# Scalable, transparent, and post-quantum secure computational integrity

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

February 19, 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 naïvely.

*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

**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 ($\mathbf{p}$) 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 $\mathbf{p}$, 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" $cm_{\mathbf{t}}$ for the state at time $\mathbf{t}$ of a large dataset $D_{\mathbf{t}}$ [33]. Typically $cm_{\mathbf{t}}$ is negligible in length[2] compared to $D_{\mathbf{t}}$, 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 $|cm_{\mathbf{t}}|$ (the bit-length of $cm_{\mathbf{t}}$), rather than like $T_C$ and $|D_{\mathbf{t}}|$; at the very least verification time/communication should be strictly less than $T_C$ and $|D_{\mathbf{t}}|$.

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 *polynomially* compared to naïve computation time ($T_P = T_C^{\mathbf{O}(1)}$) whereas verification time decreases *exponentially* with respect to it ($T_V = \log^{\mathbf{O}(1)} T_C$).

A ZK system based on the PCP Theorem (ZK-PCP) [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 [74] for interactive solutions, and common access to a random function[6] (the "random oracle model" [50]) for non-interactive ones [85] — are not known to be susceptible to attacks by large-scale quantum computers; we call such solutions *post-quantum secure*. The anticipated increase in scale of quantum computers [43] and the call for post-quantum cryptographic protocols, e.g., by the USA National Institute of Standards and Technology (NIST) [37], highlight the importance of a post-quantum secure ZK solution.

Second, ZK-PCPs are *proof of knowledge* (POK) systems, or, when realized as described above, *argument of knowledge* (ARK) systems [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 $\mathbf{D_t}$ and Presidential Candidate DNA profile $\mathbf{p}$ whose commitments were previously announced (see Definition 3.3).

Third, and most important, ZK-PCPs are *transparent* (or "public randomness"[7]), which means that the randomness[8] used by the verifier is *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 reliantly 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) *transparency*, (ii) *universality* — apply to any efficient computation C, even if it requires auxiliary (and possibly confidential) input like $\mathbf{D_t}$ above, (iii) *confidentiality (ZK)* — do not compromise auxiliary inputs like $\mathbf{D_t}$, (iv) *post-quantum security*, (v) *proof/argument of knowledge* and (vi) *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 [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 [22, 94], called an *interactive oracle proof (IOP)*[9], a common generalization of the IP, PCP, and interactive PCP (IPCP) models [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 [22], which is typically realized using a standard hash function. As a strict generalization of IP/PCP/IPCP, the IOP model offers several

[6] 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 [10] and following works.

[7] Transparent systems are also known as *Arthur-Merlin* protocols [4].

[8] Randomness is necessary for ZK proof systems for non-trivial computations [57, Section 3.2].

[9] Reingold et al. [94] use the name "Probabilistically Checkable Interactive Proofs" (PCIP).

advantages. Most relevant to this work is the improved *prover scalability* of IOPs, described below; this advantage holds both asymptotically — as input size $\mathbf{n} \to \infty$ (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[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 $T_C \cdot \log^{\mathbf{O}(1)} T_C$; 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 naïve 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)[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 $\mathbf{t}$ has posted (say, on a block-chain) a hiding commitment $\mathrm{cm}_{\mathbf{t}}$ to the state $D_{\mathbf{t}}$ of the database at that point in time. The Police now claims that the DNA profile $\mathbf{p}$ of the soon-to-be-appointed and alleged-to-be-corrupt Presidential Candidate, does not appear in $D_{\mathbf{t}}$ 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 $\mathbf{p}$ and $D_{\mathbf{t}}$).

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 $\mathrm{cm}_{\mathbf{t}}$ to the state $D_{\mathbf{t}}$ of a CODIS database is assumed to be public information (say, published at time $\mathbf{t}$ on a blockchain), as is a commitment $\mathrm{cm}_{\mathbf{p}}$ to the profile

[10] https://github.com/elibensasson/SCI-POC

[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 $\mathrm{cm}_{\mathbf{p}}$ publicly. Assume that the Police declares

*"The value is the result of the match search for the profile with commitment* $\mathrm{cm}_{\mathbf{p}}$ *in the database with commitment* $\mathrm{cm}_{\mathbf{t}}$*"* (*)

The answer 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 — $\mathrm{cm}_{\mathbf{t}}$; $\mathrm{cm}_{\mathbf{p}}$ and **A** — and two confidential inputs: (i) a DNA profile database $D^0$ and individual DNA profile $\mathbf{p}^0$. The computation C terminates successfully if and only if the public inputs ($\mathrm{cm}_{\mathbf{t}}$; $\mathrm{cm}_{\mathbf{p}}$; **A**) and the confidential ones ($D^0$; $\mathbf{p}^0$) satisfy three conditions: (i) the commitment $\mathrm{cm}^0$ of the confidential input $D^0$ equals the public input $\mathrm{cm}_{\mathbf{t}}$; (ii) the commitment $\mathrm{cm}^0_{\mathbf{p}}$ of the confidential input $\mathbf{p}^0$ equals the public input $\mathrm{cm}_{\mathbf{p}}$; and (iii) the output of the match search for the confidential input $\mathbf{p}^0$ in the confidential dataset $D^0$ leads to the publicly announced outcome ; see Appendix E.5 for details.

Let $|D(\mathbf{n})|$ denote the bit-length of a dataset $D(\mathbf{n})$ that contains **n** profiles (each profile is 40 bytes long); let $\mathrm{CC}(\mathbf{n})$ denote the communication complexity of the ZK-STARK for $D(\mathbf{n})$, i.e., the total number of bits communicated between prover and verifier; similarly, let $T_C(\mathbf{n})$ denote the time needed to naïvely verify C by executing it on D with **n** entries, and let $T_V(\mathbf{n})$ 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:

$$_{\text{time}}(\mathbf{n}) = \frac{T_V(\mathbf{n})}{T_C(\mathbf{n})}; \qquad _{\text{size}}(\mathbf{n}) = \frac{\mathrm{CC}(\mathbf{n})}{|D(\mathbf{n})|} \tag{1}$$

The rate measures (and thresholds defined next) depend on C and the system S, so the notation $^{(S;C)}_{\text{time}}$ 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 naïve 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 $\mathbf{n}_0$ such that for all $\mathbf{n} \quad \mathbf{n}_0$ the rate is less than 1,

$$_{\text{time}} = \min\{\mathbf{n}_0 \mid 8\mathbf{n} \quad \mathbf{n}_0 \quad _{\text{time}}(\mathbf{n}) < 1\}; \qquad _{\text{size}} = \min\{\mathbf{n}_0 \mid 8\mathbf{n} \quad \mathbf{n}_0 \quad _{\text{size}}(\mathbf{n}) < 1\} \tag{2}$$

Figure 1 shows the rate measures for the DPM problem on a double logarithmic scale. The time compression threshold is at $_{\text{time}} = 2{:}8 \quad 10^5$ and the witness-size threshold is $_{\text{size}} = 9 \quad 10^3$. The largest database for which we could generate a proof during our tests is $\mathbf{n}_{\max} = 2^{20} \quad 1{:}14 \quad 10^6$ DNA profiles; larger databases require more disk space and RAM than was available to us. Each profile occupies 40 bytes so $|D_{\mathbf{n}_{\max}}| \quad 43$ megabytes. The time-rate for $\mathbf{n}_{\max}$ is $_{\text{time}}(\mathbf{n}_{\max}) = 1{=}6$ and the witness-size rate is $_{\text{size}}(\mathbf{n}_{\max}) = 1{=}100$. 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.)

