \bar{Q}(j,1)$ are routed exactly the same way as the embedded back-to-back De Bruijn routing in $\bar{Q}(\text{ctrl},1), \bar{Q}(\text{ctrl},1)$.

C.4.2 Proof of soundness (Item 2)

\textit{Proof.} Assume $V_{\text{PAIR}} \rightarrow APR(x, k, R) \in APR$ for $x = (F, T, w, P, C, B)$, and let $w_{\text{APR}}$ be a witness for it. We show $x \in \text{PAIR}$ by constructing a witness $w_{\text{PAIR}} = (\hat{w}, \pi)$ for it. We define:

- for every $j \in [w]$ the mapping $w_{j, \text{PAIR}} : [T] \rightarrow F$ by:
  \begin{align}
  w_{j, \text{PAIR}}[i] \triangleq w_{\text{APR}}(j, 0)(\left(\frac{i}{\gamma} \% \zeta\right) + s) \cdot Z_{B,(j,0)} \left(\left(\frac{\gamma}{\gamma} \% \zeta\right) + s\right) + E_{B,(j,0)} \left(\left(\frac{\gamma}{\gamma} \% \zeta\right) + s\right) \quad (47)
  \end{align}

denoting by $w_{\text{APR}}^{\text{low}}$ the low degree extension of the actual witness.

- we define:
  \begin{align}
  \pi(i) = j \iff w_{\text{APR}}^{\text{low}}(\left(\frac{\gamma}{\gamma} \% \zeta\right) + s) = (\frac{\gamma}{\gamma} \% \zeta) + s \quad (48)
  \end{align}

for $i, j \in [T]$. Otherwise we say $\pi(i) = \infty$.

We show $w_{\text{PAIR}}$ satisfies $x$. Let $\bar{Q}(r, x) \triangleq x \cdot Z_{B,(r)} + E_{B,(r)}$. Boundary constraints: $w_{\text{PAIR}}$ satisfies the boundary constraints $B$ by construction, as having $(i, j, \alpha) \in B$ implies $Z_{B,(j,0)} \left(\left(\frac{\gamma}{\gamma} \% \zeta\right) + s\right) = 0$ and $E_{B,(j,0)} \left(\left(\frac{\gamma}{\gamma} \% \zeta\right) + s\right) = \alpha$, thus $w_{j, \text{PAIR}}[i] = \alpha$ as required.

General technique for rest of soundness proof: In what follows we show that in any case where $\pi$ is not a permutation, or at least one of $P_{\pi}, P_{\pi} \neq P_{\pi}$ is not satisfied by $w_{\text{PAIR}}$ then there exists $\phi \in \Phi$ such that $\phi_{\text{PAIR}} \notin \text{RS}[F, L_{\text{cmp}}, P_{\text{cmp}}]$. The general method we use is assuming by contradiction $\phi_{\text{PAIR}} \in \text{RS}[F, L_{\text{cmp}}, P_{\text{cmp}}]$, and denoting by $\psi : F \rightarrow F$ its low-degree extension. We notice the evaluation of every such $\psi$ is over $L_{\text{cmp}}$ representable as some rational function, implied by definition of $\phi$, $\psi(x) = \frac{p(x)}{q(x)}$ where (i) $\deg p < 2k + t + \lceil \log(t+1) \rceil + d$; (ii) $\deg q < 2k + t + \lceil \log(t+1) \rceil$; and (iii) there exists some $x_0$ such that $q(x_0) = 0$ while $p(x_0) \neq 0$. In particular it must hold $q(x_\psi(x)) = p(x)$ for all $x \in L_{\text{cmp}}$, but both equation sides are polynomial of degree less than $2k + t + \lceil \log(t+1) \rceil + d + 2t + \lceil \log(t+1) \rceil < |L_{\text{cmp}}|$ that do not agree on $x_0$, thus they cannot agree on any set of size $|L_{\text{cmp}}|$, contradicting their agreement over $L_{\text{cmp}}$. 

61
**Satisfaction of $P_T$:** Assume by contradiction there is $P \in P_T$ and $j \in [T-1]$ such that $P(w[j], w[j+1]) \neq 0$. In case $g^j%\zeta \in H_0$ the contradiction is achieved using $P'_0 \in \Phi P_T$ (Equation (43)), otherwise $g^j%\zeta \in H_1$ and the contradiction is achieved using $P'_1 \in \Phi P_T$ (Equation (44)).

$\pi$ is a permutation: By Theorem G.5 it is sufficient to show $\bar{Q}_{(ctrl,0)}, \bar{Q}_{(ctrl,1)}$ is an affine embedding of a back-to-back De Bruijn of degree $t$ inducing a permutation $\pi'$ such that (i) the domain of $\pi'$ is $H$, (ii) $\pi'(0) = 0$. By Definition G.4 it is sufficient to show:

1. For all $x \in L$ it holds $\bar{Q}_{(ctrl,0)}(x + s) = x$
2. For all $x \in L$ it holds $\bar{Q}_{(ctrl,0)}(x + g^1) = \bar{Q}_{(ctrl,1)}(x + g^t)$
3. For every $r, c, l \in \{0, 1\}$, and every $x \in H_r + (W_c \setminus \{0, g^1\})$, $Q_{(ctrl,l)}(x)$ equals to $Q_{(ctrl,l)} \circ n_{r,c}^{out,b}(x)$ for some $b \in \{0, 1\}$
4. $\bar{Q}_{(ctrl,0)}(s) = \bar{Q}_{(ctrl,1)}(s)$

We now use the general technique (Appendix C.4.2), that if any of the stated above constraints does not hold there is $\phi \in \Phi$ such that $\phi_N^N[\bar{X}^{APR}] \notin RS[\mathbb{F}, L_{cmp}, \rho_{cmp}]$ by providing polynomials $p, q \in \mathbb{F}[x]$ as required by the technique.

1. In case Item 1 does not hold, the contradiction is achieved using $\frac{q_{X_{((ctrl,0),n_{id})}} X_{loc-s}}{Z_{H+s}(X_{loc})} \in \Phi_{ctrl}$
2. In case Item 2 does not hold, the contradiction is achieved using $\frac{q_{X_{((ctrl,0),n_{id})}} X_{((ctrl,1),n_{id})}}{Z_{H+s}(X_{loc})} \in \Phi_{ctrl}$
3. In case Item 3 does not hold, the contradiction is achieved using the corresponding polynomial of the form $S_{r,c}(X_{loc}) \cdot q_{mv}(X_{((ctrl,l),n_{id})}, X_{((ctrl,l),n_{id}^{out,0})}, X_{((ctrl,l),n_{id}^{out,1})}) \in \Phi_{ctrl}$
4. In case Item 4 does not hold, the contradiction is achieved using $\frac{q_{X_{((ctrl,0),n_{id})}} X_{((ctrl,1),n_{id})}}{X_{loc-s}} \in \Phi_{ctrl}$

**Satisfaction of $P_{\eta}$:** For every $j \in [w]$, we notice the pair $\bar{Q}_{(j,0)}, \bar{Q}_{(j,1)}$ is routed exactly the same as $\bar{Q}_{(ctrl,0)}, \bar{Q}_{(ctrl,1)}$, as otherwise a contradiction is achievable using $\Phi_{cp}$. Thus we conclude $\bar{Q}_{(ctrl,0)}((g^i%\zeta) + s) = \bar{Q}_{(ctrl,1)}((g^i%\zeta) + s)$ for every $i \in [T-1]$. Assuming there is $P \in P_{\eta}$ and $i \in [T-1]$ such that $P(w[i], w[\pi(i)]) \neq 0$, then the contradiction is achieved using the corresponding polynomial in $\Phi_{P_{\eta}}$. □

**C.4.3 Knowledge extraction (Item 3)**

*Proof:* The knowledge extractor is described in Equation (47) and Equation (48), and the soundness proof in Appendix C.4.2 shows it is indeed an extractor. □

**C.4.4 Instance properties (Item 4h)**

The proof of Item 4h is mostly straightforward, and very similar to the proof of Item 4h in Appendix C.2.4. We provide a full proof only to it being 5-overlapping.
\( x_{\text{APR}} \) is 5-overlapping:

Proof. Define the following 3 disjoint affine subspaces of \( L \):

\[
\begin{align*}
    S_x &\triangleq L_{\text{cmp}} = \text{span}\{g^i \mid 0 \leq i < k + R + t + d\} + g^{1+k+R+t+d} \\
    S_{g_x} &\triangleq \text{span}\{g^{i+1} \mid 0 \leq i < k + R + t + d\} + g^{2+k+R+t+d} \\
    S_{g_{x+1}} &\triangleq \text{span}\{g^{i+1} \mid 0 \leq i < k + R + t + d\} + (1 + g^{2+k+R+t+d})
\end{align*}
\]

We show there are at most 5 neighbor to each space, mapping elements of \( L_{\text{cmp}} \) to them. We notice that for every \( x \in L_{\text{cmp}} \):

(i) \( n^{\text{id}}(x) \in S_x \),

(ii) \( n^{\text{cyc}}_0(x), n^{\text{rout},b}_r(x) \in S_{g_x} \) whenever \( b + r + c = 0 \), and

(iii) \( n^{\text{cyc}}_1(x), n^{\text{rout},b}_c(x) \in S_{g_{x+1}} \) whenever \( b + r + c = 1 \). Concluding our proof.

D Algebraic linking IOP (ALI)

In this section we describe the second part of the reduction described in Appendix B (see Figure 8), namely, the ALI protocol. This protocol receives a pair \( (x_{\text{APR}},w_{\text{APR}}) \) as described in Appendix B.3 and Definition B.10, uses a single round of interaction in which the verifier sends public randomness, and ends with a pair of instances of the RS proximity testing problem from Definition B.14.

The ALI protocol is described in Appendix D.1. The main properties achieved by it were presented earlier in Theorem B.15, and we prove this Theorem in Appendix D.2. The section ends in Appendix D.3 with a justification for Conjectures B.16 and B.17 that suggest better soundness (and smaller query and verifier complexity) for the ALI protocol.

D.1 The Algebraic Linking IOP (ALI) protocol

The protocol below is a generalization of the “duplex PCP” 2-round IOP protocol from [17, Section 6.2] to the case of \( \kappa \)-independent APR instances. There, the verifier sent randomness to reach a random linear combination of the assignment and a “mask” polynomial. Since we have numerous assignments and constraints, the verifier adds randomness to check a random linear combination of all assignments and another random linear combination to check all constraints. The end result of this is that even though the APR witness has many algebraic registers, and a single codeword per register, totaling \(|T| + 1\) many RS-codewords, at the end of the ALI protocol we need to test proximity only for a pair of purported RS codewords.

ALI protocol

Input:

- Verifier has a \( \kappa \)-independent instance \( \mathbb{x} = x_{\text{APR}} = (\mathbb{F}, \mathcal{T}, \mathcal{N}, \Phi, L, L_{\text{cmp}}, \bar{\rho}, \rho_{\text{cmp}}) \)

- Prover has \( \mathbb{x} \) and a witness \( \mathbb{w} = w_{\text{APR}} = \{w_\tau \in V_\tau, \tau \in \mathcal{T}\} \) satisfying \( \mathbb{x} \), where each \( w_\tau \) was sampled uniformly from the \( \kappa|L| \)-independent space \( V_\tau \) described in Definition B.11.

Protocol:

1. Prover samples uniformly and independently random

- \( f_{\text{mask}} \in \text{RS}[\mathbb{F}, L, \rho_{\text{max}}] \);
- \( g_{\text{mask}} \in \text{RS}[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \);

63
and sends to verifier the oracle \( \mathcal{D}_{\text{assignment}} \triangleq (w, f_{\text{mask}}, g_{\text{mask}}) \).

2. Verifier performs the following:

   (a) sample and send to prover a sequence \( R \) of \( 2|T| + |\Phi| \) uniformly random \( \mathbb{F} \)-elements containing:
   - \( r_{\tau, 0}, r_{\tau, 1} \) for every \( \tau \in \mathcal{T} \)
   - \( r_\phi \) for every \( \phi \in \Phi \)

   (b) define the random constraint \( \phi_R : \mathbb{F}^{|\mathcal{V}|} \rightarrow \mathbb{F} \) by \( \phi_R(\alpha) \triangleq \sum_{\phi \in \Phi} r_\phi \cdot \phi(\alpha) \)

   (c) invoke the two following RS-IOPP sub-protocols, accepting if and only if both sub-protocols accept:

   i. verify proximity of the function \( f^{(0)} : L \rightarrow \mathbb{F} \) to the code \( \text{RS}[^F, L, \rho_{\max}] \) where
   \[
   \forall x \in L, \quad f^{(0)}(x) \triangleq f_{\text{mask}}(x) + \sum_{\tau \in \mathcal{T}} \left( r_{\tau, 0} + r_{\tau, 1} \cdot x \right)^{|L| \cdot (\rho_{\max} - \rho_\tau)} \cdot w_{\tau}(x) \quad (49)
   \]

   ii. verify proximity of the function \( g^{(0)} : L_{\text{cmp}} \rightarrow \mathbb{F} \) to the code \( \text{RS}[^F, L_{\text{cmp}}, \rho_{\text{cmp}}] \) where
   \[
   \forall x \in L_{\text{cmp}}, \quad g^{(0)}(x) \triangleq g_{\text{mask}}(x) + \phi_R(\alpha_{w, \mathcal{N}}(x)) \quad (50)
   \]

### D.2 Proof of the \( ALI \) reduction Theorem B.15

Our proof follows the order of items stated in Theorem B.15. As typical for IP and PCP statements, the most intricate parts of our proof are those dealing with soundness, and, to a lesser extent, those discussing zero knowledge and knowledge extraction.

#### D.2.1 Completeness — Part 2

Suppose \( \hat{w} \) satisfies \( x \) according to Definition B.10. Then by definition of a \( \kappa \)-independent APR, the (random) witness \( \hat{w} \) sampled by the prover in step 1 of the \( ALI \) satisfies \( x \) as well. This means that the following holds:

- \( \hat{w}_\tau \in \text{RS}[^F, L, \rho_\tau] \) for each \( \tau \in \mathcal{T} \) and \( f^{(0)} \) is in the linear span of \( \hat{w} \); thus \( f^{(0)} \in \text{RS}[^F, L, \rho_{\max}] \).
- \( \phi_{\mathcal{N}}[\hat{w}] \in \text{RS}[^F, L_{\text{cmp}}, \rho_{\text{cmp}}] \) for each \( \phi \in \Phi \) and \( \phi_R \) is in the linear span of \( \{ \phi_{\mathcal{N}}[\hat{w}] \} \), thus \( \phi_R \in \text{RS}[^F, L_{\text{cmp}}, \rho_{\text{cmp}}] \) as well, concluding \( g^{(0)} \in \text{RS}[^F, L_{\text{cmp}}, \rho_{\text{cmp}}] \).

Therefore, the completeness of the RS-IOPP used in step 2c of the \( ALI \), implies that our verifier accepts the proof above with probability 1, i.e., the \( ALI \) protocol has perfect completeness.

#### D.2.2 Soundness — Part 3

Our proof of soundness requires a few preliminary statements, stated next. The proof of soundness follows in the next sub-section.

**Preliminaries** The following claim discusses interpolants, using the definition and notation from Appendix B.1.

**Claim D.1.** Let \( S \subseteq \mathbb{F} \) and \( d, k \in \mathbb{N} \) satisfy \( d + 2k < |S| \). Given \( f : S \rightarrow \mathbb{F} \) define \( \hat{f}(x) \triangleq x^k \cdot f(x) \). Suppose that both \( \deg(\text{interpolant} f) \) \(< d + k \) and \( \deg(\text{interpolant} \hat{f}) \) \(< d + k \). Then we also have \( \deg(\text{interpolant} f) \) \(< d \).
Proof. Let \( P(X) = \text{interpolant}_f \) and let \( Q(X) \triangleq X^k \cdot P(X) \); both \( P, Q \) are viewed as members of \( \mathbb{F}[X] \). By assumption \( \deg(P) < d + k \), so \( \deg(Q) < d + 2k < |S| \). The multi-point evaluation of \( Q \) on domain \( S \) is precisely the function \( f \). The uniqueness of the interpolant, along with the observation \( \deg(Q) < |S| \), imply that \( \text{interpolant}_f = Q \). Therefore, the assumption \( \deg(\text{interpolant}_f) < d + k \) gives \( \deg(Q) < d + k \). By construction \( \deg(P) = \deg(Q) - k \) and this completes the proof.

The next lemma says that linear spaces whose members are “close on average” to a linear error correcting code, have small support.

**Lemma D.2** (Proximity to codes implies small support). Let \( C \subset \mathbb{F}^S \) be an \( \mathbb{F} \)-linear code of blocklength \( \leq |\mathbb{F}| \) and relative distance \( \delta \). Fix \( c > 6/\delta \). Suppose \( V \subset \mathbb{F}^S \) satisfies

\[
\Pr_{v \in \text{span}(V)} \left[ \Delta_H(v, C) \leq \frac{1}{c} \right] > 1/|\mathbb{F}|.
\]

Then there exists \( S' \subset S \) of density \( \mu(S'/S) \geq 1 - \frac{2}{c} \) such that \( V|_{S'} \subset C|_{S'} \).

The proof of the lemma above requires a result from [95] (stated as Lemma 1.6 there). We recall and prove that lemma next, then prove Lemma D.2.

**Lemma D.3** (Average distance amplification). Let \( C \subset \mathbb{F}^S \) be a linear space. If \( f_1, \ldots, f_k \in \mathbb{F}^S \) are such that there exists \( f_i \) that is \( \epsilon \)-far from \( C \) in relative Hamming distance, then

\[
\Pr_{r_1, \ldots, r_k \in \mathbb{F}} \left[ \Delta_H \left( \sum_{i=1}^k r_i f_i, C \right) \leq \frac{\epsilon}{2} \right] \leq 1/|\mathbb{F}|
\]

**Proof of Lemma D.2.** By Lemma D.3 we have

\[
\forall v \in \text{span}(V), \quad \Delta_H(v, C) \leq \frac{2}{c}.
\]

Since \( 2/c < \delta/2 \) by assumption, we conclude that the codeword of \( C \) that is closest to \( v \in \text{span}(V) \) is unique, denote it by \( \tilde{v} \). Define \( S_v \triangleq \{ x \in S \mid v(x) \neq \tilde{v}(x) \} \) and for \( V' \subseteq \text{span}(V) \) let \( S_{V'} = \bigcup_{v \in V'} S_v \). To prove Lemma D.2 it suffices to show

\[
\mu \left( S_{\text{span}(V)}/S \right) \leq \frac{2}{c}
\]

by setting \( S' = S \setminus S_{\text{span}(V)} \).

We prove (52) by way of contradiction, namely, we show that \( \mu(S_{\text{span}(V)}/S) > \frac{2}{c} \) and (51) together imply that (51) is false, so (52) holds.

Write \( v = \tilde{v} + v' \) where \( v' \) has relative Hamming weight \( \mu(S_{v'}/S) \). Abusing notation, we identify \( v \) with \( v' \) and henceforth assume the codeword closest to \( v \) is \( 0 \) and that \( S_v \) denotes the support of \( v \), i.e., the set of its nonzero entries.

Since (51) implies \( \mu(S_v/S) < \delta/3 \), if (52) is false then there exists some \( V' \subset V \) such that

\[
\mu(S_{V'}/S) \in \left( \frac{2}{c}, \frac{4}{c} \right) \subseteq \left( \frac{2}{c}, \delta - \frac{2}{c} \right)
\]

The containment follows because \( 4/c < 2\delta/3 < \delta - 2/c \). By linearity of expectation, the expected support size of a random word in \( \text{span}(V') \) is precisely \( (1 - 1/|\mathbb{F}|)|S_{V'}| \) which is strictly greater than \( (1 - 1/|S_{V'}|)|S_{V'}| \) because \( |S_{V'}| < |S| \leq |\mathbb{F}| \). Thus, it must be the case that some \( v \in \text{span}(V') \) is fully supported on \( S_{V'} \), which means that the relative support size of \( v \) is in \( \left( \frac{2}{c}, \delta - \frac{2}{c} \right) \); we conclude \( v \) has relative distance \( \mu(S_{V'}/S) > 2/c \) from \( C \), contradicting (51) and completing the proof.
Soundness analysis

Proof of Item 3 of Theorem B.15. We prove the contrapositive: If neither item 3a nor item 3b of Theorem B.15 hold, which means both of the following items hold:

1. \[ \Pr \left[ \Delta_H \left( f^0, RS[F, L, \rho_{\text{max}}] \right) \leq \frac{\delta}{2\varepsilon} \right] > 1/|F| \]
2. \[ \Pr \left[ \Delta_H \left( g^0, RS[F, L_{\text{cmp}}, \rho_{\text{cmp}}] \right) \leq \frac{\delta}{2\varepsilon} \right] > 1/|F| \]

Then \( z \in \text{APR} \). Details follow.

We apply Lemma D.2 to Item 1 above, while setting the constant \( c \) in that lemma to

\[ c \triangleq \frac{2\zeta}{\delta} \tag{53} \]

The assumptions of Lemma D.2 hold because \( \zeta > 3 \) (cf. Equation (7)) and \( \delta \leq 1 - \rho_{\text{max}}, \) hence \( c > 6/(1 - \rho_{\text{max}}) \) as required by Lemma D.2. By that lemma we deduce the existence of a set \( S \subset L, \mu(S/L) \leq \frac{2}{\varepsilon} \) such that for all \( \tau \in \mathcal{T} \) we have both

\[ w_{\tau \mid L \setminus S} \in RS[F, L, \rho_{\text{max}}]|_{L \setminus S} \text{ and } (x^{L \setminus (\rho_{\text{max}} - \rho)} \cdot w_{\tau})|_{L \setminus S} \in RS[F, L, \rho_{\text{max}}]|_{L \setminus S}. \tag{54} \]

Let \( d = \deg (w_{\tau \mid L \setminus S}) < \rho_{\text{max}} \cdot |L| \) and \( k = |L| \cdot (\rho_{\text{max}} - \rho_{\tau}) \). We have

\[ d + 2k < |L| (\rho_{\text{max}} + 2(\rho_{\text{max}} - \rho_{\tau})) < |L| \cdot 3\rho_{\text{max}} < |L \setminus S|. \]

The last inequality follows from \( \rho_{\text{max}} \leq 1/4 \) and \( |S| \leq 2/c \leq 1/4 \) (see Equations (7) and (53)). So by Claim D.1 we conclude

\[ \forall \tau \in \mathcal{T} \quad w_{\tau \mid L \setminus S} \in RS[F, L, \rho_{\tau}]|_{L \setminus S}. \tag{55} \]

Let \( H_0 = \{ x \in L_{\text{cmp}} \mid N(x) \cap S \neq \emptyset \} \); a union bound gives \( |H_0| \leq \Theta |S| \). Let \( w'_{\tau} \) be the low degree extension of \( w_{\tau \mid L \setminus S} \) to domain \( L \), noticing \( w'_{\tau} \in RS[F, L, \rho_{\tau}] \). Let \( \mathcal{W} = \{ w_{\tau} \mid \tau \in \mathcal{T} \} \). Recall the assumption that \( z \) has \( \delta \)-distance, and let \( C \) be the linear code of minimal distance \( \delta \) that contains \( RS[F, L_{\text{cmp}}, \rho_{\text{cmp}}] \) as required by Item 2). By Equation (55) and Item 2 we conclude \( \phi_N[w'] \subset C \) for each \( \phi \in \Phi \). Thus, to complete our soundness analysis, we need only show that \( \phi_N[w'] \subset RS[F, L_{\text{cmp}}, \rho_{\text{cmp}}] \) for each \( \phi \in \Phi \).

Consider the set of functions

\[ \{ \phi_N[w'] \mid \phi \in \Phi \} \cup \{ g_{\text{mask}} \} \]

and let \( V \) denote the linear span of this set. Since, by assumption, \( \phi_N[w'] \) agrees with \( \phi_N[w] \) on \( L_{\text{cmp}} \setminus H_0 \), we conclude that \( g^0 \mid_{L_{\text{cmp}} \setminus H_0} \in V_{L_{\text{cmp}} \setminus H_0}. \) Therefore, if there exists even one member of \( \{ \phi_N[w'] \mid \phi \in \Phi \} \) that does not belong to \( RS[F, L_{\text{cmp}}, \rho_{\text{cmp}}] \), then Lemma D.3 implies

\[ \Pr \left[ \Delta_H \left( g^0, RS[F, L_{\text{cmp}}, \rho_{\text{cmp}}] \right) \leq \frac{1}{2} (\delta - \mu(H_0/L_{\text{cmp}})) \right] \leq 1/|F| \]

which contradicts Item 2 stated at the beginning of this proof, because \( \frac{\delta}{2\varepsilon} \leq \frac{1}{2} (\delta - \mu(H_0/L_{\text{cmp}})) \) by our choice of \( \zeta \) in Equation (7). Therefore we conclude that

\[ \forall \phi \in \Phi, \quad \phi_N[w'] \subset RS[F, L_{\text{cmp}}, \rho_{\text{cmp}}] \tag{56} \]

and hence \( w' \) satisfies \( z \), completing the soundness analysis. \( \square \)

Remark D.4 (Potential for improvement of soundness analysis). The \( \times 2\zeta \) loss in soundness, compared to distance \( \delta \), is due to two factors. Lemma D.3 “costs” a \( \times 2 \) factor, and the union bound used in the proof above incurs another \( \times \left( 1 + \Theta \frac{|L|}{L_{\text{cmp}}} \right) \) loss. It remains an interesting open problem to decide if either factor is actually required (cf. Conjectures B.16 and B.17).
D.2.3 Knowledge Extraction — Part 4

The proof of Item 4 of Theorem B.15 relies on the following lemma. After proving it, we complete the proof of knowledge extraction. Recall we assume $\varphi$ has $\delta$-distance.

**Lemma D.5.** Suppose there exists $L'_{\text{cmp}} \subseteq L_{\text{cmp}}$ such that all of the following hold:

1. $\mu(L'_{\text{cmp}}/L_{\text{cmp}}) > 1 - \delta$
2. $\phi_N[w]|_{L'_{\text{cmp}}} \in \text{RS}[F, L_{\text{cmp}}, \rho_{\text{cmp}}]|_{L'_{\text{cmp}}}$ for every $\phi \in \Phi$
3. $\omega_{\tau}|_{L'} \in \text{RS}[F, L, \rho_{\tau}]|_{L'}$ for each $\tau \in \mathcal{T}$

Then $\varphi \in \text{APR}$ and the assignment $\omega' := \{\omega'_\tau \mid \tau \in \mathcal{T}\}$ where $\omega'_\tau$ is the low-degree extension of $\omega_{\tau}|_{L'}$ to $L$ witnesses $\varphi \in \text{APR}$.

**Proof.** By definition of $\omega'$ and by assumption 2 we conclude $\omega'_\tau \in \text{RS}[F, L, \rho_{\tau}]$ for all $\tau \in \mathcal{T}$. By assumption 1 we know there is some codeword $w \in \text{RS}[F, L_{\text{cmp}}, \rho_{\text{cmp}}]$ that is within distance $\delta$ of $\phi_N[w]$, while $\phi_N[\omega']$ is a codeword of $C$ having relative distance at least $\delta$. As $\varphi$ is a $\delta$-distance instance, thus $\phi_N[\omega'] \in \text{RS}[F, L, \rho_{\tau}]$. Finally we conclude $(\varphi, \omega') \in \text{R_{APR}}$ by definition of $\text{R_{APR}}$.

**Lemma D.6.** There exists a Las Vegas (randomized) polynomial time algorithm $E$ that satisfies the following condition. Given as input a $\delta$-distance instance $\varphi$ with $\rho_{\text{max}} \leq 1/4$ and $\omega \in (F^L)^T$ for which there exists $L'_{\text{cmp}}$ that satisfies the assumptions of Lemma D.5 with respect to $\varphi$ and $\omega$, the output of $E$ on input $(\varphi, \omega)$ is a witness $\omega' \subset (F^L)^T$ that satisfies $\varphi$.

**Proof of Theorem B.15, Part 4.** As argued in the proof of soundness above (and using the notation there), if neither item 3a nor item 3b of Theorem B.15 hold, then Equation (55) holds, and moreover, the low degree extension $\omega'$ of $\omega|_{\tau}|_{L \setminus S}$ (for $S$ defined there) is a satisfying assignment. Thus, our extractor will find this $\omega'$ (with high probability), using the polynomial time Guruswami-Sudan (GS) list-decoding algorithm [65].

Recall that for $\text{RS}[F, S, \rho], |S| = n$ and $f : S \rightarrow F$, the GS algorithm runs in time $\text{poly}(n)$ and outputs a list $L_f \subset \text{RS}[F, S, \rho], |L_f| = \text{poly}(n)$ that contains every codeword that agrees with $f$ on more than a $\sqrt{\rho}$-fraction of entries. In other words, when given $f : L \rightarrow F$ that agrees with some codeword $w \in \text{RS}[F, L, \rho]$ on more than a $\sqrt{\rho}$-fraction of entries, the GS algorithm will return $w$ as part of $L_f$.

Notice that in our case, the assumption $\rho_{\text{max}} \leq 1/4$ along with the upper bound $|S| \leq |L|/4$, which follows from Equations (7) and (53), implies that the set of inputs on which $w \in \omega$ and $w' \in \omega'$ is of size at least $\sqrt{\rho}$. We apply the following extractor.

\[ E(\varphi, \omega) \]

1. Sample uniformly random $\vec{a} \in \mathbb{F}^T$ and compute $f : L \rightarrow \mathbb{F}$ by $f(x) \overset{\Delta}{=} \sum_{\tau \in T} a_{\tau} \omega_{\tau}(x)$
2. Let $L_f$ be the output of the GS list-decoding algorithm on $f$.
3. For each $g \in L_f$,
   a) let $S_g = \{x \in L_{\text{cmp}} \mid \forall (\tau, N) \in \mathcal{N}, f(N(x)) = g(N(x))\}$;
   b) let $\omega' = \{\omega'_\tau \mid \tau \in \mathcal{T}\}$ where $\omega'_\tau$ is the low-degree extension of $\omega_{\tau}|_{\mathcal{N}(S_g)}$;

67
Clearly \( E \) runs in polynomial time in its input because \( |L_f| = \text{poly}(|L|) \). Additionally, it is a one-sided error algorithm because if it returns "success" then \( w' \) indeed satisfies \( x \). Thus it only remains to analyze the probability of failure, given by the following claim.

**Claim D.7.** Suppose \( L'_\text{cmp} \) satisfies all properties of Lemma D.5 and is of maximal size with respect to property 2 of that Lemma. Then

\[
\Pr_a \left[ \exists g \in L_f, S_g = L'_\text{cmp} \right] \geq 1 - \frac{|L'_\text{cmp}| \delta}{|F|}.
\]

Assuming the claim, when examining \( g \in L_f \) with \( S_g = L'_\text{cmp} \) we have \( w_{\tau}|_{N(S_g)} \in \text{RS}[F, L, \rho_{\tau}]|_{N(S_g)} \) for any \( \tau \in \mathcal{T} \) by assumption and the low degree extension \( w'_{\tau} \) of \( w_{\tau}|_{N(S_g)} \) witnesses \( x \in \text{APR} \). This completes the proof of Lemma D.6 but for the proof of Claim D.7, which appears next. \( \square \)

**Proof of Claim D.7.** The assumption that \( L'_\text{cmp} \) is maximal with respect to property 2 of Lemma D.5 means that for each \( \alpha \in L'_\text{cmp} \setminus L'_{\text{cmp}} \) there exists \( \beta_{\alpha} \in \mathcal{N}(\alpha) \) and some \( \tau_{\alpha} \in \mathcal{T} \) such that \( w_{\tau_{\alpha}}(\beta_{\alpha}) \) does not agree with the low-degree extension of \( w_{\tau_{\alpha}}|_{N(L'_{\text{cmp}})} \) that we shall denote by \( w'_{\tau_{\alpha}} \). Let \( e_{\tau_{\alpha}} : L \rightarrow F \) be the "error function" related to \( w'_{\tau_{\alpha}} \), defined as \( e_{\tau_{\alpha}}(x) = w'_{\tau_{\alpha}}(x) - w_{\tau_{\alpha}}(x) \). We have

\[
\Pr_a \left[ f(\beta_{\alpha}) = \sum_{\tau \in \mathcal{T}} a_{\tau} w'_{\tau}(\beta_{\alpha}) \right] = \Pr_a \left[ \sum_{\tau \in \mathcal{T}} a_{\tau} e_{\tau}(\beta_{\alpha}) = 0 \right] = \frac{1}{|F|}
\]

the last equality holds because by assumption \( e_{\tau_{\alpha}}(\beta_{\alpha}) \neq 0 \). Applying a union bound to \( \alpha \in L_{\text{cmp}} \setminus L'_{\text{cmp}} \) we conclude that (57) holds with probability at least \( 1 - \frac{|L_{\text{cmp}}| - |L'_{\text{cmp}}|}{|F|} \geq 1 - \frac{|L_{\text{cmp}}| \delta}{|F|} \), and this completes the proof. \( \square \)

### D.2.4 Perfect Zero-Knowledge — Part 5

Our proof follows [17, Section 6] but we make the simplifying assumption that the verifier’s first message is the sequence of randomness \( R \) mentioned in Step 2a (see Remark D.8). The ALI protocol assumes two RS-IOPP systems, used in steps 2(c)i and 2(c)ii there, each with its own prover and verifier, let \( P_f, P_g \) denote the two provers, respectively. Our STIK (and STARK) instantiates both \( P_f, P_g \) to be the FRI prover (for different RS code parameters) but our proof of zero knowledge works for any choice of RS-IOPP because our simulator uses the relevant RS-IOPP prover(s) in a black-box manner. We assume messages from the verifier have a canonical format that indicates which RS-IOPP prover is being addressed, and which oracle is being queried among \( \bigcirc_{\text{assignment}} \) and the various oracles produced by the pair of RS-IOPP protocols.

**The simulator** Our straight-line PZK simulator is denoted Sim. Given a verifier \( V^* \) and instance \( x \), the simulator starts by sampling uniformly random functions \( f^{(0)} \in \text{RS}[F, L, \rho_{\text{max}}] \) and \( g^{(0)} \in \text{RS}[F, L_{\text{cmp}}, \rho_{\text{cmp}}] \) and recording them. The simulator also instantiates the two RS-IOPP provers — \( P_f \) and \( P_g \) corresponding to steps Item 2(c)i, Item 2(c)ii of the ALI protocol — with \( f^{(0)} \) and \( g^{(0)} \), respectively. It also invokes \( V^* \) and records the first message, which is the randomness \( R \) provided by \( V^* \). The simulator now continues to run \( V^* \). All messages and queries directed by \( V^* \) to one of the two RS-IOPP protocols (dealing with \( f^{(0)} \) and \( g^{(0)} \)) are managed by Sim by invoking the corresponding IOPP prover(s) \( P_f, P_g \).
To complete the description of Sim we need only explain how it answers queries to \( O_{\text{assignment}} = (w, f_{\text{mask}}, g_{\text{mask}}) \). Recall that \( w \) is a collection of functions, each with domain \( L \), and this is also the domain of \( f_{\text{mask}} \); the domain of \( g_{\text{mask}} \) is \( L_{\text{cmp}} \). As in [17, Section 6], Sim maintains a set of partial functions \( w^*, f_{\text{mask}}^*, g_{\text{mask}}^* \) (with the same domains as \( w, f_{\text{mask}}, g_{\text{mask}} \)); all these functions are initialized with \(*\) values that indicate “undetermined”. When a function in \( O_{\text{assignment}} \) is queried, Sim answers with the value recorded in \( w^*, f_{\text{mask}}^*, g_{\text{mask}}^* \), if determined, and will otherwise determine it, i.e., change its value from \(*\) to some element of \( \mathbb{F} \). The process by which undetermined values get determined is described next:

1. A query \( x_0 \in L \) sent to a function \( f \in w \cup \{ f_{\text{mask}} \} \), is determined jointly for all functions \( f' \in w \cup \{ f_{\text{mask}} \} \). Consider Equation (49). The term on the left hand side, \( f^{(0)}(x_0) \), is already fixed by Sim. On the right hand side, the terms
   \[
   \left\{ r^\gamma_{\tau,0} + r^\gamma_{\tau,1} \cdot x_0 | L \cdot (\rho_{\max} - \rho_{\gamma}) \mid \tau \in T \right\}
   \]
   are all fixed. The only undetermined values are those of \( f_{\text{mask}}^*(x_0) \) and \( \{ w^*_\tau(x_0) \mid \tau \in T \} \). Thus, our simulator determines these undetermined values by sampling a uniformly random solution to the linear constraint imposed by \( f^{(0)}(x_0) \) and Equation (58).

2. A query \( y_0 \in L_{\text{cmp}} \), sent to the function \( g_{\text{mask}} \), is determined thus. The left hand side of Equation (50) is already determined by Sim. For all \( x_0 \in \mathcal{N}(y_0) \), our simulator determines the value of all \( \{ w^*_\tau(x_0) \mid \tau \in T \} \) and \( f_{\text{mask}}^*(x_0) \) using the process described in Item 1. After all such values \( w^*_\tau(x_0) \) are determined for \( x_0 \in \mathcal{N}(y_0) \), notice that the rightmost summand of Equation (50) is also determined. Thus, Sim determines \( g_{\text{mask}}^*(y_0) \) to be the unique field element that causes the linear constraint of Equation (50) to be satisfied.

**Perfect zero knowledge** First, notice Sim is straight-line, i.e., it never restarts \( V^* \). To prove perfect zero knowledge, we shall show that the distribution sampled by Sim interacting with \( V^* \) on a satisfiable instance \( \kappa \), is the same as the distribution on transcripts of the interaction between \( V^* \) and an honest prover holding a witness for \( \kappa \) and operating as described in ALI. Notice the following facts about the distribution supplied by the honest prover:

1. Each \( w_\tau \) is sampled uniformly and independently from a \( \kappa |L| \)-wise independent space;
2. \( \phi_R (\omega_\kappa L \mathcal{N}(y_0)) \) is determined by \( R \) and \( \{ w_\tau(x_0) \mid x_0 \in \mathcal{N}(y_0), \tau \in T \} \);
3. the pair \( (f_{\text{mask}}, g_{\text{mask}}) \) is sampled uniformly from \( RS[\mathbb{F}, L, \rho_{\max}] \times RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \);
4. consequently, independently of \( R \) and \( \kappa \), the pair \( (f^{(0)}, g^{(0)}) \) is sampled uniformly from \( RS[\mathbb{F}, L, \rho_{\max}] \times RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \).

Item 4 above relies on the completeness property, which says that if \( \kappa \) satisfies \( \kappa \) then the rightmost summand of Equation (50) is a codeword of \( RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \).

Consequently, for every fixing of the first verifier message \( R \), and for every subset \( S \subset L, |S| < \kappa |L| \), the distribution on \( \kappa |S|, f_{\text{mask}}|S|, f^{(0)}|S \) generated by the honest ALI prover is the uniform distribution on field elements satisfying the linear constraint of Equation (49) for each \( x \in S \). By construction, the distribution supplied by Sim invoking \( V^* \) which makes these queries \( S \), is precisely the same distribution.

Next, assume the aforementioned \( S \) includes all \( x_0 \in \mathcal{N}(S') \), where \( S' \subset L_{\text{cmp}} \) is the set of queries made by \( V^* \) to \( g_{\text{mask}} \). By Item 2, the distribution on the rightmost term of Equation (50) generated by the
honest ALI prover is the exact same distribution as that supplied by Sim invoking V^*. By Items 3 and 4 above, the distribution on g^{(0)}|_{S'}, g_{mask}|_{S'} is thus the uniform distribution on field elements satisfying the linear constraint of Equation (50) for every y_0 \in S'. By construction, Sim produces the same distribution on g^{(0)}|_{S'}, g_{mask}|_{S'}.

Finally, the distribution of messages between V^* and the sub-provers P_1, P_2 used as part of the RS-IOPP protocols are, by construction, the same distribution as provided by Sim invoking V^* because both the honest ALI prover and the simulator invoke the same sub-provers P_1, P_2 and supply them with the exact same uniformly random inputs f^{(0)} and g^{(0)}.

We have shown that the distribution output by the straight-line simulator Sim invoking V^* is equal to the distribution output by V^* interacting with an honest prover on a satisfiable instance. This completes the proof of Item 5 of Theorem B.15.

Remark D.8. Inspection reveals that the proof of perfect zero knowledge appearing in [17, Section 6] can be adapted to our case (details omitted). That proof is more complicated, as it is designed to address the case where the verifier may query the first oracle (O_assignment) even before sending the randomness R. We point out that in all concrete STARK realizations — both interactive (iSTARK) and non-interactive (nSTARK) — this assumption is unrealistic.

D.2.5 Arithmetic complexity — Part 6

In this section we prove Item 6 of Theorem B.15. Verifier complexity is as stated because the verifier’s only action during the ALI protocol is to sample the randomness R. Regarding prover complexity, we follow the steps of the protocol:

1. In step 1 the prover samples random functions w, f_{mask}, g_{mask}. For each member of w this requires 3 \cdot |L| \log |L| arithmetic operations, as stated in Item 4h of Theorem B.12, and Theorem B.2.

2. In the beginning of Item 2(c)i the prover computes f^{(0)}, by performing point wise computation using Equation (49), at a cost of 2|L||T| multiplications, and 2|L|(|T| + 1) additions over \mathbb{F}. The total arithmetic complexity of this part is less than 5|T| \cdot |L|.