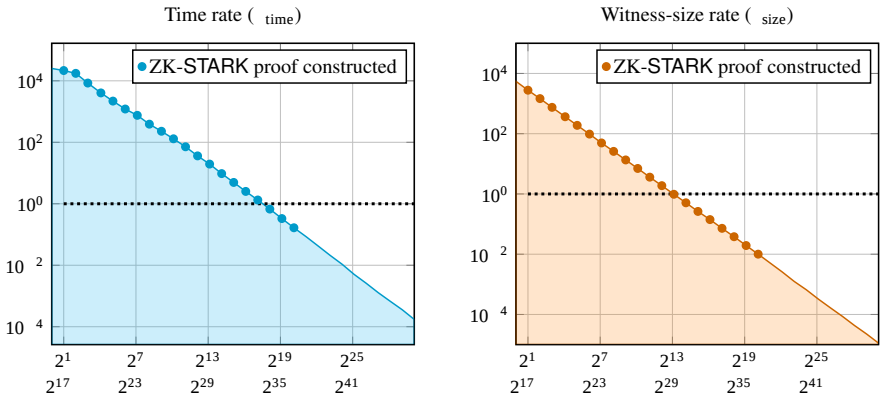


Figure 1: The time (left) and witness-size (right) rate functions of the DPM benchmark ZK-STARK as a function of (i) number of entries (**n**) in the database (upper horizontal axis) and (ii) number of multiplication gates (lower horizontal axis). Database size, in bytes, is 40 **n** and processing a single profile corresponds to a circuit with $2^{16}$ multiplication gates (bottom right entry of Figure 4, explained in Sections 1.3.2 and 2.2). Verifier complexity is independent of prover complexity, so it was measured (on a "standard" laptop; cf. Appendix A.0.2 for machine specifications) even for values of **n** that are larger than those for which a proof was generated; the values of **n** for which a proof was generated are marked by full circles.

## 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 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*[12]. 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 $T_C$.

- **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 $T_C$ 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-STARK, 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-STARK, 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 $T_C$ 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-STARK, but thus far has been realized only for a single hPKC system [23].

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

| | **prover scalability (quasilinear time)** | **verifier scalability (polylogarithmic time)** | **Transparency (public randomness)** | **Post-quantum security** |
|---|---|---|---|---|
| hPKC | Yes | Only repeated computation | No | No |
| DLP | Yes | No | Yes | No |
| IP | Yes | No | Yes | No |
| MPC | Yes | No | Yes | Yes |
| IVC+hPKC | Yes | Yes | No | No |
| ZK-STARK | Yes | Yes | Yes | Yes |

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 $F_{\mathbf{p}}$, 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[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 $\mathbf{n}$ entries (see Figure 4):

- circuit depth is $\text{depth}_{\mathbf{n}} = 62 \quad \mathbf{n}$;
- circuit width is $w = 81$;
- witness complexity is $\text{wit}_{\mathbf{n}} = 40 \quad \mathbf{n}$ bytes;
- multiplication complexity is $\text{mult}_{\mathbf{n}} = 1467 \quad 62 \quad \mathbf{n} = 90954 \quad \mathbf{n} \quad 2^{16:4} \quad \mathbf{n}$.

As discussed at length in Appendix A, we measured the full ZK-STARK system (prover+verifier) with 60 bits of security for $\mathbf{n} = 2^{\mathbf{k}}; \mathbf{k} = 1;:::;20$, i.e., for arithmetic circuits with depth up to $\text{depth}_{\mathbf{n}} \quad 2^{25:9}$, and with up to $2^{36:4}$ multiplication gates over $F_{2^{64}}$; the (nonadaptive) verifier alone was measured even for larger inputs, up to $\mathbf{n} = 2^{36}$ (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[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

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

[13] the FRI system requires $\mathbf{p}$ to contain a sufficiently large multiplicative subgroup of order $2^{t+O(1)}$; such prime fields abound, as implied by Linnik's Theorem [80].

[14] 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 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.

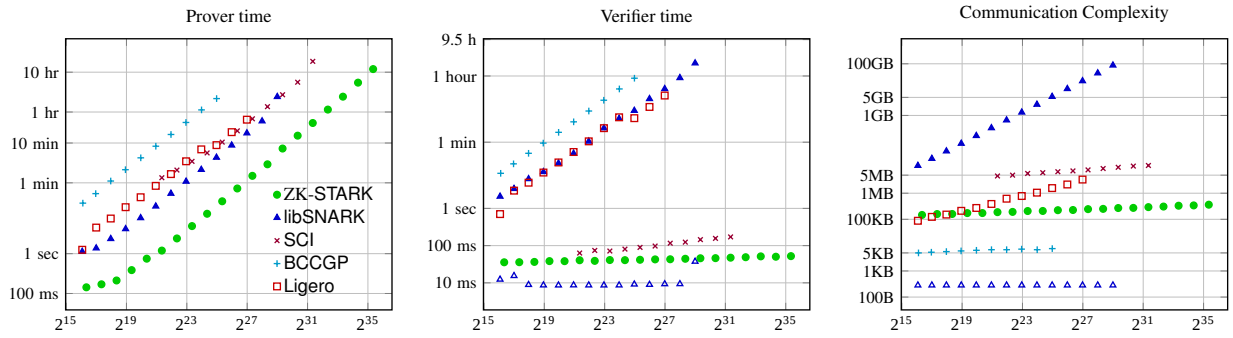


Figure 3: An "apples-to-apples" comparison of different realized proof systems as function of computation size, measured by number of multiplication gates. All systems were tested on the same server (specs below) and executed a computation of size and structure corresponding to the "exhaustive subset-sum" 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.)

**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 w mult), and Hyrax works best on small depth, massively parallel, circuits (depth = **O**(1) and w; mult 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 mult (and, often, like mult), whereas our ZK-STARK scales like w + log 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 10 seconds for a circuit with $2^{28}$ 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[15], with $\mathbf{n} = 2^{36}$ entries (profiles), $\text{wit}_{\mathbf{n}}$ 2:5-terabyte size witnesses and arithmetic circuits with $\text{mult}_{\mathbf{n}} = 2^{52}$ multiplication gates and depth $2^{40}$ (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 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 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 $70\sqrt{\text{mult}_{\mathbf{n}}}$ field elements [1, Section 5.3], and for Hyrax it scales like $\text{wit}^{1=\mathbf{k}} + 10\ \text{depth}\ \log w$ field elements for arbitrary integer $\mathbf{k}$ [109, Section 1]; increasing $\mathbf{k}$ decreases CC but also increases verification time (which is at least $\text{wit}=(\text{wit}^{1=\mathbf{k}})$). Using the estimate for Hyrax, a quick calculation shows that for a circuit arising from our DPM computation with, say $\mathbf{n} = 2^{13}$ profiles, the CC of Hyrax would reach several megabytes, compared with ZK-STARK CC that is less than 1 megabyte even for $\mathbf{n} = 2^{36}$ 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 10 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).

[15] We stress that current hardware does not support generating proofs for such large instances, as discussed later.

# 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 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

*" is the result of executing* C *for* T *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 $\mathbf{S}$ F and rate parameter 2 (0; 1), the Reed-Solomon code RS[F; **S**; ] is the family of functions $\mathbf{f} : \mathbf{S}$ ! F that are evaluations of polynomials of degree < j**S**j. The RPT problem assumes a verifier is given oracle access to $\mathbf{f}$, 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 $\mathbf{f}$ and the auxiliary PCPP/IOPP oracle(s), between the case that $\mathbf{f}$ 2 RS[F; **S**; ] and the case that $\mathbf{f}$ is 0:1-far from (all members of) RS[F; **S**; ] 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 j**S**j arithmetic operations in F — and verifier arithmetic complexity that is *strictly logarithmic*: 21 log j**S**j arithmetic operations; additionally, the proof can be constructed in log j**S**j 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 ( (j**S**j log$^{\mathbf{O}(1)}$ j**S**j)). 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

$$P = \{\mathbf{P}_1(\mathbf{X};\mathbf{Y}); \ldots; \mathbf{P}_s(\mathbf{X};\mathbf{Y})\}$$

of low degree polynomials with coefficients in F over a pair of variable sets $\mathbf{X} = (\mathbf{X}_1; \ldots; \mathbf{X}_w)$ and $\mathbf{Y} = (\mathbf{Y}_1; \ldots; \mathbf{Y}_w)$ 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 $(\mathbf{x};\mathbf{y}) \in F^w \times F^w$ corresponds to a single valid transition (or "cycle") of C if and only if

$$\mathbf{P}_1(\mathbf{x};\mathbf{y}) = \ldots = \mathbf{P}_s(\mathbf{x};\mathbf{y}) = 0;$$

i.e., if and only if $(\mathbf{x};\mathbf{y})$ is a common solution of the AIR system P. The following parameters of P determine prover and verifier complexity, so minimizing them is a major goal of this phase. The *degree* of the AIR is $\deg(P) = \max_{i=1}^{s} \deg(\mathbf{P}_i)$; 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 ($\mathbf{n}$) 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 $\mathbf{n}$ elements is $c \cdot \mathbf{n}$. 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 P. Our ZK-STARK uses the binary (characteristic 2) field $F_{2^{64}}$ 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 $\mathbf{n}$ denote this degree by $d^{max}(\mathbf{n})$. Prior state-of-the-art [27, 20, 38, 13] gave

$$d^{max}_{old}(\mathbf{n}) = \mathbf{n} \cdot c \cdot w \cdot d + \mathbf{n} \cdot c \cdot s; \tag{3}$$