3. In step 2(c)ii the prover computes g^{(0)} and (its proximity proof is discussed later), by performing a complete point wise computation using |L_{cmp}| \cdot (T_{arith}(\Phi) + |\Phi|) multiplications, and |L_{cmp}| \cdot (T_{arith}(\Phi) + |\Phi| + 1) additions over \mathbb{F}. This accounts for the rightmost summand of Equation (8).

Summing up, the total cost is as stated in Equation (8), and this completes our complexity analysis for the ALI protocol.

D.3 Conjectured soundness

In this section we discuss the rationale behind Conjectures B.16 and B.17, starting with the latter one.

Pure pseudo-prover Informally, Conjecture B.16 captures the intuition that pure pseudo-provers achieve the largest soundness-error against general instances. A pure pseudo-prover is one that generates pure pseudo-assignments O_{assignment}^* = (w^*, f^*_{mask}, g^*_{mask}); a pure pseudo-assignment satisfies

$$\text{span} (w^*, f^*_{mask}) \subseteq \text{RS}[\mathbb{F}, L, \rho_{max}] \bigvee \text{span} (\{\phi(\alpha_{w^*,\mathcal{N}}) | \phi \in \Phi\}, g^*_{mask}) \subseteq \text{RS}[\mathbb{F}, L_{cmp}, \rho_{cmp}].$$
In words, a pure pseudo-assignment selects \( O^*_\text{assignment} \), such that either each \( w^*_\tau \) is a codeword of the relevant code \( RS[\mathbb{F}, L, \rho_L] \), in which case the resulting constraint polynomials will be, with very high probability, maximally far from \( RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \), or else \( P^* \) fixes the intended values of each \( \phi \in \Phi \) to correspond to a member \( g_\phi \in RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \) and then find, for each \( y \in L_{\text{cmp}} \), a setting for the values of \( w(N(y)) \) so that \( \phi(\alpha_{w^*,N})(y) = g_\phi(y) \); in this case, the resulting \( w \) will likely be maximally far from \( RS[\mathbb{F}, L, \rho_{\text{max}}] \).

Mixed pseudo-prover The “attack” imagined to support Conjecture B.17 is the following: on input \( z \), the pseudo-prover starts with some \( w^* \subseteq RS[\mathbb{F}, L, \rho_{\text{max}}] \), which leads to \( \{ \phi(\alpha_{w^*,N}) \mid \phi \in \Phi \} \) being maximally far from \( RS[\mathbb{F}, L_{\text{cmp}}, \rho_{\text{cmp}}] \). Next, for some \( \epsilon \) fraction of the entries of \( L_{\text{cmp}} \), the attacker “reverse-engineers” a change to an \( \epsilon \)-fraction of the entries of \( w^* \) as to make each member of \( \{ \phi(\alpha_{w^*,N}) \mid \phi \in \Phi \} \) \( \epsilon \)-closers to low-degree. Assuming the spaces \( L_0, L'_0 \) are good for \( z \), as we do in Conjecture B.17, this modification “ruins” an \( \epsilon \cdot 2^n |L_{\text{cmp}}| \) fraction of the entries of \( w^* \), as expressed by Equation (9). We stress that we do not know how to efficiently instantiate this attack for general APR instances because the “reverse-engineering” step above may be hard to solve on general instances.

E An algebraic intermediate representation of the DNA profile match

Let us describe the algebraic intermediate representation (abbrev. AIR, see Section 2.2) of the DNA profile match (DPM) program, that uses the Rijndael-160 based hash function. This exposition will show how we achieve the quantities that are specified in Tables 4 and 5 (i.e., the width \( w \), the cycles \( c \), etc.), by providing a “bottom up” description of our algebraic construction.

We first recount the Rijndael block cipher, but from an algebraic perspective (Appendix E.1). We then explain our AIR constraints for the Rijndael cipher (Appendices E.2 and E.3). Following that, we describe the AIR for the transformation from a block cipher to a cryptographic hash function (Appendix E.4). Finally, we show our to implement the AIR of the logic of the DPM program, that performs an exhaustive search to compare the loci pairs that are stored in the elements of a hashchain (Appendix E.5).

The Advanced Encryption Standard (AES) instantiates Rijndael with 128-bit block size and 128-bit, 192-bit, or 256-bit key sizes. We denote by Rijndael-160 the cipher with 160-bit block size and 160-bit key size, and hence output (cipher-text) size of 160 bits. Assuming that Rijndael-160 is an ideal cipher (cf. Appendix E.4), it can be used to build a collision-resistant hash function (CRHF) with an 80-bit security parameter.

It should be noted that Rijndael with 192-bit (256-bit) block and key sizes can be used to build CRHF with a security parameter of 96 bits (128 bits), and that these stronger parameters entail a rather mild overhead in our algebraic construction (see Table 4). However, the STARK construction that we benchmark has 60 bits of security, and therefore the stronger hash functions will not provide better security with our benchmarked system.

Our reference code will be available at https://github.com/elibensasson/STARK.

Notation. Let \( g \) denote a primitive element of \( \mathbb{F}_{2^{64}} \), i.e., \( (g) = \mathbb{F}_{2^{64}}^* \). We assume throughout that field elements are represented according to a standard basis \( \{1, g, g^2, g^3, \ldots, g^{63}\} \), rather than a normal basis. We denote by \( R(t) \in \mathbb{F}_{2^{64}} \) the content of the algebraic register \( R \) at cycle \( t \) of the execution (e.g., \( K00(0) \) is the field element that resides in the register \( K00 \) during the first cycle).
E.1 Algebraic description of the Rijndael cipher

The input to the Rijndael cipher can be regarded as a plain-text array of \(4n\) elements and a key array of \(4n\) elements, such that each element resides in \(\mathbb{F}_{2^n}\). Rijndael executes in \(n + 6\) rounds, where each round consists of the following four steps (except for the last round that skips Step 3):

1. **SubBytes** - Given byte \(x\), compute \(y = Mx^{-1} + b\), where \(M \in \mathbb{F}_{2^8}^{8 \times 8}\), \(b \in \mathbb{F}_{2^8}^{8 \times 1}\) are constants.

2. **ShiftRows** - For \(i \in \{1, 2, 3, 4\}\), perform \(i\) cyclic shifts (rightwards) of the \(i\)th row of the plain-text matrix.

3. **MixColumns** - For \(j \in \{1, 2, \ldots, n\}\), multiply the \(j\)th column of the plain-text matrix by the following constant circulant MDS matrix:

\[
\begin{bmatrix}
P0[j](t+1) \\
P1[j](t+1) \\
P2[j](t+1) \\
P3[j](t+1)
\end{bmatrix} =
\begin{bmatrix}
g0 & g1 & 1 & 1 \\
1 & g0 & g1 & 1 \\
1 & 1 & g0 & g1 \\
g1 & 1 & 1 & g0
\end{bmatrix}
\begin{bmatrix}
P0[j](t) \\
P1[j](t) \\
P2[j](t) \\
P3[j](t)
\end{bmatrix}.
\]

Here, \(g_0\) is a specific field element of \(\mathbb{F}_{2^8}\) with \(\langle g_0 \rangle = 51\), and \(g_1 \triangleq g_0 + 1\) generates \(\mathbb{F}_{2^8}^*\). We note that MixColumns also can be defined as computing the following linear combinations:

\[
\begin{align*}
P0[j](t+1) &= g_0 \cdot P0[j](t) + g_1 \cdot P1[j](t) + P2[j](t) + P3[j](t) \\
P1[j](t+1) &= P0[j](t) + g_0 \cdot P1[j](t) + g_1 \cdot P2[j](t) + P3[j](t) \\
P2[j](t+1) &= P0[j](t) + P1[j](t) + g_0 \cdot P2[j](t) + g_1 \cdot P3[j](t) \\
P3[j](t+1) &= g_1 \cdot P0[j](t) + P1[j](t) + P2[j](t) + g_0 \cdot P3[j](t)
\end{align*}
\]

4. **AddRoundKey**:

   - **Key-Scheduler**
     - Using \(Rcon(t) \triangleq g_0^{t-1}\), the first column in the new key-matrix is computed according to:

\[
\begin{bmatrix}
K00(t+1) \\
K10(t+1) \\
K20(t+1) \\
K30(t+1)
\end{bmatrix} =
\begin{bmatrix}
\text{SubBytes}(K14(t)) + K00(t) + Rcon(t) \\
\text{SubBytes}(K24(t)) + K10(t) \\
\text{SubBytes}(K34(t)) + K20(t) \\
\text{SubBytes}(K04(t)) + K30(t)
\end{bmatrix}.
\]

   - The other key elements are computed as: \(K[i, j](t+1) = K[i, j-1](t+1) + K[i, j](t)\).

   - The new key is added by combining each byte of the current plain-text with the corresponding byte of the key, using bitwise exclusive-OR.

E.2 Implementation technique of the Rijndael cipher

Per our complexity measures (cf. Section 2.2), we wish to construct an efficient representation of a hash function by using algebraic constraints. However, the Rijndael cipher is computed over \(\mathbb{F}_{2^8}\) with field operations modulo the irreducible polynomial \(x^8 + x^4 + x^3 + x + 1\), while the operations in our IOP system are over \(\mathbb{F}_{2^{64}}\), defined using a different primitive polynomial. The properties of finite fields entail that for
any field $\mathbb{F}_{p^m}$ and $k|m$, there exists a subfield $\mathbb{F}_{pk}$. Therefore, there is an isomorphism between $\mathbb{F}_{2^8}$ and a subfield of $\mathbb{F}_{2^64}$.

We obtain such an isomorphism by mapping a primitive element of $\mathbb{F}_{2^64}$ to a primitive element of the subfield with $F' \cong \mathbb{F}_{2^8}$, so that the mapping is implied by their powers. This is done by finding an element of $\mathbb{F}_{2^64}$ with order $2^8 - 1$.

We then transform all constants needed for Rijndael to their representation in $F'$, and perform all the field operations in $F'$. Importantly, this enables an efficient constraint for the SubBytes step of Rijndael, since we can represent the field inverse via a single multiplication in $F'$. Specifically, by using an auxiliary element $z \in F'$, the constraint $y = x^{-1}$ can be represented via $y \cdot z = x$. By contrast, a naïve implementation of the inverse operation would require auxiliary elements $\{b_i\}_{i=0}^7$, booleanity constraints $\cup\{b_i(b_i+1)\}_{i=0}^7$, and a polynomial of degree 8 with 256 summands.

The full SubBytes S-box is defined according to $x \rightarrow M \cdot x^{-1} + b$, where $M \in F_2^{8 \times 8}$ and $b \in F_2^{8 \times 1}$ are constants. Adding the constant $b$ is a simple field addition in $F'$, whereas the multiplication by the constant matrix $M$ can be represented using a linear transformation $T : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$. Using algebraic properties, we have that any linear transformation can be represented by a linearized polynomial [78, Chaper 3.4]. We obtain the linearized polynomial $C(x) = \sum_{i=0}^7 c_i x^i$ by finding coefficients $\{c_i\}_{i=0}^7$ that satisfy $C(a_i) = b_i$, where $(a_0, a_1, \ldots, a_7)$ is a basis for the domain of $T$ and $(b_0, b_1, \ldots, b_7)$ is a basis for the range of $T$.

While the degree of $C(x)$ is 128, the degree of our constraint polynomial for the entire Rijndael-160 computation is in fact only 8. At the high-level, the degree reduction is achieved via a decomposition $C'(x) = C_1(C_2(C_3(x)))$ with 3 auxiliary field elements, using an AIR such as $\{z' + C_3(x), z'' + C_2(z'), z = C_1(z'')\}$. Per Section 2.3, this AIR is translated into a single constraint $(z' = C_3(z)) \land (z'' = C_2(z')) \land (z = C_1(z''))$, where the logical-AND is accomplished using the ALI protocol, i.e., random coefficients that are picked by the verifier and sent to the prover in the next round of interaction (this round is used simultaneously for zero-knowledge masking, cf. Appendix D.1). For better efficiency, the exact implementation uses repeated squaring/quadrupling of the coefficients $\{c_i\}_{i=0}^7$, rather than the polynomial composition $C_1(C_2(C_3(x)))$.

The ShiftRows operation is implemented together with SubBytes, by simply placing the results of the SubBytes S-box in the appropriate registers for the next cycle (cf. Appendix E.3). The MixColumns operations is implemented in a single cycle, using the linear combination that we described above to perform field additions and multiplications by the constant $g_0$. The AddRoundKey operation is done at the same cycle that we compute MixColumns, using the aforementioned efficient SubBytes S-box implementation.

### E.3 State machine of the Rijndael cipher

The algebraic execution trace (cf. Section 2) of the Rijndael-160 hash function is shown in the following table. We enforce boundary constraints on the first and last rows (i.e., cycle 0 and cycle T).

<table>
<thead>
<tr>
<th>P00 ··· P34(0)</th>
<th>K00 ··· K34(0)</th>
<th>INV1 ··· INV5(0)</th>
<th>W11 ··· W53(0)</th>
<th>F1(0)</th>
<th>F2(0)</th>
<th>RC(0)</th>
<th>INVRC(0)</th>
<th>STATE(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P00 ··· P34(t)</td>
<td>K00 ··· K34(t)</td>
<td>INV1 ··· INV5(t)</td>
<td>W11 ··· W53(t)</td>
<td>F1(t)</td>
<td>F2(t)</td>
<td>RC(t)</td>
<td>INVRC(t)</td>
<td>STATE(t)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P00 ··· P34(T)</td>
<td>K00 ··· K34(T)</td>
<td>INV1 ··· INV5(T)</td>
<td>W11 ··· W53(T)</td>
<td>F1(T)</td>
<td>F2(T)</td>
<td>RC(T)</td>
<td>INVRC(T)</td>
<td>STATE(T)</td>
</tr>
</tbody>
</table>
\((\text{INV}_1(t)P_{00}(t) + 1)(P_{00}(t) \land \text{INV}_1(t)) \land \\
(\text{W}_{11}(t) + \text{INV}_1^4(t)) \land (\text{W}_{12}(t) + \text{W}_{11}^4(t)) \land (\text{W}_{13}(t) + \text{W}_{12}^4(t)) \land \\
(P_{00}(t+1) + c_0 \cdot \text{INV}_1(t) + c_1 \cdot \text{INV}_1^2(t) + c_2 \cdot \text{W}_{11}(t) + c_3 \cdot \text{W}_{11}^2(t) \\
+ c_4 \cdot \text{W}_{12}(t) + c_5 \cdot \text{W}_{12}^2(t) + c_6 \cdot \text{W}_{13}(t) + c_7 \cdot \text{W}_{13}^2(t) + b)\)

Figure 9: Constraints polynomial for the Rijndael-160 SubBytes S-box.

Each of these 40 registers is a field element of \(\mathbb{F}_{2^{64}}\) that resides in the subfield \(\mathbb{F}' \cong \mathbb{F}_{2^8}\), and hence contains only 8 bits of information. Per Appendix E.2, this is done to support a native inversion operation for the SubBytes S-box. In each cycle, the registers INVI, INV2, INV3, INV4, INV5 are used primarily as the auxiliary field elements that compute the inverses for SubBytes.

For \(i \in \{1, \ldots, 5\}\), the registers \(W_i1, W_i2, W_i3\) store the repeated quadrupling that are used to compute the powers of INVI: \(W_i1 = \text{INV}_i^4, W_i2 = \text{W}_i^4 = \text{INV}_i^{16}, W_i3 = \text{W}_i^2 = \text{W}_i^{16} = \text{INV}_i^{64}\). Our constraints will then also square these registers, for example \(W_i3 \cdot W_i3 = \text{INV}_i^{128}\).

The registers \(F_1, F_2\) are inner flags that specify the current step in the Rijndael loop. Every round of Rijndael takes 4 steps, and our algebraic constraints use the values of \(F_1, F_2\) to enforce the requirements of the current step.

The register \(RC\) is used to compute \(\text{Rcon}(i)\) in round \(i\) of the Rijndael loop. The register \(INVRC\) is used for the inverse of \(RC\), in order to tell when to stop the Rijndael iterations. The register \(STATE\) is an external flag that specifies whether we compute the Rijndael cipher or some additional logic (i.e., \(STATE\) would be unnecessary for a single invocation of Rijndael-160).

We provide an excerpt of the algebraic constraints of a single SubBytes S-box in Figure 9.

Overall, the width of the computation is 65, per the above description. The Rijndael-160 cipher requires 11 rounds where each round consists of SubBytes, ShiftRows, MixColumns, and AddRoundKey (except for the last round that lacks MixColumns). Each round takes 5 cycles in our implementation, hence an entire invocation of Rijndael-160 takes 55 cycles. The prover needs to compute a total of \(55 \cdot 65 = 3575\) field elements for a single invocation of Rijndael-160.

### E.4 From encryption to hash function: Davies-Meyer

The Rijndael-160 block cipher can be converted into a hash function by using the Davies-Meyer transformation: \(\text{hash}(B, K) = E_K(B) \oplus B\), where \(E\) is the Rijndael-160 cipher in our case. The resulting \(\text{hash}(B, K)\) is collision-resistant under the assumption that for any key \(K\) the function \(E_K(\cdot)\) is an independent random
permutation, see [73, Theorem 6.5]. Let us also note that constructions with additional overhead can give a CRHF under milder assumptions (e.g., that $E_K(\cdot)$ is a pseudo-random permutation), see for example [32, Table 3].

To implement the Davies-Meyer construction, the execution trace (cf. Section 2.2) requires saving the 160 bits of $B$ while computing the output of the Rijndael-160 cipher. As discussed, our implementation expands $B$ into 20 registers that contain elements of $\mathbb{F}_{2^{64}}$. Since each of these 20 registers holds only 8 bits of information, we can compress and save $B$ in 3 registers that hold 64 bits of information.

The compression method is quite simple: by treating $\mathbb{F}_{2^{64}}$ as an extension field of $F' \cong \mathbb{F}_{2^8}$ of degree 8, we set a basis of the extension field and encode 8 registers of $B$ into an element of $\mathbb{F}_{2^{64}}$ by using them as the coefficients of the basis elements. It can in fact be proved that $(1, g_0, g_0^2, \ldots, g_0^7)$ is such a basis. Thus, the encoding is done as $B_0 = \sum_{k=0}^{7} p_k \cdot g_0^k$, where the values $p_k$ are taken from the registers $P_{ij}$. We encode $B_2, B_3$ in the same manner, except that for $B_3$ the last 4 coefficients are set to 0.

In order to feed the output digest into the next Rijndael-160 invocation (see Appendix E.5), we require the output to be in the “uncompressed” form that spans 20 registers. Thus, for $i \in \{0, 1, 2\}$, we decompress $B_i$ after the Rijndael-160 cipher computation, and then add the uncompressed values to the cipher’s output. This is done by letting the prover supply 20 field element $\{p_k\}_{k=0}^{19}$ nondeterministically (with $p_k = 0$ for $k \in \{20, 21, 22, 23\}$), and enforcing the algebraic constraint $B_i + \sum_{k=0}^{7} p_{k+8i} \cdot g_0^k = 0$ for $i \in \{0, 1, 2\}$. However, this linear combination is unique only if the coefficients $p_k \in F'$ for $k \in \{0, \ldots, 19\}$. This is achieved using Fermat’s little theorem, which states that for any finite field $F$, $\forall x \in F : x^{256} = x$. In our case the constraints are $\cup \{p_k^{256} + p_k = 0\}_{k=0}^{19}$, and we again reduce their total degree to 8 by using repeated quadrupling.

The state machine thus requires 3 additional registers for every cycle:

<table>
<thead>
<tr>
<th>$B_0(t)$</th>
<th>$B_1(t)$</th>
<th>$B_2(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_0(t+1)$</td>
<td>$B_1(t+1)$</td>
<td>$B_2(t+1)$</td>
</tr>
<tr>
<td>$B_0(t)$</td>
<td>$B_1(t)$</td>
<td>$B_2(t)$</td>
</tr>
</tbody>
</table>

We show the crux of the algebraic constraints for compression and decompression in Figure 10.

**Figure 10: Constraint polynomials for compression and decompression**
The compression requires one cycle before each computation of the Rijndael-160 cipher, and the decompression requires two cycles afterwards. Thus, the number of cycles for the Rijndael-160 hash function is 58, the width is 68, and the total number of field elements that the prover needs to compute is $58 \cdot 68 = 3944$.

E.5 DNA profile match (DPM)

The high-level pseudo-code of the DNA profile match is given as Program 1. The database records are assumed to reside in a hashchain of $N$ elements, as illustrated in Figure 11. More precisely, the hashchain is computed using the Merkle-Damgard construction with Rijndael-160 as the compression function. The hash of the last element commits to the entire database, and is verified as a boundary constraint (line XV). Notice that we harness the power of nondeterminism to supply the values of the chain elements during the exhaustive search, which implies that for an arbitrary $N$ the program can operate with only a small constant number of auxiliary variables. Due to the collision resistance of the underlying compression function and the Merkle-Damgard construction [83, 45], this use of nondeterminism is secure.

An element of the chain is a DNA profile according to the Combined DNA Index System (CODIS) format; it is comprised of Short Tandem Repeat (STR) counts for 20 “core loci”; we use an 8 bit integer to record a single STR value, and we encode the integer by a single element of $F'$. Since a single DNA profile requires 20 pairs of STR values (2 per loci), each record (profile) is stored in two consecutive elements of the hashchain. Thus, a database $D(n)$ of $n$ profiles requires $N = 2n$ chain elements, and Program 1 consists of logic that alternates between odd and even elements.

Program 1 DNA profile match

<table>
<thead>
<tr>
<th>Explicit inputs: $n$, $cm_l$, $cm_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nondeterministic inputs: ${VAL_{i,j}}<em>{i \in {1,2}, j \in {1,2,\ldots,20}}, {W_i}</em>{i \in {0,1,2,\ldots,N}}$</td>
</tr>
<tr>
<td>I: if $cm_p \neq \text{hash160}({VAL_{1,j}}<em>{j \in {1,2,\ldots,20}}, {VAL</em>{2,j}}_{j \in {1,2,\ldots,20}})$ then return false end if</td>
</tr>
<tr>
<td>II: $k \leftarrow 1$, flag $\leftarrow 0$, $h \leftarrow 0$, $T \leftarrow 0$, $N \leftarrow 2n$</td>
</tr>
<tr>
<td>III: while $k \neq g^N$ do</td>
</tr>
<tr>
<td>IV: Parse $(L_1, R_1, L_2, R_2, \ldots, L_{10}, R_{10}) = W_j$, $j = \log g_k$</td>
</tr>
<tr>
<td>V: if flag = 0 then</td>
</tr>
<tr>
<td>VI: $T_1 \leftarrow \text{CheckPairs}(VAL_{1,1}, VAL_{1,2}, L_1, R_1, VAL_{2,1}, VAL_{2,2}, L_2, R_2, \ldots, VAL_{10,1}, VAL_{10,2}, L_{10}, R_{10})$</td>
</tr>
<tr>
<td>VII: else</td>
</tr>
<tr>
<td>VIII: $T_2 \leftarrow \text{CheckPairs}(VAL_{11,1}, VAL_{11,2}, L_1, R_1, VAL_{12,1}, VAL_{12,2}, L_2, R_2, \ldots, VAL_{20,1}, VAL_{20,2}, L_{10}, R_{10})$</td>
</tr>
<tr>
<td>IX: $T \leftarrow \text{MatchingResult}(T_1, T_2, T)$</td>
</tr>
<tr>
<td>X: end if</td>
</tr>
<tr>
<td>XI: $h \leftarrow \text{hash160}(h, W_j)$</td>
</tr>
<tr>
<td>XII: $k \leftarrow g \cdot k$</td>
</tr>
<tr>
<td>XIII: flag $\leftarrow 1 - \text{flag}$</td>
</tr>
<tr>
<td>XIV: end while</td>
</tr>
<tr>
<td>XV: if $cm_l \neq h$ then return false else return $T$ end if</td>
</tr>
</tbody>
</table>

To validate that the prover does not skip over some prefix of the chain in the exhaustive search, the total number of hash invocations $N$ is also checked by the verifier as a boundary constraint. We also note that Program 1 increments its counter via a field multiplication with the generator $g$ of $F'^n_{254}$, thereby avoiding integer arithmetics.

The register $T$ stores the output, and is verified in the last cycle via a boundary constraint. The output

76
can be either 1) “perfect match”, meaning that an exact match between the input (i.e., the commitment \(cm_p\) that the prover decommits in ZK into 20 STR pairs) and a profile in the hashchain was found, or 2) “partial match”, meaning that the exhaustive search found a profile in the hashchain such that \(\forall j \in \{1, \ldots, 20\}\) at least one STR value of its \(j\)th pair matches a value of the \(j\)th STR pair of the input, or 3) “no match”.

We provide high-level pseudocode of the perfect/partial match logic in Figures 12 and 13, and the corresponding algebraic constraints in Figures 14 and 15. The \(\text{InnerMatch}(t)\) and \(\text{Match}(t)\) registers of Figure 15 correspond to \(T_1\) and \(T_2\) in Figure 13, and \(\text{Match}(t+1)\) corresponds to \(T\). In Figure 14, the registers \(\text{PAIR1}(t)\) and \(\text{PAIR2}(t)\) are compared to the \(i\)th loci pair of the input. Each such comparison relies on 5 auxiliary registers: \(\text{INVTMP1}(t), \ldots, \text{INVTMP4}(t)\) to compute field inverses, and \(\text{NewFlag}(t)\) that represents the current inner matching (the register \(\text{LastFlag}(t)\) is derived from the previous iteration).

Since the prover (but not the verifier) knows the decommitment of \(cm_p\), the first invocation of Rijndael-160 in Program 1 is executed with completely nondeterministic inputs (that occupy 40 registers), and the output is constrained to be equal to \(cm_p\). Due to the ZK guarantee of our proof system, this implies semantic security (by contrast, comparing the database records to 40 explicit constants that commit to STR strings is not semantically secure because the same value may appear more than once, especially if Program 1 is executed multiple times with the same committed database \(cm_t\)). In each of the next invocations, those nondeterministic values should be compared against the current database record (i.e., 20 registers that are supplied nondeterministically by the prover). However, keeping the initial 40 values throughout the entire execution of Program 1 is inefficient. Since each of those 40 registers contains only 8 bits of information, we use the same technique as in Appendix E.4 to compress this data into 5 auxiliary registers.

After each invocation of Rijndael-160, we decompress these 5 registers in order to execute the comparison logic (cf. Figures 12 to 15) for STR pairs with the newly supplied database record. Here too we must verify the decompressed registers by using Fermat’s little theorem, and this requires two cycles as there are not enough temporary registers to compute the repeated quadrupling in a single cycle.

Program 1 also requires an extra register for the counter \(k = g^j\), two registers for handling the perfect/partial match constraints, three more registers that handle the possible states, and two additional auxiliary registers. Overall, the width of the witness is 81, the number of cycles is 62, and the total number of field elements that the prover computes is \(81 \cdot 62 \cdot N = 5022 \cdot N\).
CheckPairs(VAL1,1, VAL1,2, L1, R1, VAL2,1, VAL2,2, L2, R2, ... , VAL10,1, VAL10,2, L10, R10)

I:  \( t \leftarrow 2 \)

II:  \textbf{for}  \( j \in \{1, \ldots, 10\} \) \textbf{do}
III:  \textbf{if}  \((\text{VAL}_{j,1} = L_j)\) AND \((\text{VAL}_{j,2} = R_j)\) \textbf{then} \text{Continue} \textbf{end if}
IV:  \textbf{if}  \((\text{VAL}_{j,1} = R_j)\) AND \((\text{VAL}_{j,2} = L_j)\) \textbf{then} \text{Continue} \textbf{end if}
V:  \textbf{if}  \((\text{VAL}_{j,1} = L_j)\) OR \((\text{VAL}_{j,1} = R_j)\) OR \((\text{VAL}_{j,2} = L_j)\) OR \((\text{VAL}_{j,2} = R_j)\) \textbf{then}
VI:  \( t \leftarrow 1 \)
VII:  \textbf{else}
VIII:  \textbf{return} 0
IX:  \textbf{end if}
X:  \textbf{end for}
XI:  \textbf{return}  \( t \)

Figure 12: CheckPairs subroutine of Program 1.

MatchingResult(\( T_1, T_2, T \))

I:  \textbf{if}  \( T_1 = T_2 = 2 \) \textbf{then} \text{return} 2 \textbf{end if}
II:  \textbf{if}  \((T_1 = 0)\) OR \((T_2 = 0)\) \textbf{then} \text{return} T \textbf{end if}
III:  \textbf{if}  \( T = 0 \) \textbf{then} \text{return} 1 \textbf{end if}

Figure 13: MatchingResult subroutine of Program 1.
(PAIR1(t) + values_{i,0}) \cdot (PAIR1(t) + values_{i,1}) \cdot [ \\
(PAIR2(t) + values_{i,0}) \cdot (PAIR2(t) + values_{i,1}) \cdot \text{NewFlag}(t) \land \\
\text{LastFlag}(t) \cdot (\text{NewFlag}(t) + 1) \cdot [ \\
((PAIR2(t) + values_{i,0}) \cdot \text{INVTMP1}(t) + 1) \land \\
((PAIR2(t) + values_{i,1}) \cdot \text{INVTMP2}(t) + 1)] \land \\
(PAIR2(t) + values_{i,0}) \cdot (PAIR2(t) + values_{i,1}) \cdot [ \\
\text{LastFlag}(t) \cdot (\text{NewFlag}(t) + 1) \cdot [ \\
((PAIR1(t) + values_{i,0}) \cdot \text{INVTMP3}(t) + 1) \land \\
((PAIR1(t) + values_{i,1}) \cdot \text{INVTMP4}(t) + 1)] \land \\
(\text{NewFlag}(t) + 1) \cdot (\text{LastFlag}(t) + X) \cdot \text{NewFlag}(t)
}

Figure 14: Algebraic constraints for one pair in the CheckPairs subroutine.

InnerMatch(t) \cdot (InnerMatch(t) + 1) \cdot (Match(t+1) + X) \land \\
Match(t) \cdot (Match(t) + 1) \cdot (Match(t+1) + X) \land \\
InnerMatch(t) \cdot (InnerMatch(t) + 1) \cdot (Match(t) + X) \cdot (Match(t+1) + 1) \land \\
(InnerMatch(t) + 1) \cdot (InnerMatch(t) + X) \cdot (Match(t) + 1) \cdot (Match(t) + X) \cdot Match(t+1)

Figure 15: Algebraic constraints for the MatchingResult subroutine.
The popular Secure Hash Algorithm 2 (SHA2) family [92] requires modular addition and cyclic shifts which are not particularly “binary field friendly”. Nevertheless, we constructed a rather efficient AIR for it (first row of Table 4), using field-specific constraints. A notable example is a constraint system that “extracts” the $i$th bit from $\alpha \in \mathbb{F}_{2^t}$ for any $i < t$; this system uses only a pair of constraints of degree 2 (notice the number of constraints and their degree is independent of $t$); we believe this bit-extraction constraint set will be useful for other computations.

We provide here notation and several basic lemmas, which facilitate the bit extraction technique that our efficient SHA-256 hash function implementation is based on.

The trace of an element $y \in \mathbb{F}_{2^m}$ is defined as $\text{Tr}_{m|2}(y) \triangleq \sum_{i=0}^{m-1} y^{2^i}$.

**Proposition 1.** For any $\mathbb{F}_2$-linear function $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2$, there exists a field element $\alpha_f \in \mathbb{F}_{2^m}$ such that $\forall y \in \mathbb{F}_{2^m} : f(y) = \text{Tr}_{m|2}(\alpha_f \cdot y)$.

**Proposition 2.** For every $c \in \mathbb{F}_{2^m}$, the equation $y^2 + y = c$ has solutions in $\mathbb{F}_{2^m}$ if and only if $\text{Tr}_{m|2}(c) = 0$. Notice that if $y_0$ is a solution of $y(y + 1) = c$ then $y_0 + 1$ is the other solution, since the field characteristic is 2.

**Definition 1.** Let $\text{isZero}(\alpha, w, v)$ be the polynomial $\text{isZero}(\alpha, w, v) \triangleq w^2 + w + \alpha \cdot v$.

**Lemma 1.** The $i$th coefficient in the standard basis representation of $y \in \mathbb{F}_{2^m}$ is 0 if and only if there exists $w \in \mathbb{F}_{2^m}$ such that $0 = \text{isZero}(\alpha_i, w, y)$, where $\alpha_i \in \mathbb{F}_{2^m}$ is some field element that depends only on $i$.

**Proof.** The function $f_i : \mathbb{F}_2^m \rightarrow \mathbb{F}_2$, $f_i(y) = \begin{cases} 0, & i\text{th coefficient of } y \text{ is } 0 \\ 1, & i\text{th coefficient of } y \text{ is } 1 \end{cases}$ is $\mathbb{F}_2$-linear. It follows from Proposition 1 and Proposition 2 that the required $\alpha_i$ exists. Furthermore, it is straightforward to pre-compute the constant $\alpha_i$ by solving linear equations for the trace of basis elements.

**G** Affine rearrangeable networks

**G.1** Combinatorial representation of back-to-back De Bruijn routing

The following classical results regarding De Bruijn networks [47] can be found, e.g., in [77]. We use Theorem G.3 below in the statement and proof of Theorem B.13. Therefore, we provide a self-contained account of the needed results for the sake of completeness.

**Definition G.1** (De Bruijn butterfly network). The De Bruijn butterfly network of dimension $n$ is a directed graph $G = (V, E)$ over the vertices $V \triangleq \{0, \ldots, n\} \times \{0, 1\}^n$ and edges

$$E \triangleq \{[(i, w), (i + 1, \text{csr}(w))], [(i, w), (i + 1, \text{csr}(w) \oplus 1)] \mid 0 \leq i < n, w \in \{0, 1\}^n\}$$

where $\text{csr} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is the cyclic shift-right operation, and $[x \oplus 1] : \{0, 1\}^n \rightarrow \{0, 1\}^n$ flips the leftmost bit in $x$, and does not change any other bit. Let $C$ be a set of colors. We say $\chi : V \rightarrow C$ is a
coloring of $G$ if for every vertex $v = (i, w)$ with $i < n$, there is a successor $u$ of $v$ (i.e. $vu \in E$) such that $\chi(v) = \chi(u)$.

**Definition G.2** (back-to-back De Bruijn routing). A back-to-back De Bruijn routing of dimension $n$ is a pair of coloring functions $\bar{\chi} = (\chi_0, \chi_1)$ of the De Bruijn butterfly network of dimension $n$, such that for all $w \in \{0, 1\}^n$ (i) $\chi_0(n, w) = \chi_1(n, w)$, and (ii) for any $w' \neq w$, $\chi_0(0, w') \neq \chi_0(0, w')$. In case $|\mathcal{C}| = 2^n$ we define the function induced by $\bar{\chi}$, $\pi_\bar{\chi} : \mathcal{C} \rightarrow \mathcal{C}$ to be $\pi_\bar{\chi}(\chi_0(0, w')) = \chi_1(0, w')$.

**Theorem G.3** (back-to-back De Bruijn is rearrangeable). For every set of colors $\mathcal{C}$ of size $2^n$, the set of functions induced by back-to-back De Bruijn routings of dimension $n$ and colors from $\mathcal{C}$ is exactly the set of permutations over $\mathcal{C}$. Moreover, given a permutation $\pi : \mathcal{C} \rightarrow \mathcal{C}$, there is an algorithm constructing a back-to-back De Bruijn routing $\bar{\chi}$ inducing $\pi$ with time complexity $O(n \cdot 2^n)$.

**Proof.** By Definition G.2, the function induced by a back-to-back routing must be a permutation, thus it is sufficient to show an algorithm producing a back-to-back De Bruijn routing given a permutation $\pi$. The algorithm is recursive and based on the Beneš network routing algorithm [28], and provided in Algorithm 3. The algorithm as provided in Algorithm 3 does not reach the stated complexity, because it contains many move operations, which can be easily eliminated using simple index translation. \hfill \Box

**Procedure 2** Auxiliary method for routing

**Inputs:** two vectors $u, v$ of length $2^n$ indexed by $\{0, 1\}^n$ with distinct entries, where $u$ is a permutation of $v$

**Output:** vectors $v^0, v^1, u^0, u^1$ each of length $2^{n-1}$ indexed by $\{0, 1\}^{n-1}$ with (i) each element of $v$ is in $v^0$ or $v^1$, (ii) for every $w \in \{0, 1\}^{n-1}$ and $i, b \in \{0, 1\}$, $v_{bw} = v^i_x$ or $u_{bw} = u^i_x$ only if $x = w$, and (iii) $u^i$ is a permutation of $v^i$.

I: function SPLIT($n, u, v$)
II: initialize $v^0, v^1, u^0, u^1$ as partial mappings
III: while $v^0, v^1, u^0, u^1$ not yet well defined do
IV: if $\exists w \in \{0, 1\}^{n-1}, b, i \in \{0, 1\}$ such that $v_{bw} \in u^i$ but $v_{bw} \notin v^i$ then
V: $v^i_{bw} \leftarrow v_{bw}$
VI: $u^i_{bw} \leftarrow u_{bw}$
VII: continue
VIII: else if $\exists w \in \{0, 1\}^{n-1}, b, i \in \{0, 1\}$ such that $u_{bw} \in v^i$ but $u_{bw} \notin u^i$ then
IX: $u^i_{bw} \leftarrow u_{bw}$
X: $v^i_{bw} \leftarrow v_{bw}$
XI: else
XII: pick arbitrary $w \in \{0, 1\}^{n-1}$ such that $v^i_{0w} \notin \{v^0_{iw}, v^1_{iw}\}$
XIII: $v^i_{0w} \leftarrow v^i_{iw}$ for $i \in \{0, 1\}$
XIV: end if
XV: end while
XVI: return $v^0, v^1, u^0, u^1$
XVII: end function

**G.2** Affine embedding of back-to-back De Bruijn routing

**Definition G.4** (Alegbraic De Bruijn). Let $\mathbb{F}$ be a field of characteristic 2 with primitive element $g$, and fix some integer $t$. Let $\lceil \log (t + 1) \rceil$ be a primitive binary polynomial of degree $\lceil \log (t + 1) \rceil$. Define $\text{H} \triangleq \ldots$
Program 3 Routing permutation over back-to-back De Bruijn

Inputs: two vectors $u, v$ of length $2^n$ indexed by $\{0, 1\}^n$ and distinct entries, where $u$ is a permutation of $v$

Output: back-to-back De Bruijn routing $\bar{\chi}$, represented as a pair of $2^n \times (n + 1)$ tables, with induced function satisfying $\pi_{\bar{\chi}}(w) = u_w$ for all $w$

I: function ROUTE_DE_BRUJIN($n, u, v$)
II: $\triangleright$ Recursion stopping condition
III: if $n = 1$ then
IV: if $v_0 = u_0$ then
V: $\bar{\chi} = \chi = \begin{bmatrix} v_0 & v_0 \\ v_1 & v_1 \end{bmatrix}$
VI: else
VII: $\bar{\chi} = \begin{bmatrix} v_0 & v_0 \\ v_1 & v_1 \end{bmatrix}, \chi = \begin{bmatrix} v_1 & v_0 \\ v_0 & v_1 \end{bmatrix}$
VIII: end if
IX: end if
X: $\triangleright$ General case
XI: $(v_0, v_1, u_0, u_1) \leftarrow$ SPLIT($n, u, v$)
XII: $\bar{\chi}^0 \leftarrow$ ROUTE_DE_BRUJIN($n - 1, u_0, v_0$)
XIII: $\bar{\chi}^1 \leftarrow$ ROUTE_DE_BRUJIN($n - 1, u_1, v_1$)
XIV: for all $w \in \{0, 1\}^n$ do
XV: $\chi_0(0, w) \leftarrow v_w$
XVI: $\chi_1(0, w) \leftarrow u_w$
XVII: end for
XVIII: for all $(i, w) \in [n] \times \{0, 1\}^{n-1}$ do
XIX: represent $w$ by $w = x \cdot y$ where $|y| = i - 1$
XX: $\chi_0(i, x \cdot 0 \cdot y) \leftarrow \chi_0^0(i - 1, w)$
XXI: $\chi_0(i, x \cdot 1 \cdot y) \leftarrow \chi_0^0(i - 1, w)$
XXII: $\chi_1(i, x \cdot 0 \cdot y) \leftarrow \chi_1^0(i - 1, w)$
XXIII: $\chi_1(i, x \cdot 1 \cdot y) \leftarrow \chi_1^0(i - 1, w)$
XXIV: end for
XXV: end function
\[ \text{span} \{ g_i | 0 \leq i < t \}, \quad \text{and} \quad W \overset{\Delta}{=} \text{span} \{ g_{t+i} | 0 \leq i < \lceil \log (t+1) \rceil \}. \]

A pair of mappings \( \chi_0, \chi_1 : H \oplus W \to \mathbb{F} \) is an affine embedding of back-to-back De Bruijn of degree \( t \) if:

- for every \( x \neq x' \in H \), \( \chi_0(x + g_{t+1}) \neq \chi_0(x' + g_{t+1}) \)
- for every \( x \in H \), \( \chi_0(x + g^t) = \chi_1(x + g^t) \)
- for every \( b \in \{0,1\} \), \( x = \sum x_i g^i \in \text{span} \{ g^i | 0 \leq i < t \} \) and every non zero \( y = \sum y_i g^i \in \text{span} \{ g^i | 0 \leq i < \lceil \log (t+1) \rceil \} \), denoting by \( z = x + g^t y \), if \( (x_{t-1}, y_{\lceil \log (t+1) \rceil-1}) = (r, c) \) for \( r, c \in \{0,1\} \) then \( \chi_b(z) \) equals to one of:
  - \( \chi_b(g \cdot z + r \cdot (g^t + 1) + c \cdot (g^t \cdot \xi)) \)
  - \( \chi_b(g \cdot z + r \cdot (g^t + 1) + c \cdot (g^t \cdot \xi) + 1) \)

Let \( C \) be the image of \( \chi \) over \( H \). The function induced by \( \chi_0, \chi_1 \) is a mapping \( \pi : C \to \mathbb{F} \) mapping defined by \( \chi_0(x + g^{t+1}) \mapsto \chi_1(x + g^{t+1}) \) for every \( x \in H \).

**Theorem G.5 (Affine permutation).** The set of all mappings induced be affine embeddings of back-to-back De Bruijn of degree \( t \) is exactly the set of all permutations over \( 2^t \) elements of \( \mathbb{F} \).
Aurora: Transparent Succinct Arguments for R1CS

Eli Ben-Sasson  
eli@cs.technion.ac.il  
Technion

Alessandro Chiesa  
alexch@berkeley.edu  
UC Berkeley

Michael Riabzev  
mriabzev@cs.technion.ac.il  
Technion

Nicholas Spooner  
nick.spooner@berkeley.edu  
UC Berkeley

Madars Virza  
madars@mit.edu  
MIT Media Lab

Nicholas P. Ward  
npward@berkeley.edu  
UC Berkeley

September 4, 2018

Abstract

We design, implement, and evaluate a zero knowledge succinct non-interactive argument (SNARG) for Rank-1 Constraint Satisfaction (R1CS), a widely-deployed NP language undergoing standardization. Our SNARG has a transparent setup, is plausibly post-quantum secure, and uses lightweight cryptography. A proof attesting to the satisfiability of \( n \) constraints has size \( O(\log^2 n) \); it can be produced with \( O(n \log n) \) field operations and verified with \( O(n) \). At 128 bits of security, proofs are less than 250 kB even for several million constraints, more than 10× shorter than prior SNARGs with similar features.

A key ingredient of our construction is a new Interactive Oracle Proof (IOP) for solving a univariate analogue of the classical sumcheck problem [LFKN92], originally studied for multivariate polynomials. Our protocol verifies the sum of entries of a Reed–Solomon codeword over any subgroup of a field.

We also provide libiop, a library for writing IOP-based arguments, in which a toolchain of transformations enables programmers to write new arguments by writing simple IOP sub-components. We have used this library to specify our construction and prior ones, and plan to open-source it.

Keywords: zero knowledge; interactive oracle proofs; succinct arguments; sumcheck protocol
Contents

1 Introduction 3
  1.1 The need for a transparent setup .................................................. 3
  1.2 Our goal ................................................................................. 4
  1.3 Our contributions ..................................................................... 5
  1.4 Prior implementations of transparent SNARGs ............................... 6

2 Techniques 9
  2.1 Our interactive oracle proof for R1CS ............................................. 9
  2.2 A sumcheck protocol for univariate polynomials ............................... 10
  2.3 Efficient zero knowledge from algebraic techniques ........................... 11
  2.4 Perspective on our techniques ..................................................... 12

3 Roadmap 13

4 Definitions 14
  4.1 Codes .................................................................................. 14
  4.2 Representations of polynomials ..................................................... 14
  4.3 The fast Fourier transform .......................................................... 14
  4.4 Subspace polynomials ............................................................... 14
  4.5 Interactive oracle proofs ............................................................ 15
  4.6 Zero knowledge ...................................................................... 16
  4.7 Reed–Solomon encoded IOP ....................................................... 17

5 Univariate sumcheck 19
  5.1 Zero knowledge ...................................................................... 21
  5.2 Amortization .......................................................................... 22

6 Univariate lincheck 24

7 Univariate rowcheck 26

8 An RS-encoded IOP for rank-one constraint satisfaction 28
  8.1 Zero knowledge ...................................................................... 30
  8.2 Amortization .......................................................................... 32

9 From RS-encoded provers to arbitrary provers 33
  9.1 Zero knowledge ...................................................................... 36

10 Aurora: an IOP for rank-one constraint satisfaction (R1CS) 38

11 libiop: a library for IOP-based non-interactive arguments 41
  11.1 Library for IOP protocols .......................................................... 41
  11.2 BCS transformation ............................................................... 41
  11.3 Portfolio of IOP protocols and sub-components ............................. 42

12 Evaluation 43
  12.1 Performance of Aurora ............................................................. 43
  12.2 Comparison of Ligero, Stark, and Aurora ..................................... 43

A Proof of Lemma 5.4 46

B Adaptation of Ligero to the R1CS relation 47
  B.1 Interleaved lincheck .................................................................. 47
  B.2 The fasted rowcheck .................................................................. 48
  B.3 Interleaved ZKIPCP for R1CS ..................................................... 49
  B.4 From encoded IPCP to regular IPCP ........................................... 53

C Additional comparisons 55
  C.1 Comparison of the LDTs in Ligero, Stark, and Aurora .................. 55
  C.2 Comparison of the IOPs in Ligero, Stark, and Aurora .................. 55

Acknowledgments 57

References 57
1 Introduction

A zero knowledge proof is a protocol that enables one party (the prover) to convince another (the verifier) that a statement is true without revealing any information beyond the fact that the statement is true. Since their introduction [GMR89], zero knowledge proofs have become fundamental tools not only in the theory of cryptography but also, more recently, in the design of real-world systems with strong privacy properties.

For example, zero knowledge proofs are the core technology in Zcash [BCGGMTV14; Zca], a popular cryptocurrency that preserves a user’s payment privacy. While in Bitcoin [Nak09] users broadcast their private payment details in the clear on the public blockchain (so other participants can check the validity of the payment), users in Zcash broadcast encrypted transaction details and prove, in zero knowledge, the validity of the payments without disclosing what the payments are.

Many applications, including the aforementioned, require that proofs are succinct, namely, that proofs scale sublinearly in the size of the witness for the statement, or perhaps even in the size of the computation performed to check the statement. This strong efficiency requirement cannot be achieved with statistical soundness (under standard complexity assumptions) [GH98], and thus one must consider proof systems that are merely computationally sound, known as argument systems [BCC88]. Many applications further require that a proof consists of a single non-interactive message that can be verified by anyone; such proofs are cheap to communicate and can be stored for later use (e.g., on a public ledger). Constructions that satisfy these properties are known as (publicly verifiable) succinct non-interactive arguments (SNARGs) [GW11].

In this work we present Aurora, a zero knowledge SNARG for (an extension of) arithmetic circuit satisfiability whose proof size is polylogarithmic in the circuit size. Aurora also has attractive features: it uses a transparent setup, is plausibly post-quantum secure, and only makes black-box use of fast symmetric cryptography (any cryptographic hash function modeled as a random oracle).

Our work makes an exponential asymptotic improvement in proof size over Ligero [AHIV17], a recent zero knowledge SNARG with similar features but where proofs scale as the square root of the circuit size. For example, Aurora’s proofs are 20× smaller than Ligero’s for circuits with a million gates (which already suffices for representative applications such as Zcash).

Our work also complements and improves on Stark [BBHR18a], a recent zero knowledge SNARG that targets computations expressed as bounded halting problems on random access machines. While Stark was designed for a different computation model, we can still study its efficiency when applied to arithmetic circuits. In this case Aurora’s prover is faster by a logarithmic factor (in the circuit size) and Aurora’s proofs are concretely much shorter, e.g., 15× smaller for circuits with a million gates.

The efficiency features of Aurora stem from a new Interactive Oracle Proof (IOP) that solves a univariate analogue of the celebrated sumcheck problem [LFKN92], in which query complexity is logarithmic in the degree of the polynomial being summed. This is an exponential improvement over the original multi-variate protocol, where communication complexity is (at least) linear in the degree of the polynomial. We believe this protocol and its analysis are of independent interest.

1.1 The need for a transparent setup

The first succinct argument is due to Kilian [Kil92], who showed how to use collision-resistant hashing to compile any Probabilistically Checkable Proof (PCP) [BFLS91; FGLSS96; AS98; ALMSS98] into a corresponding interactive argument. Micali then showed how a similar construction, in the random oracle model, yields succinct non-interactive arguments (SNARGs). Subsequent work [IMSX15] noted that if the underlying PCP is zero knowledge then so is the SNARG. Unfortunately, PCPs remain very expensive, and this approach has not led to SNARGs with good concrete efficiency.
In light of this, a different approach was initially used to achieve SNARG implementations with good concrete efficiency [PGHR13; BCGTV13]. This approach, pioneered in [Gro10; GGPR13; Lip13; BCIOP13], relied on combining certain linearly homomorphic encodings with lightweight information-theoretic tools known as linear PCPs [IKO07; BCIOP13; SBVBPW13]; this approach was refined and optimized in several works [BCTV14b; BCTV14a; CFHKKNPZ15; Gro16; BISW17; GM17]. These constructions underlie widely-used open-source libraries [SCI] and deployed systems [Zca], and their main feature is that proofs are very short (a few hundred bytes) and very cheap to verify (a few milliseconds).

Unfortunately, the foregoing approach suffers from a severe limitation, namely, the need for a central party to generate system parameters for the proof system. Essentially, this party must run a probabilistic algorithm, publish its output, and “forget” the secret randomness used to generate it. This party must be trustworthy because knowing these secrets allows forging proofs for false assertions. While this may sound like an inconvenience, it is a colossal challenge to real-world deployments. When using cryptographic proofs in distributed systems, relying on a central party negates the benefits of distributed trust and, even though it is invoked only once in a system’s life, a party trusted by all users typically does not exist!

The responsibility for generating parameters can in principle be shared across multiple parties via techniques that leverage secure multi-party computation [BCGTV15; BGG17; BGM17]. This was the approach taken for the launch of Zcash [The], but it also demonstrated how unwieldy such an approach is, involving a costly and logistically difficult real-world multi-party “ceremony”. Successfully running such a multi-party protocol was a singular feat, and systems without such expensive setup are decidedly preferable.

Some setup is unavoidable because if SNARGs without any setup existed then so would sub-exponential algorithms for SAT [Wee05]. Nevertheless, one could still aim for a “transparent setup”, namely one that consists of public randomness, because in practice it is cheaper to realize. Recent efforts have thus focused on designing SNARGs with transparent setup (see discussion in Section 1.4).

1.2 Our goal

The goal of this paper is to obtain transparent SNARGs that satisfy the following desiderata.

- **Post-quantum security.** Practitioners, and even standards bodies [NIS16], have a strong interest in cryptographic primitives that are plausibly secure against efficient quantum adversaries. This is motivated by the desire to ensure long-term security of deployed systems and protocols.

- **Concrete efficiency.** We seek proof systems that not only exhibit good asymptotics (in proof length and prover/verifier time) but also demonstrably offer good efficiency via a prototype.

The second bullet warrants additional context. Most proof systems support an NP-complete problem, so they are in principle equivalent under polynomial-time reductions. Yet, whether such protocols can be efficiently used in practice actually depends on: (a) the particular NP-complete problem “supported” by the protocol; (b) the concrete efficiency of the protocol relative to this problem. This creates a complex tradeoff.

Simple NP-complete problems, like boolean circuit satisfaction, facilitate simple proof systems; but reducing the statements we wish to prove to boolean circuits is often expensive. On the other hand, one can design proof systems for rich problems (e.g., an abstract computer) for which it is cheap to express the desired statements; but the resulting proof systems might require expensive tools to support these rich problems.

Our goal is concretely-efficient proof systems that support rank-1 constraint satisfaction (R1CS), which is the following natural NP-complete problem: given a vector \( v \in \mathbb{F}^k \) and three matrices \( A, B, C \in \mathbb{F}^{m \times n} \), can one augment \( v \) to \( z \in \mathbb{F}^n \) such that \( Az \circ Bz = Cz \)? (We use “\( \circ \)” to denote the entry-wise product.)

We choose R1CS because it strikes an attractive balance: it generalizes circuits by allowing “native” field arithmetic and having no fan-in/fan-out restrictions, but it is simple enough that one can design efficient proof
systems for it. Moreover, R1CS has demonstrated strong empirical value: it underlies real-world systems [Zca] and there are compilers that reduce program executions to it (see [WB15] and references therein). This has led to efforts to standardize R1CS formats across academia and industry [Zks].

1.3 Our contributions

In this work we study Interactive Oracle Proofs (IOPs) [BCS16; RRR16], a notion of “multi-round PCPs” that has recently received much attention [BCGV16; BCFGRS17; BBCGGHRSTV17; BBHR18b; BBHR18a; BKS18]. These types of interactive proofs can be compiled into non-interactive arguments in the random oracle model [BCS16], and in particular can be used to construct transparent SNARGs. Building on this approach, we present several contributions: (1) an IOP protocol for R1CS with attractive efficiency features; (2) the design, implementation, and evaluation of a transparent SNARG for R1CS, based on our IOP protocol; (3) a generic library for writing IOP-based non-interactive arguments. We now describe each contribution.

(1) IOP for R1CS. We construct a zero knowledge IOP protocol for rank-1 constraint satisfaction (R1CS) with linear proof length and logarithmic query complexity.

Given an R1CS instance \( \mathcal{C} = (A, B, C) \) with \( A, B, C \in \mathbb{F}^{m \times n} \), we denote by \( N = \Omega(m + n) \) the total number of non-zero entries in the three matrices and by \( |\mathcal{C}| \) the number of bits required to represent these; note that \(|\mathcal{C}| = \Theta(N \log |\mathbb{F}|)\). One can view \( N \) as the number of “arithmetic gates” in the R1CS instance.

**Theorem 1.1** (informal). There is an \( O(\log N) \)-round IOP protocol for R1CS with proof length \( O(N) \) over alphabet \( \mathbb{F} \) and query complexity \( O(\log N) \). The prover uses \( O(N \log N) \) field operations, while the verifier uses \( O(N) \) field operations. The IOP protocol is public coin and is a zero knowledge proof of knowledge.

The core of our result is a solution to a univariate analogue of the classical sumcheck problem [LFKN92]. Our protocol (including zero knowledge and soundness error reduction) is relatively simple: it is specified in a single page (see Fig. 5 in Section 10), given a low-degree test as a subroutine. The low degree test that we use is a recent highly-efficient IOP for testing proximity to the Reed–Solomon code [BBHR18b].

(2) SNARG for R1CS. We design, implement, and evaluate Aurora, a zero knowledge SNARG for R1CS with several notable features: (a) it only makes black-box use of fast symmetric cryptography (any cryptographic hash function modeled as a random oracle); (b) it has a transparent setup (users merely need to “agree” on which cryptographic hash function to use); (c) it is plausibly post-quantum secure (there are no known efficient quantum attacks against this construction). These features follow from the fact that Aurora is obtained by applying the transformation of [BCS16] to our IOP for R1CS.

In terms of asymptotics, given an R1CS instance \( \mathcal{C} \) over \( \mathbb{F} \) with \( N \) gates (and here taking for simplicity \( \mathbb{F} \) to have size \( 2^{O(\lambda)} \) where \( \lambda \) is the security parameter), Aurora provides proofs of length \( O(\lambda(\log^2 N)) \); these can be produced in time \( O(\lambda(N \log N)) \) and checked in time \( O(\lambda(N)) \).

For example, when setting our implementation to a security level of 128 bits over a 192-bit finite field, proofs range from 50 kB to 250 kB for instances of up to millions of gates; producing proved takes on the order of several minutes and checking proofs on the order of several seconds. (See Section 12 for details.)

Overall, as indicated in Fig. 2, we achieve the smallest proof size among (plausibly) post-quantum non-interactive arguments for circuits, more than an order of magnitude. Other approaches achieve smaller proof sizes by relying on (public-key) cryptographic assumptions that are vulnerable to quantum adversaries.

(3) **libiop:** a library for non-interactive arguments. We provide libiop, a codebase that enables the design and implementation of non-interactive arguments based on IOPs. The codebase uses the C++ language and has three main components: (1) a library for writing IOP protocols; (2) a realization of [BCS16]’s
transformation, mapping any IOP written with our library to a corresponding non-interactive argument; (3) a portfolio of IOP protocols, including Ligero [AHIV17], Stark [BBHR18a], and ours.

We plan to open-source libiop under a permissive software license for the community, so that others may benefit from its portfolio of IOP-based arguments, and may even write new IOPs tailored to new applications. We believe that our library will serve as a powerful tool in meeting the increasing demand by practitioners for transparent non-interactive arguments.

1.4 Prior implementations of transparent SNARGs

We summarize prior work that has designed and implemented transparent SNARGs; see Fig. 2.1

Based on asymmetric cryptography. Bulletproofs [BCCGP16; BBBPWM17] proves the satisfaction of an $N$-gate arithmetic circuit via a recursive use of a low-communication protocol for inner products, achieving a proof with $O(\log N)$ group elements. Hyrax [WTSTW17] proves the satisfaction of a layered arithmetic circuit of depth $D$ and width $W$ via proofs of $O(D \log W)$ group elements; the construction applies the Cramer–Damgård transformation [CD98] to doubly-efficient Interactive Proofs [GKR15; CMT12]. Both approaches use Pedersen commitments, and so are vulnerable to quantum attacks. Also, in both approaches the verifier performs many expensive cryptographic operations: in the former, the verifier uses $O(N)$ group exponentiations; in the latter, the verifier’s group exponentiations are linear in the circuit’s witness size. (Hyrax allows fewer group exponentiations but with longer proofs; see [WTSTW17].)

Based on symmetric cryptography. The “original” SNARG construction of Micali [Mic00; IMSX15] has advantages beyond transparency. First, it is unconditionally secure given a random oracle, which can be instantiated with extremely fast symmetric cryptography.2 Second, it is plausibly post-quantum secure, in that there are no known efficient quantum attacks. But the construction relies on PCPs, which remain expensive.

IOPs are “multi-round PCPs” that can also be compiled into non-interactive arguments in the random oracle model [BCS16]. This compilation retains the foregoing advantages (transparency, lightweight cryptography, and plausible post-quantum security) and, in addition, facilitates greater efficiency, as IOPs have superior efficiency compared to PCPs [BCGV16; BCGR17; BBCGGHPRSTV17; BBHR18b; BBHR18a].

In this work we follow the above approach, by constructing a SNARG based on a new IOP protocol. Two recent works have also taken the same approach, but with different underlying IOP protocols, which have led to different features. We provide both of these works as part of our library (Section 11), and experimentally compare them with our protocol (Section 12). The discussion below is a qualitative comparison.

- Ligero [AHIV17] is a non-interactive argument that proves the satisfiability of an $N$-gate circuit via proofs of size $O(\sqrt{N})$ that can be verified in $O(N)$ cryptographic operations. As summarized in Fig. 1, the IOP underlying Ligero achieves the same oracle proof length, prover time, and verifier time as our IOP. However, we reduce query complexity from $O(\sqrt{N})$ to $O(\log N)$, which is an exponential improvement, at the expense of increasing round complexity from 2 to $O(\log N)$. The arguments that we obtain are still non-interactive, but our smaller query complexity translates into shorter proofs (see Fig. 2).

- Stark [BBHR18a] is a non-interactive argument for bounded halting problems on a random access machine. Given a program $P$ and a time bound $T$, it proves that $P$ accepts within $T$ steps on a certain abstract

---

1We omit a discussion of prior works without implementations, or that study non-transparent SNARGs; we refer the reader to the survey of Walfish and Blumberg [WB15] for an overview of sublinear proof systems. We also note that recent work [BBCPGL18] has used lattice cryptography to achieve sublinear zero knowledge arguments that are plausibly post-quantum secure, which leaves the exciting question of whether these recent protocols can lead to efficient implementations.

2Some cryptographic hash functions, such as BLAKE2, can process almost 1 gibibyte per second [ANWOW13].
computer (when given suitable nondeterministic advice) via succinct proofs of size \(\text{polylog}(T)\). Moreover, verification is also succinct: checking a proof takes time only \(|P| + \text{polylog}(T)\), which is polynomial in the size of the statement and much better than “naive verification” which takes time \(\Omega(|P| + T)\).

The main difference between Stark and Aurora is the computational models that they support. While Stark supports uniform computations specified by a program and a time bound, Aurora supports non-uniform computations specified by an explicit circuit (or constraint system). Despite this difference, we can compare the cost of Stark and Aurora with respect to the explicit circuit model, since one can reduce a given \(N\)-gate circuit (or \(N\)-constraint system) to a corresponding bounded halting problem with \(|P|, T = \Theta(N)\).

In this case, Stark’s verification time is the same as Aurora’s, \(O(N)\); this is best possible because just reading an \(N\)-gate circuit takes time \(\Omega(N)\). But Stark’s prover is a logarithmic factor more expensive because it uses a switching network to verify a program’s accesses to memory. Stark’s prover uses an IOP with oracles of size \(O(N \log N)\), leading to an arithmetic complexity of \(O(N \log^2 N)\). (See Figs. 1 and 2.)

Both Stark and Aurora have proof size \(O(\log^2 N)\), but additional costs in Stark (e.g., due to switching networks) result in Stark proofs being one order of magnitude larger than Aurora proofs. That said, we view Stark and Aurora as complementing each other: Stark offers savings in verification time for succinctly represented programs, while Aurora offers savings in proof size for explicitly represented circuits.
Table 1: Asymptotic comparison of the information-theoretic proof systems underlying Ligero, Stark, and Aurora, when applied to an $N$-gate arithmetic circuit.

† An IPCP [KR08] is a PCP oracle that is checked via an Interactive Proof; it is a special case of an IOP.

Table 2: Comparison of some non-interactive zero knowledge arguments for proving statements of the form “there exists a secret $w$ such that $C(x, w) = 1$” for a given arithmetic circuit $C$ of $N$ gates (and depth $d$) and public input $x$ of size $k$. The table is grouped by “technology”, and for simplicity assumes that the circuit’s underlying field has size $2^{O(\lambda)}$ where $\lambda$ is the security parameter. Approximate proof sizes are given for $N = 10^6$ gates over a cryptographically-large field, and a security level of 128 bits; some proof sizes may differ from those reported in the cited works because size had to be re-computed for the security level and $N$ used here; also, [ZGKPP17a] reports no implementation.

‡ Given a per-circuit preprocessing step.

‡‡ A tradeoff between proof size and verifier time is possible; see [WTSTW17].
2 Techniques

Our main technical contribution is a linear-length logarithmic-query IOP for R1CS (Theorem 1.1), which we use to design, implement, and evaluate a transparent SNARG for R1CS. Below we summarize the main ideas behind our protocol, and postpone to Sections 11 and 12 discussions of our system. In Section 2.1, we describe our approach to obtain the IOP for R1CS; this approach leads us to solve the univariate sumcheck problem, as discussed in Section 2.2; finally, in Section 2.3, we explain how we achieve zero knowledge. In Section 2.4 we conclude with a wider perspective on the techniques used in this paper.

2.1 Our interactive oracle proof for R1CS

The R1CS relation consists of instance-witness pairs \(((A, B, C, v), w)\), where \(A, B, C\) are matrices and \(v, w\) are vectors over a finite field \(F\), such that \((Az) \circ (Bz) = Cz\) for \(z := (1, v, w)\) and \(\circ\) denotes the entry-wise product.\(^3\) For example, R1CS captures arithmetic circuit satisfaction: \(A, B, C\) represent the circuit’s gates, \(v\) the circuit’s public input, and \(w\) the circuit’s private input and wire values.\(^4\)

We describe the high-level structure of our IOP protocol for R1CS, which has linear proof length and logarithmic query complexity. The protocol tests satisfaction by relying on two building blocks, one for testing the entry-wise vector product and the other for testing the linear transformations induced by the matrices \(A, B, C\). Informally, we thus consider protocols for the following two problems:

- **Rowcheck**: given vectors \(x, y, z \in F^m\), test whether \(x \circ y = z\), where \(\circ\) denotes entry-wise product.
- **Lincheck**: given vectors \(x \in F^m, y \in F^n\) and a matrix \(M \in F^{m \times n}\), test whether \(x = My\).

One can immediately obtain an IOP for R1CS when given IOPs for the rowcheck and lincheck problems. The prover first sends four oracles to the verifier: the satisfying assignment \(z\) and its linear transformations \(y_A := Az, y_B := Bz, y_C := Cz\). Then the prover and verifier engage in four IOPs in parallel:

- An IOP for the lincheck problem to check that \(y_A = Az\). Likewise for \(y_B\) and \(y_C\).
- An IOP for the rowcheck problem to check that \(y_A \circ y_B = yC\).

Finally, the verifier checks that \(z\) is consistent with the public input \(v\). Clearly, there exist \(z, y_A, y_B, y_C\) that yield valid rowcheck and lincheck instances if and only if \((A, B, C, v)\) is a satisfiable R1CS instance.

The foregoing reduces the goal to designing IOPs for the rowcheck and lincheck problems.

As stated, however, the rowcheck and lincheck problems only admit “trivial” protocols in which the verifier queries all entries of the vectors in order to check the required properties. In order to allow for sublinear query complexity, we need the vectors \(x, y, z\) to be encoded via some error-correcting code. We use the Reed–Solomon (RS) code because it ensures constant distance with constant rate while at the same time it enjoys efficient IOPs of Proximity [BBHR18b].

Given an evaluation domain \(L \subseteq F\) and rate parameter \(\rho \in [0, 1]\), RS \([L, \rho]\) is the set of all codewords \(f : L \to F\) that are evaluations of polynomials of degree less than \(\rho|L|\). Then, the encoding of a vector \(v \in F^S\) with \(S \subseteq F\) and \(|S| < \rho|L|\) is \(\hat{v}|L \in F^L\) where \(\hat{v}\) is the unique polynomial of degree \(|S| - 1\) such that \(\hat{v}|S = v\). Given this encoding, we consider “encoded” variants of the rowcheck and lincheck problems.

\(^3\)Throughout, we assume that \(F\) is “friendly” to FFT algorithms, i.e., \(F\) is a binary field or its multiplicative group is smooth.

\(^4\)The reader may be familiar with a standard arithmetization of circuit satisfaction (used, e.g., in the inner PCP of [ALMSS98]). Given an arithmetic circuit with \(m\) gates and \(n\) wires, each addition gate \(x_i \leftarrow x_j + x_k\) is mapped to the linear constraint \(x_i = x_j + x_k\) and each product gate \(x_i \leftarrow x_j \cdot x_k\) is mapped to the quadratic constraint \(x_i = x_j \cdot x_k\). The resulting system of equations can be written as \(A \cdot ((1, x) \otimes (1, x)) = b\) for suitable \(A \in F^{m \times (n+1)^2}\) and \(b \in F^{m}\). However, this reduction results in a quadratic blowup in the instance size. There is an alternative reduction due to [Mei12; GGPR13] that avoids this.
• **Univariate rowcheck** (Definition 7.1): given a subset $H \subseteq \mathbb{F}$ and codewords $f, g, h \in \text{RS} [L, \rho]$, check that $\hat{f}(a) \cdot \hat{g}(a) = \hat{h}(a)$ for all $a \in H$. (This is a special case of the definition that we use later.)

• **Univariate lincheck** (Definition 6.1): given subsets $H_1, H_2 \subseteq \mathbb{F}$, codewords $f, g \in \text{RS} [L, \rho]$, and a matrix $M \in \mathbb{F}^{H_1 \times H_2}$, check that $\hat{f}(a) = \sum_{b \in H_2} M_{a, b} \cdot \hat{g}(b)$ for all $a \in H_1$.

Given IOPs for the above problems, we can now get an IOP protocol for R1CS roughly as before. Rather than sending $z, Az, Bz, Cz$, the prover sends their encodings $f_z, f_{Az}, f_{Bz}, f_{Cz}$. The prover and verifier then engage in rowcheck and lincheck protocols as before, but with respect to these encodings.

For these encoded variants, we achieve IOP protocols with linear proof length and logarithmic query complexity, as required. For both cases, we do not use any routing and instead use a standard technique (dating back at least to [BFLS91]) to reduce the given testing problem to a sumcheck instance. However, since we are not working with multivariate polynomials, we cannot rely on the usual (multivariate) sumcheck protocol. Instead, we present a novel protocol that realizes a univariate analogue of the classical sumcheck protocol, and use it as the testing “core” of our IOP protocol for R1CS. We discuss univariate sumcheck next.

**Remark 2.1.** The verifier receives as input an explicit (non-uniform) description of the set of constraints, namely, the matrices $A, B, C$. In particular, the verifier runs in time that is at least linear in the number of non-zero entries in these matrices (if we consider a sparse-matrix representation for example).

### 2.2 A sumcheck protocol for univariate polynomials

A key ingredient in our IOP protocol is a *univariate* analogue of the classical (multivariate) sumcheck protocol [LFKN92]. Recall that the classical sumcheck protocol is an IP for claims of the form $\sum_{\bar{a} \in H^m} f(\bar{a}) = 0$, where $f$ is a given polynomial in $\mathbb{F}[X_1, \ldots, X_m]$ of individual degree $d$ and $H$ is a subset of $\mathbb{F}$. In this protocol, the verifier runs in time $\text{poly}(m, d, \log |\mathbb{F}|)$ and accesses $f$ at a single (random) location. The sumcheck protocol plays a fundamental role in computational complexity (it underlies celebrated results such as $\text{IP} = \text{PSPACE}$ [Sha92] and $\text{MIP} = \text{NEXP}$ [BFL91]) and in efficient proof protocols [GKR15; CMT12; TRMP12; Tha13; Tha15; WHGSW16; WJBSTWW17; ZGKPP17a; ZGKPP17b; WTSTW17].

We work with univariate polynomials instead, and need a univariate analogue of the sumcheck protocol (see previous subsection): how can a prover convince the verifier that $\sum_{\bar{a} \in H} f(\bar{a}) = 0$ for a given polynomial $f \in \mathbb{F}[X]$ of degree $d$ and subset $H \subseteq \mathbb{F}$? Designing a “univariate sumcheck” is not straightforward because univariate polynomials (the Reed–Solomon code) do not have the tensor structure used by the sumcheck protocol for multivariate polynomials (the Reed–Muller code). In particular, the sumcheck protocol has $m$ rounds, each of which reduces a sumcheck problem to a simpler sumcheck problem with one variable fewer. When there is only one variable, however, it is not clear to what simpler problems one can reduce.

Using different ideas, we design a natural protocol for univariate sumcheck in the cases where $H$ is an additive or multiplicative coset in $\mathbb{F}$ (i.e., a coset of an additive or multiplicative subgroup of $\mathbb{F}$).

**Theorem** (informal). The univariate sumcheck protocol over additive or multiplicative cosets has an $O(\log d)$-round IOP with proof complexity $O(d)$ over alphabet $\mathbb{F}$ and query complexity $O(\log d)$. The IOP prover uses $O(d \log |H|)$ field operations and the IOP verifier uses $O(\log d + \log^2 |H|)$ field operations.

We now provide the main ideas behind the protocol, when $H$ is an additive coset in $\mathbb{F}$.

Suppose for a moment that the degree $d$ of $f$ is less than $|H|$ (we remove this restriction later). A theorem of Byott and Chapman [BC99] states that the sum of $f$ over (an additive coset) $H$ is zero if and only if the coefficient of $X^{|H| − 1}$ in $f$ is zero. In particular, $\sum_{a \in H} f(a)$ is zero if and only if $f$ has degree less than $|H| − 1$. Thus, the univariate sumcheck problem over $H$ when $d < |H|$ is equivalent to low-degree testing.
The foregoing suggests a natural approach: test that $f$ has degree less than $|H| - 1$. Without any help from the prover, the verifier would need at least $|H|$ queries to $f$ to conduct such a test, which is as expensive as querying all of $H$. However, the prover can help by engaging with the verifier in an IOP of Proximity for the Reed–Solomon code. For this we rely on the recent construction of Ben-Sasson et al. [BBHR18b], which has proof length $O(d)$ and query complexity $O(\log d)$.

In our setting, however, we need to also handle the case where the degree $d$ of $f$ is larger than $|H|$. For this case, we observe that we can split any polynomial $f$ into two polynomials $g$ and $h$ such that $f(x) \equiv g(x) + \prod_{\alpha \in H} (x - \alpha) \cdot h(x)$ with $\deg(g) < |H|$ and $\deg(h) < d - |H|$; in particular, $f$ and $g$ agree on $H$, and thus so do their sums on $H$. This observation suggests the following extension to the prior approach: the prover sends $g$ (as an oracle) to the verifier, and then the verifier performs the prior protocol with $g$ in place of $f$. Of course, a cheating prover may send a polynomial $g$ that has nothing to do with $f$, and so the verifier must also ensure that $g$ is consistent with $f$. To facilitate this, we actually have the prover send $h$ rather than $g$; the verifier can then “query” $g(x)$ as $f(x) - \prod_{\alpha \in H} (x - \alpha) \cdot h(x)$; the prover then shows that $f, g, h$ are all of the correct degrees.

A similar reasoning works when $H$ is a multiplicative coset in $\mathbb{F}$ (see Remark 5.6). It remains an interesting open problem to establish whether the foregoing can be extended to any subset $H$ in $\mathbb{F}$.

**Remark 2.2** (vanishing vs. summing). The following are both linear subcodes of the Reed–Solomon code:

\[
\text{VanishRS}[\mathbb{F}, L, H, d] := \{ f : L \to \mathbb{F} \mid f \text{ has degree less than } d \text{ and is zero everywhere on } H \},
\]

\[
\text{SumRS}[\mathbb{F}, L, H, d] := \{ f : L \to \mathbb{F} \mid f \text{ has degree less than } d \text{ and sums to zero on } H \}.
\]

Our univariate sumcheck protocol is an IOP of Proximity for SumRS, and is reminiscent of IOPs of Proximity for VanishRS (e.g., see [BBHR18a]). Nevertheless, there are also intriguing differences between the two cases. For example, while it is known how to test proximity to VanishRS for general $H$, we only know how to test proximity to SumRS when $H$ is a coset. Additionally, our IOP protocol for R1CS from Section 2.1 can be viewed as a reduction from checking satisfaction of R1CS to testing proximity to SumRS; we do not know how to carry out a similar reduction to VanishRS. Indeed, there is an interactive reduction from VanishRS to SumRS, but no reduction in the other direction is known.

### 2.3 Efficient zero knowledge from algebraic techniques

The ideas discussed thus far yield an IOP protocol for R1CS with linear proof length and logarithmic query complexity. However these by themselves do not provide zero knowledge.

We achieve zero knowledge by leveraging recent algebraic techniques [BCGV16]. Informally, we adapt these techniques to achieve efficient zero knowledge variants of key sub-protocols, including the univariate sumcheck protocol (see Section 5.1) and low-degree testing (see Section 9.1), and combine these to achieve a zero knowledge IOP protocol for R1CS (see Sections 8.1 and 10).

We summarize the basic intuition for how we achieve zero knowledge in our protocols.

First, we use bounded independence. Informally, rather than encoding a vector $z \in \mathbb{F}^H$ by the unique polynomial of degree $|H| - 1$ that matches $z$ on $H$, we instead sample uniformly at random a polynomial of degree, say, $|H| + 9$ conditioned on matching $z$ on $H$. Any set of 10 evaluations of such a polynomial are independently and uniformly distributed in $\mathbb{F}$ (and thus reveal no information about $z$), provided these evaluations are outside of $H$. To ensure this latter condition, we choose the evaluation domain $L$ of Reed–Solomon codewords to be disjoint from $H$. Thus, for example, if $H$ is a linear space (an additive subgroup of $\mathbb{F}$) then we choose $L$ to be an affine subspace (a coset of some additive subgroup), since the
underlying machinery for low-degree testing (e.g., [BBHR18b]) requires codewords to be evaluated over algebraically-structured domains. All of our protocols are robust to these variations.

Bounded independence alone does not suffice, though. For example, in the sumcheck protocol, consider the case where the input vector \( z \in \mathbb{F}^H \) is all zeroes. The prover samples a random polynomial \( f \) of degree \( |H| + 9 \), such that \( \hat{f}(a) = 0 \) for all \( a \in H \), and sends its evaluation \( \hat{f} \) over \( L \) disjoint from \( H \) to the verifier. As discussed, any ten queries to \( f \) result in ten independent and uniformly random elements in \( \mathbb{F} \). Observe, however, that when we run the sumcheck protocol on \( f \), the polynomial \( g \) (the remainder of \( \hat{f} \) when divided by \( \prod_{\alpha \in H} (x - \alpha) \)) is the zero polynomial: all randomness is removed by the division.

To remedy this, we use self-reducibility to reduce a sumcheck claim about the polynomial \( f \) to a sumcheck claim about a random polynomial. The prover first sends a random Reed–Solomon codeword \( r \), along with the value \( \beta := \sum_{a \in H} r(a) \). The verifier sends a random challenge \( \rho \in \mathbb{F} \). Then the prover and verifier engage in the univariate sumcheck protocol with respect to the new claim “\( \sum_{a \in H} \rho f(a) + r(a) = \beta \)”.

Since \( r \) is uniformly random, \( \rho f + r \) is uniformly random for any \( \rho \), and thus the sumcheck protocol is performed on a random polynomial, which ensures zero knowledge. Soundness is ensured by the fact that if \( f \) does not sum to 0 on \( H \) then the new claim is true with probability \( 1/|\mathbb{F}| \) over the choice of \( \rho \).

2.4 Perspective on our techniques

A linear-length logarithmic-query IOP for a “circuit-like” NP-complete relation like R1CS (Theorem 1.1) may come as a surprise. We wish to shed some light on our IOP construction by connecting the ideas behind it to prior ideas in the probabilistic checking literature, and use these connections to motivate our construction.

A significant cost in all known PCP constructions with good proof length is using routing networks to reduce combinatorial objects (circuits, machines, and so on) to structured algebraic ones;\(^5\) routing also plays a major role in many IOPs [BCGV16; BCFG17; BCGHPRSTV17; BBHR18a]. While it is plausible that one could adapt routing techniques to route the constraints of an R1CS instance (similarly to [PS94]), such an approach would likely incur logarithmic-factor overheads, precluding linear-size IOPs.

A recent work [BCGRS17] achieves linear-length constant-query IOPs for boolean circuit satisfaction without routing the input circuit. Unfortunately, [BCGRS17] relies on other expensive tools, such as algebraic-geometry (AG) codes and quasilinear-size PCPs of proximity [BS08]; moreover, it is not zero knowledge. Informally, [BCGRS17] tests arbitrary (unstructured) constraints by invoking a sumcheck protocol [LFKN92] on a \( O(1) \)-wise tensor product of AG codes; this latter is then locally tested via tools in [BS06; BS08].

One may conjecture that, to achieve an IOP for R1CS like ours, it would suffice to merely replace the AG codes in [BCGRS17] with the Reed–Solomon code, since both codes have constant rate. But taking a tensor product exponentially deteriorates rate, and testing proximity to that tensor product would be expensive.

An alternative approach is to solve a sumcheck problem directly on the Reed–Solomon code. Existing protocols are not of much use here: the multivariate sumcheck protocol relies on a tensor structure that is not available in the Reed–Solomon code, and recent IOP implementations either use routing [BCGGHPRSTV17; BBHR18a] or achieve only sublinear query complexity [AHIV17].

Instead, we design a completely new IOP for a sumcheck problem on the Reed–Solomon code. We then combine this solution with ideas from [BCGRS17] (to avoid routing) and from [BCGV16] (to achieve zero knowledge) to obtain our linear-length logarithmic-query IOP for R1CS. Along the way, we rely on recent efficient proximity tests for the Reed–Solomon code [BBHR18b].

---

\(^5\)Polishchuk and Spielman [PS94] reduce boolean circuit satisfaction to a trivariate algebraic coloring problem with “low-degree” neighbor relations, by routing the circuit’s wires over an arithmetized routing network. Ben-Sasson and Sudan [BS08] reduce nondeterministic machine computations to a univariate algebraic satisfaction problem by routing the machine’s memory accesses over another arithmetized routing network. Routing is again a crucial component in the linear-size sublinear-query PCPs of [BKKMS13].
3 Roadmap

In Section 4 we provide necessary definitions about codes, proof systems, and other notions. Subsequent sections describe subprotocols, presented as Reed–Solomon encoded IOPs, which are IOPs for which soundness only holds against provers whose messages are Reed–Solomon codewords of specified rates, that are later compiled into standard IOPs. Specifically: the sumcheck protocol is in Section 5, the rowcheck protocol in Section 7, and the lincheck protocol in Section 6; the latter two are interactive reductions to univariate sumcheck. In Section 8 we combine the rowcheck and lincheck protocols to obtain an RS-encoded IOP for R1CS. In Section 9 we explain how to transform RS-encoded IOPs to standard IOPs, and in Section 10 we apply this transformation to our RS-encoded IOP for R1CS. Fig. 3 summarizes the structure of our IOP for R1CS. Finally, in Section 11 we describe our implementation and in Section 12 we report on its evaluation.

Throughout, we focus on the case where all relevant domains are additive cosets (affine subspaces) in $\mathbb{F}$; the case where domains are multiplicative cosets is similar, with only minor modifications (see Remark 5.6).

**Figure 3: Structure of our IOP for R1CS in terms of key sub-protocols.**
4 Definitions

Given a relation \( R \subseteq S \times T \), we denote by \( L(R) \subseteq S \) the set of \( s \in S \) such that there exists \( t \in T \) with \((s, t) \in R\); for \( s \in S \), we denote by \( R|_s \subseteq T \) the set \( \{ t \in T : (s, t) \in R \} \). Given a set \( S \) and strings \( v, w \in S^n \) for some \( n \in \mathbb{N} \), the fractional Hamming distance \( \Delta(v, w) \in [0, 1] \) is \( \Delta(v, w) := \frac{1}{n} \{ i : v_i \neq w_i \} \).