[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 ($\mathbf{n}$) of the input dataset.

| | State size w | Cycles c | Degree d | System size s | # gates | # + gates | total # gates |
|---|---|---|---|---|---|---|---|
| SHA2 | 56 | 3762 | 11 | 1065 | 720 | 3194 | 2708640 $2^{21:3}$ |
| AES128+DM | 62 | 48 | 8 | 327 | 486 | 1033 | 23328 $2^{14:5}$ |
| Rij160+DM | 68 | 58 | 8 | 318 | 891 | 1390 | 51678 $2^{15:7}$ |
| DPM | 81 | 62 | 8 | 626 | 1467 | 1520 | 90954 $2^{16:4}$ |

Figure 4: Main complexity parameters of basic cryptographic primitives and the benchmark DNA profile match search program. The first four measures are explained in Section 2.2. The 5th and 6th columns measure total number of multiplication and addition gates in the AIR; the last column measures total number of multiplication gates (product of second and 5th columns) for the sake of comparison with other ZK systems that use this measure (cf. Section 1.3.2).

which leads to concretely large values (see first column of Figure 5). Our ZK-STARK reduces $d^{max}$ to

$$d^{max}_{ZK\ STARK}(\mathbf{n}) = \mathbf{n}\ \ c\ \ d \tag{4}$$

which results in a multiplicative savings factor of $6{:}5\ \ 10^4$–$1{:}8\ \ 10^5$ 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 $\mathbf{P}_1;\ldots;\mathbf{P}_s$. This round of interaction completely removes the second summand of (3).

**Register-based encoding** The naïve computation performed by the prover can be recorded by an *execution trace*, a two-dimensional array with $c\ \ \mathbf{n}$ rows and w columns, in which each row represents the state of the computation at a single point in time[19] and each column corresponds to an *algebraic register* tracked over all $c\ \ \mathbf{n}$ cycles. Prior systems, like [13], encoded the *full* execution trace by a *single* Reed-Solomon codeword, leading to degree $\mathbf{n}\ \ c\ \ w$; this degree is then multiplied by d to account for application of the afore-mentioned "local map" to the codeword, resulting in the first summand of (3). Our ZK-STARK uses a *separate* Reed-Solomon codeword for each register[20], leading to w many codewords, each of lower degree $\mathbf{n}\ \ c$. 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 $d^{max}$ 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 $6{:}5\ \ 10^4$–$1{:}8\ \ 10^5$ for the computations discussed in Section 2.2 and Figure 4.

[20]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.

| | $d_{old}^{max}(1)$ | $d_{ZK\ STARK}^{max}(1)$ | $d_{old}^{max}/d_{ZK\ STARK}^{max}$ |
|---|---|---|---|
| SHA2 | 6323922 | 41382 | 18491 |
| AES128+DM | 39504 | 384 | 6584 |
| Rij160+DM | 49996 | 464 | 6896 |
| DPM | 78988 | 496 | 10192 |

Figure 5: The maximal degree ($d^{max}$) 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 $d_{ZK\ STARK}^{max}$ over the prior state of the art.

## 2.4 Low degree extension and composition degree

Decreasing $d^{max}$ affords our ZK-STARK greater scalability. But, eventually, as the input size $\mathbf{n}$ grows, so does $d^{max}$. 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* $\mathbf{S}; \mathbf{S}^0$ *of a field* F *satisfying* $|\mathbf{S}^0| > |\mathbf{S}|$ *and a function* $\mathbf{f} : \mathbf{S} \to F$*, the* low degree extension (LDE) *of* $\mathbf{f}$ *to* $\mathbf{S}^0$ *is the function* $\mathbf{f}^0 : \mathbf{S}^0 \to F$ *that has the same interpolating polynomial as that of* $\mathbf{f}$.

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 $3\ |\mathbf{S}^0| \log |\mathbf{S}^0|$ for the sets $\mathbf{S}^0$ that our system requires (see Theorem B.2). Prior additive FFTs [36, 106, 51] required $(\mathbf{N} \log \mathbf{N} \log\log \mathbf{N})$ 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-STIK in which prover complexity is *strictly* quasi-linear, *and* verifier complexity is *strictly* logarithmic, in the size of the execution trace $c\ \mathbf{n}\ w$; see Lemma B.5.

## 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  denote the number of output bits of the cryptographic hash function used to construct a Merkle tree in our system; let $AP_{\mathbf{total}}$ denote the total number of authentication path nodes in all

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

subtrees of Merkle trees whose leaves are query answers, and let $\mathbf{q_{total}}$ denote the total number of queries, made to all proof oracles. The total communication complexity (CC) of the proof system is

$$CC = \mathbf{q_{total}} \log |F| + AP_{\mathbf{total}} \tag{5}$$

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

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

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

*The* computational integrity (CI) language $L_{CI}$ *is the projection of the binary relation* $R_{CI}$ *onto its first coordinate; alternatively,* $L_{CI} \stackrel{4}{=} \{x = (\mathbf{M}; \mathbf{x}; \mathbf{y}; \mathbf{T}; \mathbf{S}) \mid \exists w \ (x; w) \in R_{CI}\}$.

The space bound $\mathbf{S}$ in the definition above is unneeded, because $\mathbf{S} \quad \mathbf{T}$. However, IOP systems for space bounded computations ($\mathbf{S} = \mathbf{o}(\mathbf{T})$) are simpler and, often, concretely more efficient (this holds for our DPM computation). Thus, we treat space-bounded computations separately and dedicate and 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* $\in [0; 1]$ *denote* soundness error*. An* Interactive Oracle Proof (IOP) *system* $S$ *for* $R$ *with soundness* *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* $x$*, and the input of the prover is* $(x; w)$ *for some string* $w$*. The number of interactive rounds, denoted* $r(x)$*, 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* $\langle P(x; w) \ \$ \ V(x) \rangle$ *the output of* $V$ *after interacting with* $P$*; this output is either* accept *or* reject*.*

[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* $\Pr[\langle P(x; w) \$ V(x)\rangle = \text{accept}] = 1$
- **soundness** *If* $x \notin L$ *then for any* $P^*$, $\Pr[\langle P^* \$ V(x)\rangle = \text{accept}]$

*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* $L \in \text{NTIME}(\mathbf{T}(\mathbf{n}))$ *for* $\mathbf{T}(\mathbf{n}) \geq \mathbf{n}$ *and let* $S = (P; V)$ *be an IOP for* L *with soundness error* $\varepsilon(\mathbf{n}) < 1$*. 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* $\mathbf{n}$*, both of the following hold:*
  1. **scalable verifier:** $tv(\mathbf{n}) = \text{poly}(\mathbf{n}; \log \mathbf{T}(\mathbf{n}); \log 1/\varepsilon(\mathbf{n}))$
  2. **scalable prover:** $tp(\mathbf{n}) = \mathbf{T}(\mathbf{n}) \cdot \text{poly}(\mathbf{n}; \log \mathbf{T}(\mathbf{n}); \log 1/\varepsilon(\mathbf{n}))$
- **proof of knowledge** *if there exists a* knowledge error function $\kappa(\mathbf{n}) \in [0; 1]$ *and a randomized extractor* E *that, given oracle access to any prover* $P^*$ *that causes the verifier to accept* x *with probability* $\mathbf{p}(\mathbf{n}) > \kappa(\mathbf{n})$*, outputs in expected time* $\text{poly}\left(\frac{\mathbf{T}(\mathbf{n})}{\mathbf{p}(\mathbf{n}) - \kappa(\mathbf{n})}\right)$ *a witness* w *such that* $(x; w) \in R$*.*
- **privacy preservation** *if there exists a randomized simulator* Sim *that samples (perfectly) the distribution on transcripts of interactions between* V *and* P*, and runs in time* $\text{poly}(\mathbf{T}(\mathbf{n}))$*.*