4.1 Codes

Interleaved codes. Given linear codes \( C_1, \ldots, C_m \subseteq \mathbb{F}^n \) with alphabet \( \mathbb{F} \), we denote by \( \prod_{i=1}^m C_i \subseteq (\mathbb{F}^m)^n \equiv \mathbb{F}^{m \times n} \) the linear “interleaved” code with alphabet \( \mathbb{F}^m \) that equals the set of all \( m \times n \) matrices whose \( i \)-th row is in \( C_i \). If \( C_1 = \cdots = C_m \), we write \( C^m \) for \( \prod_{i=1}^m C_i \). Since the alphabet is \( \mathbb{F}^m \), the Hamming distance is taken column-wise: for \( A, A' \in \mathbb{F}^{m \times n} \), \( \Delta(A, A') := \frac{1}{n} \{ j \in [n] : \exists i \in [m] \text{ s.t. } A_{i,j} \neq A'_{i,j} \} \).

The Reed–Solomon code. Given a subset \( L \) of a field \( \mathbb{F} \) and \( \rho \in (0, 1) \), we denote by \( \text{RS}[L, \rho] \subseteq \mathbb{F}^L \) all evaluations over \( L \) of univariate polynomials of degree less than \( \rho|L| \). That is, a word \( c \in \mathbb{F}^L \) is in \( \text{RS}[L, \rho] \) if there exists a polynomial \( p \) of degree less than \( \rho|L| \) such that \( c_a = p(a) \) for every \( a \in L \). We denote by \( \text{RS}[L, (\rho_1, \ldots, \rho_n)] := \prod_{i=1}^n \text{RS}[L, \rho_i] \) the interleaving of Reed–Solomon codes with rates \( \rho_1, \ldots, \rho_n \).

4.2 Representations of polynomials

We frequently move from univariate polynomials over \( \mathbb{F} \) to their evaluations on chosen subsets of \( \mathbb{F} \), and back. We use plain letters like \( f, g, h, \pi \) to denote evaluations of polynomials, and “hatted letters” \( \hat{f}, \hat{g}, \hat{h}, \hat{\pi} \) to denote corresponding polynomials. This bijection is well-defined only if the size of the evaluation domain is larger than the degree. Formally, if \( f \in \text{RS}[L, \rho] \) for \( L \subseteq \mathbb{F} \), \( \rho \in (0, 1] \), then \( \hat{f} \) is the unique polynomial of degree less than \( \rho|L| \) whose evaluation on \( L \) equals \( f \). Likewise, if \( \hat{f} \in \mathbb{F}[X] \) with \( \deg(f) < \rho|L| \), then \( f_L := \hat{f}|_L \in \text{RS}[L, \rho] \) (but we will drop the subscript when the choice of subset is clear from context).

4.3 The fast Fourier transform

We often rely on polynomial arithmetic, which can be efficiently performed via fast Fourier transforms and their inverses. In particular, polynomial evaluation and interpolation over an (affine) subspace of size \( n \) of a finite field can be performed in \( O(n \log n) \) field operations via an additive FFT [LCH14]. Because in practice the number of FFTs we perform is important, when discussing complexities we use the notation \( \text{FFT}(\mathbb{F}, m) \) for the cost of a single additive FFT (or IFFT) on a subspace of \( \mathbb{F} \) of size \( m \).

Remark 4.1. Strictly, an additive FFT evaluates a polynomial of degree \( d \) on a subspace of size \( d + 1 \). To evaluate on a larger subspace (of size \( n \)), one can run an FFT over each coset of the smaller space inside the larger one at a cost of \( \frac{n^2}{2} \cdot O(d \log d) = O(n \log d) \). We will suppress this technicality when it appears, and upper bound the cost of such an evaluation by an FFT on a subspace of size \( n \).

4.4 Subspace polynomials

Let \( \mathbb{F} \) be an extension field of a prime field \( \mathbb{F}_p \), and \( H \) be a subset of \( \mathbb{F} \). We denote by \( \mathbb{Z}_H \) the unique nonzero polynomial of degree at most \( |H| \) that is zero on \( H \). If \( H \) is an (affine) subspace of \( \mathbb{F} \), then \( \mathbb{Z}_H \) is called an (affine) subspace polynomial. In this case, there exist \( c_1, \ldots, c_k, d \in \mathbb{F} \), where \( k \) is the dimension of \( H \), such that \( \mathbb{Z}_H(X) = \sum_{i=0}^k c_i X^i + d \) (and, furthermore, if \( H \) is linear, then \( d = 0 \)). See [LN97, Chapter 3.4] and [BCGT13, Remark C.8] for how to find the coefficients \( c_i, d \) in \( O((\dim H)^2) \) field operations.
of this type are called linearized because they are $\mathbb{F}_p$-affine maps: if $H = H_0 + \beta$ for a subspace $H_0 \subseteq \mathbb{F}$ and shift $\beta \in \mathbb{F}$, then $\mathbb{Z}_H(X) \equiv \mathbb{Z}_{H_0}(X) - \mathbb{Z}_{H_0}(\beta)$, and $\mathbb{Z}_{H_0}$ is an $\mathbb{F}_p$-linear map.

4.5 Interactive oracle proofs

The information-theoretic protocols in this paper are Interactive Oracle Proofs (IOPs) [BCS16; RRR16], which combine aspects of Interactive Proofs [Bab85; GMR89] and Probabilistically Checkable Proofs [BFLS91; AS98; ALMSS98], and also generalize the notion of Interactive PCPs [KR08].

A $k$-round public-coin IOP has $k$ rounds of interaction. In the $i$-th round of interaction, the verifier sends a uniformly random message $m_i$ to the prover; then the prover replies with a message $\pi_i$ to the verifier. After $k$ rounds of interaction, the verifier makes some queries to the oracles it received and either accepts or rejects.

An IOP system for a relation $R$ with round complexity $k$ and soundness error $\epsilon$ is a pair $(P, V)$, where $P, V$ are probabilistic algorithms, that satisfies the following properties. (See [BCS16; RRR16] for details.)

Completeness: For every instance-witness pair $(x, w)$ in the relation $R$, $(P(x, w), V(x))$ is a $k$-round interactive oracle protocol with accepting probability 1.

Soundness: For every instance $x \notin L(R)$ and unbounded malicious prover $\hat{P}$, $(\hat{P}, V(x))$ is a $k$-round interactive oracle protocol with accepting probability at most $\epsilon(n)$.

Like the IP model, a fundamental measure of efficiency is the round complexity $k$. Like the PCP model, two additional fundamental measures of efficiency are the proof length $p$, which is the total number of alphabet symbols in all of the prover’s messages, and the query complexity $q$, which is the total number of locations queried by the verifier across all of the prover’s messages.

We say that an IOP system is non-adaptive if the verifier queries are non-adaptive, namely, the queried locations depend only on the verifier's inputs and its randomness. All of our IOP systems will be non-adaptive.

Since the verifier is public coin, its behavior in the interactive part of the protocol is easy to describe. We can therefore think of $V$ as a randomized algorithm which, given its prior random messages and oracle access to the prover’s messages, makes queries to the prover’s messages and either accepts or rejects.

The foregoing division allows us to separately consider the randomness and soundness error for these two phases, which is useful for a more fine-grained soundness-error reduction. Letting $r_i$ and $r_q$ be the randomness complexities of interaction and query phases respectively, the quantities $\epsilon_i$ and $\epsilon_q$ satisfy the following relation (for all instances $x \notin L(R)$ and malicious provers $\hat{P}$):

$$\Pr \left[ \Pr_{r \leftarrow \{0,1\}^n} [V^{\pi_1, \ldots, \pi_k}(x, m_1, \ldots, m_k, r) = 1] \geq \epsilon_q \quad \left( m_1, \ldots, m_k \leftarrow \{0,1\}^{r_i}, \pi_1, \ldots, \pi_k \leftarrow (\hat{P}, (m_1, \ldots, m_k)) \right) \right] \leq \epsilon_i$$

That is, the probability that random messages make $V$ accept with probability at least $\epsilon_q$ (over internal randomness) is at most $\epsilon_i$. In particular, the overall soundness error is at most $\epsilon_i + \epsilon_q$. Note that an IOP with $\epsilon_i = 0$ is a PCP, an IOP with $\epsilon_q = 0$ is an IP, and an IOP with both $\epsilon_i = \epsilon_q = 0$ is a deterministic (NP) proof.

Given the above, consider a “semi-black-box” example of soundness-error reduction: the interactive phase is run once, and then we repeat the query phase $\ell$ times with fresh randomness. This yields an IOP with query complexity $\ell \cdot q$, randomness complexity $r_i + \ell \cdot r_q$, and soundness error $\epsilon_i + \epsilon_q^\ell$, but with the same proof length and number of rounds. The running time of the prover is unchanged, and the verifier runs in time $O(\ell \cdot t_V)$. By comparison, repetition of the entire protocol yields proof length $\ell \cdot p$ and $\ell \cdot k$ rounds, for soundness error $(\epsilon_i + \epsilon_q)^\ell$; the prover runs in time $O(\ell \cdot t_P)$ and the verifier in time $O(\ell \cdot t_V)$. 

15
4.5.1 IOPs of proximity

An IOP of Proximity extends an IOP the same way that PCPs of Proximity extend PCPs. An IOPP system for a relation \( R \) with round complexity \( k \), soundness error \( \varepsilon \), and proximity parameter \( \delta \) is a pair \((P, V)\) that satisfies the following properties.

**Completeness:** For every instance-witness pair \((x, w)\) in the relation \( R \), \((P(x, w), V^w(x))\) is a \( k(n) \)-round interactive oracle protocol with accepting probability 1.

**Soundness:** For every instance-witness pair \((x, w)\) with \( \Delta(w, R|x) \geq \delta(n) \) and unbounded malicious prover \( P \), \((\hat{P}, V^w(x))\) is a \( k(n) \)-round interactive oracle protocol with accepting probability at most \( \varepsilon(n) \).

Efficiency measures for IOPPs are as for IOPs, except that we also count queries to the witness. Namely, if \( V \) makes at most \( q_w \) queries to \( w \) and at most \( q_x \) queries across all prover messages, the query complexity is \( q := q_w + q_x \). Like with IOPs, we divide public-coin IOPPs into an interaction phase and a query phase.

**Low-degree testing.** For the purposes of this paper, a low-degree test is an IOPP for the Reed–Solomon relation \( R_{RS} := \{(x,\rho) : L \subseteq \mathbb{F}, \rho \in (0, 1), p \in RS[L, \rho]\} \). In this case \( \varepsilon \) and \( \delta \) are functions of \( \rho \).

4.6 Zero knowledge

The definitions of unconditional (perfect) zero knowledge that we use for IOPs and for IOPPs follow those in [GIMS10; IW14; BCFGRS17]. We first define the notion of a view and of straightline access; after that we define zero knowledge for IOPs and for IOPPs in a way that suffices for our purposes.

**Definition 4.2.** Let \( A, B \) be algorithms and \( x, y \) strings. We denote by \( \text{View} (B(y), A(x)) \) the view of \( A(x) \) in an interactive oracle protocol with \( B(y) \), i.e., the random variable \((x, r, a_1, \ldots, a_n)\) where \( x \) is \( A \)'s input, \( r \) is \( A \)'s randomness, and \( a_1, \ldots, a_n \) are the answers to \( A \)'s queries into \( B \)'s messages.

**Definition 4.3.** An algorithm \( B \) has straightline access to an algorithm \( A \) if \( B \) interacts with \( A \), without rewinding, by exchanging messages with \( A \) and answering any oracle queries along the way.

We denote by \( B^A \) the concatenation of \( A \)'s random tape and \( B \)'s output when it has straightline access to \( A \). (Since \( A \)'s random tape could be super-polynomially large, \( B \) cannot sample it for \( A \) and then output it; instead, we restrict \( B \) not to see it, and we prepend it to \( B \)'s output.)

For IOPs, we consider unconditional (perfect) zero knowledge against bounded-query verifiers.

**Definition 4.4.** An IOP system \((P, V)\) for a relation \( R \) is (perfect) zero knowledge against query bound \( b \) if there exists a simulator algorithm \( S \) such that for every \( b \)-query algorithm \( \hat{V} \) and instance-witness pair \((x, w) \in R, S^\hat{V} (x) \) and \( \text{View} (P(x, w), \hat{V}(x)) \) are identically distributed. (An algorithm is \( b \)-query if, on input \( x \), it makes at most \( b(|x|) \) queries to any oracles it has access to.) Moreover, \( S \) must run in time \( \text{poly}(|x| + q_{\hat{V}}(|x|)) \), where \( q_{\hat{V}}(\cdot) \) is \( \hat{V} \)'s query complexity.

For zero knowledge against arbitrary polynomial-time adversaries, it suffices for \( b \) to be superpolynomial. Note that \( S \)'s running time is required to be polynomial in the input size \( |x| \) and the actual number of queries \( \hat{V} \) makes (as a random variable) and, in particular, may be polynomial even if \( b \) is not. We do not restrict \( \hat{V} \) to make queries only at the end of the interaction; all of our protocols will be zero knowledge against the more general class of verifier that can, at any time, make queries to any oracle it has already received.

For IOPPs, we consider unconditional (perfect) zero knowledge against unbounded-query verifiers.
Definition 4.5. An IOPP system $(P, V)$ for a relation $R$ is (perfect) zero knowledge against unbounded queries if there exists a simulator algorithm $S$ such that for every algorithm $\tilde{V}$ and instance-witness pair $(x, w) \in R$, the following two random variables are identically distributed:
\[
\left( S^{x,w}(x), q_S \right) \text{ and } \left( \text{View} \ (P(x, w), \tilde{V}^{x,w}(x)), q_{\tilde{V}} \right),
\]
where $q_S$ is the number of queries to $w$ made by $S$, and $q_{\tilde{V}}$ is the number of queries to $w$ or to prove messages made by $\tilde{V}$. Moreover, $S$ must run in time $\text{poly}(|x| + q_{\tilde{V}}(|x|))$, where $q_{\tilde{V}}(\cdot)$ is $\tilde{V}$’s query complexity.

4.7 Reed–Solomon encoded IOP

We typically first describe IOPs for which soundness only holds against provers whose messages are Reed–Solomon codewords of specified rates and on which certain rational constraints hold, and later “compile” them into standard IOPs.\(^6\) This facilitates focusing on a protocol’s key ideas, and leaves handling provers that do not respect this restriction to generic tools. We first define what we mean by a polynomial relation.

Definition 4.6. A rational constraint is a pair $(C, \sigma)$ where $C = (N, D), N : \mathbb{F}^{1+\ell} \rightarrow \mathbb{F}$, $D : \mathbb{F} \rightarrow \mathbb{F}$ are arithmetic circuits and $\sigma \in (0, 1]$ is a rate parameter. A rational constraint $(C, \sigma)$ and an interleaved word $f \in (L \rightarrow \mathbb{F})^k$ jointly define a codeword $C[f] : L \rightarrow \mathbb{F}$, given by $C[f](\alpha) := \frac{\alpha^{N(N(\alpha, f_1(\alpha), ..., f_k(\alpha))}}{D(\alpha)}$ for all $\alpha \in L$. A rational constraint $(C, \sigma)$ is satisfied by $f$ if $C[f] \in \text{RS} [L, \sigma]$.\(^7\)

An Reed–Solomon encoded IOP (RS-encoded IOP) for a relation $R$ is a tuple $(P, V, (\tilde{\rho}_i)_{i=1}^k)$, where $P$ and $V$ are probabilistic algorithms and $\tilde{\rho}_1 \in (0, 1][\ell_1], ..., \tilde{\rho}_k \in (0, 1][\ell_k]$, that satisfies the following properties.

Completeness: For every instance-witness pair $(x, w)$ in the relation $R$, $(P(x, w), V(x))$ is a $k(n)$-round interactive oracle protocol, where the $i$-th message of $P$ is a codeword of $\text{RS} [L, \tilde{\rho}_i]$, and $V$ outputs a set of rational constraints that are satisfied with respect to the prover’s messages with probability $1$.

Soundness: For every instance $x \notin L(R)$ and unbounded malicious prover $\tilde{P}$ whose $i$-th message is a codeword of $\text{RS} [L, \tilde{\rho}_i]$, $(\tilde{P}, V(x))$ is a $k(n)$-round interactive oracle protocol wherein the set of rational constraints output by $V$ are satisfied with respect to the prover’s messages with probability at most $\epsilon(n)$.

The maximum rate $\rho_{\text{max}}$ of a Reed–Solomon encoded IOP is the maximum over the rates of the codewords to be sent by the prover and those induced by the verifier’s rational constraints. To formally define it, we first introduce the notion of degree function for an arithmetic circuit $C : \mathbb{F}^{\ell+1} \rightarrow \mathbb{F}$. Given $d_1, ..., d_\ell \in \mathbb{N}$, define $D_C(d_1, ..., d_\ell)$ to be the smallest integer $\varepsilon$ such that for all $p_i \in \mathbb{F}^{\leq d_i}[X]$ there exists a polynomial $q \in \mathbb{F}^{\leq \varepsilon}[X]$ such that $C(X, p_1(X), ..., p_\ell(X)) \equiv q(X)$. Given $L \subseteq \mathbb{F}$ and $p \in (0, 1]^\ell$, we abuse notation and write $D_C(p)$ for $D_C(p_1[L], ..., p_\ell[L])$/L (L will typically be clear from context). Given this notation, and letting $\tilde{\rho} := (\tilde{\rho}_1, ..., \tilde{\rho}_k)$, the maximum rate $\rho_{\text{max}}$ equals the maximum rate in both $\tilde{\rho}$ and $\{\sigma + \text{deg}(D), D_N(\tilde{\rho})\}_{\sigma \in V, (C, \sigma) \in \varepsilon}$.\(^8\)

---

\(^6\)Rational constraints enable us to capture useful optimizations that involve testing “virtual oracles” implicitly derived from oracles sent by the prover. Such optimizations ultimately reduce proof length in the resulting SNARGs as discussed, e.g., in [BBHR18a].

\(^7\)For $\alpha \in L$, if $D(\alpha) = 0$ then we define $C[f](\alpha) := \perp$. Note that if this holds for some $\alpha \in L$ then, for any word $f$ and rate parameter $\sigma$, the rational constraint $(C, \sigma)$ is not satisfied by $f$; in particular, the completeness condition does not hold.

\(^8\)This definition may appear mysterious, but it is naturally motivated by the proof of Theorem 9.1.
Remark 4.7. The model of RS-encoded IOPs does not forbid the verifier from making queries to messages. However, in all of our protocols to achieve soundness it suffices for the rational constraints output by the verifier to be satisfied (and so the verifier does not make any queries). For this reason, we do not consider query complexity when discussing RS-encoded IOPs. Naturally, after we “compile” an RS-encoded IOP into a corresponding (regular) IOP, the resulting verifier will make queries to the proof; for details, see Section 9.

4.7.1 Proximity

In an RS-encoded IOP of Proximity (RS-encoded IOPP), soundness must hold only if prover messages are Reed–Solomon codewords and the witness is a tuple of Reed–Solomon codewords. Formally, a Reed–Solomon IOPP system for a relation $R \subseteq \{0,1\}^n \times \text{RS}[L,\vec{\rho}_w]$ is a tuple $(P,V,(\vec{p}_i)_{i=1}^k)$, where $P$ and $V$ are probabilistic algorithms, that satisfies the properties below. Note that the rational constraints output by the verifier may now also take the witness as input; the definition of maximum rate is modified accordingly.

Completeness: For every instance-witness pair $(x,w)$ in the relation $R$, $(P(x,w),V^w(x))$ is a $k(n)$-round interactive oracle protocol with accepting probability 1, where the $i$-th message of $P$ is a codeword of RS $[L,\vec{p}_i]$, and $V$ outputs a set of rational constraints that are satisfied with respect to the witness and the prover’s messages with probability 1.

Soundness: For every instance-witness pair $(x,w)$ with $w \in (\text{RS}[L,\vec{\rho}_w] \setminus R|_x)$ and unbounded malicious prover $\tilde{P}$ whose $i$-th message is a codeword of RS $[L,\vec{p}_i]$, $(\tilde{P},V^w(x))$ is a $k(n)$-round interactive oracle protocol wherein the set of rational constraints output by $V$ are satisfied with respect to the witness and the prover’s messages with probability at most $\varepsilon(n)$.

While the soundness condition does not consider “distance” of candidate witnesses to $R|_x$ (as in Section 4.5.1), we think of the notion above as an IOPP because soundness holds with respect to a particular witness provided as an oracle to the verifier. (This is analogous to “exact” PCPPs in [IW14].)

4.7.2 Zero knowledge

The definition of zero knowledge for RS-encoded IOPs (resp., RS-encoded IOPPs) equals that for IOPs (resp., IOPPs). This is because the definitions of RS-encoded IOPs and (standard) IOPs differ only in the soundness condition. Note that while the honest verifiers that we consider never make queries, a malicious verifier may do so. Indeed, we must allow malicious verifiers to make queries in order to “lift” zero knowledge guarantees from an RS-encoded IOP to a corresponding (regular) IOP, and thereby achieve the notion of zero knowledge against a given query bound $b$ stated in Section 4.6. We further note that the structure of the compiler that performs this lifting (see Section 9) motivates a definition of query bound $b$ that can lead to more efficient constructions. Namely, since all of the prover messages and witnesses are over the same domain $L$, we merely count the number of distinct queries to this common domain, i.e., if a malicious verifier queries multiple prover messages (or witnesses) at the same position $\alpha \in L$, we consider it a single query.
5 Univariate sumcheck

We describe UNIVARIATE SUMCHECK, an RS-encoded IOPP for testing whether a low-degree univariate polynomial \( \hat{f} \) sums to zero on a given subspace \( H \subseteq \mathbb{F} \). This protocol is a univariate analogue of the multi-variate sumcheck protocol [LFKN92].

If \( \hat{f} \) has degree less than \( d \), then \( \hat{f} \) can be uniquely decomposed into polynomials \( \hat{g}, \hat{h} \) of degrees less than \( |H| \) and \( d - |H| \) (respectively) such that \( \hat{f} \equiv \hat{g} + Z_H \cdot \hat{h} \), where \( Z_H \) is the vanishing polynomial of \( H \) (see Section 4.4). This implies that \( \sum_{a \in H} \hat{f}(a) = \sum_{a \in H} (\hat{g}(a) + Z_H(a) \cdot \hat{h}(a)) = \sum_{a \in H} \hat{g}(a) \). By Lemma 5.4 below, this latter expression is equal to \( \beta \sum_{a \in H} a^{|H| - 1} \), where \( \beta \) is the coefficient of \( X^{|H| - 1} \) in \( \hat{g}(X) \). Note that \( \sum_{a \in H} a^{|H| - 1} \neq 0 \) since otherwise this would imply that all functions sum to zero on \( H \).

Thus, \( \sum_{a \in H} \hat{f}(a) = 0 \) if and only if \( \beta = 0 \).

This suggests the following RS-encoded IOP (actually an RS-encoded PCPP). The prover sends \( g, h \) (the evaluations of \( \hat{g}, \hat{h} \)). The verifier now must check that (a) \( \hat{f} \equiv \hat{g} + Z_H \cdot \hat{h} \), and (b) the coefficient of \( X^{|H| - 1} \) in \( \hat{g} \) is zero. For both conditions we use the definition of an RS-encoded IOPP: the verifier outputs a rational constraint specifying that the polynomial \( \hat{f} - Z_H \cdot \hat{h} \) is of degree less than \( |H| - 1 \), which corresponds to forcing the coefficient of \( X^{|H| - 1} \) to be zero. In the final (non-encoded) IOPP protocol this will correspond to testing proximity of \( \hat{f} - Z_H \cdot \hat{h} \) to a Reed–Solomon code with rate parameter \((|H| - 1)/|L|\).

Below we consider the more general case of testing that the sum equals a given \( \mu \in \mathbb{F} \) (rather than zero).

**Definition 5.1** (sumcheck relation). The relation \( R_{\text{SUM}} \) is the set of all pairs \(((\mathbb{F}, L, H, \rho, \mu), f)\) where \( \mathbb{F} \) is a finite field, \( L, H \) are affine subspaces of \( \mathbb{F} \), \( \rho \in (0, 1) \), \( \mu \in \mathbb{F} \), \( f \in \text{RS}[L, \rho] \), and \( \sum_{a \in H} \hat{f}(a) = \mu \).

**Theorem 5.2.** There exists an RS-encoded IOPP (Protocol 5.3) for the sumcheck relation \( R_{\text{SUM}} \) with the following parameters:

<table>
<thead>
<tr>
<th>alphabet</th>
<th>( \Sigma )</th>
<th>( \mathbb{F} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of rounds</td>
<td>( k )</td>
<td>( = 1 )</td>
</tr>
<tr>
<td>proof length</td>
<td>( p )</td>
<td>( = 2</td>
</tr>
<tr>
<td>randomness</td>
<td>( r )</td>
<td>( = 0 )</td>
</tr>
<tr>
<td>soundness error</td>
<td>( \varepsilon )</td>
<td>( = 0 )</td>
</tr>
<tr>
<td>prover time</td>
<td>( t_p )</td>
<td>( = O(</td>
</tr>
<tr>
<td>verifier time</td>
<td>( t_v )</td>
<td>( = O(</td>
</tr>
<tr>
<td>maximum rate</td>
<td>( \rho_{\text{max}} )</td>
<td>( = \rho )</td>
</tr>
</tbody>
</table>

**Protocol 5.3** (UNIVARIATE SUMCHECK). Let \( w = f \in \text{RS}[L, \rho] \) be the witness oracle, and let \( \hat{f} \) be the unique polynomial of degree at most \( \rho |L| \) that agrees with \( f \). The RS-encoded IOP protocol \((P, V)\) for \( R_{\text{SUM}} \) proceeds as follows.

1. \( P \) computes the unique polynomials \( \hat{g} \) and \( \hat{h} \) and unique element \( \beta \in \mathbb{F} \) such that \( \text{deg}(\hat{g}) < |H| - 1 \), \( \text{deg}(\hat{h}) < \rho |L| - |H| \), and \( \hat{f}(X) \equiv \hat{g}(X) + \beta X^{|H| - 1} + Z_H(X) \hat{h}(X) \).
2. \( P \) sends \( \hat{h} := \hat{h}|_L \in \text{RS}[L, \rho - |H|/|L|] \) to \( V \).
3. \( V \) computes \( \xi := \sum_{a \in H} a^{|H| - 1} \) (this can be done efficiently as explained below), and accepts if and only if \( p \in \text{RS}[L, |H|/|L|] \) where \( \hat{p}(X) := \xi \cdot \hat{f}(X) - \mu \cdot X^{|H| - 1} - \xi \cdot Z_H(X) \hat{h}(X) \). In the formalism of RS-encoded IOPs (see Section 4.7), this corresponds to the rational constraint \((C, \sigma) := ((N, D), |H|/|L|)\) where \( N(X, Z_1, Z_2) := \xi \cdot Z_1 - \mu \cdot X^{|H| - 1} - \xi \cdot Z_H(X) \cdot Z_2 \) and \( D(X) := 1 \).

**Proof.** Completeness and soundness rely on the following lemma:
Lemma 5.4 ([BC99, Theorem 1], restated). Let \( H \) be an affine subspace of \( \mathbb{F} \), and let \( \hat{g}(x) \) be a univariate polynomial over \( \mathbb{F} \) of degree (strictly) less than \( |H| - 1 \). Then

\[
\sum_{a \in H} \hat{g}(a) = 0.
\]

We provide a self-contained proof of this statement in Appendix A, when \( \mathbb{F} \) is an extension field of \( \mathbb{F}_2 \).

Completeness. Consider \( f \in \text{RS} \{ L, \rho \} \) with \( \sum_{a \in H} \hat{f}(a) = \mu \). Then, by definition of \( g, h \) and Lemma 5.4,

\[
\mu = \sum_{a \in H} \left( \hat{g}(a) + \beta \cdot a^{|H| - 1} + Z_{H}(a)\hat{h}(a) \right) = \beta \xi.
\]

Therefore,

\[
\begin{align*}
\xi \cdot \hat{f}(X) - \mu \cdot X^{|H| - 1} - \xi \cdot Z_{H}(X)\hat{h}(X) \\
\equiv \xi \cdot (\hat{g}(X) + \beta X^{|H| - 1} + Z_{H}(X)\hat{h}(X)) - \mu \cdot X^{|H| - 1} - \xi \cdot Z_{H}(X)\hat{h}(X) \\
\equiv \xi \cdot \hat{g}(X) + \beta X^{|H| - 1} - \mu \cdot X^{|H| - 1} \equiv \xi \cdot \hat{g}(X).
\end{align*}
\]

Hence \( \hat{p}(X) \equiv \xi \cdot \hat{g}(X) \), and so \( p \in \text{RS}[L, |H|^{-1} L] \).

Soundness. Consider \( f \in \text{RS} \{ L, \rho \} \) with \( \sum_{a \in H} \hat{f}(a) = \mu' \neq \mu \). We show that for any \( h \in \text{RS}[L, |H|^{-1} L] \), \( p \notin \text{RS}[L, |H|^{-1} L] \). Suppose towards contradiction that \( p \in \text{RS}[L, |H|^{-1} L] \). Then, by Lemma 5.4, we have that \( \sum_{a \in H} \hat{p}(a) = 0 \). But also \( \sum_{a \in H} \hat{p}(a) = \sum_{a \in H} (\xi \cdot \hat{f}(a) - \mu \cdot a^{|H| - 1}) = \xi (\mu' - \mu) \neq 0 \), since \( \xi \neq 0 \); this is a contradiction.

Efficiency. For computational efficiency of the verifier, we use an additional lemma due to [BC99].

Lemma 5.5 ([BC99], implicit in the proof of Theorem 1). If \( H \) is an affine subspace of \( \mathbb{F} \), then \( \sum_{a \in H} a^{|H| - 1} \) equals the linear term of \( Z_{H} \).

The verifier runs in time \( O(\log^2 |H|) \): its work consists of finding the linear term of \( Z_{H} \), which can be achieved via a divide-and-conquer algorithm, and evaluating \( Z_{H} \) at a single point. The prover runs in time \( O(|L| \log |H|) + 3 \cdot \text{FFT}(\mathbb{F}, |L|) \): the polynomial division can be performed by interpolating (one IFFT) over \( L \) to obtain the coefficients of \( f \), running a divide-and-conquer algorithm to obtain the \( O(\log |H|) \) coefficients of \( Z_{H} \), and then performing standard symbolic polynomial division. Given the coefficients of \( \hat{h} \) and \( \hat{g} \), the evaluations \( h \) and \( g \) can be computed using two FFTs.

Remark 5.6. The univariate sumcheck relation states that \( H, L \) are affine subspaces of \( \mathbb{F} \) (Definition 5.1). One can define a similar relation where \( H, L \) are multiplicative cosets in \( \mathbb{F} \), in which case Theorem 5.2 holds essentially unchanged. The protocol is similar to Protocol 5.3, except that \( \hat{g} \) and \( \hat{h} \) are such that \( \hat{f}(X) = X \cdot \hat{g}(X) + \beta + Z_{H}(X)\hat{h}(X) \). The rational constraint becomes \( (C, \sigma) := ((N, D), |H|^{-1} L) \) where \( N(X, Z_1, Z_2) := |H| \cdot Z_1 - \mu - |H| \cdot Z_2, D(X) := X \). Correctness of this protocol follows from the fact that, if \( H \) is a multiplicative coset, \( \sum_{a \in H} \hat{p}(a) = \hat{p}(0) \cdot |H| \) for all polynomials \( \hat{p} \) with \( \deg(\hat{p}) < |H| \).
5.1 Zero knowledge

We describe how to modify Protocol 5.3 to achieve zero knowledge; the modification is an adaptation of algebraic techniques from [BCGV16; BCFGRS17]. The prover first sends a random Reed–Solomon codeword \( q \in \text{RS}[L,\rho] \). The verifier then replies with a random “challenge” element \( c \in F \). Finally, the prover and verifier engage in Protocol 5.3 with respect to the “virtual” oracle \( p := c \cdot f + r \), and new target value \( c \cdot \mu + \sum_{a \in H} q(a) \). Since \( p \) is an (almost) uniformly random Reed–Solomon codeword, one can efficiently simulate the sumcheck prover with input \( p \). We obtain the following theorem.

**Theorem 5.7.** There exists an RS-encoded IOPP (Protocol 5.8) for the sumcheck relation \( R_{\text{SUM}} \) (Definition 5.1), which is zero knowledge against unbounded queries, with the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet</td>
<td>( F )</td>
</tr>
<tr>
<td>number of rounds</td>
<td>( k = 1 )</td>
</tr>
<tr>
<td>proof length</td>
<td>( p = 3</td>
</tr>
<tr>
<td>randomness</td>
<td>( r = \log</td>
</tr>
<tr>
<td>soundness error</td>
<td>( \varepsilon = \frac{1}{</td>
</tr>
<tr>
<td>prover time</td>
<td>( t_p = O(</td>
</tr>
<tr>
<td>verifier time</td>
<td>( t_v = O(</td>
</tr>
<tr>
<td>maximum rate</td>
<td>( \rho_{\max} = \rho )</td>
</tr>
</tbody>
</table>

**Protocol 5.8.** Let \( f \in \text{RS}[L,\rho] \) be the witness oracle. Let \( (P_{\text{SUM}}, V_{\text{SUM}}) \) be the RS-encoded IOP for univariate sumcheck (Protocol 5.3). The zero knowledge RS-encoded IOP \( (P, V) \) for univariate sumcheck proceeds as follows.

1. \( P \) samples \( q \in \text{RS}[L,\rho] \) uniformly at random and sends it to \( V \), along with \( \beta := \sum_{a \in H} q(a) \).
2. \( V \) samples \( c \in F \) uniformly at random, and sends it to \( P \).
3. \( P \) and \( V \) invoke \( (P_{\text{SUM}}(x', c \cdot f + q), V_{\text{SUM}}^{c \cdot f + q}(x')) \), where \( x' := (F, L, H, \rho, c \cdot \mu + \beta) \).

**Proof.**

**Completeness.** Follows from the completeness of univariate sumcheck.

**Soundness.** Suppose that \( \sum_{a \in H} \hat{f}(a) = \alpha \neq \mu \). Let \( \beta' := \sum_{a \in H} q'(a) \), where \( q' \) is sent by \( \hat{P} \) in the first round. Then \( \sum_{a \in H} (c \cdot \hat{f} + q')(a) = c \cdot \alpha + \beta' \), which is equal to \( c \cdot \mu + \beta \) if and only if \( c = \frac{\beta - \beta'}{\alpha - \mu} \), which happens with probability \( 1/|F| \) for any fixed \( \beta, \beta' \). Hence with probability \( 1 - 1/|F| \), \( (x', c \cdot \mu + \beta) \notin R_{\text{SUM}} \), and soundness follows by the soundness of the standard protocol.

**Zero knowledge.** We describe a simulator \( S \) that, given straightline access to a (malicious) verifier \( \hat{V} \) and oracle access to a witness oracle \( f \in \text{RS}[L,\rho] \), perfectly simulates \( \hat{V} \)’s view in the real protocol.

1. Sample \( q_{\text{sim}} \in \text{RS}[L,\rho] \) uniformly at random and start simulating \( \hat{V} \).
2. Answer any query to \( f \) by querying \( f \), and answer any query to \( q \) by querying \( q_{\text{sim}} \). Let \( Q_{\text{sim}} \subseteq L \) be \( \hat{V} \)’s queries to \( q \) from the beginning of the simulation until the next step.
3. Send \( \beta_{\text{sim}} := \sum_{a \in H} \hat{q}_{\text{sim}}(a) \) to \( \hat{V} \).
4. Receive \( \hat{c}_{\text{sim}} \in F \) from \( \hat{V} \).
5. Sample \( p_{\text{sim}} \in \text{RS}[L,\rho] \) uniformly at random such that, for every \( q \in Q_{\text{sim}} \), \( p_{\text{sim}}(q) = \hat{c}_{\text{sim}} \cdot \hat{f}(q) + q_{\text{sim}}(q) \) and \( \sum_{a \in H} p_{\text{sim}}(a) = \hat{c}_{\text{sim}} \cdot \mu + \beta_{\text{sim}} \); this requires \( |Q_{\text{sim}}| \) queries to \( f \). (Note that if \( |Q_{\text{sim}}| > \rho |L| \) then \( p_{\text{sim}} \equiv \hat{f} + r_{\text{sim}} \).)
6. Answer any query to \( f \) by querying \( f \) (as before), and answer any query to \( q \) by querying \( p_{\text{sim}} - \hat{c}_{\text{sim}} \cdot \hat{f} \).
Note that $S$ runs in polynomial time, and the number of queries it makes to $f$ is exactly the number of queries that $\tilde{V}$ makes to $f$ and $q$.

To see that $\tilde{V}$'s view is perfectly simulated, we consider a hybrid experiment in which the “hybrid prover” reads all of $f$ (like the honest prover in the real world) but can modify messages after they are sent (like the simulator in the ideal world).

1. Sample $q \in \text{RS} [L, \rho]$ uniformly at random and start simulating $\tilde{V}$.
2. Send $q$ to $\tilde{V}$, along with $\beta := \sum_{a \in H} q(a)$. Let $Q \subseteq L$ be $\tilde{V}$'s queries to $q$ from the beginning of the simulation until the next step.
3. Receive $\tilde{c} \in \mathbb{F}$ from $\tilde{V}$.
4. Sample $p \in \text{RS} [L, \rho]$ uniformly at random such that, for every $q \in Q$, $p(q) = \tilde{c} \cdot f(q) + q(q)$ and $\sum_{a \in H} p(a) = \tilde{c} \cdot \mu + \beta$.
5. Replace $q$ with $p - \tilde{c} \cdot f$.
6. Simulate the interaction of $P_{\text{SUM}}(x', p_{\text{sim}})$ and $\tilde{V}$.

The distribution of $\tilde{V}$'s view in the real protocol is identical to the distribution of $\tilde{V}$'s view in the above experiment. In particular, all of $\tilde{V}$'s queries to $q$ after its replacement by $p - \tilde{c} \cdot f$ have the correct distribution. Moreover, it is not hard to see that $\tilde{V}$'s view in the above experiment and $S$'s output are identically distributed.

**Efficiency.** Most of the parameters are seen from the protocol description. We require the prover to send $r \in \text{RS} [L, \rho]$ uniformly at random, which can be done by choosing $\rho |L|$ coefficients uniformly at random and performing one FFT to evaluate that polynomial over $L$. \hfill \Box

### 5.2 Amortization

Given $\ell$ instance-witness pairs for univariate sumcheck $((\mathbb{F}, L, H, \rho_i, \mu_i), f_i)_{i \in [\ell]}$, we want to test that all of them are in $\mathcal{R}_{\text{SUM}}$. This is achieved with an $\ell$-fold increase in complexity, but we want to do this much more efficiently. This will be crucial in our final protocol. We first state formally the relation we will test.

**Definition 5.9 (\ell-sumcheck relation).** The relation $\mathcal{R}^\ell_{\text{SUM}}$ is the set of all $\ell$-tuples $((z_1, \ldots, z_\ell), (f_1, \ldots, f_\ell))$ such that for all $i = 1, \ldots, \ell$, $z_i = (\mathbb{F}, L, H, \rho_i, \mu_i)$, and $(z_i, f_i) \in \mathcal{R}_{\text{SUM}}$.

The idea is to have the verifier choose $z_1, \ldots, z_\ell \in \mathbb{F}$ uniformly at random and send them to the prover, and then to test that $\sum_{a \in H} \sum_{i=1}^\ell z_i f_i(a) = \sum_{i=1}^\ell z_i \mu_i$. Completeness is easy to see, and soundness follows from properties of random linear combinations. The verifier runtime is increased only by an additive $\ell$ term, which corresponds to sending $z_1, \ldots, z_\ell$ and querying each $f_i$ in one position. Crucially, the proof length is unchanged, and the prover still only performs three FFTs. We obtain the following lemma.

**Lemma 5.10.** There is an RS-encoded IOPP for the univariate $\ell$-sumcheck relation (Definition 5.9) with the following parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet</td>
<td>$\mathbb{F}$</td>
</tr>
<tr>
<td>number of rounds</td>
<td>$1$</td>
</tr>
<tr>
<td>proof length</td>
<td>$2</td>
</tr>
<tr>
<td>randomness</td>
<td>$\ell \log</td>
</tr>
<tr>
<td>soundness error</td>
<td>$1/</td>
</tr>
<tr>
<td>prover time</td>
<td>$O(</td>
</tr>
<tr>
<td>verifier time</td>
<td>$O(\log^2</td>
</tr>
<tr>
<td>maximum rate</td>
<td>$\rho_{\text{max}} = \rho$</td>
</tr>
</tbody>
</table>
for any instance \( \mathbf{\bar{x}} = (x_1, \ldots, x_\ell) = ( (F, L, H, \rho_i, \mu_i))_{i=1}^\ell \), where \( \rho = \max_i \rho_i \).

Protocol 5.11. Let \( \rho = \max_i \rho_i \), and let \( f_1, \ldots, f_\ell \in RS[L, \rho] \) be the witness oracles. Let \( (P_{SUM}, V_{SUM}) \) be the standard RS-encoded IOP for univariate sumcheck (Protocol 5.3). The RS-encoded IOP protocol for univariate \( \ell \)-sumcheck proceeds as follows.