*A (fully) scalable and transparent IOP of knowledge will be denoted by* STIK*. For* $\mathbf{T}(\mathbf{n}) = \text{poly}(\mathbf{n})$*, a privacy-preserving* STIK *has* perfect[23] zero knowledge *(ZK-*STIK*) but for* $\mathbf{T}(\mathbf{n}) = \mathbf{n}^{!(1)}$ *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 x (see Remark B.1); we use $\mathrm{tp}^{\mathrm{F}}$ and $\mathrm{tp}^{\mathrm{F}}$ 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.5), 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 NTimeSpace($\mathbf{T}(\mathbf{n}); \mathbf{S}(\mathbf{n})$) is the class of nondeterministic languages that are decidable in simultaneous time $\mathbf{T}(\mathbf{n})$ and space $\mathbf{S}(\mathbf{n})$.

**Theorem 3.4** (ZK-STIK for space bounded computations)**.** *Let* L *be a language in* NTimeSpace($\mathbf{T}(\mathbf{n}); \mathbf{S}(\mathbf{n})$)$; \mathbf{T}(\mathbf{n})$ $\mathbf{n}$ *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* $\mathrm{err}(\mathbf{n}) = 2^{(\mathbf{n})}$

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

*In particular, for* $\mathbf{S}(\mathbf{n}) = \mathrm{poly}\log \mathbf{T}(\mathbf{n})$*, this IOP is fully scalable, i.e., the system is a wi-*STIK*; if, additionally,* $\mathbf{T}(\mathbf{n}) = \mathrm{poly}(\mathbf{n})$*, 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 NEXP)**.** *Let* $\mathrm{L} \; 2 \; \mathrm{NTIME}(\mathbf{T}(\mathbf{n}))\mathbf{; T}(\mathbf{n}) \quad \mathbf{n}$ *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* $\mathrm{err}(\mathbf{n}) = 2^{(\mathbf{n})}$

- *perfect completeness and soundness error* $\mathrm{err}(\mathbf{n}) \quad 2^{(\mathbf{n})}$ *for instances of size* $\mathbf{n}$
- *knowledge extraction bound* $\mathrm{err}'(\mathbf{n}) = \mathbf{O}(\mathrm{err}(\mathbf{n}))$
- *round complexity* $\mathrm{r}(\mathbf{n}) = \frac{\log \mathbf{T}(\mathbf{n})}{2} + \mathbf{O}(1)$

- *query complexity* $\mathbf{O}(\ (\mathbf{n})\ \log \mathbf{T}(\mathbf{n}))$*, each query is an element of a binary field* $\mathrm{F}; \mathrm{jFj} = 2^{\mathrm{n}}$ *for* $\mathrm{n} = \ (\mathbf{n}) + \log \mathbf{T}(\mathbf{n}) + \log\log \mathbf{T}(\mathbf{n}) + \mathbf{O}(1)$.
- *verifier arithmetic complexity* $\mathrm{tv}^{\mathrm{F}}(\mathbf{n}) = \mathbf{O}(\ (\mathbf{n})\ \log \mathbf{T}(\mathbf{n}))$,
- *prover arithmetic complexity* $\mathrm{tp}^{\mathrm{F}}(\mathbf{n}) = \mathbf{O}(\mathbf{T}(\mathbf{n}) \log^2 \mathbf{T}(\mathbf{n}))$,
- *proof length* $\mathbf{O}(\mathbf{T}(\mathbf{n}) \log \mathbf{T}(\mathbf{n}))$*, measured in field elements.*

*For* $\mathbf{T}(\mathbf{n}) = \mathrm{poly}(\mathbf{n})$ *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


[1] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 24th ACM Conference on Computer and Communications Security*, October 2017.

[2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in FOCS '92.

[3] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in FOCS '92.

[4] László Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429, 1985.

[5] László Babai and Lance Fortnow. Arithmetization: A new method in structural complexity theory. *computational complexity*, 1(1):41–66, 1991.

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

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

[8] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM Journal on Computing*, 38(5):1661–1694, 2008. Preliminary version appeared in CCC '02.

[9] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 390–420, 1993.

[10] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[11] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Hr astad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Proceedings of the 8th Annual International Cryptology Conference*, CRYPTO '89, pages 37–56, 1988.

[12] Eli Ben-Sasson. Universal and affordable computational integrity. `https://www.youtube.com/watch?v=Q4nWoEKUtgU&t=32s`, May 2013.

[13] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear PCPs. *IACR Cryptology ePrint Archive*, 2016:646, 2016.

[14] Eli Ben-Sasson, Iddo Bentov, Ynon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity (2nd revision). *Electronic Colloquium on Computational Complexity (ECCC)*, 24:134, 2017.

[15] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. On probabilistic checking in perfect zero knowledge. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:156, 2016.

[16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Short interactive oracle proofs with constant query complexity, via composition and sumcheck. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:46, 2016.

[17] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasilinear-size zero knowledge from linear-algebraic PCPs. In *Proceedings of the 13th Theory of Cryptography Conference*, TCC '16, pages 33–64, 2016.

[18] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, 2014.

[19] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference*, ITCS '13, pages 401–414, 2013.

[20] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 585–594, 2013.

[21] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 90–108, 2013.

[22] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. *Interactive Oracle Proofs*, pages 31–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[23] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Proceedings of the 34th Annual International Cryptology Conference*, CRYPTO '14, pages 276–294, 2014. Extended version at `http://eprint.iacr.org/2014/595`.

[24] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium*, Security '14, pages 781–796, 2014. Extended version at `http://eprint.iacr.org/2013/879`.

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

[26] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

[27] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008. Preliminary version appeared in STOC '05.

[28] Václav E Beneš. *Mathematical theory of connecting networks and telephone traffic*, volume 17. Academic press, 1965.

[29] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 111–120, 2013.

[30] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *Proceedings of the 10th Theory of Cryptography Conference*, TCC '13, pages 315–333, 2013.

[31] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 327–357, 2016.

[32] Joppe W. Bos, Onur Özen, and Martijn Stam. Efficient hashing using the aes instruction set. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 507–522. Springer, 2011.

[33] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[34] Arthur Breitman. Scaling tezos. `https://hackernoon.com/scaling-tezo-8de241dd91bd`, 2017.

[35] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Efficient range proofs for confidential transactions. Cryptology ePrint Archive, Report 2017/1066, 2017. `https://eprint.iacr.org/2017/1066`.

[36] David G. Cantor. On arithmetical algorithms over finite fields. *J. Comb. Theory, Ser. A*, 50(2):285–300, 1989.

[37] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[38] Alessandro Chiesa and Zeyuan Allen Zhu. Shorter arithmetization of nondeterministic computations. *Theor. Comput. Sci.*, 600:107–131, 2015.

[39] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Proceedings of the 1st Symposium on Innovations in Computer Science*, ICS '10, pages 310–331, 2010.

[40] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[41] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*, ITCS '12, pages 90–112, 2012.

[42] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proceedings of the VLDB Endowment*, 5(1):25–36, 2011.

[43] R. Courtland. Google aims for quantum computing supremacy [news]. *IEEE Spectrum*, 54(6):9–10, June 2017.

[44] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[45] Ivan Bjerre Damgr ard. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, 1989.

[46] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. *Square Span Programs with Applications to Succinct NIZK Arguments*, pages 532–550. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[47] NG de Bruijn. A combinatorial problem. *Proc. Akademe Van Westeschappen*, 49(2):758–764, 1946.