1. \( V \) chooses \( z_1, \ldots, z_\ell \in F \) uniformly at random, and sends them to \( P \).
2. \( P \) and \( V \) invoke \( (P_{SUM}(x^*, w^*), V_{SUM}(x^*)) \), where \( x^* := (F, L, H, \rho, \sum_{i=1}^\ell z_i \mu_i), w^* := \sum_{i=1}^\ell z_i f_i \).

Proof.

Completeness. Suppose that, for all \( i \in [\ell], (x_i, f_i) \in R_{SUM} \). Then for any choice of \( z_1, \ldots, z_\ell \in F \),
\[
\sum_{a \in H} \sum_{i=1}^\ell z_i f_i(a) = \sum_{i=1}^\ell z_i \mu_i, \text{ so } (x^*, w^*) \in R_{SUM}.
\]

Soundness. Suppose that, for some \( i \in [\ell], (x_i, f_i) \notin R_{SUM} \). Then since \( z_1, \ldots, z_\ell \in F \) are uniformly random, \( \sum_{a \in H} \sum_{i=1}^\ell z_i f_i(a) = \sum_{i=1}^\ell z_i \mu_i \) (i.e., \( (x^*, w^*) \in R_{SUM} \)) with probability at most \( 1/|F| \).

Efficiency. The efficiency of the system corresponds to a single invocation of univariate sumcheck. The prover, in addition to the cost of running \( P_{SUM} \), pays \( O(\ell \cdot |L|) \) to construct \( w^* \). The verifier pays only an additive \( O(\ell) \) to pick \( z_1, \ldots, z_\ell \) and construct \( x^* \). \( \square \)
6 Univariate lincheck

We describe Univariate Lincheck, an RS-encoded IOPP for verifying linear relations on Reed–Solomon codewords. Given $H_1, H_2 \subseteq \mathbb{F}$, $f_1, f_2 \in \text{RS}[L, \rho]$, and a coefficient matrix $M \in \mathbb{F}^{H_1 \times H_2}$, we want to check that $f_1|_{H_1} = M \cdot f_2|_{H_2}$, where $\cdot$ is standard matrix multiplication over $\mathbb{F}$. The next definition captures this.

**Definition 6.1** (lincheck relation). The relation $\mathcal{R}_{\text{LIN}}$ is the set of all pairs $\{(\mathbb{F}, L, H_1, H_2, \rho, M), (f_1, f_2)\}$ where $\mathbb{F}$ is a finite field, $L, H_1, H_2$ are affine subspaces of $\mathbb{F}$, $\rho \in (0, 1)$, $f_1, f_2 \in \text{RS}[L, \rho]$, $M \in \mathbb{F}^{H_1 \times H_2}$, and $\forall a \in H_1$ $f_1(a) = \sum_{b \in H_2} M_{a,b} \cdot f_2(b)$.

To build intuition, consider that, given vectors $x \in \mathbb{F}^m, y \in \mathbb{F}^n$ and a matrix $M \in \mathbb{F}^{m \times n}$, a simple probabilistic test for the claim “$x = My$” is to check that $\langle r, x - My \rangle = 0$ for a random $r \in \mathbb{F}^m$. Indeed, if $x \neq My$ then $\Pr_r[\langle r, x - My \rangle = 0] = 1/|\mathbb{F}|$. However, this approach would require the verifier to sample $m$ random field elements, and send these to the prover. A straightforward modification (used also, e.g., in [BFLS91, §5.2]) requires only a single random field element and incurs only a modest increase in soundness error. Namely, letting $h(X) := \langle \tilde{X}, x - My \rangle$ where $\tilde{X} := (1, X, \ldots, X^{m-1})$, if $x \neq My$ then $h(X)$ is a non-zero polynomial of degree less than $m$ over $\mathbb{F}$, and thus $\Pr_{r \in \mathbb{F}}[h(\alpha) = 0] \leq m/|\mathbb{F}|$. The verifier now merely has to sample and send $\alpha \in \mathbb{F}$, and the prover must then prove the claim “$h(\alpha) = 0$” to the verifier. This latter claim is in fact a claim about sums: one can rewrite $h(X)$ as $\langle \tilde{X}, x \rangle - (M^T \tilde{X}, y)$ and, expanding the inner products, we obtain the two-sum expression $h(\alpha) = \sum_{i=1}^m \alpha^{i-1} x_i - \sum_{j=1}^n (\sum_{i=1}^m M_{i,j} \alpha^{i-1}) y_j$.

We now return to the RS-encoded version of the problem (defined above), and explain how the prover can handle the claim “$h(\alpha) = 0$” via the univariate sumcheck protocol.

We can think of $\hat{f_1}$ and $\hat{f_2}$ as the low-degree extensions of some $x \in \mathbb{F}^{H_1}$ and $y \in \mathbb{F}^{H_2}$ with $m := |H_1|$ and $n := |H_2|$. The verifier samples and sends $\alpha \in \mathbb{F}$ to the prover; the prover and verifier each compute the low-degree extension $\hat{p}_\alpha^{(1)}$ of $\tilde{\alpha} := (1, \alpha, \ldots, \alpha^{m-1})$, and the low-degree extension $\hat{p}_\alpha^{(2)}$ of $M^T \tilde{\alpha}$. We can then write $h(\alpha) = \sum_{a \in H_1} \hat{p}_\alpha^{(1)}(a) \hat{f}_1(a) - \sum_{b \in H_2} \hat{p}_\alpha^{(2)}(b) \hat{f}_2(b)$. In sum, we reduced the claim “$h(\alpha) = 0$” to a sumcheck instance of the polynomial $\hat{p}_\alpha^{(1)}(\cdot) \hat{f}_1(\cdot)$ over $H_1$ and one of the polynomial $\hat{p}_\alpha^{(2)}(\cdot) \hat{f}_2(\cdot)$ over $H_2$.

While $h(\alpha)$ equals zero in the honest case, the value of each summation may reveal information. Therefore, to ensure zero knowledge, we combine these two summations into a single summation over the affine space $H_1 \circ H_2$, defined to be the smallest affine space that contains both $H_1$ and $H_2$ (and note that if $H_1, H_2$ are linear subspaces then $H_1 \circ H_2 = H_1 + H_2$). Since the precise choice of $H_1, H_2$ is not important, for efficiency we will typically choose $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$ in order to minimize $|H_1 \circ H_2|$.

**Theorem 6.2.** Protocol 6.3 below is an RS-encoded IOPP for $\mathcal{R}_{\text{LIN}}$ (Definition 6.1) with parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet</td>
<td>$\mathbb{F}$</td>
</tr>
<tr>
<td>number of rounds</td>
<td>1</td>
</tr>
<tr>
<td>proof length</td>
<td>$2</td>
</tr>
<tr>
<td>randomness</td>
<td>$\log</td>
</tr>
<tr>
<td>soundness error</td>
<td>$</td>
</tr>
<tr>
<td>prover time</td>
<td>$O(|M| +</td>
</tr>
<tr>
<td>verifier time</td>
<td>$O(|M| + s) + t(V_{\text{SUM}}; \mathbb{F},</td>
</tr>
<tr>
<td>maximum rate</td>
<td>$\rho_{\text{max}} = \rho + s/</td>
</tr>
</tbody>
</table>

where $s := |H_1 \circ H_2|$ and $\|M\|$ is the number of nonzero entries of $M$.

**Protocol 6.3 (Univariate Lincheck).** Denote by $(P_{\text{SUM}}, V_{\text{SUM}})$ the RS-encoded IOPP for univariate sumcheck (Protocol 5.3). The RS-encoded IOPP $(P, V)$ for univariate lincheck works as follows.
1. $P$ and $V$ agree in advance on an ordering $\gamma: H_1 \to \{0, \ldots, |H_1| - 1\}$ of $H_1$.
2. $V$ draws a uniformly random $\alpha \in \mathbb{F}$ and sends it to $P$.
   The element $\alpha$ and the witness codewords $f_1, f_2 \in \text{RS}[L, \rho]$ jointly define several polynomials:
   - $\hat{p}_\alpha^{(1)}$ is the unique polynomial of degree less than $s$ s.t. $\hat{p}_\alpha^{(1)}(a) = \alpha^{\gamma(a)}$ for all $a \in H_1$, and $\hat{p}_\alpha^{(1)}(b) = 0$ for all $b \in H_1 \cap H_2 \setminus H_1$;
   - $\hat{p}_\alpha^{(2)}$ is the unique polynomial of degree less than $s$ s.t. $\hat{p}_\alpha^{(2)}(b) = \sum_{a \in H_1} M_{a,b} \cdot \alpha^{\gamma(a)}$ for all $b \in H_2$, and $\hat{p}_\alpha^{(2)}(a) = 0$ for all $a \in H_1 \cap H_2 \setminus H_2$;
   - $\hat{q}_\alpha(X) := \hat{f}_1(X)\hat{p}_\alpha^{(1)}(X) - \hat{f}_2(X)\hat{p}_\alpha^{(2)}(X)$.
   Observe that $q_\alpha = \hat{q}_\alpha|_L \in \text{RS}[L, \rho']$ where $\rho' := \rho + \frac{n}{|H_1|}$.
3. $P$ and $V$ run $(P_{\text{SUM}}(\mathbf{x}', q_\alpha), V_{\text{SUM}}(\mathbf{x}'))$ where $\mathbf{x}' := (\mathbb{F}, L, H_1 \cap H_2, \rho', \mu = 0)$. Note that $V$ can use its oracles $f_1, f_2$ to simulate access to the oracle $q_\alpha$.
4. $V$ accepts if and only if $V_{\text{SUM}}$ accepts.

Proof. Completeness and soundness rely on the fact that, by rearranging terms, for every $\alpha \in \mathbb{F}$ it holds that:

$$h(\alpha) := \sum_{b \in H_1 \cap H_2} \hat{q}_\alpha(b) = \sum_{a \in H_1} \hat{f}_1(a)\alpha^{\gamma(a)} - \sum_{b \in H_2} \sum_{a \in H_1} M_{a,b} \hat{f}_2(b)\alpha^{\gamma(a)}$$

$$= \sum_{a \in H_1} \left( \hat{f}_1(a) - \sum_{b \in H_2} M_{a,b} \hat{f}_2(b) \right) \cdot \alpha^{\gamma(a)}.$$

Completeness. Suppose that, for all $a \in H_1$, $\hat{f}_1(a) = \sum_{b \in H_2} M_{a,b} \hat{f}_2(b)$. For every $\alpha \in \mathbb{F}$, $h(\alpha) = 0$ and thus $\sum_{b \in H_2} \hat{q}_\alpha(b) = 0$. Completeness of the univariate sumcheck implies that $V_{\text{SUM}}$ always accepts.

Soundness. Suppose that there exists $a \in H_1$ such that $\hat{f}_1(a) \neq \sum_{b \in H_2} M_{a,b} \hat{f}_2(b)$. This implies that $h$ is a nonzero polynomial of degree less than $|H_1|$, and so $\Pr_{\alpha \in \mathbb{F}}[h(\alpha) = 0] < |H_1|/|\mathbb{F}|$. If $h(\alpha) \neq 0$, then $\sum_{b \in H_2} \hat{q}_\alpha(b) \neq 0$ and in this case $V_{\text{SUM}}$ rejects.

Efficiency. Both parties run the univariate sumcheck as a subroutine. In addition, the prover needs to compute $q_\alpha = \hat{q}_\alpha|_L$ (the evaluation of $\hat{q}_\alpha$ over $L$), for example as follows: (i) evaluate $\hat{p}_\alpha^{(1)}$ over $L$ in time $O(s) + \text{FFT}(\mathbb{F}, s) + \text{FFT}(\mathbb{F}, |L|)$; (ii) evaluate $\hat{p}_\alpha^{(2)}$ over $L$ in time $O(\|M\| + s) + \text{FFT}(\mathbb{F}, s) + \text{FFT}(\mathbb{F}, |L|)$; (iii) compute $q_\alpha$ from these components in time $O(|L|)$. The verifier only needs to access $q_\alpha$ at a single point $r \in L$, which can be done in time $O(\|M\| + s)$. □
7 Univariate rowcheck

We describe Univariate Rowcheck, an RS-encoded IOPP for simultaneously testing satisfaction of a given arithmetic constraint on a large number of inputs. The next definition captures this.

**Definition 7.1** (rowcheck relation). The relation $\mathcal{R}_\text{ROW}$ is the set of all pairs $\left( (\mathbb{F}, L, H, \rho, w, c), (f_1, \ldots, f_w) \right)$ where $\mathbb{F}$ is a finite field, $L, H$ are affine subspaces of $\mathbb{F}$, $\rho \in \{0, 1\}$, $w \in \mathbb{N}$, $c : \mathbb{F}^w \to \mathbb{F}$ is an arithmetic circuit, $f_1, \ldots, f_w \in \text{RS}[L, \rho]$, and $\forall a \in H \ c(f_1(a), \ldots, f_w(a)) = 0$.

Standard techniques for testing membership in the vanishing subcode of the Reed–Solomon code directly imply a 1-message RS-encoded IOPP for the above problem [BS08]. Namely, the system of equations $\{c(\hat{f}_1(a), \ldots, \hat{f}_w(a)) = 0\}_{a \in H}$ is equivalent to the statement “there exists $g \in \text{RS}[L, \rho \deg(c) - \frac{|H|}{|F|}]$ such that $\hat{g}(X) \cdot \prod_{a \in H} (X - \alpha) \equiv c(\hat{f}_1(X), \ldots, \hat{f}_w(X))$”. Therefore, the prover can send $g$ to the verifier, who can probabilistically check the identity at a random point of $L$, with a soundness error of $\rho \deg(c)$.

We do not take the above approach, and instead probabilistically reduce the system of equations to a sumcheck instance, via a standard method [BFLS91, §5.2]. The reason is that, in our protocol for R1CS (see Section 8) we already “pay” for one sumcheck protocol execution inside the lincheck protocol (see Section 6), and additional sumcheck instances come essentially for free. We now explain the reduction.

The polynomial $h(X) := \sum_{a \in H} c(\hat{f}_1(a), \ldots, \hat{f}_w(a))X^{\gamma(a)}$ (where $\gamma$ is an ordering of $H$ starting at zero) is a polynomial of degree less than $|H|$ that is the zero polynomial if and only if all equations are satisfied. In particular, if even one equation is not satisfied, $h(\alpha) \neq 0$ with probability at least $1 - |H|/|\mathbb{F}|$ over a random choice of $\alpha \in \mathbb{F}$. Also note that $h(\alpha)$ is of the form $\sum_{a \in H} g_\alpha(a)$ where $H$ is a subspace and, given $f_1, \ldots, f_w$, it is easy to evaluate $g_\alpha$ at any point in $L$. This suggests a protocol: the verifier sends a random $\alpha \in \mathbb{F}$ to the prover, and then the two run univariate sumcheck on the claim “$\sum_{a \in H} g_\alpha(a) = 0$”.

One issue, however, is that $g_\alpha$ is not a low-degree polynomial.

We resolve this as in [BFLS91, §5.2] by summing a different function: $\tilde{g}_\alpha(X) := c(\hat{f}_1(X), \ldots, \hat{f}_w(X)) \cdot p_\alpha(X)$, where $p_\alpha$ is the unique polynomial of degree less than $|H|$ such that $p_\alpha(a) = \alpha^{\gamma(a)}$ for all $a \in H$; note that $\tilde{g}_\alpha$ has degree less than $\rho |L| \cdot \deg(c) + |H|$. Clearly $\tilde{g}_\alpha \neq g_\alpha$ (in particular, $\tilde{g}_\alpha$ is low-degree) but it is easy to see that $\tilde{g}_\alpha$ agrees with $g_\alpha$ on all points in $H$. Thus $h(\alpha) = \sum_{a \in H} \tilde{g}_\alpha(a)$ and, since $\tilde{g}_\alpha$ is also easy to evaluate, we can run univariate sumcheck on this sum instead.

**Theorem 7.2.** Protocol 7.3 below is an RS-encoded IOPP for $\mathcal{R}_\text{ROW}$ (Definition 7.1) with parameters:

<table>
<thead>
<tr>
<th>alphabet</th>
<th>$\Sigma$</th>
<th>$\mathbb{F}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of rounds</td>
<td>$k$</td>
<td>$1$</td>
</tr>
<tr>
<td>proof length</td>
<td>$p$</td>
<td>$2</td>
</tr>
<tr>
<td>randomness</td>
<td>$r$</td>
<td>$\log</td>
</tr>
<tr>
<td>soundness error</td>
<td>$\varepsilon$</td>
<td>$</td>
</tr>
<tr>
<td>prover time</td>
<td>$t_p$</td>
<td>$O(</td>
</tr>
<tr>
<td>verifier time</td>
<td>$t_v$</td>
<td>$O(</td>
</tr>
<tr>
<td>maximum rate</td>
<td>$\rho_{\text{max}}$</td>
<td>$\rho \deg(c) +</td>
</tr>
</tbody>
</table>

**Protocol 7.3 (Univariate Rowcheck).** Denote by $(P_{\text{SUM}}, V_{\text{SUM}})$ the RS-encoded IOPP for univariate sumcheck (Protocol 5.3). The RS-encoded IOPP $(P, V)$ for univariate rowcheck works as follows.

1. $P$ and $V$ agree in advance on an ordering $\gamma : H \to \{0, \ldots, |H| - 1\}$ of $H$.
2. $V$ draws a uniformly random $\alpha \in \mathbb{F}$ and sends it to $P$.

The element $\alpha$ and witness codewords $f_1, \ldots, f_w \in \text{RS}[L, \rho]$ define two polynomials:

- $\hat{p}_\alpha$ is the unique polynomial of degree less than $|H|$ s.t. $\hat{p}_\alpha(a) = \alpha^{\gamma(a)}$ for all $a \in H$;
• \( \hat{q}_\alpha \) is the polynomial \( \hat{q}_\alpha(X) := c(\hat{f}_1(X), \ldots, \hat{f}_w(X)) \cdot \hat{p}_\alpha(X) \).

Observe that \( q_\alpha := \hat{q}_\alpha|_L \in \mathrm{RS}[L, \rho'] \) where \( \rho' := \rho \deg(c) + |H|/|L| \).

3. \( P \) and \( V \) run \( (P_{\mathrm{SUM}}(x', q_\alpha), V_{\mathrm{SUM}}(x')) \) where \( x' := (\mathbb{F}, L, H, \rho', \mu = 0) \). Note that \( V \) can use its oracles \( f_1, \ldots, f_w \) to simulate access to the oracle \( q_\alpha \).

4. \( V \) accepts if and only if \( V_{\mathrm{SUM}} \) accepts.

**Proof.** Completeness and soundness rely on the fact that for every \( \alpha \in \mathbb{F} \) it holds that:

\[
    h(\alpha) := \sum_{a \in H} \hat{q}_\alpha(a) = \sum_{a \in H} c(\hat{f}_0(a), \ldots, \hat{f}_w(a)) \cdot \hat{p}_\alpha(a) = \sum_{a \in H} c(\hat{f}_0(a), \ldots, \hat{f}_w(a)) \cdot \alpha^{\gamma(a)} .
\]

**Completeness.** Suppose that, for all \( a \in H \), \( c(\hat{f}_0(a), \ldots, \hat{f}_w(a)) = 0 \). For every \( \alpha \in \mathbb{F} \), \( h(\alpha) = 0 \) and thus \( \sum_{a \in H} \hat{q}_\alpha(a) = 0 \). Completeness of the univariate sumcheck implies that \( V_{\mathrm{SUM}} \) always accepts.

**Soundness.** Suppose there exists \( a \in H \) such that \( c(\hat{f}_0(a), \ldots, \hat{f}_w(a)) \neq 0 \). This implies that \( h \) is a nonzero polynomial of degree less than \( |H| \), and so \( \Pr_{a \sim \mathbb{F}} \left[ \sum_{a \in H} \hat{q}_\alpha(a) = 0 \right] = \Pr_{a \sim \mathbb{F}} [h(\alpha) = 0] < |H|/|\mathbb{F}| \). If \( h(\alpha) \neq 0 \), then \( \sum_{a \in H} \hat{q}_\alpha(a) \neq 0 \) and in this case \( V_{\mathrm{SUM}} \) rejects with probability 1.

**Efficiency.** Both parties run the univariate sumcheck as a subroutine. In addition, the prover needs to compute \( q_\alpha = \hat{q}_\alpha|_L \) (the evaluation of \( \hat{q}_\alpha \) over \( L \)), for example as follows: (i) compute \( p_\alpha = \hat{p}_\alpha|_L \) (the evaluation of \( \hat{p}_\alpha \) over \( L \)) in time \( O(|H|) + \mathrm{FFT}(\mathbb{F}, |H|) + \mathrm{FFT}(\mathbb{F}, |L|) \); (ii) follow the definition of \( \hat{q}_\alpha \) to compute \( q_\alpha \) from \( p_\alpha \), the arithmetic circuit \( c \), and witnesses \( f_1, \ldots, f_w \), in time \( O(|L| \cdot |c|) \). The verifier only needs to access \( q_\alpha \) at a single point \( r \in L \), which can be done by accessing each of \( f_1, \ldots, f_w \) at \( r \), evaluating \( c \) on the result, and multiplying this latter with \( \hat{p}_\alpha(r) \), all of which takes time \( O(|H| + |c|) \). \( \square \)
8 An RS-encoded IOP for rank-one constraint satisfaction

We describe an RS-encoded IOP for rank-one constraint satisfaction (R1CS). An R1CS instance consists of matrices $A, B, C \in \mathbb{F}^{m \times (n+1)}$ and explicit input $v \in \mathbb{F}^k$, and it is satisfiable if there exists $w \in \mathbb{F}^{n-k}$ such that $Az \cdot Bz = Cz$ where $z = (1, v, w) \in \mathbb{F}^{n+1}$ and $\circ$ denotes entry-wise (Hadamard) product.

**Definition 8.1** (R1CS relation). The relation $\mathcal{R}_{\text{R1CS}}$ is the set of all pairs $((F, k, n, m, A, B, C, v), w)$ where $F$ is a finite field, $k, n, m \in \mathbb{N}$ denote the number of inputs, variables and constraints respectively ($k \leq n$), $A, B, C$ are $m \times (1 + n)$ matrices over $F$, $v \in F^k$, and $w \in F^{n-k}$, such that for all $i \in [m]$ $\left( \sum_{j=0}^{n} A_{i,j}z_j \right) \cdot \left( \sum_{j=0}^{n} B_{i,j}z_j \right) = \left( \sum_{j=0}^{n} C_{i,j}z_j \right)$, where $z := (1, v, w) \in F^{n+1}$.

We describe how to obtain an RS-encoded IOP for R1CS by using RS-encoded IOPPs for rowcheck and lincheck (which we obtained in Sections 6 and 7 respectively).

Let $H_1, H_2$ be subspaces of $F$ such that $|H_1| = m$ and $|H_2| = n+1$, and view $A, B, C$ as matrices in $F^{H_1 \times H_2}$. The prover first sends four oracles: $f_z$ that (purportedly) is the low-degree extension of $z : H_2 \rightarrow F$; and $f_{Az}, f_{Bz}, f_{Cz}$ that (purportedly) are the low-degree extensions of $Az, Bz, Cz : H_1 \rightarrow F$. The verifier uses the lincheck protocol to test that, indeed, $f_{Az}$ is a low-degree extension of $Az$, and likewise for $f_{Bz}, f_{Cz}$. Then the verifier uses the rowcheck protocol to test that $f_{Az}(a) \cdot f_{Bz}(a) = f_{Cz}(a)$ for all $a \in H_1$.

The above protocol almost works, with the one problem being that the prover could cheat by sending $f_z$ that is inconsistent with the explicit input $v$. We remedy this by (roughly) having the prover send the low-degree extension $f_w$ of $w$ instead of $f_z$. The verifier only needs to query one point of $f_w$, which it can do by making one query to $f_w$ and evaluating the low-degree extension of $v$ at one point.

The above protocol uses three linchecks and one rowcheck, each of which is a probabilistic reduction to sumcheck; this means running the sumcheck protocol four times (in parallel). The sumcheck protocol is relatively expensive, so we use the optimization of bundling these four sumcheck instances (see Section 5.2). We also save computation by choosing the same challenge $\alpha$ for each of the linchecks and the rowcheck.

Below we provide details about the foregoing intuition. After that we provide additional subsections that explain how to modify the “basic” protocol to achieve additional goals: in Section 8.1 we describe how to achieve zero knowledge; in Section 8.2 we describe how to amortize the cost of verifying the satisfaction of multiple R1CS instances (sharing the same matrices) at the same time.

**Theorem 8.2.** Protocol 8.3 below is an RS-encoded IOP for $\mathcal{R}_{\text{R1CS}}$ (Definition 8.1) with parameters:

| alphabet $\Sigma$ | $= \mathbb{F}$ |
| number of rounds $k$ | $= 2$ |
| proof length $p$ | $= 6|L|$ |
| randomness $r$ | $= 8 \log |F|$ |
| soundness error $\varepsilon$ | $= \frac{m+1}{|F|}$ |
| prover time $t_p$ | $= O(|L| \cdot \log(n + m) + \|A\| + \|B\| + \|C\|)$ |
| | $+ 7 \cdot \text{FFT}(F, \max(n, m)) + 10 \cdot \text{FFT}(F, |L|)$ |
| verifier time $t_v$ | $= O(\|A\| + \|B\| + \|C\| + n + m)$ |
| maximum rate $\rho_{\text{max}}$ | $= \frac{2\max(n, m+1) + n}{|L|}$ |

for any instance $\pi = (\mathbb{F}, k, n, m, A, B, C, v)$ and any affine subspace $L$ of $\mathbb{F}$.

**Protocol 8.3.** The prover $P$ and verifier $V$ both receive as input an R1CS instance $(\mathbb{F}, k, n, m, A, B, C, v)$, and the prover $P$ also receives as input a corresponding R1CS witness $w$; as above, $z := (1, v, w) \in \mathbb{F}^{n+1}$.

Below, $(P_{\text{LIN}}, V_{\text{LIN}})$ denotes the RS-encoded IOPP for univariate lincheck (Protocol 6.3) and $(P_{\text{ROW}}, V_{\text{ROW}})$ the RS-encoded IOPP for univariate rowcheck (Protocol 7.3).
Let $H_1, H_2$ be two affine subspaces of $F$ with $|H_1| = m$ and $|H_2| = n + 1$ such that $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$; this implies that $H_1 \cap H_2 = H_1 \cup H_2$. (We assume without loss of generality that $m, n + 1$, and $k + 1$ are powers of $\text{char}(F)$.) Let $\gamma : H_1 \cup H_2 \to \{0, \ldots, |H_1 \cup H_2| - 1\}$ be an ordering on $H_1 \cup H_2$ such that $\gamma(H_i) = \{0, \ldots, |H_i| - 1\}$ for $i \in \{1, 2\}$. We view $A, B, C$ as matrices in $F^{H_1 \times H_2}$ via this ordering.

1. **Compute LDE of the input.** Letting $H_2^{\leq k} := \{b \in H_2 : 0 \leq \gamma(b) \leq k\}$, $P$ and $V$ construct $\hat{f}_{(1,v)}(X)$, the unique polynomial of degree less than $|H_2^{\leq k}| = k + 1$ such that, for all $b \in H_2^{\leq k}$,

$$\hat{f}_{(1,v)}(b) = \begin{cases} 1 & \text{if } \gamma(b) = 0, \\ v_i & \text{if } \gamma(b) = i \text{ and } i \in \{1, \ldots, k\}. \end{cases}$$

2. **Witness and auxiliary oracles.** $P$ sends to $V$ the oracle codewords $f_w \in \text{RS}[L, \frac{n-k}{|H_2^{\leq k}|}]$ and $f_{A_2}, f_{B_2}, f_{C_2} \in \text{RS}[L, \frac{n-k}{|H_2^{\leq k}|}]$ defined as follows.

- $f_w := \hat{f}_w|_L$ where $\hat{f}_w$ is the unique polynomial of degree less than $n - k$ such that

$$\forall b \in H_2 \text{ with } k < \gamma(b) \leq n, \quad \hat{f}_w(b) = \frac{w_{\gamma(b) - k} - \hat{f}_{(1,v)}(b)}{Z_{H_2^{\leq k}}(b)}.$$

- $f_{A_2} := \hat{f}_{A_2}|_L$ where $\hat{f}_{A_2}$ is the unique polynomial of degree less than $m$ such that, for all $a \in H_1$,

$$\hat{f}_{A_2}(a) = \sum_{b \in H_2} A_{a,b} \cdot z_{\gamma(b)} = (Az)_a. \quad \text{The other codewords, } f_{B_2} \text{ and } f_{C_2}, \text{ are defined similarly.}$$

The above implicitly define the “virtual oracle” $f_z := \hat{f}_z|_L$ where $\hat{f}_z(X) := \hat{f}_w(X) \cdot Z_{H_2^{\leq k}}(X) + \hat{f}_{(1,v)}(X)$. Note that $\hat{f}_z(b) = z_{\gamma(b)}$ for all $b \in H_2$, and $f_z \in \text{RS}[L, \frac{n+1}{|H_2^{\leq k}|}]$.

3. **Run subprotocols.** Letting $\rho := \max(m, n + 1)/|L|$, $P$ and $V$ run the following in parallel:

- (a) $(P_{\text{LIN}}(z^A_{\text{LIN}}, (f_{A_2}, f_z)), V_{\text{LIN}}^{f_{A_2}, f_z}(z^A_{\text{LIN}}))$ with $z^A_{\text{LIN}} := (F, L, H_1, H_2, \rho, A)$.
- (b) $(P_{\text{LIN}}(z^B_{\text{LIN}}, (f_{B_2}, f_z)), V_{\text{LIN}}^{f_{B_2}, f_z}(z^B_{\text{LIN}}))$ with $z^B_{\text{LIN}} := (F, L, H_1, H_2, \rho, B)$.
- (c) $(P_{\text{LIN}}(z^C_{\text{LIN}}, (f_{C_2}, f_z)), V_{\text{LIN}}^{f_{C_2}, f_z}(z^C_{\text{LIN}}))$ with $z^C_{\text{LIN}} := (F, L, H_1, H_2, \rho, C)$.
- (d) $(P_{\text{ROW}}(z_{\text{ROW}}, (f_{A_2}, f_{B_2}, f_{C_2})), V_{\text{ROW}}^{f_{A_2}, f_{B_2}, f_{C_2}}(z_{\text{ROW}}))$ with $z_{\text{ROW}} := (F, L, H_1, \rho', 3, c)$, $\rho' := \frac{m}{|L|}$,

$$c(X, Y, Z) := XY - Z.$$

4. $V$ accepts if and only if all of the above subverifiers accept.

**Proof.**

**Completeness.** Suppose that $w \in \mathbb{F}^{m-k}$ is a valid witness for the instance $(F, k, n, m, A, B, C, v)$, and define $z := (1, v, w) \in \mathbb{F}^{m+1}$. By construction, $\hat{f}_z$ is a low-degree extension of $z$ over $H_2$ (i.e., $\hat{f}_z(b) = z_{\gamma(b)}$ for all $b \in H_2$). Therefore, for all $a \in H_1$ it holds that $\hat{f}_z(a) = \sum_{b \in H_2} A_{a,b} \cdot z_{\gamma(b)} = (Az)_a$, and so Step 3a (lincheck on $(f_{A_2}, f_z)$) always accepts. By the same argument, Steps 3b and 3c always accept. Finally, the fact that $w$ is a valid witness implies that, for all $a \in H_1$, it holds that $\hat{f}_z(a) \cdot \hat{f}_w(a) - \hat{f}_C(a) = \sum_{b \in H_2} A_{a,b} \cdot z_{\gamma(b)} \cdot \sum_{j=0}^n B_{a,b} \cdot z_{\gamma(b)} - \sum_{j=0}^n C_{a,b} \cdot z_{\gamma(b)} = 0$, which means that Step 3d always accepts.

**Soundness.** Suppose that the instance $(F, k, n, m, A, B, C, v)$ is not satisfiable, i.e., for all $w \in \mathbb{F}^{m-k}$, letting $z := (1, v, w)$, there exists $i \in \{m\}$ such that $(\sum_{j=0}^n A_{i,j} \cdot z_j) \cdot (\sum_{j=0}^n B_{i,j} \cdot z_j) \neq (\sum_{j=0}^n C_{i,j} \cdot z_j)$. Let the oracles sent by a malicious prover be $f_{w}, f_{A}, f_{B}, f_{C}$, and let $f'_z := \hat{f}'_z|_H$ where $\hat{f}'_z(X) := \hat{f}_w(X) \cdot Z_{H_2^{\leq k}}(X) + \hat{f}_{(1,v)}(X)$. We distinguish between multiple cases.

(i) There exists $a \in H_1$ for which $\hat{f}'_z(a) \neq \sum_{b \in H_2} A_{a,b} \cdot \hat{f}_z(b)$. Then, by soundness of the lincheck protocol, Step 3a accepts with probability $\frac{|H_1|}{|H_2^{\leq k}|}$. 


(ii) If analogous statements hold for $f_B'$ or $f_C'$, then analogous conclusions hold for Step 3b or Step 3c.
(iii) For all $a \in H_1$ it holds that $f_A'(a) = \sum_{b \in H_2} A_{a,b} f_Z'(b)$, and likewise for $f_B', f_C'$. Then by assumption there exists $a \in H_1$ such that $c(f_A'(a), f_B'(a), f_C'(a)) = f_A'(a) \cdot f_B'(a) - f_C'(a) \neq 0$. We conclude that, by soundness of the rowcheck protocol, Step 3d accepts with probability $\frac{|H_1|}{|F|}$.

The soundness error is given by maximizing over the above cases.

**Optimization: one sumcheck suffices.** We can view both the rowcheck and linecheck protocols as probabilistic interactive reductions to sumcheck. In particular:
- Given an instance-witness pair $(\langle F, L, H, \rho, w, c \rangle, (f_1, \ldots, f_w))$, the rowcheck protocol outputs an instance-witness pair $(\langle F, L, H, \rho \deg(c) + \frac{|H|}{|L|}, 0 \rangle, q_\alpha)$ for sumcheck.
- Given an instance-witness pair $(\langle F, L, H_1, H_2, \rho, M \rangle, (f_1, f_2))$, the linecheck protocol outputs an instance-witness pair $(\langle F, L, H_1 \cup H_2, \rho + \frac{|H_1\cup H_2|}{|L|}, 0 \rangle, q_\alpha)$ for sumcheck.

We can save costs across these four sumcheck instances by via one execution of our amortized sumcheck protocol (see Lemma 5.10 in Section 5.2), which yields the parameters in the theorem statement. Note that the amortized sumcheck protocol relies on all summations being taken over the same space; the reductions yield sumchecks over $H_1$ and $H_1 \cup H_2$. If $H_2 \not\subseteq H_1$, then these are already the same; if $H_1 \not\subseteq H_2$, then we can define $\tilde{p}_\alpha$ in the rowcheck protocol so that it is zero on $H_2 \setminus H_1$, and sum over $H_1 \cup H_2$.

**Optimization: re-use $\alpha$.** The rowcheck and linecheck protocols instruct the verifier to sample a uniformly random $\alpha \in F$ and send it to the prover. Naively, the verifier would choose $\alpha_1, \ldots, \alpha_4 \in F$ uniformly and independently, and send $(\alpha_1, \ldots, \alpha_4)$ to the prover. However, this means that the verifier must compute $\tilde{p}_\alpha_i$ for each $i$. We observe that choosing $\alpha_1 = \ldots = \alpha_4 \in F$ uniformly at random does not affect our soundness analysis, which means that the verifier only has to compute $\tilde{p}_\alpha$ for one $\alpha$. This will become more important later in Section 8.2 when we consider amortizing multiple instances.

**Efficiency.** The prover computes $A_z, B_z, C_z$ and their low-degree extensions, along with the low-degree extensions of $w$ and $z$, in time $O(\|A\| + \|B\| + \|C\| + n + m) + 5 \cdot \text{FFT}(F, |L|)$. The verifier evaluates $f_z$ at a single point in $L$, which costs $O(n + m)$. Summing over the costs of the subprotocols and amortizing the sumcheck cost across the four instances yields the stated expressions.

### 8.1 Zero knowledge

We describe how to modify Protocol 8.3 to achieve zero knowledge against bounded-query malicious verifiers; the modification is an adaptation of algebraic techniques from [BCGV16; BCFGRS17]. Essentially, instead of providing the unique low-degree extensions of $w, A_z, B_z, C_z$, the prover provides randomized low-degree extensions that are over a domain $L \subseteq F$ chosen such that $(H_1 \cup H_2) \cap L = \emptyset$ (in particular, $L$ will be affine so that $0_F \not\in L$). This ensures that a bounded number of queries to the witness and auxiliary oracles does not reveal any information about $w$. Then, both prover and verifier use our zero knowledge sumcheck protocol (see Protocol 5.8 in Section 5.1) instead of the “plain” sumcheck protocol used above.

**Theorem 8.4.** For any $b: \mathbb{N} \to \mathbb{N}$, Protocol 8.5 below is an RS-encoded IOP for $\mathcal{R}_{R1CS}$ (Definition 8.1) that
is zero knowledge against query bound $b$ with parameters:

| alphabet | $\Sigma = \mathbb{F}$ |
| number of rounds | $k = 2$ |
| proof length | $p = |L|$ |
| randomness | $r = 8 \log |\mathbb{F}|$ |
| soundness error | $\varepsilon = \frac{m+1}{|\mathbb{F}|}$ |
| prover time | $t_p = O(|L| \cdot \log(n+m) + \|A\| + \|B\| + \|C\| + 7 \cdot \text{FFT}(\mathbb{F}, \max(m,n)) + 11 \cdot \text{FFT}(\mathbb{F}, |L|))$ |
| verifier time | $t_v = O(|A| + \|B\| + \|C\| + n + m)$ |
| maximum rate | $\rho_{\text{max}} = \frac{2 \max(m,n+1)+m+8}{|L|}$ |

for any instance $\pi = (\mathbb{F}, k, n, m, A, B, C, v)$.

**Protocol 8.5** (ZK variant of Protocol 8.3). We use the same notation as in Protocol 8.3, with the only additional constraint that $(H_1 \cup H_2) \cap L = \emptyset$.

1. **Compute LDE of the input.** Same as Step 1 in Protocol 8.3.
2. **Witness and auxiliary oracles.** $P$ sends to $V$ the oracle codewords $f_w \in \text{RS}[L, \frac{n-k+b}{|L|}]$ and $f_{Az}, f_{Bz}, f_{Cz} \in \text{RS}[L, \frac{m+b}{|L|}]$ defined as follows.
   - $f_w := \hat{f}_w|_L$ where $\hat{f}_w$ is a random polynomial of degree less than $n-k+b$ such that
     \[ \forall b \in H_2 \text{ with } k < \gamma(b) \leq n, \quad \hat{f}_w(b) = \frac{w_\gamma(b) - \hat{f}_{1,v}(b)}{\mathbb{Z}_{H_2}^\gamma(b)}. \]
   - $f_{Az} := \hat{f}_{Az}|_L$ where $\hat{f}_{Az}$ is a random polynomial of degree less than $m+b$ such that, for all $a \in H_1$,
     \[ \hat{f}_{Az}(a) = \sum_{b \in H_2} A_{a,b} \cdot z(b) = \bar{A}_{a}. \]
     The other codewords, $f_{Bz}$ and $f_{Cz}$, are defined similarly.
   - As before, the above implicitly define the “virtual oracle” $f_z := \hat{f}_z|_L$ where $\hat{f}_z(X) := \hat{f}_w(X) \cdot \mathbb{Z}_{H_2}^\gamma(X) + \hat{f}_{1,v}(X)$, $\hat{f}_z(b) = z(b)$ for all $b \in H_2$, but now $f_z \in \text{RS}[L, \frac{n-k+b}{|L|}]$ since $\hat{f}_w$ has higher degree.
3. **Run subprotocols.** The same as Step 3 in Protocol 8.3, except that $P$ and $V$ run the (amortized) zero knowledge sumcheck protocol (see Protocol 5.8 in Section 5.1).
4. $V$ accepts if and only if all of the above subverifiers accept.

**Proof.** Completeness and soundness follow almost directly from the proof of Theorem 8.2, so we do not discuss them. Before discussing zero knowledge, we note that the round complexity can be reduced to 2 by running the first round of the zero knowledge sumcheck protocol (Protocol 5.8) in parallel with the first round of Protocol 8.5. We now argue the zero knowledge guarantee (see Definition 4.4): we need to construct a probabilistic simulator $S$ that, given as input a satisfiable RICS instance $(\mathbb{F}, k, n, m, A, B, C, v)$ and straightline access to a $b$-query malicious verifier $\hat{V}$, outputs a view that is identically distributed as $\hat{V}$’s view when interacting with an honest prover.

At a high level, $S$ simulates the oracles $f_w, f_{Az}, f_{Bz}, f_{Cz}$ by answering each query with uniformly random field elements. Given these, it runs the simulator for the amortized zero knowledge sumcheck, answering the subverifiers’ queries to the virtual oracle by “querying” the appropriate locations of $f_w, f_{Az}, f_{Bz}, f_{Cz}$. More precisely, on input $\pi$, the simulator operates as follows.