[48] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 155–164, 2004.

[49] Cynthia Dwork, Uriel Feige, Joe Kilian, Moni Naor, and Shmuel Safra. Low communication 2-prover zero-knowledge proofs for NP. In *Proceedings of the 11th Annual International Cryptology Conference*, CRYPTO '92, pages 215–227, 1992.

[50] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of the 6th Annual International Cryptology Conference*, CRYPTO '86, pages 186–194, 1986.

[51] Shuhong Gao and Todd D. Mateer. Additive fast fourier transforms over finite fields. *IEEE Trans. Information Theory*, 56(12):6265–6272, 2010.

[52] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 465–482, Berlin, Heidelberg, 2010. Springer-Verlag.

[53] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '13, pages 626–645, 2013.

[54] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 99–108, New York, NY, USA, 2011. ACM.

[55] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1069–1083, Austin, TX, 2016. USENIX Association.

[56] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik*, 38(1):173–198, 1931.

[57] Oded Goldreich. Randomness and computation. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 507–539. Springer, 2011.

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

[59] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Preliminary version appeared in STOC '85.

[60] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '10, pages 321–340, 2010.

[61] Jens Groth. Efficient zero-knowledge arguments from two-tiered homomorphic commitments. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 431–448, 2011.

[62] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326, 2016.

[63] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 581–612, 2017.

[64] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 415–432, 2008.

[65] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Trans. Information Theory*, 45(6):1757–1767, 1999.

[66] Michael Hearn. Corda and SGX: a privacy update. `https://www.corda.net/2017/06/corda-sgx-privacy-update/`, 2017.

[67] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox, March 2017.

[68] Rosamond Hutt. Beyond bitcoin: 4 surprising uses for blockchain, Dec 2016.

[69] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity*, CCC '07, pages 278–291, 2007.

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

[71] Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. On zero-knowledge PCPs: Limitations, simplifications, and applications, 2015. Available at `http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf`.

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

[73] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

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

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

[76] Protocol Labs. Filecoin: A decentralized storage network. `https://filecoin.io/filecoin.pdf`, 2017.

[77] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[78] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, second edition edition, 1997.

[79] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. Novel polynomial basis and its application to reed-solomon erasure codes. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, pages 316–325, Washington, DC, USA, 2014. IEEE Computer Society.

[80] UV Linnik. On the least prime in an arithmetic progression. i. the basic theorem. *Matematicheskii Sbornik*, 15(2):139–178, 1944.

[81] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*, TCC '12, pages 169–189, 2012.

[82] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[83] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology (CRYPTO '89)*, pages 218–238, Berlin - Heidelberg - New York, 1990. Springer.

[84] S. Micali, M. Rabin, and J. Kilian. Zero-knowledge sets. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 80–91, Oct 2003.

[85] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[86] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2009.

[87] Federal Bureau of Investigation. Combined dna index system (CODIS). https://www.fbi.gov/services/laboratory/biometric-analysis/codis. Retrieved September 2017.

[88] Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, Oakland '13, pages 238–252, 2013.

[89] M. Peck. A blockchain currency that beat s bitcoin on privacy [news]. *IEEE Spectrum*, 53(12):11–13, December 2016.

[90] Evgenya Pergament. Algebraic RAM. Master's thesis, Technion — Israel Institute of Technology, 2017.

[91] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, pages 194–203, 1994.

[92] FIPS PUB. Secure hash standard (shs). *FIPS PUB 180*, 4, 2012.

[93] Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

[94] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016.

[95] Guy N Rothblum, Salil Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 793–802. ACM, 2013.

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

[97] Jae Hong Seo. Round-efficient sub-linear zero-knowledge arguments for linear algebra. In *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, pages 387–402, 2011.

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

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

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

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

[102] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82. ACM, 1987.

[103] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), September 1997. Retrieved October 2017.

[104] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 71–89, 2013.

[105] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Proceedings of the 5th Conference on Theory of Cryptography*, TCC'08, pages 1–18, Berlin, Heidelberg, 2008. Springer-Verlag.

[106] Joachim von zur Gathen and Jürgen Gerhard. Fast algorithms for taylor shifts and certain difference equations. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, ISSAC '97, Maui, Hawaii, USA, July 21-23, 1997*, pages 40–47, 1997.

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

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

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

[110] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.

[111] R. S. Winternitz. A secure one-way hash function built from des. In *1984 IEEE Symposium on Security and Privacy*, pages 88–88, April 1984.

[112] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vsql. Cryptology ePrint Archive, Report 2017/1146, 2017. `https://eprint.iacr.org/2017/1146`.

# 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 $\mathbf{n} = 2^{14}$ 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 $\mathbf{n}$. 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 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.

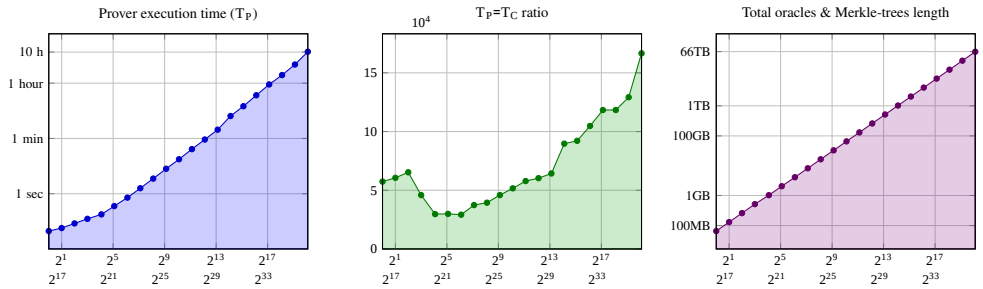


Figure 6: On the left we plot, on a double-logarithmic scale, prover running time as a function of the number of entries ($\mathbf{n}$) in the DNA profile database. On the middle we plot the ratio between proving time to naïve execution time; the horizontal axis is logarithmic ($\log \mathbf{n}$) and 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 24, Speed: 2666 MHz)
- SWAP : 3.2TB NVMe (1.6TB 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 it's 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. Morevoer, 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 specificatin below), verification complexity does not increase significantly.

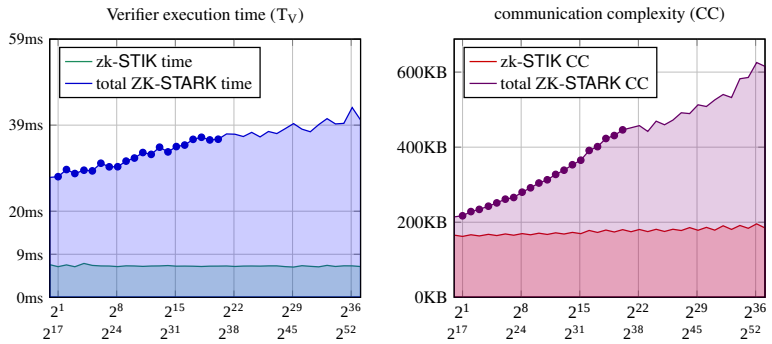


Figure 7: Verifier running time (left) and communication complexity (right) as a function of the number of entries (**n**) in the DNA database size; for both plots the horizontal axis is logarithmic and the vertical one is linear. In both plots, the lower line measure the complexity of the ZK-STIK verifier, 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.

**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)

# B From AIR to ZK-STARK

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.

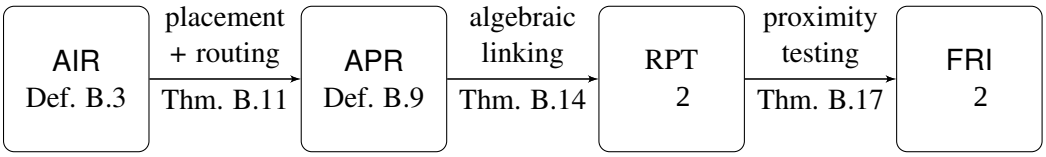


Figure 8: The reduction from an AIR instance to a pair of RPT problems, solved using the FRI protocol.