1. Prepare a table $T$ for $(f_{Az}, f_{Bz}, f_{Cz}, f_w)$ which is initially empty. Whenever we “query” an oracle $f_z$ at a point $a \in L$, where $x$ is one of $Az, Bz, Cz, w$, if $(a, b_{Az}, b_{Bz}, b_{Cz}, b_w) \in T$ then output $b_x$; otherwise, choose $b_{Az}, b_{Bz}, b_{Cz}, b_w \in \mathbb{F}$ uniformly at random, add $(a, b_{Az}, b_{Bz}, b_{Cz}, b_w)$ to $T$ and output $b_x$. 31
2. “Send” the oracles $f_{A_2}, f_{B_2}, f_{C_2}, f_w$ to $\tilde{V}$. In parallel, “send” the first prover message $r$ in the univariate ZK sumcheck protocol (Protocol 5.8), and use the simulator for that protocol to answer queries to $r$.
3. Run the prover for each subprotocol in Step 3, except that we do not explicitly construct any witness (we think of them as arithmetic circuits with oracle gates), and we do not run the sumcheck protocol.
4. Pass the instances constructed in the previous step to the simulator for the zero knowledge 4-sumcheck protocol. When the subsimulator queries one of the oracles, evaluate the corresponding circuit and use $T$ to look up the necessary values of $f_{A_2}, f_{B_2}, f_{C_2}, f_w$.

The subsimulator makes the same number of queries to the “amortized” virtual oracle as the verifier makes to the sumcheck proof oracle; in particular, this is at most $b$. By inspecting the virtual oracle, we see that the answer to a query $a \in \mathbb{F}$ depends only on $f_{A_2}(a), f_{B_2}(a), f_{C_2}(a), f_w(a)$ (and $f_v(a)$, which is known). Hence $|T| \leq b$. It remains to show that $T$ is consistent with the real view.

We look at $f_w$; the other cases are essentially identical. We can write $f_w$ as

$$f_w := \hat{f}_w(X) + \mathbb{Z}_{H_2^{\leq k}}(X) \cdot R(X),$$

where $R(X)$ is a uniformly random polynomial of degree less than $b$ and $H_2^{\leq k} := H_2 \setminus H_2^{>k}$. Since $\mathbb{Z}_{H_2^{>k}}$ is nonzero outside of $H_2^{>k}$, any vector $(\hat{f}_w(a))_{a \in Q}$ is distributed uniformly in $\mathbb{F}^Q$ for any $Q \subseteq \mathbb{F}$ such that $Q \cap H_2 = \emptyset$ and $|Q| \leq b$. In particular, this holds for the set of query positions asked by the subsimulator, even if they are chosen adaptively based on the answers to previous queries.

### 8.2 Amortization

We describe efficiency savings that can be made when one considers multiple R1CS instances with the same constraints (but different inputs). More precisely, we seek an RS-encoded IOP for the following relation:

**Definition 8.6** ($\ell$-wise R1CS relation). The relation $\mathcal{R}_{\text{R1CS}}^{\ell}$ is the set of all pairs $((x_1, \ldots, x_{\ell}), (w_1, \ldots, w_{\ell}))$ such that, for every $i \in \{1, \ldots, \ell\}$, $x_i = (\mathbb{F}, k, n, m, A, B, C, v^{(i)})$ and $(x_i, w_i) \in \mathcal{R}_{\text{R1CS}}$.

We have already obtained an RS-encoded IOP for $\mathcal{R}_{\text{R1CS}}$ (Protocol 8.3), so we can obtain an RS-encoded IOP for $\mathcal{R}_{\text{R1CS}}^{\ell}$ by running this IOP in parallel $\ell$ times. Note, however, that the running time of both the prover and the verifier increases by a multiplicative factor of $\ell$.

We modify this strategy to ensure that the verifier’s running time increases by only an additive factor in $\ell$, for a total of $O(|A| + |B| + |C| + n + m + \ell)$. This is significant because, as $\ell$ increases, the amortized per-instance cost becomes constant. The modification follows from an observation used in the proof of Theorem 8.2: we choose the same random $\alpha \in \mathbb{F}$ for all rowcheck and lincheck instances that result from the $\ell$ parallel executions, and then amortize all of the resulting sumcheck instances (see Section 5.2). The verifier then only has to evaluate the auxiliary lincheck and rowcheck polynomials once.

**Corollary 8.7.** For every $\ell \in \mathbb{N}$ there exists an RS-encoded IOP for $\mathcal{R}_{\text{R1CS}}^{\ell}$ (Definition 8.6) with parameters:

| alphabet | $\Sigma$ | $\mathbb{F}$ |
| number of rounds | $k$ | 2 |
| proof length | $\ell \cdot 6|L|$ |
| randomness | $r$ | $(\ell + 1) \cdot \log |\mathbb{F}|$ |
| soundness error | $\varepsilon$ | $\frac{m+1}{|L|}$ |
| prover time | $t_p$ | $\ell \cdot O(|L| \cdot \log(n + m) + |A| + |B| + |C|)$ + $7 \cdot \text{FFT}(\mathbb{F}, \max(n, m)) + 10 \cdot \text{FFT}(\mathbb{F}, |L|)$ |
| verifier time | $t_v$ | $O(|A| + |B| + |C| + n + m + \ell)$ |
| maximum rate | $\rho_{\text{max}}$ | $\frac{2\max(m, n+1)+m}{|L|}$ |
9 From RS-encoded provers to arbitrary provers

In prior sections we have designed IOP protocols based on the simplifying assumption that a malicious prover is restricted to sending Reed–Solomon codewords of prescribed rates. In this section we describe how to transform any IOP protocol that is sound under this assumption into one that is sound against all provers.

This by itself should not be surprising: the probabilistic checking literature is rich with such transformations, which are enabled by the tools of low-degree testing and self-correction. However, our goal here is to obtain a transformation that is particularly efficient for the setting of this paper, as we now explain.

There is a straightforward approach to using low-degree testing, which we now spell out since it serves as a comparison point. Suppose that we have a low-degree test for RS $[L, \rho]$ with soundness error $\varepsilon_{LDT}$ and proximity parameter $\delta_{LDT}$, and we wish to transform a given RS-encoded IOP $(P, V, (\tilde{\rho}_i)_{i=1}^k)$ into a corresponding IOP that is sound against all provers. Let us assume for simplicity that $\tilde{\rho}_1 = \cdots = \tilde{\rho}_k = (\rho)$ for some $\rho \in (0, 1]$, that is, each prover message consists of one codeword in RS $[L, \rho]$.

The naive approach is to individually run the low-degree test on each prover message. If all tests pass with probability greater than $\varepsilon_{LDT}$, then every message $\tilde{\pi}_i$ is $\delta_{LDT}$-close to some codeword $\pi_i \in$ RS $[L, \rho]$. If the verifier makes $q$ uniform queries, the probability that any one of these queries does not “see” $(\pi_i)_{i=1}^k$ is at most $q \cdot \delta_{LDT}$. Conditioned on the verifier “seeing” $(\pi_i)_{i=1}^k$, the verifier’s acceptance probability is exactly the same as in the RS-encoded protocol.

While the foregoing approach “works”, it has two inefficiencies. First, it runs one low-degree test for each purported codeword, which is undesirable because low-degree tests are expensive. Second, the soundness error of the RS-encoded IOP typically decreases by increasing $q$, which creates a trade-off with the soundness error $q \cdot \delta_{LDT}$ of the transformation.

We address the first problem by testing a random linear combination of the $\pi_i$, following an idea introduced in [RVW13] (in the context of interactive proofs of proximity) and applied in [AHIV17] (in the context of interactive PCPs of proximity). The verifier samples $a_1, \ldots, a_k \in \mathbb{F}$ uniformly and independently at random, and sends these to the prover; the prover and verifier then engage in a low-degree test for the “virtual oracle” $\tilde{\pi} := \sum_{i=1}^k a_i \tilde{\pi}_i$. If $\tilde{\pi}_i \in$ RS $[L, \rho]$ for all $i$, then $\tilde{\pi} \in$ RS $[L, \rho]$. If instead $\tilde{\pi}_i$ is $\delta$-far from RS $[L, \rho]$ for some $i$ (and $\delta$ small enough), then one can show that $\tilde{\pi}$ is also $\delta$-far with high probability. Thus, a single low-degree test is run, regardless of the number of oracles $k$.

We address the second problem by using an observation due to [BBHR18a] about testing rational constraints (see Section 4.7). In the encoded protocols in this work (and in [BBHR18a]), soundness entails testing both that the prover’s messages are low-degree and that they satisfy some existentially-quantified polynomial equations; for example, “message $f$ is low-degree and there is a low-degree $g$ such that $f \equiv g \cdot Z_H$”. The standard way to test this property is for the prover to send $g$; the verifier can then check the relation by querying at a uniformly random point in the domain, but this creates the aforementioned trade-off. However, [BBHR18a] observe that the verifier can simulate queries to $g$ itself, given query access to $f$, since $g(\alpha) = f(\alpha) / Z_H(\alpha)$ (when $Z_H(\alpha) \neq 0$). Thus the prover does not have to send $g$, but only has to show that $g$ is low-degree. In all of our protocols, this observation results in RS-encoded IOPs with $q = 0$, and we will assume that this is the case in the transformation described in this section.

In this exposition we have made the simplifying assumption that the desired rate for each codeword in each proof is the same. In our protocols (in particular, the sumcheck protocol) this will not be the case, and so we must also handle differing rates. In some settings it suffices to test for proximity to $\text{RS} [L, \max_i (\rho_i)]^k$, but not in our setting. This is because the soundness of univariate sumcheck relies on $g$ being close to $\text{RS} [L, (|I| - 1)/|L|]$; soundness breaks if $g$ is merely close to (say) RS $[L, |I|/|L|]$ (or RS codes with bigger rates). Following [BS08], we instead multiply each $\tilde{\pi}_i$ by an appropriately-chosen random polynomial...
and then take a linear combination. We show that if \( \tilde{\Pi} \) is \( \delta \)-far from RS \( [L, (\rho_1, \ldots, \rho_k)] \) then with high probability \( \Delta(\tilde{\Pi}, RS [L, \vec{\sigma}]) > \delta \); then we show that given this, the verifier’s acceptance probability is bounded as required.

**Theorem 9.1.** Suppose that we are given:

- an RS-encoded IOP \((P_R, V_R, (\tilde{\rho}_i)_{i=1}^k)\) for a relation \( R \);
- an IOPP \((P_{LDT}, V_{LDT})\) for the RS code RS \([L, \rho]\) for \( \rho := \rho_{\max} \).

Then we can combine these two ingredients to obtain an IOP \((P, V)\) for \( R \) with the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet</td>
<td>( \Sigma = \Sigma^R )</td>
</tr>
<tr>
<td>number of rounds</td>
<td>( k = k^R + k^{LDT} )</td>
</tr>
<tr>
<td>proof length</td>
<td>( \rho = \rho^R + \rho^{LDT} )</td>
</tr>
<tr>
<td>query complexity</td>
<td>( q_\pi = q^LDT \cdot \sum_{i=1}^k \ell_i )</td>
</tr>
<tr>
<td>randomness</td>
<td>((r_i, r_q) = (r_i^{LDT} + (\sum_{i=1}^k \ell_i) + c) \log</td>
</tr>
<tr>
<td>soundness error</td>
<td>((\varepsilon_i, \varepsilon_q) = (\varepsilon_i^{LDT} + \frac{\ell^{LDT}</td>
</tr>
<tr>
<td>prover time</td>
<td>( t_P = O(t_P^R + t_P^{LDT}) )</td>
</tr>
<tr>
<td>verifier time</td>
<td>( t_V = O(t_V^R + t_V^{LDT}) )</td>
</tr>
</tbody>
</table>

provided \( \delta^{LDT} < \min(\frac{1-\varepsilon_i}{2}, \frac{1-\rho}{2}) \), and where \( c \) is the maximum size of a constraint set output by \( V_{RS} \). (Parameters with superscript \( \text{“}R\text{”} \) and \( \text{“}LDT\text{”} \) denote parameters for \((P_R, V_R)\) and \((P_{LDT}, V_{LDT})\) respectively.)

Recall that \( \ell_i^{R} \) is the height of the \( i \)-th prover message, i.e., the \( i \)-th prover message has alphabet \( F^{\ell_i^{R}} \).

**Protocol 9.2.** Letting \((P_R, V_R)\) and \((P_{LDT}, V_{LDT})\) be as in the theorem statement, we need to construct an IOP \((P, V)\) for \( R \). The prover \( P \) and verifier \( V \) both receive as input an instance \( z \), and the prover \( P \) also receives as input a corresponding witness \( w \).

1. **RS-encoded IOP for \( R \).** \( P \) and \( V \) simulate \((P_R(z, w), V_R(z))\). During this protocol, the prover sends oracle codewords \( \pi_1 \in RS [L, \tilde{\rho}_1], \ldots, \pi_k \in RS [L, \tilde{\rho}_k] \), and the verifier outputs a set of rational constraints \( C \). Let \( \ell := \sum_{i=1}^k \ell_i + |C|, \tilde{\rho} = (\tilde{\rho}_1, \ldots, \tilde{\rho}_k), \tilde{\sigma} := (\sigma)_{(C, \sigma) \in C}, \) and \( \tilde{\sigma} := (\tilde{\rho}, \tilde{\sigma}) \in (0, 1)^\ell \).

2. **Random linear combination.** \( V \) samples \( \tilde{z} \in F^{2\ell} \) uniformly at random and sends it to \( P \).

3. **Low-degree test.** \( P \) and \( V \) simulate \((P_{LDT}(\tilde{z}^T, \Pi), V_{LDT}\tilde{z}^T, \Pi))\) where \( \Pi := [\Pi_0] \in F^{2\ell \times L} \) is as follows:

   - \( \Pi_0 \in F^{\ell \times L} \) is the matrix obtained by “stacking” vertically the matrices \( \pi_1, \ldots, \pi_k \), and \( \Pi_0 \) is obtained by stacking \( \Pi_0 \) with \( (C[\Pi_0])_{(C, \sigma) \in C} \).

   - \( \Pi_1 \in F^{\ell \times L} \) is the matrix whose entries are \((\Pi_1)_{i,a} := a^{\ell + \sigma_i |L|} \cdot (\Pi_0)_{i,a} \) for all \( i \in \{1, \ldots, \ell\}, \) \( a \in L \).

4. \( V \) accepts if and only if \( V_{LDT} \) accepts.

**Proof.**

**Completeness.** If \( \pi_i \in RS [L, \tilde{\rho}] \) for all \( i \), then \( \tilde{z}^T \Pi \in RS [L, \rho] \) and thus \( P_{LDT} \) makes \( V_{LDT} \) accept. Completeness then follows immediately from the completeness of the RS-encoded IOP \((P_R, V_R)\).

**Soundness.** Suppose that \( z \notin L(R) \) and fix a malicious prover; let \( \delta := \delta^{LDT} \). During the protocol, the prover sends oracles \( \tilde{\pi}_1, \ldots, \tilde{\pi}_k \); let \( \tilde{P} := [\tilde{P}_i] \) be as in the protocol description but with respect to the messages \( \tilde{\pi}_i \). We argue that the verifier accepts with probability at most \( \max(\varepsilon_i^{LDT} + \frac{\delta |L|}{|F|}, 1) \). To do this, we first show that it must hold that \( \Delta(\tilde{\Pi}_0, RS [L, \tilde{\sigma}]) > \delta \); then we show that given this, the verifier’s acceptance probability is bounded as required.
Let $E$ be the event that the verifier accepts in the query phase with probability greater than $\varepsilon_{q,LDT}^\rho$, given the transcript of the interactive phase. Observe that

$$
\Pr[E] = \Pr[E \mid \Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta] \cdot \Pr[\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta] \\
+ \Pr[E \mid \Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) \leq \delta] \cdot \Pr[\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) \leq \delta] \\
\leq \Pr[E \mid \Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta] + \Pr[\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) \leq \delta].
$$

We bound each of these terms individually.

- **The probability of $E$ when $\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta$.** First we argue that if $\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta$ then $\Delta(\hat{\Pi}_0, RS[L, \rho])^{2\delta} > \delta$; then we cite a claim stating that, given this, a random linear combination of $\hat{\Pi}$ is $\delta$-far from $RS[L, \rho]$ with high probability; finally we derive the bound of the aforementioned probability.

**Claim 9.3.** For any $\delta < (1 - 2\rho)/2$, if $\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta$ then $\Delta(\hat{\Pi}_0, RS[L, \rho])^{2\delta} > \delta$.

**Proof.** Suppose by way of contradiction that $\Delta(\hat{\Pi}, RS[L, \rho])^{2\delta} \leq \delta$, and let $\hat{\Pi} = [\hat{\Pi}^0_0]_{i=1}^\ell \in RS[L, \rho]^\ell$ be such that $\Delta(\hat{\Pi}, \Pi) \leq \delta$. For each $i$, let $p_i, p'_i \in RS[L, \rho]$ be the $i$-th rows of $\hat{\Pi}_0, \hat{\Pi}_1$ respectively. We argue that $p_i \in RS[L, \sigma_i]$ for every $i$, which implies $\Pi_0 \in RS[L, \bar{\sigma}]$, so $\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) \leq \Delta(\hat{\Pi}_0, \Pi_0) \leq \Delta(\hat{\Pi}, \Pi) \leq \delta$, which is a contradiction.

Suppose towards contradiction that there exists $i$ such that $p_i \in RS[L, \rho] \setminus RS[L, \sigma_i]$. Then $q := p_i \cdot X(\rho - \sigma_i)^{|L|} \in RS[L, 2\rho - \sigma_i] \setminus RS[L, \rho];$ in particular, $q \neq p'_i$, which implies that $\Delta(q, p'_i) \geq 1 - (2\rho - \sigma_i)$. However, because $\Delta(\hat{\Pi}, RS[L, \rho])^{2\delta} \leq \delta$, we have that, letting $\hat{p}_i$ be the $i$-th row of $\hat{\Pi}_0$, $\Delta(\hat{p}_i \cdot X(\rho - \sigma_i)^{|L|}, q) = \Delta(\hat{p}_i, p_i) \leq \delta$ and $\Delta(\hat{p}_i \cdot X(\rho - \sigma_i)^{|L|}, p'_i) \leq \delta$. By the triangle inequality we have that $\Delta(q, p'_i) \leq 2\delta < 1 - (2\rho - \sigma_i)$, which is a contradiction. \hfill \qed

**Claim 9.4.** Let $\rho > (1 - \rho)/4$, if $\Delta(\hat{\Pi}, RS[L, \rho])^{2\delta} > \delta$ then

$$
\Pr_{\varepsilon_q \sim P} \left[ \Delta(\hat{\varepsilon}_T^T \hat{\Pi}, RS[L, \rho]) > \delta \right] > 1 - (\delta |L| + 1)/|F|.
$$

(For some choices of rate parameter $\rho$, a recent result of [BKS18] yields a stronger statement.)

Combining the two claims: if $\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta$ then, with probability at least $1 - (\delta |L| + 1)/|F|$, $\Delta(\hat{\varepsilon}_T^T \hat{\Pi}, RS[L, \rho]) > \delta = \delta^{LDT}$. Since $\varepsilon'_q \geq \varepsilon_q^{LDT}$, $\Pr[E \mid \Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) > \delta] \leq \varepsilon_q^{LDT} + \frac{|L| + 1}{\rho |F|}$.

- **The probability that $\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) \leq \delta$.** Let $\pi_1 \in RS[L, \bar{\rho}_1], \ldots, \pi_k \in RS[L, \bar{\rho}_k]$ be the closest codewords to the prover’s messages $\bar{\pi}_1, \ldots, \bar{\pi}_k$. We can construct a prover $\hat{P}$ for the encoded IOP, which sends messages $\pi_1, \ldots, \pi_k$. We show that if $\Delta(\hat{\Pi}_0, RS[L, \bar{\sigma}]) \leq \delta$, then for all $(C, \sigma) \in C$ it holds that $C[\hat{\Pi}_0] \in RS[L, \sigma]$. By the soundness of the encoded IOP, this occurs with probability at most $\varepsilon_{RK}^\rho$.

Take some $(C, \sigma) \in C$, and let $\pi_C \in RS[L, \sigma]$ be the (unique) closest codeword to $C[\hat{\Pi}_0]$. By assumption, we have that $\Delta(\pi_C, C[\hat{\Pi}_0]) \leq \delta$. Let $(N(D), C) := C$; then $\Delta(\pi_C \cdot D, N[\Pi_0]) \leq \delta$. Since $\pi_C \cdot D[\Pi_0] \in RS[L, \sigma + \deg(D)]$ and $N[\Pi_0] \in RS[L, D_N(\bar{\rho})]$, we have that $\pi_C \cdot D \equiv N[\Pi_0]$ since $\delta < 1 - \rho_{max}^{N} \leq 1 - \max(\sigma + \deg(D), D_N(\bar{\rho}))$. In particular, this implies that $D$ divides $N[\Pi_0]$ as a polynomial, and so $C[\hat{\Pi}_0] \in RS[L, D_N(\bar{\rho}) - \deg(D)]$; thus $C[\hat{\Pi}_0] = \pi_C \in RS[L, \sigma]$.

\footnote{Note that $\bot \notin C[\hat{\Pi}_0]$ because otherwise the completeness condition of the RS-encoded IOP would fail to hold.}
9.1 Zero knowledge

We describe how to modify the transformation above to preserve zero knowledge, thereby showing how to efficiently convert an RS-encoded IOP with a zero knowledge guarantee into a corresponding IOP with the same zero knowledge guarantee. The transformation uses the random self-reducibility of Reed–Solomon proximity testing, which implies that the low-degree test used in the transformation need not be zero knowledge (the only requirement is that its honest prover must run in polynomial time). In particular, the honest prover in the new protocol will send, in addition to the messages of the underlying RS-encoded IOP, a random codeword \( r \), which is added to the linear combination of messages that are tested for proximity to RS.

**Theorem 9.5.** Suppose that we are given:
- an RS-encoded IOP \( (P_R, V_R, (\tilde{p}_i)_{i=1}^{k_R}) \) for a relation \( \mathcal{R} \) that is zero knowledge against \( b \) queries;
- an IOPP \( (P_{LDT}, V_{LDT}) \) for the RS code \( \text{RS}[L, \rho] \) with \( \rho := \rho_{\max} \) and a polynomial-time honest prover (not necessarily zero knowledge).

Then we can combine these two ingredients to obtain an IOP \( (P, V) \) for \( \mathcal{R} \), also zero knowledge against \( b \) queries, with the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet</td>
<td>( \Sigma = \Sigma^R )</td>
</tr>
<tr>
<td>number of rounds</td>
<td>( k = k^R + k^{LDT} + 1 )</td>
</tr>
<tr>
<td>proof length</td>
<td>( p = p^R + p^{LDT} +</td>
</tr>
<tr>
<td>query complexity</td>
<td>( q_\pi = q_\pi^LDT + q_\pi^LDT \sum_{i=1}^{k} \ell_i^R )</td>
</tr>
<tr>
<td>randomness</td>
<td>( (r_\ell, r_\ell) = (r^R_\ell + r_\ell^{LDT} + (\sum_{i=1}^{k} \ell_i^R + c) \log</td>
</tr>
<tr>
<td>soundness error</td>
<td>( (\varepsilon_1, \varepsilon_q) = (\varepsilon^R_\ell + \varepsilon_\ell^{LDT} + \frac{\delta^{LDT}L+1}{</td>
</tr>
<tr>
<td>prover time</td>
<td>( t_p = O(t_p^R + t_p^{LDT}) )</td>
</tr>
<tr>
<td>verifier time</td>
<td>( t_V = O(t_V^R + t_V^{LDT}) )</td>
</tr>
</tbody>
</table>

provided \( \delta^{LDT} < \min(\frac{1}{2}, \frac{1-r}{4}) \), and where \( c \) is the maximum size of a constraint set output by \( V_R \).

(Parameters with superscript “\( \mathcal{R} \)” and “\( \text{LDT} \)” denote parameters for \( (P_R, V_R) \) and \( (P_{LDT}, V_{LDT}) \) respectively; highlights denote parameter differences with Theorem 9.1.)

**Protocol 9.6.** Letting \( (P_R, V_R) \) and \( (P_{LDT}, V_{LDT}) \) be as in the theorem statement, we need to construct an IOP \( (P, V) \) for \( \mathcal{R} \). The prover \( P \) and verifier \( V \) both receive as input an instance \( x \), and the prover \( P \) also receives as input a corresponding witness \( w \).

1. **RS-encoded IOP for \( \mathcal{R} \).** \( P \) and \( V \) simulate \( (P_R(x, w), V_R(x)) \). In the course of this protocol, the prover sends oracle codewords \( \pi_1 \in \text{RS}[L, \tilde{p}_1], \ldots, \pi_k \in \text{RS}[L, \tilde{p}_k] \), and the verifier specifies a set of rational constraints \( C \). Let \( \ell := \sum_{i=1}^{k} \ell_i + |C| \).

2. **Random linear combination.** \( V \) samples \( \tilde{z} \in \mathbb{F}_2^\ell \) uniformly at random and sends it to \( P \).

3. **Low-degree test.** \( P \) and \( V \) simulate \( (P_{LDT}(\tilde{z}^T \Pi + r), V_{LDT}^{\tilde{z}^T \Pi + r}) \) where \( \Pi := [\Pi_i] \in \mathbb{F}_2^{2\ell \times L} \) is defined as in Protocol 9.2.

4. \( V \) accepts if only if \( V_{LDT} \) accepts.

**Proof.** Completeness and soundness follow almost immediately from those of Protocol 9.2. Indeed, we can view Protocol 9.6 as Protocol 9.2 modified so that \( (P_R, V_R) \) begins with an additional “dummy” round where the prover just sends a random codeword. (Note that we can fix \( \tilde{z} \)'s random coefficient for \( r \) to be 1 almost without loss of generality since distance to Reed–Solomon codewords is preserved under multiplication by a nonzero constant.) We now focus on arguing the zero knowledge property.
Let $S_R$ be the simulator for $(P_R, V_R)$, witnessing zero knowledge against $b$ queries. The simulation guarantee for $S_R$ is that, for any $V_R$ that makes at most $b$ distinct queries across all oracles, View$(P_R(z, w), V_R)$ and the output of $S_R^{V_R}(x)$ are identically distributed.

Consider the simulator $S$ for $(P, V)$ that, given a malicious verifier $\tilde{V}$, constructs a new malicious verifier $\tilde{V}_R$ (defined below), then runs $S_R$ on $\tilde{V}_R$, and finally outputs what $\tilde{V}_R$ outputs given its simulated view.

1. Start running $\tilde{V}$.
2. Sample $r_{\text{sim}} \in \text{RS } [L, \rho]$ uniformly at random, and answer $\tilde{V}$’s queries to $r$ with $r_{\text{sim}}$; let $Q_{\text{sim}}$ be the verifier’s queries to $r$ in this phase.
3. For $k_R$ rounds, forward $\tilde{V}$’s messages to the prover. Answer all of $\tilde{V}$’s queries to the received oracles honestly. Receive a set of rational constraints $C$ from $\tilde{V}$.
4. Receive $\tilde{z} \in \mathbb{R}^{2\ell}$ from $\tilde{V}$.
5. For every $q \in Q_{\text{sim}}$, query every oracle received at $q$. For each oracle defined by a rational constraint $(\tilde{C}, \tilde{\sigma}) \in \mathcal{C}$, evaluate $\tilde{C}$ at $q$.
6. Let $p_0^{(q)} \in \mathbb{R}^\ell$ be the value of each oracle at point $q$, $p_1^{(q)}$ be given by $(p_1^{(q)})_i = q^{\rho - \sigma_i}(p_0^{(q)})_i$ for $i \in [\ell]$, and $p^{(q)} \in \mathbb{R}^{2\ell}$ be the concatenation of $p_0^{(q)}, p_1^{(q)}$.
7. Sample $p_{\text{sim}} \in \text{RS } [L, \rho]$ uniformly at random such that, for every $q \in Q_{\text{sim}}$, $p_{\text{sim}}(q) = \tilde{z}^T p^{(q)} + r_{\text{sim}}(q)$.
8. Now when $\tilde{V}$ queries $r$ at $q$, query every oracle received at $q$ and answer with $p_{\text{sim}}(q) - \tilde{z}^T p^{(q)}$.
9. Simulate the interaction of $P_{\text{LDT}}(p)$ and $\tilde{V}$.
10. Output the view of the simulated $\tilde{V}$.

For every query that $\tilde{V}$ makes to $r$, $\tilde{V}_R$ makes a query to every oracle it has received in the same location. Similarly, for each query $\tilde{V}$ makes to any other oracle, $\tilde{V}_R$ makes at most one query to some received oracle. Hence $\tilde{V}_R$ makes at most $b$ distinct queries across all oracles, and so the simulation guarantee holds.

To show zero knowledge, we exhibit the following hybrid experiment, in which the view of $\tilde{V}$ is identically distributed to the output of the simulator.

1. Run the honest prover $P_R(z, f)$; let $\Pi$ be as in the protocol.
2. Sample $r \in \text{RS } [L, \rho]$ uniformly at random and send it to $\tilde{V}$. Let $Q \subseteq L$ be the verifier’s queries to $r$ in this phase.
3. Receive $\tilde{z} \in \mathbb{R}^{2\ell}$ from $\tilde{V}$.
4. Sample $p \in \text{RS } [L, \rho]$ uniformly at random such that, for every $q \in Q$, $p(q) = (\tilde{z}^T \Pi)_q + r(q)$.
5. Replace $r$ with $p - \tilde{z}^T \Pi$.
6. Simulate the interaction of $P_{\text{LDT}}(p)$ and $\tilde{V}$.

One can verify that the view of $\tilde{V}$ in this hybrid is also identically distributed to the view of $\tilde{V}$ in the real protocol. In particular, all answers to $\tilde{V}$’s queries to $r$ after its replacement by $p$ are correctly distributed. 

37
10 Aurora: an IOP for rank-one constraint satisfaction (R1CS)

We describe the IOP for R1CS (Definition 8.1) that comprises the main technical contribution of this paper, and also underlies the SNARG for R1CS that we have designed and built (more about this in Section 11).

For the discussions below, we introduce notation about the low-degree test in [BBHR18b], known as “Fast Reed–Solomon IOPP” (FRI): given a subspace $L$ of a binary field $\mathbb{F}$ and rate $\rho \in (0, 1)$, we denote by $\varepsilon^\text{FRI}_i(\mathbb{F}, L)$ and $\varepsilon^\text{FRI}_q(L, \rho)$ the soundness error of the interactive and query phases in FRI (respectively) when testing proximity to RS $[L, \rho]$. In [BBHR18b], it is proved that $\varepsilon^\text{FRI}_i(\mathbb{F}, L) \leq 3|L|/|\mathbb{F}|$ and $\varepsilon^\text{FRI}_q(L, \rho) \leq 1 - (1 - 3\rho - 4|L|^{-1/2})/4$; much better values for both are conjectured to hold (see Appendix C.1).

We first provide a “barebones” statement with constant soundness error and no zero knowledge.

**Theorem 10.1.** There is an IOP for $R_{\text{R1CS}}$ (Definition 8.1) over binary fields $\mathbb{F}$ that, given an R1CS instance having $n$ variables and $m$ constraints, letting $\rho \in (0, 1)$ be a constant and $L$ be any subspace of $\mathbb{F}$ such that $2(\max(m, n + 1) + m \leq \rho|L|$, has the following parameters:

| alphabet $\Sigma$ | $= \mathbb{F}$ |
| number of rounds $k$ | $= O(\log |L|)$ |
| proof length $\rho$ | $= (6 + \frac{1}{3}|L|)$ |
| query complexity $q_\pi$ | $= O(\log |L|)$ |
| randomness $(r_i, r_q)$ | $= (O(\log |L| \cdot \log |\mathbb{F}|), O(\log |L|))$ |
| soundness error $(\varepsilon_i, \varepsilon_q)$ | $= \left(\frac{m+1}{|\mathbb{F}|}, \varepsilon^\text{FRI}_i(\mathbb{F}, L), L, \rho \right)^{\lambda_i}$ |
| prover time $t_p$ | $= O(\log(n + m) + \|A\| + \|B\| + \|C\|)$ |
| verifier time $t_v$ | $= O(\log(n + m) + \|A\| + \|B\| + \|C\|)$ |

**Proof.** Apply the transformation in Section 9 (see Theorem 9.1) to two ingredients: (a) the RS-encoded IOP for R1CS in Section 8 (see Theorem 8.2); and (b) the FRI low-degree test. The resulting protocol is sound against all malicious provers (and not just provers that send oracles that are Reed–Solomon codewords). \(\square\)

Next, we provide a statement that additionally has parameters for controlling the soundness error, is zero knowledge, and includes other (whitebox) optimizations; the proof is analogous except that we use zero knowledge components (the RS-encoded IOP of Theorem 8.4 and the transformation of Theorem 9.5). The resulting IOP protocol, fully specified in Fig. 5, underlies our SNARG for R1CS (see Section 11).

**Theorem 10.2.** There is an IOP for $R_{\text{R1CS}}$ (Definition 8.1) over binary fields $\mathbb{F}$ that, given an R1CS instance having $n$ variables and $m$ constraints, letting $\rho \in (0, 1)$ be a constant and $L$ be any subspace of $\mathbb{F}$ such that $2(\max(m, n + 1) + b) + m \leq \rho|L|$, is zero knowledge against $b$ queries and has the following parameters:

| alphabet $\Sigma$ | $= \mathbb{F}$ |
| number of rounds $k$ | $= O(\log |L|)$ |
| proof length $\rho$ | $= (4 + 2\lambda_i + \lambda_1^{\lambda_2^{\lambda_3}})|L|$ |
| query complexity $q_\pi$ | $= O(\lambda_1^{\lambda_2^{\lambda_3}} \log |L|)$ |
| randomness $(r_i, r_q)$ | $= (O(\lambda_i^{\lambda_2^{\lambda_3}} \log |L| \log |\mathbb{F}|), O(\lambda_1^{\lambda_2^{\lambda_3}} \lambda_2^{\lambda_3} \log |L|))$ |
| soundness error $(\varepsilon_i, \varepsilon_q)$ | $= \left(\frac{m+1}{|\mathbb{F}|}, \varepsilon^\text{FRI}_i(\mathbb{F}, L), L, \rho \right)^{\lambda_i}$ |
| prover time $t_p$ | $= \lambda_i \cdot O(|L| \cdot \log(n + m) + \|A\| + \|B\| + \|C\|)$ |
| verifier time $t_v$ | $= \lambda_i \cdot O(|L| + \|B\| + \|C\|)$ |
Setting $b \geq q_\pi$ ensures honest-verifier zero knowledge.

Given an R1CS instance $(\mathbb{F}, k, n, m, A, B, C, v)$, we fix subspaces $H_1, H_2 \subseteq \mathbb{F}$ such that $|H_1| = m$ and $|H_2| = n + 1$ (padding to the nearest power of 2 if necessary) with $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$, and a sufficiently large affine subspace $L \subseteq \mathbb{F}$ such that $L \cap (H_1 \cup H_2) = \emptyset$. Fig. 4 below gives polynomials and codewords used in the protocol figure (Fig. 5). We also define $\xi := \sum_{a \in H_1 \cup H_2} a^{H_1 \cup H_2} - 1$.

<table>
<thead>
<tr>
<th>Polynomial</th>
<th>Degree</th>
<th>Values that define the polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{a_1}$</td>
<td>$m - 1$</td>
<td>$\tilde{p}_{a_1}(a) = \alpha^{\gamma(a)}$ for $a \in H_1$</td>
</tr>
<tr>
<td>$p_{a_1}'$</td>
<td>max($m - 1, n$)</td>
<td>$\tilde{p}_{a_1}'(a) = \begin{cases} \alpha^{\gamma(a)} &amp; \text{for } a \in H_1 \ 0 &amp; \text{for } a \in (H_1 \cup H_2) \backslash H_1 \end{cases}$</td>
</tr>
<tr>
<td>$p_{a_1}^{(M)}$</td>
<td>max($m - 1, n$)</td>
<td>$\tilde{p}<em>{a_1}^{(M)}(b) = \begin{cases} \sum</em>{a \in H_1} M_{a,b} \cdot \alpha^{\gamma(a)} &amp; \text{for } b \in H_2 \ 0 &amp; \text{for } b \in (H_1 \cup H_2) \backslash H_2 \end{cases}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Codeword</th>
<th>Code</th>
<th>Polynomial that defines the codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_w$</td>
<td>RS $\left[ L, \frac{n-k+b}{</td>
<td>L</td>
</tr>
<tr>
<td>$f_{Mz}$</td>
<td>RS $\left[ L, \frac{m+b}{</td>
<td>L</td>
</tr>
</tbody>
</table>

Figure 4: Polynomials and codewords used in the IOP protocol given in Fig. 5.
Sample (as in Fig. 4):
- \( f_w \in \mathbb{RS} \left[ L, \frac{n-k+b}{|L|} \right] \)
- \( f_{A_0}, f_{B_0}, f_{C_0} \in \mathbb{RS} \left[ L, \frac{m+b}{|L|} \right] \)

**reduction from RICS to sumcheck**

\[
\begin{array}{c}
p = \left\{ f_w, f_{A_0}, f_{B_0}, f_{C_0} \right\} \\
\end{array}
\]

\[
\begin{array}{c}
p = \left\{ f_w \cdot Z_{H_2}^{\alpha} + f(w) \right\} \\
\end{array}
\]

repeat for \( i = 1, \ldots, \lambda_i \) in parallel:

**rowcheck/lincheck**

Sample \( r_i \leftarrow \mathbb{RS} \left[ L, \frac{t}{|L|} \right] \)

for univariate sumcheck below, and compute

\[
\mu_i := \sum_{a \in H_1 \cup H_2} r_i(a) 
\]

\[
\alpha_i, \tilde{s}_i 
\]

\[
\alpha_i \leftarrow \mathbb{F}, \tilde{s}_i \leftarrow \mathbb{F}^4 
\]

**virtual oracle for rowcheck**

\[
q_{i,1} := p_{\alpha} \cdot (f_{A_0} \cdot f_{B_0} - f_{C_0}) 
\]

**virtual oracle for lincheck**

\[
q_{i,2} := f_{A_0} \cdot p_{\alpha} - f_{z} \cdot p_{\alpha}^{(A)} 
\]

\[
q_{i,3} := f_{B_0} \cdot p_{\alpha} - f_{z} \cdot p_{\alpha}^{(B)} 
\]

\[
q_{i,4} := f_{C_0} \cdot p_{\alpha} - f_{z} \cdot p_{\alpha}^{(C)} 
\]

**amortized zero knowledge univariate sumcheck**

Compute:

\[
g_{i} \in \mathbb{RS} \left[ L, \frac{m-1}{|L|} \right], h_{i} \in \mathbb{RS} \left[ L, \frac{t-m}{|L|} \right] 
\]

s.t.

\[
\hat{r}_i(X) + \sum_{j=1}^{4} s_{i,j} \hat{q}_{i,j}(X) = \hat{g}_i(X) + \xi^{-1} \mu_i \cdot X^{1)_{H_1 \cup H_2} - 1} + Z_{H_1 \cup H_2}(X) \cdot \hat{r}_i(X) 
\]

For all \( a \in L, g_i(a) := 
\]

\[
r_i(a) + \sum_{j=1}^{4} s_{i,j} q_{i,j}(a) 
\]

\[
- \xi^{-1} \mu_i \cdot a^{1)_{H_1 \cup H_2} - 1} - Z_{H_1 \cup H_2}(a) \cdot r_i(a) 
\]

**low-degree test**

\[
\begin{array}{c}
\Pi := \left[ \begin{array}{c}
f_w \\
f_{A_0} \\
f_{B_0} \\
f_{C_0} \\
r_1 \vdots r_{\lambda_i} \\
h_1 \vdots h_{\lambda_i} \\
g_1 \vdots g_{\lambda_i} \\
(X^{t-n+1} g_{\lambda_i}) 
\end{array} \right] \\
\end{array}
\]

\[
\begin{array}{c}
\Pi := \left[ \begin{array}{c}
f_{w} \\
f_{A_0} \\
f_{B_0} \\
f_{C_0} \\
r_1 \vdots r_{\lambda_i} \\
h_1 \vdots h_{\lambda_i} \\
g_1 \vdots g_{\lambda_i} \\
(X^{t-n+1} g_{\lambda_i}) 
\end{array} \right] \\
\end{array}
\]

\[
\begin{array}{c}
\tilde{y}_1, \ldots, \tilde{y}_{\lambda_i}^{LDT} \\
\end{array}
\]

\[
\begin{array}{c}
\tilde{y}_1, \ldots, \tilde{y}_{\lambda_i}^{LDT} \leftarrow \mathbb{F}^{4+\lambda_i} \\
\end{array}
\]

**RS proximity test:**

\[
\text{FRl} \left( \tilde{y}_i^{\Pi} + r_{LDT} \right) 
\]

For \( i = 1, \ldots, \lambda_i \)

Figure 5: Diagram of the zero knowledge IOP for RICS that proves Theorem 10.2.
11 libiop: a library for IOP-based non-interactive arguments

We provide libiop, a codebase that enables the design and implementation of IOP-based non-interactive arguments. The codebase uses the C++ language and has three main components: (1) a library for writing IOP protocols; (2) a realization of the [BCS16] transformation, mapping any IOP written with our library to a corresponding non-interactive argument; (3) a portfolio of IOP protocols, including our new IOP protocol for R1CS and IOP protocols from [AHIV17] and [BBHR18a]. We discuss each of these components in turn.

11.1 Library for IOP protocols

We provide a library that enables a programmer to write IOP protocols. Informally, the programmer provides a blueprint of the IOP by specifying, for each round, the number and sizes of oracle messages (and non-oracle messages) sent by the prover, as well as the number of random bytes subsequently sent by the verifier. For the prover, the programmer specifies how each message is to be computed. For the verifier, the programmer specifies how oracle queries are generated and, also, how the verifier’s decision is computed based on its random choices and information received from the prover. Notable features of our library include:

- Support for writing new IOPs by using other IOPs as sub-protocols. This includes juxtaposing or interleaving selected rounds of these sub-protocols. This latter feature not only facilitates reducing round complexity in complex IOP constructions but also makes it possible to take advantage of optimizations such as column hashing (discussed in Section 11.2) when constructing a non-interactive argument.

- A realization of the transformation described in Section 9, which constructs an IOP by combining an encoded IOP (as defined in Section 4.7) and a low-degree test (as defined in Section 4.5.1). This is a powerful paradigm (it applies to essentially all published IOP protocols) that reduces the task of writing an IOP to merely providing suitable choices of these two simpler ingredients.

11.2 BCS transformation

We realize the transformation of [BCS16], by providing code that maps any IOP written in our library into a corresponding non-interactive argument (which consists of a prover algorithm and a verifier algorithm).

We use BLAKE2b [ANWOW13] to instantiate the random oracle in the [BCS16] transformation (our code allows to conveniently specify alternative instantiations). This hash function is an improvement to BLAKE (a finalist in the SHA-3 competition) [AMP14], and its performance on all recent x86 platforms is competitive with the most performant (and often hardware-accelerated) hash functions [CS17]. Moreover, BLAKE2b can be configured to output digests of any length between 1 and 64 bytes (between 8 and 512 bits in multiples of 8). When aiming for a security level of $\lambda$ bits, we only need the hash function to output digests of $2\lambda$ bits, and our code automatically sets this length.

Our code incorporates additional optimizations that, while simple, are generic and effective.

One is column hashing, which informally works as follows. In many IOP protocols (essentially all published ones, including Ligero [AHIV17] and Stark [BBHR18a]), the prover sends multiple oracles over the same domain in the same round, and the verifier accesses all of them at the same index in the domain. The prover can then build a Merkle tree over columns consisting of corresponding entries of the oracles, rather than building separate Merkle trees for each or a single Merkle tree over their concatenation. This reduces a non-interactive proof’s length, because the proof only has to contain a single authentication path for the desired column, rather than authentication paths corresponding to the indices across all the oracles.
Another optimization is path pruning. When providing multiple authentication paths relative to the same root (in the non-interactive argument), some digests become redundant and can thus be omitted. For example, if one considers the authentication paths for all leaves in a particular sub-tree, then one can simply provide the authentication path for the root of the sub-tree. A simple way to view this optimization is to provide the smallest number of digests to authenticate a set of leaves.

### 11.3 Portfolio of IOP protocols and sub-components

We use our library to realize several IOP protocols:

- **Aurora**: our IOP protocol for R1CS (specifically, the one provided in Fig. 5 in Section 10).
- **Ligero**: an adaptation of the IOP protocol in [AHIV17] to R1CS. While the protocol(s) in [AHIV17] are designed for (boolean or arithmetic) circuit satisfiability, the same ideas can be adapted to support R1CS at no extra cost. This simplifies comparisons with R1CS-based arguments, and confers additional expressivity. For convenience, we provide the foregoing adaptation in Appendix B.
- **Stark**: the IOP protocol in [BBHR18a] for Algebraic Placement and Routing (APR), a language that is a “succinct” analogue of algebraic satisfaction problems such as R1CS. (See [BBHR18a] for details.)

Each of the above IOPs is obtained by specifying an encoded IOP and a low-degree test. As explained in Sections 11.1 and 11.2, our library compiles these into an IOP protocol, and the latter into a non-interactive argument. This toolchain enables specifying protocols with few lines of code (see Fig. 6), and also enhances code auditability.

The IOP protocols above benefit from several algebraic components that our library also provides.

- **Finite field arithmetic.** We support prime and binary fields. Our prime field arithmetic uses Montgomery representation [Mon85]. Our binary field arithmetic uses the carryless multiplication instructions [Gue11]; these are ubiquitous in x86 CPUs and, being used in AES-GCM computations, are highly optimized.
- **FFT algorithms.** The choice of FFT algorithm depends on whether the R1CS instance (and thus the rest of the protocol) is defined over a prime or binary field. In the former case, we use the radix-2 FFT (whose evaluation domain is a multiplicative coset of order $2^a$ for some $a$) [CT65]. In the latter case, we use an additive FFT (whose evaluation domain is an affine subspace of the binary field) [Can89; GM10; BC14; LCH14; LAH16]. We also provide the respective inverse FFTs, and variants for cosets of the base domains.

**Remark 11.1.** Known techniques can be used to reduce given programs or general machine computations to low-level representations such as R1CS and APR (see, e.g., [BCTV14b; WSRBW15; BBHR18a]). Such techniques have been compared in prior work, and our library does not focus on these.

<table>
<thead>
<tr>
<th>encoded IOP protocol</th>
<th>lines of code</th>
<th>low-degree test</th>
<th>lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stark</td>
<td>321</td>
<td>FRI</td>
<td>416</td>
</tr>
<tr>
<td>Ligero</td>
<td>1281</td>
<td>direct</td>
<td>212</td>
</tr>
<tr>
<td>Aurora</td>
<td>1165</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: Lines of code to express various sub-components in our library.

42
12 Evaluation

In Section 12.1 we evaluate the performance of Aurora. Then, in Section 12.2 we compare Aurora with Ligero [AHIV17] and Stark [BBHR18a], two other IOP-based SNARGs. Our experiments not only demonstrate that Aurora’s performance matches the theoretical predictions implied by the protocol but also that Aurora achieves the smallest proof length of any IOP-based SNARG, more than an order of magnitude.

That said, there is still a sizable gap between the proof sizes of IOP-based SNARGs and other SNARGs that use public-key cryptographic assumptions vulnerable to quantum adversaries; see Fig. 2 for how proof sizes vary across these. It remains an exciting open problem to close this gap.

Experiments ran on a machine with an Intel Xeon W-2155 3.30GHz 10-core processor and 64GB of RAM.

12.1 Performance of Aurora

We consider Aurora at the standard security level of 128 bits, over the binary field $\mathbb{F}_{2^{192}}$. We report data on key efficiency measures of a SNARG: the time to generate a proof (running time of the prover), the length of a proof, and the time to check a proof (running time of the verifier). We also indicate how much of each cost is due to the IOP protocol, and how much is due to the BCS transformation [BCS16].

In Aurora, all of these quantities depend on the number of constraints $m$ in an R1CS instance.\textsuperscript{10} Our experiments report how these quantities change as we vary $m$ over the range $\{2^{10}, 2^{11}, \ldots, 2^{20}\}$.

**Prover running time.** In Fig. 7 we plot the running time of the prover, as absolute cost (top graph) and as relative cost when compared to native execution (bottom graph). In the case of R1CS, native execution means the time that it takes to check that an assignment satisfies the constraint system. The plot confirms the quasilinear complexity of the prover; proving times range from fractions of a second to several minutes. Proving time is dominated by the cost of running the underlying IOP prover.

**Proof size.** In Fig. 8 we plot proof size, as absolute cost (top graph) and as relative cost when compared to native witness size (bottom graph). In the case of R1CS, native witness size means the number of bytes required to represent an assignment to the constraint system. The plot shows that compression (proof size is smaller than native witness size) occurs for $m \geq 4000$. The plot also shows that proof size ranges from 50 kB to 250 kB, and that proof size is dominated by the cryptographic digests to authenticate query answers.

**Verifier running time.** In Fig. 9 we plot the running time of the verifier, as absolute cost (top graph) and as relative cost when compared to native execution (bottom graph). The plot shows that verification times range from milliseconds to seconds, and confirms that our implementation incurs a constant multiplicative overhead over native execution.

12.2 Comparison of Ligero, Stark, and Aurora

In Figs. 10 to 12 we compare costs (proving time, proof length, and verification time) on R1CS instances for three IOP-based SNARGs: Ligero [AHIV17], Stark [BBHR18a], and Aurora (this work). As in Section 12.1, we plot costs as the number of constraints $m$ increases (and with $n \approx m$ variables as explained in Footnote 10); we also set security to the standard level of 128 bits and use the binary field $\mathbb{F}_{2^{192}}$.

**Comparison of Ligero and Aurora.** Ligero natively supports R1CS so a comparison with Aurora is straightforward. Fig. 11 shows that proof size in Aurora is much smaller than in Ligero, even for a relatively

\textsuperscript{10}The number of variables $n$ also affects performance, but it is usually close to $m$ and so we take $n \approx m$ in our experiments. The number of inputs $k$ in an R1CS instance is at most $n$, and in typical applications it is much smaller than $n$, so we do not focus on it.
small number of constraints. The gap between the two only grows bigger as the number of constraints increases, as Aurora’s proof size is polylogarithmic while Ligero’s is only sublinear (an exponential gap).

**Comparison of Stark and Aurora.** Stark does not natively support the NP-complete relation R1CS but instead natively supports an NEXP-complete relation known as *Algebraic Placement and Routing* (APR). These two relations are quite different,\(^1\) and so to achieve a meaningful comparison, we consider an APR instance that *simulates* a given R1CS instance. We thus plot the costs of Stark on a hand-optimized APR instance that simulates R1CS instances. Relying on the reductions described in [BBHR18a], we wrote an APR instance that realizes a simple abstract computer that checks that a variable assignment satisfies each one of the rank-1 constraints in a given R1CS instance.

Fig. 11 shows that proof size in Aurora is much smaller than in Stark, even if both share the same asymptotic growth. This is due to the fact that R1CS and APR target different computation models (explicit circuits vs. uniform computations), so Stark incurs significant overheads when used for R1CS. Fig. 12 shows that verification time in Stark grows linearly with the number of constraints (like Ligero and Aurora); indeed, the verifier must read the description of the statement being proved, which is the entire constraint system.

\(^{11}\)Using notation for APR introduced in Appendix C.2, one can think of APR as a *succinctly-represented* system of \(|H| \cdot |C|\) equations over \(|H| \cdot |R|\) variables, in which equations have total degree at most \(D := \max_{c \in C} \deg c\). When \(D = 2\), one could be led to view APR as “comparable” to R1CS with \(m = |H| \cdot |C|\) constraints over \(n = |H| \cdot |R|\) variables. This comparison, however, is misleading in that one is simply comparing the total number of constraints and variables, ignoring what they actually represent.
Figure 7: Proving time in Aurora.  
Figure 8: Proof length in Aurora.  
Figure 9: Verification time in Aurora.  

Figure 10: Proving time in Aurora, Ligero, Stark.  
Figure 11: Proof length in Aurora, Ligero, Stark.  
Figure 12: Verification time in Aurora, Ligero, Stark.
A Proof of Lemma 5.4

Definition A.1. For any field $\mathbb{F}$, the derivative of a function $f: \mathbb{F} \to \mathbb{F}$ in a direction $a \in \mathbb{F}$ is the function $\Delta_a(f)(X) := f(a + X) - f(X)$. Given $a_1, \ldots, a_k \in \mathbb{F}$, we inductively define $\Delta_{a_1, \ldots, a_k}(f) := \Delta_{a_1}(\Delta_{a_2, \ldots, a_k}(f))$.

Let $\mathbb{F}$ be an extension field of $\mathbb{F}_2$. For $a_1, \ldots, a_k$ a basis of $H_0$,

$$\Delta_{a_1, \ldots, a_k}(f)(X) = \sum_{a \in H_0} f(X + a).$$

An alternative statement of the above is that $\Delta_{a_1, \ldots, a_k}(f)(a_0)$ is equal to the sum of $f$ over $H := a_0 + H_0$:

$$\Delta_{a_1, \ldots, a_k}(f)(a_0) = \sum_{a \in H} f(a). \tag{1}$$

For a natural number $c$ written in base 2 as $\sum_{i=0}^d c_i \cdot 2^i$ let $\text{supp}(c) = \{i : c_i \neq 0\}$ and $\text{wt}(c) = |\text{supp}(c)|$. For a polynomial $P(X) = \sum_{j \geq 0} \alpha_j X^j$ we define $\text{wt}(P) = \max\{\text{wt}(j) : \alpha_j \neq 0\}$.

The following claim is implied by [AKKLR05; KS08].

Claim A.2. For any polynomial $P \in \mathbb{F}[X]$ of positive degree, and any $a \in \mathbb{F}$,

$$\text{wt}(\Delta_a(P)) < \text{wt}(P)$$

Proof. By linearity of the operator $\Delta_a(\cdot)$, it suffices to prove the claim for a single monomial, namely, for $P(X) = X^c$ for some positive integer $c > 0$. Write $c$ in binary as $\sum_{i=0}^d c_i \cdot 2^i$ for some integer $d$. The Frobenius automorphism $X \mapsto X^{2^d}$ is $\mathbb{F}_2$-linear, meaning that $(X + Y)^{2^d} = X^{2^d} + Y^{2^d}$. Thus,

$$\Delta_a(X^c) = (X + a)^{\sum_{i=0}^d c_i 2^i} - X^c = \prod_{i=0}^d \left( X^{c_i 2^i} + a^{c_i 2^i} \right) - X^c$$

$$= \sum_{I \subseteq \text{supp}(c)} \prod_{i \in I} \left( X^{c_i 2^i} \right) \cdot \prod_{j \in [d] \setminus I} \left( a_j 2^j \right)$$

$$= \sum_{I \subseteq \text{supp}(c)} a_I \cdot X^{\sum_{i \in I} c_i 2^i}$$

Since all the exponents in the last expression are integers whose support is strictly contained in $\text{supp}(c)$, the claim follows for the case of $P(X) = X^c$, and hence for all $P \in \mathbb{F}[X]$ by linearity.

Lemma A.3. If $P(X) \in \mathbb{F}[X]$ satisfies $\text{deg}(P) < 2^k - 1$, then for any $a_1, \ldots, a_k$ that are linearly independent over $\mathbb{F}_2$ we have $\Delta_{a_1, \ldots, a_k}(P(X)) = 0$.

Proof. We have $\text{wt}(P) < k$. By Claim A.2, $\text{wt}(\Delta_{a_2, \ldots, a_k}(P)) = 0$, and so $\Delta_{a_2, \ldots, a_k}(P)$ is a constant function. By definition, the derivative of a constant function is 0, and the claim follows.

Proof of Lemma 5.4. For some $a_0, a_1, \ldots, a_k \in \mathbb{F}$, $H = a_0 + H_0$ where $H_0$ is the linear subspace with basis $a_1, \ldots, a_k$. By Eq. (1) and Lemma A.3 we conclude $\sum_{a \in H} g(a) = \Delta_{a_1, \ldots, a_k}(g)(a_0) = 0$.  \[\Box\]
B Adaptation of Ligero to the R1CS relation

We describe R1CS-Ligero, an adaptation of the Ligero protocol for the R1CS relation (Definition 8.1). This adaptation captures as a special case, and at no additional cost, the arithmetic circuits considered in [AHIV17]. The high-level structure of the protocol is analogous to that described in Section 2.1. Namely, given a satisfying assignment \( z \) to an R1CS instance with matrices \( A, B, C \), the prover computes \( y_A := Az \), \( y_B := Bz \), \( y_C := Cz \), and sends to the verifier certain encodings of \( z, y_A, y_B, y_C \). After that, the prover convinces the verifier that \( "y_M = Mz" \) for \( M \in \{ A, B, C \} \) via three suitable linecheck protocols, and that \( "y_A \circ y_B = y_C" \) via a suitable rowcheck protocol. A key aspect is that the encoding of \( N \) field elements consists of \( O(\sqrt{N}) \) Reed–Solomon codewords of block length \( O(\sqrt{N}) \) rather than a single Reed–Solomon codeword of length \( O(N) \) — this aspect is what determines the design of the aforementioned sub-protocols. As in [AHIV17], the final protocol is an IPCP, i.e., an IOP wherein only the first prover message is an oracle.

The rest of this section is structured as follows. In Appendix B.1 and Appendix B.2 we describe linecheck and rowcheck protocols for information encoded via the interleaved Reed–Solomon code. In Appendix B.3 we show how to combine these to obtain an RS-encoded IPCP for the R1CS relation; this protocol also takes care of additional goals such as zero knowledge and input consistency. In Appendix B.4 we explain how generic tools can augment this latter protocol to a standard IPCP (that is sound against all provers).

Unlike in an RS-encoded IOP, in an RS-encoded IPCP we count queries to the first (oracle) message only. All other messages are read in full by the verifier, and we charge their length to communication complexity.

B.1 Interleaved linecheck

Let \( \mathbb{F} \) be a field and \( L, H \) subsets of \( \mathbb{F} \) of sizes \( l, h \) respectively (with \( l \geq h \)). Let \( M \) be a \( m_1h \times m_2h \) matrix over \( \mathbb{F} \), for two positive integers \( m_1, m_2 \). Below we describe an RS-encoded IPCP protocol for testing that two given oracles \( F_x \in \text{RS}[L, h/l]^{m_1} \) and \( F_y \in \text{RS}[L, h/l]^{m_2} \) encode messages \( x \in \mathbb{F}^{m_1h} \) and \( y \in \mathbb{F}^{m_2h} \) such that \( x = My \). This can be viewed as the interleaved analogue of the RS-encoded IOP in Section 6, and is a modification of the “Test-Linear-Constraints-IRS” protocol in [AHIV17] in which the result of the linear transformation is encoded by an oracle rather than being known to the verifier.

The protocol below is summarized in Fig. 13, and implicitly assumes an ordering \( \gamma_H : H \rightarrow \{1, \ldots, h\} \) on \( H \). The parameter \( \lambda_q \) controls the number of query repetitions in the verifier.

1. The verifier \( V \) samples random vectors \( r_1, \ldots, r_{m_1} \in \mathbb{F}^h \) and sends these to \( P \).

2. The verifier \( V \) and prover \( P \) compute:
   - \((s_1, \ldots, s_{m_2}) := (r_1, \ldots, r_{m_1})^\top M\);
   - for \( i \in [m_1] \), the polynomial \( \hat{r}_i \) of degree less than \( h \) that evaluates to \( r_i \in \mathbb{F}^h \) on \( H \);
   - for \( i \in [m_2] \), the polynomial \( \hat{s}_i \) of degree less than \( h \) that evaluates to \( s_i \in \mathbb{F}^h \) on \( H \).

3. The prover \( P \) sends the \( 2h - 1 \) coefficients of the polynomial \( \hat{p} = \sum_{i=1}^{m_1} \hat{r}_i \cdot \hat{f}_{x,i} - \sum_{i=1}^{m_2} \hat{s}_i \cdot \hat{f}_{y,i} \), where \( \hat{f}_{x,i}, \hat{f}_{y,i} \) are the polynomials of degree less than \( h \) corresponding to the \( i \)-th row of \( F_x, F_y \).

4. The verifier \( V \) samples random indices \( \alpha_1, \ldots, \alpha_{\lambda_q} \leftarrow L \), queries \( F_x, F_y \) at \( \alpha_k \) for \( k \in [\lambda_q] \), and checks that: (a) \( \sum_{\alpha \in H} \hat{p}(\alpha) = 0 \); (b) \( \hat{p}(\alpha_k) = \sum_{i=1}^{m_1} \hat{r}_i(\alpha_k) \cdot F_x[i, \alpha_k] - \sum_{i=1}^{m_2} \hat{s}_i(\alpha_k) \cdot F_y[i, \alpha_k] \) for all \( k \in [\lambda_q] \).
Completeness. If \( x = M y \) and \( P \) sends the correct \( \hat{p} \) then, letting \( r = (r_1, \ldots, r_{m_1}) \) and \( s = (s_1, \ldots, s_{m_2}) \),

\[
\sum_{\alpha \in H} \hat{p}(\alpha) = \sum_{\alpha \in H} \left( \sum_{i=1}^{m_1} \hat{r}_i(\alpha) \cdot \hat{f}_{x,i}(\alpha) - \sum_{i=1}^{m_2} \hat{s}_i(\alpha) \cdot \hat{f}_{y,i}(\alpha) \right) \\
= \sum_{i=1}^{m_1} r_i^\top x_i - \sum_{i=1}^{m_2} s_i^\top y_i \\
= r^\top x - s^\top y \\
= r^\top (M y) - (r^\top M) y \\
= 0 ,
\]

so the verifier’s first test passes. Correctness of the second test follows directly from the definition of \( \hat{p} \).

Soundness. If \( x \neq M y \), there is a \( 1/|\mathbb{F}| \) probability over the choice of \( r_1, \ldots, r_{m_1} \) that \( (r_1, \ldots, r_{m_1})^\top x = (r_1, \ldots, r_{m_1})^\top M y \). Letting \( \hat{p} \) be the polynomial to be sent by an honest prover, if \( (r_1, \ldots, r_{m_1})^\top x \neq (r_1, \ldots, r_{m_1})^\top M y \) then \( \hat{p} \) does not sum to 0 over \( H \). If the polynomial \( \hat{p}' \) actually sent by the prover is equal to \( \hat{p} \), then \( V \) rejects (always). Otherwise, as both are polynomials of degree less than \( 2h - 1 \), \( \hat{p} \) and \( \hat{p}' \) agree on at most \( 2h - 2 \) points. The verifier accepts only if all of its queries lie in this set.

Efficiency. The prover and the verifier perform matrix multiplication by \( M \), whose cost depends on the number of nonzero entries in \( M \). Each also performs interpolations to find the polynomials \( \hat{s}_i \) and \( \hat{r}_i \).

Additionally, the prover finds \( \hat{p} \) by evaluating \( \hat{s}_i \) and \( \hat{r}_i \) over \( L \), suitably combining these with evaluations of \( \hat{f}_{x,i} \) and \( \hat{f}_{y,i} \), and interpolating the result. The verifier also evaluates \( \hat{p} \) on \( H \) for its first test, and performs simple arithmetic to check the answer of each of its queries.

Summary. The aforementioned protocol is an RS-encoded IOP with the following parameters.

| alphabet | \( \Sigma \) | \( \mathbb{F} \) |
| number of rounds | \( k \) | 1 |
| communication | \( c \) | \( 2h - 1 \) |
| query complexity | \( q \) | \( (m_1 + m_2)\lambda_q \) |
| randomness | \( (r_1, r_2) \) | \( (m_1 h \log |\mathbb{F}|, \lambda_q \log l) \) |
| soundness error | \( (\varepsilon_1, \varepsilon_2) \) | \( \left( \frac{1}{|\mathbb{F}|}, \frac{(2h-2)^2 \lambda_q}{\lambda_q} \right) \) |
| prover time | \( t_p \) | \( O(|M|) + (m_1 + m_2)\text{FFT}(\mathbb{F}, h) + (m_1 + m_2 + 1)\text{FFT}(\mathbb{F}, l) + O((m_1 + m_2)h) \) |
| verifier time | \( t_v \) | \( O(|M|) + (m_1 + m_2)\text{FFT}(\mathbb{F}, h) + \text{FFT}(\mathbb{F}, 2h) + O(\lambda_q (m_1 + m_2)h) \) |

B.2 Interleaved rowcheck

Let \( \mathbb{F} \) be a field; \( L, H \) subsets of \( \mathbb{F} \) of sizes \( l, h \) respectively (with \( l \geq h \)). Below we describe an RS-encoded IOP protocol for testing that three given oracles \( F_x, F_y, F_z \in \text{RS}[L, h/l]^m \) encode messages \( x, y, z \in \mathbb{F}^{mh} \) such that \( x \circ y = z \). This can be viewed as the interleaved analogue of the RS-encoded IOP in Section 7, and is a straightforward modification of the “Test-Quadratic-Constraints-IRS” protocol in [AHIV17].

The protocol below is summarized in Fig. 14, and implicitly assumes an ordering \( \gamma_H : H \to \{1, \ldots, h\} \) on \( H \). The parameter \( \lambda_q \) controls the number of query repetitions in the verifier.

1. The verifier \( V \) samples random \( t \in \mathbb{F}^m \) and sends \( t \) to \( P \).
2. The prover \( P \) sends the \( 2h - 1 \) coefficients of the polynomial \( \hat{p} = \sum_{i=1}^{m} t_i \cdot (\hat{f}_{x,i} \cdot \hat{f}_{y,i} - \hat{f}_{z,i}) \) where \( \hat{f}_{x,i}, \hat{f}_{y,i}, \hat{f}_{z,i} \) are the polynomials of degree less than \( h \) corresponding to the \( i \)-th row of \( F_x, F_y, F_z \).
3. The verifier $V$ samples indices $\alpha_1, \ldots, \alpha_{\lambda q} \leftarrow L$, queries $F_x, F_y, F_z$ at $\alpha_k$ for every $k \in [\lambda q]$, and checks that:

(a) $\hat{p}(H) = \{0\};$

(b) $\hat{p}(\alpha_k) = \sum_{i=1}^{m} t_i \cdot (F_x[i, \alpha_k] \cdot F_y[i, \alpha_k] - F_z[i, \alpha_k])$ for every $k \in [\lambda q]$.

**Completeness.** If $x \circ y = z$ and $P$ sends the correct $\hat{p}$ then, for every $\alpha \in H$,

$$\hat{p}(\alpha) = \sum_{i=1}^{m} t_i \cdot \left( \hat{f}_{x,i}(\alpha) \cdot \hat{f}_{y,i}(\alpha) - \hat{f}_{z,i}(\alpha) \right) = \sum_{i=1}^{m} t_i \cdot (x_i \gamma_H(\alpha) \cdot y_i \gamma_H(\alpha) - z_i \gamma_H(\alpha)) = \sum_{i=1}^{m} t_i \cdot 0 = 0,$$

so the verifier’s first test passes. Correctness of the second test follows directly from the definition of $\hat{p}$.

**Soundness.** If $x \circ y \neq z$, there is a $1/|F|$ probability over the choice of $t$ that $t^T (x \circ y - z) = 0$, where $x, y, z$ are viewed as $m \times h$ matrices. Letting $\hat{p}$ be the polynomial to be sent by an honest prover, if $t^T (x \circ y - z) \neq 0$ then $\hat{p}$ does not vanish on $H$. If the polynomial $\hat{p}'$ actually sent by the prover is equal to $\hat{p}$, then $V$ rejects (always). Otherwise, as both are polynomials of degree less than $2h - 1$, $\hat{p}$ and $\hat{p}'$ agree on at most $2h - 2$ points. The verifier accepts only if all of its queries lie in this set.

**Efficiency.** The prover obtains $\hat{p}$ by suitably combining evaluations of $\hat{f}_{x,i}, \hat{f}_{y,i}, \hat{f}_{z,i}$ and then interpolating. The verifier evaluates $\hat{p}$ on $H$ for its first test, and performs simple arithmetic to check answers to its queries.

**Summary.** The aforementioned protocol is an RS-encoded IOP with the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet</td>
<td>$\Sigma = F$</td>
</tr>
<tr>
<td>number of rounds</td>
<td>$k = 1$</td>
</tr>
<tr>
<td>communication</td>
<td>$c = 2h - 1$</td>
</tr>
<tr>
<td>query complexity</td>
<td>$q = 3m\lambda_q$</td>
</tr>
<tr>
<td>randomness</td>
<td>$(\tau_1, \tau_q) = (mh \log</td>
</tr>
<tr>
<td>soundness error</td>
<td>$(\varepsilon_1, \varepsilon_q) = \left( \frac{1}{</td>
</tr>
<tr>
<td>prover time</td>
<td>$t_p = \text{FFT}(F, l) + O(ml)$</td>
</tr>
<tr>
<td>verifier time</td>
<td>$t_v = \text{FFT}(F, 2h) + O(\lambda_q m)$</td>
</tr>
</tbody>
</table>

### B.3 Interleaved ZKIPCP for R1CS

We describe an RS-encoded IPCP protocol for the R1CS relation (see Definition 8.1). This can be viewed as an interleaved analogue of the RS-encoded IOP for R1CS in Section 8, and is a modification of the IPCP for arithmetic circuits in [AHIV17] to work for R1CS.

Let $(\mathbb{F}, k, n, m, A, B, C, v)$ be an R1CS instance and $w$ a witness for it. The prover and verifier receive the instance as input, and the prover additionally receives the witness as input. Define $z := (1, v, w) \in \mathbb{F}^{n+1}$.

Let $L, H$ be disjoint subsets of $\mathbb{F}$ of sizes $l, h$ respectively (with $l \geq h$) and let $m_2, m_1$ be integers such that $m_1 h = m$ and $m_2 h = 1 + n$. Let $b$ be the query bound for zero knowledge.

The protocol below is summarized in Fig. 15, and implicitly assumes an ordering $\gamma_H : H \rightarrow \{1, \ldots, h\}$ on $H$. The parameter $\lambda_i$ controls the number of repetitions of the sub-protocols, and $\lambda_q$ controls the number of query repetitions in the verifier.

**Oracle:** The prover $P$ sends an oracle $F \in \text{RS}[L, \hat{p}]^{m_2+3m_1+4\lambda_i}$ that is computed as follows.

Extend the witness $w \in \mathbb{F}^{n-k}$ to $\overline{w} := (0^{1+k}, w) \in \mathbb{F}^{1+n}$, and sample a random codeword $F_{\overline{w}} \in \text{RS}[L, \frac{h+b}{l}]^{m_2}$ such that the evaluation over $H$ of the interpolation of the $i$-th row of $F_{\overline{w}}$ is the $i$-th block of $h$ entries in $\overline{w}$ (note that $1+n = m_2 h$). Compute vectors $a := Az, b := Bz, c := Cz \in \mathbb{F}^{m_2}$, and sample random codewords $F_a, F_b, F_c \in \text{RS}[L, \frac{h+b}{l}]^{m_1}$ such that the evaluation over $H$ of the interpolation of the $i$-th row of $F_a, F_b, F_c$ is the $i$-th block of $h$ entries in $a, b, c$ respectively (note that $m = m_1 h$). For
every $i \in [\lambda_i]$, sample random codewords $q_i^a, q_i^b, q_i^c \in \text{RS} \left[ L, \frac{2h+b-1}{i} \right]$ such that each of $q_i^a, q_i^b, q_i^c$ sums to zero on $H$, and random codeword $q_i^{\text{ROW}} \in \text{RS} \left[ L, \frac{2h+2b-1}{i} \right]$ such that $q_i^{\text{ROW}}$ vanishes everywhere on $H$. The oracle $F$ is the vertical juxtaposition of $F_\pi, F_a, F_b, F_c$ as well as $q_i^a, q_i^b, q_i^c, q_i^{\text{ROW}}$. Note that each codeword in $F_\pi, F_a, F_b, F_c$ is b-wise independent (because of the way they are sampled), and thus any set of $b$ evaluations are uniformly distributed (in particular, they reveal no information about $w, a, b, c$).

- The interactive protocol:

1. The prover $P$ and verifier $V$ extend the public input $v$ to $\pi := (1, v, 0^{n-k})$, and compute the codeword $F_\pi \in \text{RS} \left[ L, \frac{h}{2} \right]^{m^2}$ such that the evaluation over $H$ of the interpolation of the $i$-th row of $F_\pi$ is the $i$-th block of $h$ entries in $\pi$ (note that $1 + n = m_2h$). By linearity, $F_\pi + F_m$ encodes $z = (1, v, w) \in \mathbb{F}^{n+1}$.

2. For every $i \in [\lambda_i]$, $V$ samples vectors $r_{i,1}, \ldots, r_{i,m_1} \leftarrow \mathbb{F}^h$ (for lincheck) and $t_i \in \mathbb{F}^{m_1}$ (for rowcheck).

3. For every $i \in [\lambda_i]$, the prover $P$ and verifier $V$ compute several vectors:

\[
\begin{align*}
(s^a_{i,1}, \ldots, s^a_{i,m_2}) & := (r_{i,1}, \ldots, r_{i,m_1})^\top A, \\
(s^b_{i,1}, \ldots, s^b_{i,m_2}) & := (r_{i,1}, \ldots, r_{i,m_1})^\top B, \\
(s^c_{i,1}, \ldots, s^c_{i,m_2}) & := (r_{i,1}, \ldots, r_{i,m_1})^\top C.
\end{align*}
\]

They also find the polynomial $\hat{r}_{i,i}$ of degree less than $h$ that evaluates to $r_{i,i}$ on $H$ (for $i \in [m_1]$), and the polynomials $\hat{s}^a_{i,i}, \hat{s}^b_{i,i}, \hat{s}^c_{i,i}$ of degree less than $h$ that evaluate to $s^a_{i,i}, s^b_{i,i}, s^c_{i,i}$ on $H$ (for $i \in [m_2]$).

4. For every $i \in [\lambda_i]$, $P$ responds with (the coefficients of) several polynomials:

- For every $\phi \in \{a, b, c\}$, a lincheck polynomial $\hat{p}_i^\phi$ of degree less than $2h + b - 1$ defined as

\[
\hat{p}_i^\phi := \hat{q}_i^\phi + \sum_{i=1}^{m_1} \hat{r}_{i,i} \cdot \hat{f}_{\phi,i} - \sum_{i=1}^{m_2} \hat{s}^\phi_{i,i} \cdot (\hat{f}_{\pi,i} + \hat{f}_{\overline{\pi},i})
\]

where

* $\hat{f}_{\phi,i}$ is the polynomial of degree less than $h + b$ that interpolate the $i$-th row of $F_\phi$ (for $i \in [m_1]$);
* $\hat{f}_{\pi,i}$ is the polynomial of degree less than $h + b$ that interpolates the $i$-th row of $F_\pi$ (for $i \in [m_2]$);
* $\hat{f}_{\overline{\pi},i}$ is the polynomial of degree less than $h + b$ the interpolates the $i$-th row of $F_{\overline{\pi}}$ (for $i \in [m_2]$).

- A rowcheck polynomial $\hat{p}_i^{\text{ROW}}$ of degree less than $2h + 2b - 1$ defined as

\[
\hat{p}_i^{\text{ROW}} := \hat{q}_i^{\text{ROW}} + \sum_{i=1}^{m_1} t_{i,i} \cdot (\hat{f}_{a,i} \cdot \hat{f}_{b,i} - \hat{f}_{c,i})
\]

where $\{\hat{f}_{a,i}, \hat{f}_{b,i}, \hat{f}_{c,i}\}$ are the polynomials of degree less than $h + b$ that interpolate the $i$-th row of $\{F_a, F_b, F_c\}$ respectively.

5. The verifier $V$ samples random indices $\alpha_1, \ldots, \alpha_{\lambda_q} \leftarrow L$ and, for every $k \in [\lambda_q]$, queries $F$ at $\alpha_k$ thereby obtaining

\[
F[\alpha_k] = (F_\pi[\alpha_k], F_a[\alpha_k], F_b[\alpha_k], F_c[\alpha_k], q_i^a[\alpha_k], q_i^b[\alpha_k], q_i^c[\alpha_k], q_i^{\text{ROW}}[\alpha_k]).
\]

The verifier $V$ accepts if and only if for every $i \in [\lambda_i]$ the following tests pass.
Lincheck tests. For every \( \diamond \in \{ a, b, c \}, \sum_{\alpha \in H} \hat{p}_i^\diamond(\alpha) = 0 \) and for every \( k \in [\lambda q] \) it holds that

\[
\hat{p}_i^\diamond(\alpha_k) = q_i^\diamond[\alpha_k] + \sum_{i=1}^{m_1} \hat{r}_i,i(\alpha_k) \cdot F_\diamond[i, \alpha_k] - \sum_{i=1}^{m_2} \hat{s}_i,i(\alpha_k) \cdot (F_\pi[i, \alpha_k] + F_\pi[i, \alpha_k]) .
\]

Rowcheck test. \( \hat{p}_i^{ROW}(H) = \{ 0 \} \) and for every \( k \in [\lambda q] \) it holds that

\[
\hat{p}_i^{ROW}(\alpha_k) = q_i^{ROW}[\alpha_k] + \sum_{i=1}^{m_1} t_{i,i}(\alpha_k) \cdot (F_a[i, \alpha_k] - F_c[i, \alpha_k]) .
\]

Completeness. If \( w \) is in fact a satisfying witness for the RICS instance, and the prover is honest, then the rowcheck and lincheck correctness tests pass, by arguments analogous to those made for the previous two protocols. The masking codewords \( \{ q_i^a, q_i^b, q_i^c, q_i^{ROW} \}_{i \in [\lambda]} \) are chosen so that completeness is unaffected.

Soundness. Assume that the RICS instance is not satisfiable. Let \( \tilde{F} \) be the codeword sent by the prover. Let \( \tilde{w} \) be the candidate witness encoded in \( \tilde{F} \); note that \( A \tilde{z} \circ B \tilde{z} \neq C \tilde{z} \) where \( \tilde{z} = (1, v, \tilde{w}) \). Let \( \tilde{a}, \tilde{b}, \tilde{c} \) be the alleged linear transformations of \( \tilde{z} \) encoded in \( \tilde{F} \). One of the following equations cannot hold: \( \tilde{a} = A \tilde{z}, \tilde{b} = B \tilde{z}, \tilde{c} = C \tilde{z}, \tilde{a} \circ \tilde{b} = \tilde{c} \). If one of the first three equations fails to hold, the corresponding lincheck sub-protocol will reject with high probability; if the last equation fails to hold, the rowcheck sub-protocol will reject with high probability.

The interactive phase of each of these sub-protocols is repeated \( \lambda \) times, bringing the corresponding soundness error down from \( 1/|F| \) to \( 1/|F|^{\lambda} \); the subsequent query phase is repeated \( \lambda \) times, bringing the corresponding soundness error down from \( 2^{h+2b-2} \) to \( (2^{h+2b-2})^{\lambda} \).

Note that the masking codewords \( q_i^a, q_i^b, q_i^c, q_i^{ROW} \) do not affect soundness, as we now explain. In the “no” case for the lincheck protocol, the summation \( \sum_{\alpha \in H} \hat{p}(\alpha) \) is uniform over \( F \). In the “no” case for the rowcheck protocol, there exists some \( \alpha \in H \) such that \( \hat{p}(\alpha) \) is uniform over \( F \). Thus in both cases, regardless of the (possibly malicious) choice of mask the probability that the verifier accepts remains \( 1/|F| \).

Zero knowledge. We construct a probabilistic simulator \( S \) that, given as input a satisfiable RICS instance \( (F, k, n, m, A, B, C, v) \) and straightline access to a \( b \)-query malicious verifier \( \tilde{V} \), outputs a view that is identically distributed as \( \tilde{V} \)'s view when interacting with an honest prover.

1. Use the public input \( v \) to compute \( F_\pi \in RS[L, h/2]^{m_2} \) like the honest prover does.
2. Sample \( F_\pi \in (F^{L^{m_2}}) \) and \( F_a, F_b, F_c \in (F^{L^{m_1}}) \) uniformly at random. For every \( i \in [\lambda] \), sample \( q_i^a, q_i^b, q_i^c \) in \( RS[L, 2^{h+b-1}] \) uniformly at random given that the interpolation of \( q_i^a, q_i^b, q_i^c \) sums to 0 on \( H \). For every \( i \in [\lambda] \), sample \( q_i^{ROW} \) in \( RS[L, 2^{h+b-1}] \) uniformly at random given that its interpolation vanishes everywhere on \( H \). Set \( F = (F_\pi, F_a, F_b, F_c, q_i^a, q_i^b, q_i^c, q_i^{ROW}) \), and start simulating \( \tilde{V} \).
3. Use \( F \) to answer any queries by \( \tilde{V} \). Let \( Q \subseteq L \) be the queries asked by \( \tilde{V} \) until the next step.
4. Receive a challenge \( \{ r_{i,1}, \ldots, r_{i,m_1}, t_i \}_{i \in [\lambda]} \) from \( \tilde{V} \).
5. For every \( i \in [\lambda] \), sample \( \hat{p}_i^a, \hat{p}_i^b, \hat{p}_i^c \) in \( RS[L, 2^{h+b-1}] \) uniformly at random such that each of \( \hat{p}_i^a, \hat{p}_i^b, \hat{p}_i^c \) sums to 0 on \( H \) and, for every \( \alpha \in Q \), the following hold:
   - \( \hat{p}_i^a(\alpha) = \sum_{i=1}^{m_1} \hat{r}_{i,i}(\alpha) \cdot F_a[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,i}(\alpha) \cdot (F_\pi[i, \alpha] + F_\pi[i, \alpha]) - q_i^a[\alpha] \)
   - \( \hat{p}_i^b(\alpha) = \sum_{i=1}^{m_1} \hat{r}_{i,i}(\alpha) \cdot F_b[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,i}(\alpha) \cdot (F_\pi[i, \alpha] + F_\pi[i, \alpha]) - q_i^b[\alpha] \)
   - \( \hat{p}_i^c(\alpha) = \sum_{i=1}^{m_1} \hat{r}_{i,i}(\alpha) \cdot F_c[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,i}(\alpha) \cdot (F_\pi[i, \alpha] + F_\pi[i, \alpha]) - q_i^c[\alpha] \)
6. For every \( i \in [\lambda] \), sample \( \hat{p}_i^{ROW} \) in \( RS[L, 2^{h+b-2}] \) uniformly at random such that \( \hat{p}_i^{ROW} \) evaluates to 0 everywhere on \( H \), and, for every \( \alpha \in Q \), the following holds:
\[ p_t^{\text{ROW}}(\alpha) = \sum_{i=1}^{m_1} t_{i,\alpha} \cdot (F_a[i, \alpha] \cdot F_b[i, \alpha] - F_c[i, \alpha]) - q_t^{\text{ROW}}[\alpha]. \]

7. Send \( \{p_t^a, p_t^b, p_t^c, p_t^{\text{ROW}}\}_{\alpha \in [\lambda]} \) to \( \hat{V} \).

8. Answer any query \( \alpha \in L \) by \( \hat{V} \) by using \( F_{\pi}, F_a, F_b, F_c \) (as before) but for \( q_t^a, q_t^b, q_t^c, q_t^{\text{ROW}} \) use:

\[ \begin{align*}
q_t^a[\alpha] &= \hat{p}_t^a(\alpha) - \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_a[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha]), \\
q_t^b[\alpha] &= \hat{p}_t^b(\alpha) - \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_b[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha]), \\
q_t^c[\alpha] &= \hat{p}_t^c(\alpha) - \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_c[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha]), \\
q_t^{\text{ROW}}[\alpha] &= \hat{p}_t^{\text{ROW}}(\alpha) - \sum_{i=1}^{m_1} t_{i,\alpha} \cdot (F_a[i, \alpha] \cdot F_b[i, \alpha] - F_c[i, \alpha]).
\end{align*} \]

To see that the view of \( \hat{V} \) is perfectly simulated, we consider a hybrid experiment in which a “hybrid prover” sends actual codewords for the blinding vectors (like the honest prover in the real world) but can modify messages after they are sent (like the simulator in the ideal world).