**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 $(\mathbf{L}; \mathbf{L}_{cmp})$ and different code rates ( ; $_{cmp}$). The witness in this case is a pair of purported codewords $(\mathbf{f}^{(0)}; \mathbf{g}^{(0)})$. 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.

4. Finally, for each of the two functions (oracles) $\mathbf{f}^{(0)}; \mathbf{g}^{(0)}$, 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.5 and B.8) that prove our main Theorems 3.4 and 3.5, as well as the lemma on which our ZK-STARK realization relies (Lemma B.6). 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.9) 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.13 of the RS proximity testing (RPT) problem and followed by the algebraic linking IOP (ALI) stated in Theorem B.14 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 $\mathbf{S}'$ of a finite set $\mathbf{S}$ is $(\mathbf{S}'/\mathbf{S}) \stackrel{4}{=} |\mathbf{S}'|/|\mathbf{S}|$.

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

**Binary fields** A finite field is denoted by $F$ and $F_{\mathbf{q}}$ is the field of size $\mathbf{q}$. 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 $\mathbf{H} \subseteq F$ that is an additive coset of an $F_2$-linear space, meaning $\mathbf{H} = \alpha + \mathbf{V} \stackrel{4}{=} \{\alpha + \mathbf{v} \mid \mathbf{v} \in \mathbf{V}\}$ for some fixed $\alpha \in F$ and $\mathbf{V} \subseteq F$ a linear space over the two-element field $F_2$.

**Remark B.1** (Canonical representations of binary fields)**.** *We assume canonical representation for binary fields* $F$*, given by an irreducible polynomial and a primitive element* $g \in F$ *for it (i.e.,* $g$ *generates* $F^*$*). We use the standard basis* $1; g; g^2; \ldots; g^{n-1}$ *to represent* $F_{2^n}$ *over* $F_2$*. Similarly, for a fixed* $F_2$*-subspace of dimension* $\mathbf{k}$*, the (non-zero) polynomial of degree* $2^{\mathbf{k}}$ *that vanishes on* $\mathbf{S}$ *will assumed to be known to the verifier; recall this polynomial is linearized (see [78, Section 3.4]) and has at most* $\mathbf{k}$ *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 $\mathbf{C}$ over $F$ of blocklength $\mathbf{n}$ as functions with domain $\mathbf{S}; |\mathbf{S}| = \mathbf{n}$ and range $F$, i.e., $\mathbf{C} \subseteq F^{\mathbf{S}}$; for $\mathbf{x} \in \mathbf{S}$, the $\mathbf{x}$-entry of a codeword $\mathbf{w} \in \mathbf{C}$ is denoted by $\mathbf{w}(\mathbf{x})$. For $\mathbf{v} \in F^{\mathbf{S}}$ let $\mathbf{w}_{\mathbf{v}} \in \mathbf{C}$ be a codeword that is closest to $\mathbf{v}$ in relative hamming distance, breaking

[26] 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 $\mathbf{C}$ arbitrarily and setting $\mathbf{w_v}$ to be the smallest $\mathbf{w} \in \mathbf{C}$ with minimal distance to $\mathbf{v}$).

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

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

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,* $\mathbf{S} \subseteq \mathbb{F}$ *an* $\mathbb{F}_2$*-affine space and* $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{F}$ *there exists an arithmetic circuit computing an advice string* $\mathbf{A}$ *that requires* $(\mathbf{n}+1)\text{polylog}|\mathbf{S}|$ *arithmetic operations in* $\mathbb{F}$*, and another arithmetic circuit that uses the advice* $\mathbf{A}$ *and computes the LDE of a function* $\mathbf{f} : \mathbf{S} \to \mathbb{F}$ *to domain* $\bigcup_{\mathbf{c}_i} \mathbf{S} + \mathbf{c}_i$*. This latter circuit has arithmetic complexity* $3(\mathbf{n}+1)(|\mathbf{S}| \log |\mathbf{S}|)$ *over* $\mathbb{F}$.

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

**Arithmetic complexity** For an arithmetic circuit $\mathbf{C}$ with gates of fan-in 2 over the set of gates $\{+, \times\}$, we denote by $T_{\text{arith}}(\mathbf{C})$ the arithmetic complexity of $\mathbf{C}$, 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 $\mathbf{f} : \mathbb{F}^{\mathbf{n}} \to \mathbb{F}^{\mathbf{m}}$ whose "canonical circuit" $\mathbf{C_f}$ is implicitly known, we abuse notation and define $T_{\text{arith}}(\mathbf{f}) = T_{\text{arith}}(\mathbf{C_f})$.

Given a set of functions $F = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{\mathbf{m}}\}$ mapping elements of $\mathbb{F}^{\mathbf{n}}$ to $\mathbb{F}$, we denote by $T_{\text{arith}}(F)$ the (total) arithmetic complexity of $F$ and define it as the the arithmetic of a circuit $\mathbf{C} : \mathbb{F}^{\mathbf{n}} \to \mathbb{F}^{\mathbf{m}}$ such that for any $\vec{\mathbf{v}} \in \mathbb{F}^{\mathbf{n}}$ and any $1 \le \mathbf{i} \le \mathbf{m}$, $\mathbf{C}(\vec{\mathbf{v}})_{\mathbf{i}} = \mathbf{f}_{\mathbf{i}}(\vec{\mathbf{v}})$.

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

## B.2 Algebraic Intermediate Representation (AIR)

An algebraic execution trace of a program running for T steps is represented by a w × 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 $\mathbf{L} \in \text{NTimeSpace}(\widehat{\mathbb{T}}(\mathbf{n}); \widehat{\mathbb{w}}(\mathbf{n}))$ to AIR instances with algebraic execution traces of size $T(\mathbf{n}) = \mathbf{O}(\widehat{\mathbb{T}}(\mathbf{n}))$ and width $w(\mathbf{n}) = \mathbf{O}(\widehat{\mathbb{w}}(\mathbf{n}))$, 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* $(x; w) = (x_{\text{AIR}}; w_{\text{AIR}})$ *satisfying*

1. **Instance Format:** *the instance* $x$ *is a tuple* $x = (\mathbb{F}; T; w; P; B)$ *where*

- F *is a finite field*
- T *is an integer representing a bound on running time*
- w *is an integer representing state width*
- P F[$\mathbf{X}_1;\ldots;\mathbf{X}_w;\mathbf{Y}_1;\ldots;\mathbf{Y}_w$] *is a set of* constraints. *The* degree *of* P *is* $\deg(P) \stackrel{4}{=} \max_{\mathbf{P} \in P} \deg(\mathbf{P})$
- $B$ *is a set of boundary constraints, where each boundary constraint is a tuple* $(\mathbf{i};\mathbf{j};\ )$ *for* $\mathbf{i} \in [T];\mathbf{j} \in [w];\ \in F$

2. **Witness Format:** *The witness* w *is a set of functions* $\mathbf{w}_1;\ldots;\mathbf{w}_w : [T] \to F$*; we say* w satisfies *the instance if and only if*
   (a) *For all boundary constraints* $(\mathbf{i};\mathbf{j};\ )$ *we have* $\mathbf{w}_\mathbf{j}(\mathbf{i}) =$
   (b) *For all* $\mathbf{t} \in [T\ \ 1]$ *and for all* $\mathbf{P} \in P$ *we have* $\mathbf{P}(\mathbf{w}[\mathbf{t}];\mathbf{w}[\mathbf{t}+1]) = 0$ *where* $\mathbf{w}[\mathbf{t}] = (\mathbf{w}_1(\mathbf{t});\ldots;\mathbf{w}_w(\mathbf{t}))$

*Finally,* $R_{AIR}$ *is the set of all pairs* $(x;w)$ *such that* w *satisfies* x*, and* $L_{AIR} \stackrel{4}{=} \{x \mid \exists w;(x;w) \in R_{\mathbf{AIR}}\}$.

Our reductions will use AIRs in which 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.4** (BAIR)**.** *A* binary AIR *(*BAIR*) instance is an* AIR *instance* $x = (F;T;w;P;B)$ *satisfying for some* $n;t;d \in \mathbb{N}^+$ *(i)* $|F| = 2^n$*; (ii)* $T = 2^t\ \ 1$*; and (iii)* $\deg P\ \ 2^d$*. We call* x *an* $(n;t;d)$-BAIR *instance. The relations* $R_{BAIR}$ *and language* $L_{BAIR}$ *are the natural restriction of* $R_{AIR}$ *and* $L_{AIR}$ *to binary* 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 ZK-IOP protocols [22, 15], it fully specifies all constants and uses no asymptotic notation.

**Lemma B.5** (wi-IOP of knowledge for $R_{BAIR}$)**.** *For soundness error parameter* $: \mathbb{N}^+ \to \mathbb{N}^+$*, rate integer* $R \geq 2$*, and ZK parameter* $k \geq 1$*, let* $= 1\ \ 2^{-R} 1 + 2^{-d}$*. Let relation* $R_{AIR}^{(;R;k)}$ *denote the restriction of* $R_{AIR}$ *to* $(n;t;d)$-BAIR *instances that satisfy (i)* $n > + t + d + R + k + 2$ *and (ii)* $k + t \geq 6 + \log + \log\log\frac{1}{1\ \sqrt{10}}$*. Then* $R_{AIR}^{(;R)}$ *has a ZK-IOP of knowledge with the following parameters for* $(n;t;d)$-BAIR *instances:*

- *soundness error at most* $\text{err} = 2$
- *knowledge error bound at most* $^0 = 4\ \text{err}$
- *prover arithmetic complexity*

$$tp^F \leq (9w(t+d+R+3) + |P| + w + T_{arith}(P) + 12)\ \ 2^{t+d+R+3};$$

- *verifier arithmetic complexity*

$$tv^F \leq 2\ \ |B|^2 + d\frac{+2}{\log\frac{1}{1\ \ =10}}e\ \ (8(w + |\ \ | + T_{arith}(P) + |B|) + 21(k + R + t + d + 2))\ \ ;$$

- *round complexity* $r = \frac{t+d}{2} + 2$,
- *query complexity* $q \quad 8d\frac{+2}{\log\frac{1}{1 \quad =10}}e \quad (2w + k + t + d + 2)$

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.6** (wi-IOP for $R_{BAIR}$ with improved conjectured soundness)**.** *Assuming Conjectures B.16 and B.18, the following holds. For soundness error parameter* $: N^+ \; ! \; N^+$*, rate integer* $R \quad 3$*, and ZK parameter* $k \quad 1$*, let relation* $R_{AIR}^{\emptyset( \; ;R;k)}$ *denote the restriction of* $R_{AIR}$ *to* $(n; t; d)$-BAIR *instances that satisfy (i)* $n > \max f58; \; ; t + d + R + kg + 2$ *and (ii)* $k + t > 5 + \log d\frac{+2}{R}e$*. Then* $R_{AIR}^{\emptyset( \; ;R)}$ *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.5,*
- *verifier arithmetic complexity*

$$tv^F \quad 2jBj^2 + d\frac{+2}{R}e \quad (16(w + j \quad j + T_{arith}(P) + 42 \quad (k + R + t + d + 2)))$$

- *query complexity* $q \quad 4 \quad d\frac{+2}{R+d}e \quad (w + 3 + k + t + d) + d\frac{+2}{R}e \quad (3w + 3 + k + t + d)$
- *prover arithmetic complexity*

$$tp^F \quad 9w(t + d + R + 3) + jPj + w + T_{arith}(P) + 12d\frac{+2}{n \quad (10 + 2\log n)}e \quad 2^{t+d+R+3}$$

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 ($w = \quad (T)$), leading to AIR witnesses of total size $(w \; T) = \quad (T^2)$. To reach witnesses of size $\mathbf{O}(T \log T)$, 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.7** (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* $x = (F; T; w; P_T; P \; ; B)$ *where*
   - $F$ *is a finite field*
   - $T$ *is an integer representing a bound on running time*
   - $w$ *is an integer representing state width*
   - $P_T; P \quad F[\mathbf{X}_1; : : : ; \mathbf{X}_w; \mathbf{Y}_1; : : : ; \mathbf{Y}_w]$ *are two sets of* constraints.
   - $B$ *is a set of boundary constraints, where each boundary constraint is a tuple* $(\mathbf{i}; \mathbf{j}; \quad )$ *for* $\mathbf{i} \; 2 \; [T]; \mathbf{j} \; 2 \; [w]; \quad 2 \; F$

2. **Witness Format:** *The witness* w *is a pair* $(\hat{\mathbf{w}}; \quad)$ *where* $\hat{\mathbf{w}}$ *is a sequence of* w *functions* $\mathbf{w}_1; \ldots; \mathbf{w}_w : [T] \to F$, *and* $: [T] \to [T]$ *is a permutation; we say* w satisfies *the instance if and only if*

   (a) *For all boundary constraints* $(\mathbf{i}; \mathbf{j}; \quad)$ *we have* $\mathbf{w_j}(\mathbf{i}) =$

   (b) *For all* $\mathbf{t} \in [T \quad 1]$ *and for all* $\mathbf{P} \in P_T$ *we have* $\mathbf{P}(\mathbf{w}[\mathbf{t}]; \mathbf{w}[\mathbf{t}+1]) = 0$

   (c) *For all* $\mathbf{t} \in [T \quad 1]$ *and for all* $\mathbf{P} \in P$ *we have* $\mathbf{P}(\mathbf{w}[\mathbf{t}]; \mathbf{w}[\quad(\mathbf{t})]) = 0$

*Finally,* $R_{PAIR}$ *is the set of all pairs* $(x; w)$ *such that* w *satisfies* x, *and* $PAIR \stackrel{4}{=} \{x \mid \exists w; (x; w) \in \mathbf{R_{AIRwRAM}}\}$.

*We define the sub-relation of* binary PAIR *(*BPAIR*) analogously to the definition of binary* AIR *in Definition B.4.*

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

**Lemma B.8** (wi-STIK for BPAIR)**.** *For every language* $L \in \mathrm{NTIME}(\mathbf{T}(\mathbf{n})); \mathbf{T}(\mathbf{n}) \quad \mathbf{n}$ *there exists a deterministic polynomial time reduction from* L *to* $L_{PAIR}$, *mapping an instance* $\mathbf{x}$ *of* L *to an instance* $x = (F; T; w; P_T; P \quad ; B)$ *of* $L_{PAIR}$ *that satisfies*

- $T = \mathbf{O}(\mathbf{T}(\mathbf{n}))$,
- $w = \mathbf{O}(\log \mathbf{T}(\mathbf{n}))$,
- $\deg(P_T [\ P\ ]) = \mathbf{O}(1)$, $|P_T [\ P\ ]| = \mathbf{O}(\log \mathbf{T}(\mathbf{n}))$, *and* $T_{\mathrm{arith}}(P) = \mathbf{O}(\log \mathbf{T}(\mathbf{n}))$,
- $|B| = \mathbf{O}(\mathbf{n})$.

*Furthermore, a nondeterministic witness* w *for* x *can be computed in time* $\mathbf{O}(\mathbf{T}(\mathbf{n}) \log \mathbf{T}(\mathbf{n})) + \mathbf{n}^{\mathbf{O}(1)}$, *given* $\mathbf{x}$ *and a nondeterministic witness* $\mathbf{w}$ *for the membership of* $\mathbf{x}$ *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 $A_1(F)$ denotes the 1-dimensional affine group over F, its set of elements is isomorphic to $\{\mathbf{aX} + \mathbf{b} \mid \mathbf{a} \in F; \mathbf{b} \in F\}$. An *affine graph* generated by a set $N = \{\mathbf{N}_1; \ldots; \mathbf{N}_s\} \quad A_1(F)$ is the directed graph with vertex set F and edge set $\{(\mathbf{x}; \mathbf{N}(\mathbf{x})) \mid \mathbf{x} \in F; \mathbf{N} \in N\}$.