1. Use the public input \( v \) to compute \( F_{\pi} \in \text{RS}[L, \frac{h}{t}, \frac{m_2}{t}] \) like the honest prover does.
2. Sample \( F_{\pi} \in \{0, 1\}^{m_2} \) and \( F_a, F_b, F_c \in \{0, 1\}^{m_1} \) uniformly at random. For every \( \ell \in [\lambda] \), sample \( q_t^a, q_t^b, q_t^c, q_t^{\text{ROW}} \in \text{RS}[L, \frac{2h+b-1}{t}] \) uniformly at random given that the interpolation of \( q_t^a, q_t^b, q_t^c \) sums to zero on \( H \). For every \( \ell \in [\lambda] \), sample \( q_t^{\text{ROW}} \in \text{RS}[L, \frac{2h+b-1}{t}] \) uniformly at random given that its interpolation vanishes everywhere on \( H \). Set \( F = (F_{\pi}, F_a, F_b, F_c, q_t^a, q_t^b, q_t^c, q_t^{\text{ROW}}) \), and start simulating \( \hat{V} \).
3. Use \( F \) to answer any queries by \( \hat{V} \). Let \( Q \subseteq L \) be the queries asked by \( \hat{V} \) until the next step.
4. Receive a challenge \( \{r_{i,1}, \ldots, r_{i,m_1}, t_i\}_{\alpha \in [\lambda]} \) from \( \hat{V} \).
5. For every \( \ell \in [\lambda] \), sample \( \hat{p}_t^a, \hat{p}_t^b, \hat{p}_t^c \in \text{RS}[L, \frac{2h+b-1}{t}] \) uniformly at random such that each of \( \hat{p}_t^a, \hat{p}_t^b, \hat{p}_t^c \) sums to 0 on \( H \) and, for every \( \alpha \in Q \), the following hold:
\[ \begin{align*}
\hat{p}_t^a(\alpha) &= \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_a[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha]), \\
\hat{p}_t^b(\alpha) &= \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_b[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha]), \\
\hat{p}_t^c(\alpha) &= \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_c[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha]).
\end{align*} \]
6. For every \( \ell \in [\lambda] \), sample \( \hat{p}_t^{\text{ROW}} \in \text{RS}[L, \frac{2h+b-1}{t}] \) uniformly at random such that \( \hat{p}_t^{\text{ROW}} \) evaluates to 0 everywhere on \( H \), and, for every \( \alpha \in Q \), the following hold:
\[ \hat{p}_t^{\text{ROW}}[\alpha] = \sum_{i=1}^{m_1} t_{i,\alpha} \cdot (F_a[i, \alpha] \cdot F_b[i, \alpha] - F_c[i, \alpha]) - q_t^{\text{ROW}}[\alpha]. \]
7. Send \( \{\hat{p}_t^a, \hat{p}_t^b, \hat{p}_t^c, \hat{p}_t^{\text{ROW}}\}_{\alpha \in [\lambda]} \) to \( \hat{V} \).
8. For every \( \ell \in [\lambda] \), replace \( q_t^a, q_t^b, q_t^c, q_t^{\text{ROW}} \) with the following codewords respectively:
\[ \begin{align*}
&\{\hat{p}_t^a(\alpha) - \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_a[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha])\}_{\alpha \in \ell}; \\
&\{\hat{p}_t^b(\alpha) - \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_b[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha])\}_{\alpha \in \ell}; \\
&\{\hat{p}_t^c(\alpha) - \sum_{i=1}^{m_1} \hat{r}_{i,\alpha}(\alpha) \cdot F_c[i, \alpha] + \sum_{i=1}^{m_2} \hat{s}_{i,\alpha}(\alpha) \cdot (F_{\pi}[i, \alpha] + F_{\pi}[i, \alpha])\}_{\alpha \in \ell}; \\
&\{\hat{p}_t^{\text{ROW}}(\alpha) - \sum_{i=1}^{m_1} t_{i,\alpha} \cdot (F_a[i, \alpha] \cdot F_b[i, \alpha] - F_c[i, \alpha])\}_{\alpha \in \ell}.
\end{align*} \]
9. Finish simulating the interaction with \( \hat{V} \).

The distribution of \( \hat{V} \)'s view in the real protocol is identical to the distribution of \( \hat{V} \)'s view in the above experiment. In particular, since \( \hat{V} \) makes at most \( b \) queries, the answers to its queries to \( F_a, F_b, F_c \) are uniformly random in the real world, and hence are perfectly simulated, and it is easy to check that its queries to \( q_t^a, q_t^b, q_t^c, q_t^{\text{ROW}} \) after their replacement by the new values have the correct distribution. Moreover, it is not hard to see that \( \hat{V} \)'s view in the above experiment and \( S \)'s output are identically distributed.

**Efficiency.** Both the prover and the verifier perform matrix multiplications, which take time proportional to the number of non-zero elements of the matrices. The prover also performs \( O(m_2 + m_1 + \lambda_i) \) FFTs.
over the systematic subspace $H$ (of size $h \leq l$) and the codeword subspace $L$ (of size $l$) to construct the codewords in $F$ and, later, also to construct the response polynomials. The verifier performs FFTs to evaluate the four response polynomials over $H$; then, after interpolating its challenges, the verifier also performs $O((m_2 + m_1)h)$ field operations for each interactive repetition and each query.

**Summary.** The aforementioned protocol is an RS-encoded IOP with the following parameters. One should think of $m_1, m_2, h$ as on the order of square root of the number of constraints/variables in the R1CS instance. To achieve soundness $2^{-\lambda}$, we can set $\lambda_i := \lceil \lambda / \log |F| \rceil + 1$ and $\lambda_q := \lfloor \lambda / \log(\frac{m_2 + m_1}{2^h - 2}) \rfloor + 1$ for example.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabet</td>
<td>$\Sigma := F$</td>
</tr>
<tr>
<td>Number of rounds</td>
<td>$k = 2$</td>
</tr>
<tr>
<td>Oracle length</td>
<td>$p = (m_2 + 3m_1 + 4\lambda_1)l$</td>
</tr>
<tr>
<td>Communication</td>
<td>$c = 4\lambda_1(8h + 5b - 4)$</td>
</tr>
<tr>
<td>Query complexity</td>
<td>$q = (m_2 + 3m_1 + 4\lambda_1)\lambda_q$</td>
</tr>
<tr>
<td>Randomness</td>
<td>$(r_i, r_q) = ((m_1 + 1)h\lambda_q \log</td>
</tr>
<tr>
<td>Soundness error</td>
<td>$(\varepsilon_i, \varepsilon_q) = \left( \left( \frac{1}{</td>
</tr>
<tr>
<td>Prover time</td>
<td>$t_P = O(|A| + |B| + |C|) + O(m_2 + m_1 + \lambda_i)\text{FFT}(F, l)$</td>
</tr>
<tr>
<td>Verifier time</td>
<td>$t_V = O(|A| + |B| + |C|) + O(m_2 + m_1 + \lambda_i)\text{FFT}(F, l) + O(\lambda_i \lambda_q (m_2 + m_1)h)$</td>
</tr>
</tbody>
</table>

### B.4 From encoded IPCP to regular IPCP

The IPCP for R1CS described in the prior section is RS-encoded because soundness assumes that the oracle sent by the prover is an interleaved Reed–Solomon codeword (a list of Reed–Solomon codewords over the same domain). Such a protocol can be transformed into a (regular) IPCP for R1CS by generically combining it with any low-degree test, as described in Section 9 for example. Informally, the verifier tests that a suitable linear combination of words in the oracle is close to the Reed–Solomon code. The low-degree test that we use here is the direct one, namely, the prover sends, as a message, the coefficients of the polynomial that interpolates the linear combination, and the verifier probabilistically checks that this polynomial is consistent with the oracle; this subroutine corresponds to the “Test-Interleaved” protocol in [AHIV17].

This compilation has an impact on several parameters, which we now sketch. Let $\lambda_q^{\text{LDT}}$ denote the number of linear combinations that the verifier considers, and let $\lambda_q^{\text{LDT}}$ denote the number of times that the verifier repeats the consistency check. Let $\ell := m_2 + 3m_1 + 4\lambda_1$ be the number of words in the oracle.

Randomness complexity in the interactive phase increases by $\lambda_q^{\text{LDT}} \ell \log |F|$ and in the query phase by $\lambda_q^{\text{LDT}} \log l$; query complexity increases by $\ell \lambda_q^{\text{LDT}}$; communication complexity increases by the maximum degree (plus 1) across all words. Soundness error in the interactive phase increases by $(\frac{1}{|F|^2})^{\lambda_q^{\text{LDT}}}$ (this is a fairly coarse bound) and soundness error in the query phase becomes $(\varepsilon_q + \delta^{\text{LDT}}) \lambda_q + (1 - \delta^{\text{LDT}}) \lambda_q^{\text{LDT}}$ for any proximity parameter $\delta^{\text{LDT}} < \frac{1}{4^2}$ (where $\rho := \frac{2h + 2b - 2}{|F|^2}$ is the maximum codeword rate). The choice of $\delta^{\text{LDT}}$ balances the probability of a non-codeword oracle being caught by the low-degree test with its ability to cheat on the protocol’s tests; an optimal value, to minimize overall soundness error, can be found numerically for given choices of the other parameters.
Both prover and verifier compute:
- $(s_1, \ldots, s_{m_2}) := (r_1, \ldots, r_{m_1})^T M$
- $\forall i \in [m_1], \tilde{r}_i \text{ s.t. } \hat{r}_i | H = r_i$
- $\forall i \in [m_2], \tilde{s}_i \text{ s.t. } \hat{s}_i | H = s_i$

Compute the polynomial 
\[
\hat{p} = \sum_{i=1}^{m_1} \tilde{r}_i \cdot \hat{f}_{x,i} - \sum_{i=1}^{m_2} \tilde{s}_i \cdot \hat{f}_{y,i}
\]

Sample $\alpha_1, \ldots, \alpha_{m_1} \leftarrow L$ and check that $\sum_{\alpha \in H} \hat{p}(\alpha) = 0$ and $\forall k \in [\lambda_q]$
\[
\hat{p}(\alpha_k) = \sum_{i=1}^{m_1} \tilde{r}_i(\alpha_k) \cdot F_x[i, \alpha_k] - \sum_{i=1}^{m_2} \tilde{s}_i(\alpha_k) \cdot F_y[i, \alpha_k]
\]

Figure 13: Diagram of the interleaved lincheck protocol.

Compute the polynomial 
\[
\hat{p} = \sum_{i=1}^{m} t_i \cdot (\hat{f}_{x,i} \cdot \hat{f}_{y,i} - \hat{f}_{z,i})
\]

Sample $\alpha_1, \ldots, \alpha_{m} \leftarrow L$ and check that $\hat{p}(H) = \{0\}$ and $\forall k \in [\lambda_q]$
\[
\hat{p}(\alpha_k) = \sum_{i=1}^{m} t_i \cdot (F_x[i, \alpha_k] \cdot F_y[i, \alpha_k] - F_z[i, \alpha_k])
\]

Figure 14: Diagram of the interleaved rowcheck protocol.

Sample $\alpha_1, \ldots, \alpha_{m} \leftarrow L$ and $\forall k \in [\lambda_q]$
\[
\hat{p}(\alpha_k) = \sum_{i=1}^{m_1} \tilde{r}_i(\alpha_k) \cdot (F_{x,i}[\alpha_k] + F_{y,i}[\alpha_k] - F_{z,i}[\alpha_k]) + \sum_{i=1}^{m_2} \tilde{s}_i(\alpha_k) \cdot (F_{x,i}[\alpha_k] + F_{y,i}[\alpha_k] - F_{z,i}[\alpha_k])
\]

Figure 15: Diagram of the interleaved R1CS protocol.
C Additional comparisons

We provide additional comparisons across Ligero, Stark, and Aurora: in Appendix C.1 we compare the low-degree tests that they rely on, and in Appendix C.2 we compare their underlying IOP protocols.

C.1 Comparison of the LDTs in Ligero, Stark, and Aurora

A key ingredient in Ligero [AHIV17], Stark [BBHR18a], and Aurora (this work) are low-degree tests (LDTs). Formally, each of these systems relies on an IOPP for the Reed–Solomon relation (see Section 4.5.1). The LDT is then generically “lifted” to an LDT for the interleaved Reed–Solomon code (see Section 4.1), by taking a random linear combination as in Section 9. Below (and in Fig. 16) we discuss aspects of the LDTs underlying these systems that are important in the comparison in Appendix C.2.

Direct LDT. Ligero uses a *direct* LDT: the verifier is given oracle access to a function $f: L \rightarrow \mathbb{F}$, receives from the prover $a_0, \ldots, a_{\rho|L|−1} \in \mathbb{F}$ (allegedly, coefficients of the polynomial $\hat{f}$ obtained by interpolating $f$), and checks that $f$ and $\sum_{i=0}^{\rho|L|−1} a_i X^i$ agree at a random point of $|L|$. If $f$ is $\delta$-far from RS $[L, \rho]$ then the verifier accepts with probability at most $1 - \delta$. This probability can be reduced to $(1 - \delta)^t$ via $t$ independent checks. Overall, the verifier queries $f$ at $t$ points, and reads $\rho|L|$ field elements sent by the prover. One should think of $t$ as much less than $\rho|L|$, which facilitates lifting to an LDT for the interleaved Reed–Solomon code.

FRI LDT. Stark and Aurora use FRI [BBHR18b], a LDT in which the verifier is given oracle access to a function $f: L \rightarrow \mathbb{F}$ and, in each of a sequence of rounds, sends a random field element to the prover, who replies with an oracle; at the end of the interaction, the verifier makes a certain number of queries to $f$ and the oracles, and then either accepts or rejects. (The domain $L$ here is an additive or multiplicative coset in $\mathbb{F}$ whose order is a power of 2.) In more detail, given a localization parameter $\eta \in \mathbb{N}$, the number of rounds is $\frac{\log \rho|L|}{\eta}$, and in the $i$-th round the prover sends an oracle over a domain of size $|L|/2^{\eta i}$; thus, the total number of elements sent across all oracles is less than $\sum_{i=1}^{\infty} |L|/2^{\eta i} = |L|/(2^\eta + 1)$. After the interaction, the verifier queries $f$ at a point, and every other oracle at $2^\eta - 1$ points; given the corresponding answers, the verifier performs $O(2^n \log \rho|L|)$ arithmetic operations, and then accepts or rejects. If $f$ is $\delta$-far from RS $[L, \rho]$ then the verifier accepts with probability at most $\varepsilon(\delta) := \varepsilon_1 + (1 - \min\{\delta, \delta\})^t$ for certain values of $\varepsilon_1$ and $\delta_\rho$. In [BBHR18b] it is proved that $\varepsilon_1 = 3|L|/|\mathbb{F}|$ and $\delta_\rho = (1 - 3\rho - 2^n|L|^{-1/2})/4$; in [BKS18] this was improved to $\varepsilon_1 = 2 \log |L|/\epsilon^3 |\mathbb{F}|$ and $\delta_\rho = \epsilon J_\epsilon(J_\epsilon(1 - \rho) - \epsilon \log |L|)$ for any $\epsilon > 0$, where $J_\epsilon(x) := 1 - \sqrt{1 - x}(1 - \epsilon)$. In [BBHR18b] it is conjectured that the best possible values are $\varepsilon_1 = 2^n \log^2(|L|/\epsilon \eta^2 |\mathbb{F}|)$ with $\delta_\rho = 1 - (1 - \epsilon)\rho$ for any $\epsilon > 0$.

C.2 Comparison of the IOPs in Ligero, Stark, and Aurora

Each of Ligero [AHIV17], Stark [BBHR18a], and Aurora (this work) is a (zero knowledge) IOP that is compiled into a (zero knowledge) SNARG via a transformation of Ben-Sasson et al. [BCS16]. Comparing these SNARGs (essentially) reduces to comparing the underlying IOPs, which we do below.

Construction blueprint. The IOPs in the aforementioned systems can all be viewed as combining an encoded IOP (as defined in Section 4.7) and a low-degree test (as defined in Section 4.5.1), via the transformation described in Section 9. Informally, this transformation invokes the low-degree test on a suitable random linear combination of the oracles sent by the encoded IOP prover (more generally, of “virtual” oracles implied by these), thereby ensuring that the codeword obtained by stacking these oracles is close to the interleaved Reed–Solomon code (more generally, a codeword obtained by applying a transformation to these oracles is close to the interleaved Reed–Solomon code); one can then reduce to soundness of the encoded IOP.
<table>
<thead>
<tr>
<th>LDT</th>
<th>number of queries</th>
<th>soundness error</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct</td>
<td>$t \cdot \rho</td>
<td>L</td>
</tr>
<tr>
<td>FRI</td>
<td>$t \cdot (2^\eta - 1) \cdot \log \rho</td>
<td>L</td>
</tr>
</tbody>
</table>

Figure 16: Parameters of the direct low-degree test and FRI low-degree test when invoked on a function $f: L \rightarrow \mathbb{F}$ that is $\delta$-far from RS $|L, \rho| \subseteq \mathbb{F}^k$. Note that $\delta$ always lies in $[0, 1 - \rho]$.

<table>
<thead>
<tr>
<th>IOP</th>
<th>relation</th>
<th>number of queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stark</td>
<td>APR</td>
<td>$q_{\text{FRI}}(</td>
</tr>
<tr>
<td>Ligero</td>
<td>R1CS</td>
<td>$q_{\text{DIR}}(4(h + 1), 2m/h, \rho)$</td>
</tr>
<tr>
<td>Aurora</td>
<td>R1CS</td>
<td>$q_{\text{FRI}}(6, 3m + 2b, \rho)$</td>
</tr>
</tbody>
</table>

Figure 17: Aspects of the IOPs underlying Stark, Ligero, and Aurora.

For a given soundness error, the query complexity of an IOP constructed via the blueprint above is determined by the query complexity of the underlying low-degree test, while (typically) the prover and verifier complexities are dominated by the encoded IOP’s prover and verifier complexities.

**The three IOPs.** In light of the foregoing blueprint, we describe the differences across the three IOPs by discussing the differences across the respective encoded IOPs and low-degree tests (see Fig. 17). Recall that $b$ denotes the query bound for zero knowledge (as defined in Section 4.6); the bound is later set to equal the number of queries of the honest verifier. Moreover, for notational simplicity, below we use $q(k, d, \rho)$ to denote the query complexity of a low-degree test invoked on a function $f^*: L \rightarrow \mathbb{F}$ derived entry-wise from $k$ oracles $f_i: L \rightarrow \mathbb{F}$ sent by the encoded IOP prover, with each oracle (allegedly) having degree less than $d = \rho |L|$; using a low-degree test in this way follows the general paradigm described in Section 4.1.

- **The IOP in Aurora.** The IOP in Aurora is obtained by combining an encoded IOP for R1CS (described in Section 8) and the FRI low-degree test. Given an R1CS instance with $m$ constraints, the IOP invokes the low-degree test on 6 oracles having maximal degree $3m + 2b$, resulting in $q_{\text{FRI}}(6, 3m + 2b, \rho)$ queries.

- **The IOP in Ligero.** The IOP in Ligero (adapted for R1CS) is obtained by combining an encoded IOP for R1CS and a direct low-degree test (see Appendix C.1). Given an R1CS instance with $m$ constraints and for a parameter $h \approx \sqrt{m}$, the IOP invokes the low-degree test on $4(h + 1)$ oracles of maximal degree $2m/h$, resulting in $q_{\text{DIR}}(4(h + 1), 2m/h, \rho)$ queries.

- **The IOP in Stark.** The IOP in Stark natively supports Algebraic Placement and Routing (APR), which is the following problem: given a finite field $\mathbb{F}$, subset $H \subseteq \mathbb{F}$, algebraic registers $\mathcal{R}$, neighbors $\mathcal{N}$, and set of polynomial constraints $C$, are there functions $w = (w_i: H \rightarrow \mathbb{F})_{i \in \mathcal{R}}$ such that for every element $\alpha \in H$ and every constraint $c \in C$ it holds that $c(\alpha, (w_i(f(\alpha)))_{(i, f) \in \mathcal{N}}) = 0$? (See [BBHR18a] for details.)

The IOP in Stark is obtained by combining an encoded IOP for APR and the FRI low-degree test. The latter is used twice: once on $|\mathcal{R}| + 1$ oracles of maximal degree $4(|H| + b)$; once on $|\mathcal{N}| + 1$ oracles of maximal degree $D(|H| + b)$. This results in $q_{\text{FRI}}(|\mathcal{R}| + 1, 4(|H| + b), \rho) + q_{\text{FRI}}(|\mathcal{N}| + 1, D(|H| + b), \rho)$ queries.
Acknowledgments

We thank Alexander Chernyakhovsky and Tom Gur for helpful discussions, and Aleksejs Popovs for help in implementing parts of libiop. This work was supported in part by: the Ethics and Governance of Artificial Intelligence Fund; a Google Faculty Award; the Israel Science Foundation (grant 1501/14); the UC Berkeley Center for Long-Term Cybersecurity; the US-Israel Binational Science Foundation (grant 2015780); and donations from the Interchain Foundation and Qtum.

References


Chapter 4

Discussion

Computational-integrity (also known as checking-computation [9], verifiable-computation [31, 49], and computation-delegation [37]) protocols eliminate the need to trust reporters of computation results, by allowing the latter to provide a cryptographic proof attesting to the validity of the results. The proof verification time is polylogarithmic with the length of the computation (succinct), introducing significant improvement over the naive solution for general computations, which is merely re-execution. Hereafter we refer to the reporter of the computation results as the prover, and the verifier of the results simply as the verifier. In a special case of such proof systems, the prover can attest to possessing knowledge of a secret input, such that when provided to the program, the result is some reported value (e.g., acceptance or rejection). In this case the proof does not reveal the prover’s private input.

Computational-integrity protocols were introduced around three decades ago [9]. This original work offered theoretical solutions to practical problems. An immediate practical motivation for taking such an approach comes from a different field of mathematical-proofs done by computer such as the celebrated example of [8] showing that any planar graph is four-colourable. The method used in [8] included a computer exhaustively verifying correctness of a finite set of graphs, requiring work beyond what is humanly achievable, and took over a thousand hours to complete using the computer. Obviously, anyone who did not want to repeat this heavy verification process had to put their trust in the researcher who claimed to execute it. This kind of a proof to a mathematical claim was novel, and the fact the proof cannot be easily verified by peer reviews disappointed many. Computational-integrity is a natural solution to such problems, as it can be used by researchers to generate a succinct proof for the integrity of any computation, verifiable by anyone. Unfortunately, the work of [8] predated the initial ideas of verifiable computation, introduced in [9], by roughly a decade. Even though roughly three decades have passed since the introduction of [9], it is still probably infeasible to use modern computational-integrity for problems of such complexity. Although there are implementations of succinct verifiers that can succinctly verify the correctness of a small proof, the computation overhead on the prover, compared to the naive computation, is still too high.

The line of research work we present shows what we have done in order to advance the field of computational-integrity from theory to practice. In Computational Integrity with a Public Random String from Quasi-Linear PCPs (aka SCI), we introduced the first proof-of-concept implementation of a system

1. having quasi-linear proving time, compared to naive execution time,
2. enabling polylogarithmic time verification with naive execution time, aka is succinct, and
3. being publicly-verifiable without a trusted-setup, aka *is transparent*.

In addition to the above, SCI is plausibly post-quantum secure. While quasi-linear proving time introduces asymptotic advantages over various older theoretical constructions, the uniqueness of SCI among other implementations of computational-integrity proof systems is that it is both succinct and transparent.

While SCI introduced a proof-of-concept having great asymptotic properties, the concrete efficiency of SCI was far from practical. To list the costs of computational-integrity systems we must start with understanding the high-level components used in proving and verifying computational-integrity and their effect on system performance.

1. The first component describes the computation by delineating an algebraic constraint system.
2. The second component is the proof system, which is expected to be provided with a description of the constraints, and an additional witness in the case of the prover, where the prover convinces the verifier it knows a witness satisfying the constraints.

In SCI, the proof system expects an algebraic constraint system as input. Even though there are some well-known constructions translating computations into algebraic constraint systems, they are designed to show theoretical results, thus providing only asymptotic efficiency. Moreover, to gain concrete efficiency in a specific proof system, general transformations are often inferior to solutions designed with the aim to optimize the specific system used. In SCI, we have embedded a compiler, from a dedicated assembly language called TinyRAM [20], for the algebraic constraint system used by the proof system. The usage of an assembly-like language is essential for programmers to both describe computations they want to prove or verify, and understand descriptions shared with them, as the algebraic constraint systems are far from native for the average programmer, and analysis of its soundness and completeness often requires familiarity with advanced algebraic structures. Unfortunately, the compiler, although fine-tuned for the proof system used in SCI, introduces great overheads compared to fine-tuned descriptions using algebraic constraint systems directly.

In following publications we focused on improving the properties of the proof system, measuring performance with fine-tuned manually constructed algebraic constraints to represent computations, and ignoring the development of compilers from standard programming languages, as each variant of the proof system introduces different cost models, implying different compiler requirements. Separation of the compilation logic from the proof system allowed us to focus on improving the proof system protocol, and delay research into compilation from common programming languages. Thus, we were able to reach the stage where we had efficient algebraic representation and a proof system having pleasing concrete efficiency, applicable to industrial needs, at least for some fine-tuned manually written algebraic representation.

On the other hand, when introducing a succinct computational-integrity solution, the natural approach to measure its concrete efficiency is to compare the proving and verification time to the time of the naive execution of the computation. Moreover, the choice of which computation to measure should be as neutral as possible, not introducing any specific advantages to the specific proof system used and representing the general case. That is, algebraic proof systems are best when proving algebraic statements, which is far from being the general case.

Motivated by neutrality, we benchmarked SCI by writing the benchmarks in TinyRAM and compiling them to the algebraic constraint system, passing the constraints over to the proof system,
with the performance measurements presented in Figures 1, 2, and 3. In Figure 1 we can see that the overhead introduced by proving, compared to the naive computation, is approximately eight orders of magnitude. When implementing STARK (Scalable, transparent, and post-quantum secure computational integrity), in addition to applying various theoretical improvements, we measured a statement represented directly in Algebraic Intermediate Representation, or simply AIR, and were able to achieve an overhead of only four to five orders of magnitude, compared to the naive execution (Figure 6).

The recent rise of blockchains has emphasized the need for efficient computational-integrity. There are several types of blockchain implementations and, for simplicity, we will focus on the most common one, which is a public blockchain with consensus strategy using proof-of-work (aka PoW). A blockchain can be described syntactically as a public append-only log, where anyone in the world is allowed to append to it, as long as the added entry obeys the blockchain’s semantic rules. For example, there could be a blockchain used as a payment system, where the syntax of its records describes transactions, and the semantic rules require the transaction to be valid by

1. it being signed by the payer, and
2. the payer having enough funds to execute the transaction.

This type of blockchain is very similar to the well-known Bitcoin blockchain and their popular blockchains having more complex semantic rules, e.g., the Ethereum blockchain that has semantic rules as hard to verify as when executing a general program. Notably, integrating solutions using computation-integrity protocols could reduce the load on verification, by publishing proofs for semantic correctness of records to the network. Moreover, when using proofs featuring zero-knowledge, privacy features are possible, which are impossible when semantic verification is done explicitly.

To enforce the validity of each added entry to the blockchain, it must be verified by all peers, amplifying the global cost of verifying semantic correctness by orders of magnitude. Moreover, in many blockchains the verification requires accessing older entries in the blockchain, resulting in the need to keep the entire history, introducing additional resource consumption on the network peers, sometimes as big as a few hundreds of GB, e.g., the Bitcoin blockchain has to be stored entirely and consumes more than 200GB. In addition to these semantic verification related issues, there is a less obvious issue derived from its syntactical definition. As the entries in the blockchain are required to have a well-defined order, there is a need to overcome concurrency related issues. Since many peers may try to add different entries simultaneously, a consensus mechanism to define exact order, accessible to all the peers in the network, is needed. A common method used for implementing such consensus mechanisms is to add the entries to the blockchain in rounds, where in every round, one peer is chosen at random and is eligible to add a block of ordered entries, usually called miner, while the rest of the peers are required to verify the validity of the added block. In PoW based consensus systems, miners are chosen consecutively by being the first peer able to solve a probabilistically-hard riddle defined by the previous block, e.g., finding a value \( x \), usually called nonce, such that \( \text{SHA2}(x||r) \) has many leading zeroes, and \( r \) is the hash of the previous block. As there is a need for all peers to agree on each block before its successor is defined, the riddle has to be challenging enough so it is not solved before the last block is propagated through the entire network.

For the sake of fairness, as in many networks, the miner of each round is rewarded, the time it takes for the block to propagate through the entire network should be negligible compared to the
difficulty of the riddle, e.g., in Bitcoin, the difficulty of the riddle is calibrated to be solved once every 10 minutes, and the size of each block is 1MB. This constraint introduces a physical limitation on the throughput of the network, as the size of each block affects the time it takes to propagate nearly linearly, thus nearly linearly affecting the rate of blocks, e.g., Bitcoin is limited to seven transactions per second. Such limitations make it more difficult to adopt such systems as global payment systems to be used for common payments. This issue is often referred to as the blockchains scalability problem. Computation-integrity protocols can be used to scale a blockchain by passing succinct proof-of-knowledge of a block of valid entries, where the block may be significantly larger than the proof transferred. While such solutions may overcome the rate of the network syntactically, they introduce semantic challenges, as the lack of access to the explicit entries from previous blocks could prevent one from being able to generate a new valid block.

As succinctness is required to solve scalability problems as mentioned above, and given the fact no practical transparent implementation has to date been published, the use of systems requiring a trusted setup started to spread. In a trusted setup system, there is a trusted dealer, who constructs public pair of keys, a proving key and a verification key. The key generation process reveals additional information that can be used as a backdoor, allowing proofs to any statement to be forged.

The dealer is trusted to destroy this additional information, often related as ‘toxic waste’. Advanced methods were suggested to generate the keys in a distributed manner, where the backdoor can be revealed only if all participants agree to collaborate to extract this information, moving the trust assumption from a single dealer to a group where it is sufficient to assume at least one of its members is honest.

A well-known adapter of a trusted setup based system is ZCash [1], a pioneer in the industrial usage of computational-integrity proof systems, basing its system on [51]. The target of ZCash is to provide privacy for payments over the blockchain, using the zero-knowledge features of their proof system. While in blockchains such as Bitcoin, the entries are public and used to explicitly verify their validity against the blockchain’s state, in ZCash the entries are encrypted and accompanied by a proof of computational-integrity showing knowledge of the unencrypted form, and its validity against the blockchain’s state. Although ZCash made every effort to apply best practices in the distributed keys generation process, leaks of sensitive information from the setup phase became public [35]–which is strong evidence of the need for a transparent proof system, especially when industrial-grade security is a requirement.

Zero-knowledge is an important property of several computational-integrity systems, especially in cases where these are used to implement industrial systems featuring advanced privacy properties. Unfortunately, SCI does not feature zero-knowledge, but a transformation described in [17] does provide perfect witness indistinguishability. Although witness indistinguishability is good enough for any practical usage, the theoretical property of zero-knowledge is preferable, and was not reached in [17] due to the hardness of constructing a succinct simulator for low-degree testing of [24]. In **Zero Knowledge Protocols from Succinct Constraint Detection**, we introduced succinct constraint detection and used fine analysis of [24], showing that the construction of [17] is perfect zero-knowledge. While it can be initially counterintuitive, proving zero-knowledge against succinct verifiers is harder than proving zero-knowledge against stronger verifiers, even though the formal definition implies this, given that proving a construction of a succinct simulator is obviously more challenging, or maybe even impossible. To understand this difficulty, one can compare trying to prove some statement, e.g., graph 3 colorability, in zero-knowledge to two independent parties, one
bounded only by polynomial computations, while the other can execute exponential computations as well. The exponentially bounded verifier can verify the correctness of the statements with no external help. Thus, interaction with a prover should not teach the verifier anything the verifier cannot do by itself. Moreover, the trivial protocol of explicitly sending a valid coloring is a zero-knowledge protocol. On the other hand, for a protocol to be zero-knowledge against the polynomial verifier, it has to be designed carefully to prevent leakage of knowledge the verifier could not gain by itself.

In Interactive Oracle Proofs with Constant Rate and Query Complexity, we showed asymptotic improvement in proving time for the succinct sumcheck problem. Unfortunately, this solution is based on algebraic-geometry codes, which require a heavy computational overhead. Fast Reed-Solomon Interactive Oracle Proofs of Proximity and Scalable, Transparent, and Post-Quantum Secure Computational Integrity, (aka FRI and STARK), provide a solution that can be used in practice, with a proving overhead in the order of $10^4$ to $10^5$ (STARK, Figure 6), verification over performing naïve computation (STARK, Figure 1), and featuring zero-knowledge.

SCI and STARK are systems where the verifier's computational power grows polynomially with the size of the program, and polylogarithmically with the program's execution time. Notably, this provides a great advantage to programs where a small portion of code is repeated many times, e.g., using loops. Given this property the major design goal with both these systems was to optimize them for this kind of programs. Although the property of succinct verification is important, there are many cases where the program does not fall into this category (e.g., circuits). Thus, computational succinctness is not achievable, but, nevertheless, the properties of zero-knowledge and succinct communication-complexity are required. For such cases we designed Aurora: Transparent Succinct Arguments for R1CS, based on ideas from STARK, providing an efficient solution for programs described as circuits.

We continue with a survey on industrial usage of zero-knowledge proofs. All listed projects are related to the blockchain industry and were targeted at solving various issues arising in this field. The different projects can be classified as belonging to one of three categories, based on the part they play in the different protocols:

1. **Trusted setup phase:** This category contains industrial usage of computational-integrity proof systems for sound parameter initialization. One of the most notable projects in this area is presented by Ligero [2], aiming to develop an MPC protocol for secure computation of an RSA modulus, to be used as a group of unknown order. The need in such a group is to develop a verifiable delay function (or VDF) to be used as part of the consensus protocol of the blockchain Ethereum 2.0. The main challenge is to develop such a protocol to be both practical and support thousands of participants. The team of [2] report using the Ligero zero-knowledge proof system [7] to compile an MPC protocol supporting passive malicious parties, to support active malicious parties as well.

2. **Embedded into blockchain protocol:** This category contains computational-integrity protocols used as a fundamental part of the blockchain protocol. Two notable projects in this category are ZCASH [1], and CODA [3]. Both these projects use computational-integrity constructions similar to [20]. [1] is a blockchain centered on privacy, and uses the zero-knowledge property to support encrypted records in a blockchain. The proof is used to convince the blockchain verifiers that the record, even though encrypted, does not contradict any network rule, and changes the state correctly. [3] uses a computational-integrity proof system for a completely
differen reason—succinct verification and recursive proving. The problem [3] proposes to solve is the one of the high resources required to keep the integrity proof of the blockchain state (e.g., balances for financial usage blockchains), which is the entire blockchain in standard systems. It is sugested by [3] to generate a succinct computational-integrity proof for each state-change commitment (e.g., using a Merkle tree). The recursive proving propery allows generation of each new proof, based only on the previous proof, and knowledge of the previous state.

3. applications over the blockchain protocol: The last category includes applications using the blockchain infrastructure. Notable projects in this category were developed (in addition to many more) by StarkWare Industries [4] and Matter Labs [5]. Both projects aim to improve the scalability of various applications over blockchain, focusing on financial use cases such as payments and self-custodial exchanges. In both cases, there are central parties responsible for data storage and manipulating it using various rules, similarly to classical services. The main difference from classical services is the fact that a commitment to the database is stored on the blockchain, and each change to the commitment requires a computational-integrity proof attesting to the validity of the update. Both projects aim to improve the scalability of financial transactions passed through their systems, without taking custody of the client funds.

In conclusion, we can see how the field of computation-integrity with zero-knowledge has emerged from the theoretical research, moving into practical ventures and day-to-day usage. It is a very interesting era to be working in this field. It is indeed possible that a decade from now such technology will be integrated into essential services, enabling many fantastic opportunities.
Bibliography

[1] [Online]. Available: https://z.cash/


263
פרוטוקול עבור יושרה חישובית הוא זוג אלגוריתמים אינטראקטיביים, אחד מהם נקרא מוכיח, והשני נקרא מוודא. שני הצדדים, המ八大以来 והמוודא, מסכימים על תוכנית מחשב מסוימת מראש, אשר אם הוכחתו, התוכנית עוצרת לאחרúmero מסוים של מציאותי, ובלבדה או מת绌. ב𝖝ה שעון, המוכיח ישליח למוודא את המなの, והמוודא יתבצע פעולות על תוכנית המכלולה באמצעות המ.'<n> מתכון המתייחס של💙更多信息 על ניסיון í ניסיון יומר על מודעות ומענה של תמודאה או על מקום הmoודא שלים בעיות החישובית

בעולם בו אנו חיים, אנו מאפשרים לאירגונים רבים לנהל מידע עבורנו, ולדווח לנו על תוצאות חישובים וחיוניים עבורנו שלא נפלו בהם טעות, או חמור מכך, לא שונו מ騙ים זדניים. דוגמאות לכך מגיעות הן מחשב הענן שהכתיב לבנUIApplication הוצב על גביו להרבה ארגונים, ועד לבנקים והחברות它们ようにNous נסובים, אך עם זאת,iami ומקומיות circם את ההשקפת של המ noun והנו, וכו'.

ובנוסף להתקשרות בין עם צעירים המבוקשים, אלגודות,ﻝגבוב סיפס קחשמונדות. בעד

בשל トラון המוודא הלילה, מרחיק_templates שלéta של מתאי ביו

אף על פי כן, במובנה של תחרות מתאימה, בבית של çiftים עולים אם המอนาคה של כל שיתוף

ברקע, בקשת נהוגה העיכבות של מ鼱וגים שונים, ניתן לייעל את התיאוריה של עםAWN klשיים:

בשוף העשיה השמינית של המורה הקומדית, לא ברז מיווש המשמשים את אופנים של תבניות וברבותה.

בעינה העיכובים של המיצוגים המочкиים עליות האטרוסים, ודואג למיצוג המочки מגוון, אך לא

באנר הקישון של עליית האטרוסים, אגי צור המочек ייבוא את ההשקפים המכרים, ודבר זה

מא strdupים השניםنشر יבר חשים קרין בברית. עב ראש, ובוונת האטרוסים בין עליות, הם מוסשת

איסמיטיפניא משקפת את בנית מורכבות השוית, שיום הריצה של שימש תמית שלמה לא יפישר

שימש תמית.

מתקנים

המחוק שלוג לשולחןbbing לרשראפ שימש יו-יומתיו,娀 וישראלי, ומכלכראופרה שימש יונתן-יוויאת. במקורה

המחוק שלוג של לטענת לרשראפ שימש יו-יומתיו,娀 וישראלי, ומכלכראופרה שימש יונתן-יוויאת. במקורה

לגרון, מחוקים יוני מחוק שברולטת לברך יוני מחוק עשה淇, אך המדע שיכחת, לקח אלו

מותיפהות והן לטרות לטרות של כוכב, ההבעיות קליות, רָנָּג, קליות מחזורי, ואל מוודא. את הממחוקים

מותיפהות והן לטרות לטרות של כוכב, ההבעיות קליות, רָנָּג, קליות מחזורי, ואל מוודא. את הממחוקים

לתרונומיה, בتراث אל מתרגפש אומן איצן דוי שילושי.

בישון האגרות עליה פופולית מערכות גמיסות בולטים, הבור חוסם האומן באנדר אשיניה.

דרישת מרכזיות, מחלוקיימ יושרי השושבה עד הכפיפה ליד לב יער מערכות שטח.
הניחה

המחקר נעשה בהנחיית פרופסור אליהו בן-ששון ופרופסור יובל ישי בפקולטה למדעי המחשב.

תמיכה כספית

אני מודה לטכניון, הפקולטה למדעי המחשב, מלגת הסייבר מל ברלין והמלגה ע"ש נחמיה לבציון ותת על התמיכה הכספית הנדיבה בהשתלמותי.
יוושרו חישובית להמונים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר דוקטור לפילוסופיה

מיכאל ריאבצב

הוגש לסנט הטכניון - מכון טכנולוגי לישראל

2019
תשרי תש"פ, חיפה, אוקטובר