In the following definition, the set captures both boundary, and transition, constraints of the computation. The affine graph is added to the instance description via its generating set 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 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.9** (Algebraic placement and routing (APR) problem)**.** *The relation* $R_{APR}$ *contains all pairs* $(x; w)$ *satisfying the following requirements:*

1. **Instance Format:** *the* instance x *is a tuple* $x = (F; T; N; \quad; \mathbf{L}; \mathbf{L}_{\mathrm{cmp}}; \sim; \ _{\mathrm{cmp}})$ *where*

   - F *is a finite field of characteristic* 2.

- $\mathbb{T}$ *is a set of indices called the* algebraic-register indices.
- $N : \mathbb{T} \to A_1(F)$ *is a set of pairs called* neighbors*, each pair contains an algebraic-register index, and a member of the affine group over* $F$ *(a polynomial of degree exactly* 1*). Given a set* $\mathbf{S} \subseteq F$ *we denote by* $N(\mathbf{S})$ *the set* $N(\mathbf{S}) := \{\mathbf{N}(\mathbf{x}) \mid \mathbf{x} \in \mathbf{S}, (\tau, \mathbf{N}) \in N\}$ *for some* $\tau \in \mathbb{T}$*.*
- $\mathbf{f} \subseteq F^{F^N}$ *is a set of mappings over the variables* $V := \{X_{loc}\} \cup \{X_{\mathbf{N}}\}_{\mathbf{N} \in N}$*. Assignments to any* $\tau \in \mathbb{T}$ *expressed by mappings* $\sigma : V \to F$*. Given a set* $\mathbf{S} \subseteq F$ *and a sequence of functions indexed by elements of* $\mathbb{T}$*,* $\langle \mathbf{f}_i \rangle \in \mathbf{f}^{N(\mathbf{S})^{\mathbb{T}}}$*, denote by* $\langle \mathbf{f}_i \rangle_{;N} : \mathbf{S} \to F^V$ *the mapping defined by* $\langle \mathbf{f}_i \rangle_{;N}(\mathbf{x})(X_{loc}) := \mathbf{x}$*, and for every* $(\tau, \mathbf{N}) \in N$*,* $\langle \mathbf{f}_i \rangle_{;N}(\mathbf{x})(X_{\mathbf{N}}) := \mathbf{f}_{(\tau,\mathbf{N})}(\langle \langle \mathbf{f}_i \rangle_{;N}(\mathbf{x}) \rangle_\tau)$*. Given* $\sigma \in \mathbb{T}$ *we denote by* $\mathbf{f}_\sigma \in F^{\mathbf{S}}$ *the vector defined by* $\mathbf{f}_\sigma(\mathbf{x}) := \langle \mathbf{f}_i \rangle_{;N}(\mathbf{x})(\sigma)$ *for every* $\mathbf{x} \in \mathbf{S}$*.*
- $\mathbf{L}$ *and* $\mathbf{L}_{\text{cmp}}$ *are two* $\mathbb{F}_2$*-affine subspaces of* $F$ *called the witness evaluation space and the composition evaluation space respectively;*
- $\sim \in (0;1)^{\mathbb{T}}$ *is a sequence of rates called the witness rates, and* $_{\text{cmp}} \in (0;1)$*, called the composition rate.*

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

*Let* $R_{APR}$ *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* APR *be the nondeterministic language induced by* $R_{APR}$*,*

$$APR = \{x \mid \exists \hat{w}; (x; \hat{w}) \in R_{APR}\}: \tag{6}$$

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

**Definition B.10** (APR properties)**.** *We say an instance* $x := (F; \mathbb{T}; N; \ ; \mathbf{L}; \mathbf{L}_{\text{cmp}}; \sim; \ _{\text{cmp}}) \in APR$

1. *is* -independent *if there is a sequence of sets* $\{\mathbf{V} \ \}_{\in \mathbb{T}}$ *such that each* $\mathbf{V} \ \subseteq F^{\mathbf{L}}$ *is* $|\mathbf{L}|$*-independent, and for any* $\hat{w}$ *such that* $w \in \mathbf{V}$*, it holds that* $\hat{w}$ *satisfies* $x$*. Let* $T_{\text{Sampling}}( )$ *denote the arithmetic complexity of the circuit, that on input* $w \in \mathbf{V}$ *(and random field elements), samples a uniformly random* $w \in \{\mathbf{V} \ \}_{\in \mathbb{T}}$*.*
2. *has* -distance *if:*
   - $1 \ _{\text{cmp}}$*.*
   - $1$ *for any* $\in \mathbb{T}$*.*
   - *There exists some linear code* $\mathbf{C} \ \subseteq F^{\mathbf{L}_{\text{cmp}}}$ *of relative distance at least* *, containing the code* $RS[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$*, such that for any sequence of mappings* $\mathbf{f} \in F^{\mathbf{L}^{\mathbb{T}}}$ *where* $\mathbf{f} \in RS[F; \mathbf{L}; \ ]$ *for all* $\in \mathbb{T}$*, it holds* $\forall \ ; \ _N [\hat{w}] \in \mathbf{C}$*.*
3. *is* -overlapping *if for any* $\mathbf{y} \in \mathbf{L}$ *the size of the set* $\mathbf{S}_\mathbf{y} := \{\mathbf{x} \in \mathbf{L}_{\text{cmp}} \mid \mathbf{y} \in N(\{\mathbf{x}\})\}$ *is at most* *.*

## 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.11 by $V^{\text{BAIR} \to \text{APR}}; P^{\text{BAIR} \to \text{APR}}$, respectively, and those of Theorem B.12 are denoted $V^{\text{BPAIR} \to \text{APR}}; P^{\text{BPAIR} \to \text{APR}}$.

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 apposed 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 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.11** (Algebraic placement and routing)**.** *There are two algorithms* $V^{\text{BAIR} \to \text{APR}}; P^{\text{BAIR} \to \text{APR}}$ *such that for any* $k; R; t; d \in \mathbb{N}^+$ *and* BAIR *instance* $x = (F; T; w; P; B)$ *with* $T = 2^t \quad 1 > 4$, $\deg P \quad 2^d$, *and* $|F| = 2^n$ *with* $n > 2 + k + R + t + d$*:*

1. *Perfect completeness:* $(x; w) \in \mathbf{R}_{\text{AIR}} \ ) \ (V^{\text{BAIR} \to \text{APR}}(x; k; R); P^{\text{BAIR} \to \text{APR}}(x; w; k; R)) \in \mathbf{R}_{\text{APR}}$
2. *Perfect soundness:* $x \in \text{AIR} \ ( \ V^{\text{BAIR} \to \text{APR}}(x; k; R) \in \text{APR}$
3. *Knowledge extraction: There is an efficient knowledge extractor* $E^{\text{BAIR} \to \text{APR}}$*, such that for every* $(V^{\text{BAIR} \to \text{APR}}(x; k; R); w_{\text{APR}}) \in \mathbf{R}_{\text{APR}}$*, the extractor outputs a witness* $w \quad E^{\text{BAIR} \to \text{APR}}(x; k; R; w_{\text{APR}})$ *such that* $(x; w) \in \mathbf{R}_{\text{AIR}}$
4. *Let* $x_{\text{APR}} = (F; T; N; \ ; \mathbf{L}; \mathbf{L}_{\text{cmp}}; \sim; \ _{\text{cmp}}) = V^{\text{BAIR} \to \text{APR}}(x; k; R)$*:*
   - *(a)* $|T| = w$
   - *(b)* $|N| = 3w$*, with same* 3 *neighbor polynomials for each witness index* $\quad \in T$
   - *(c)* $| \ | \quad 2|P| + |B|$ *and* $T_{\text{arith}}( \ ) \quad 2T_{\text{arith}}(P) + 3t + \mathbf{O}(w \quad |P| \quad |B|)$
   - *(d)* $\log_2 |\mathbf{L}| = 2 + k + R + t + d$
   - *(e)* $\log_2 |\mathbf{L}_{\text{cmp}}| = k + R + t + d$
   - *(f) The maximal rate* $_{\max}$ *in* $\sim$ *satisfies* $_{\max} \quad 2^{-(2+R+d)}$
   - *(g)* $_{\text{cmp}} \quad 2^{-R}$
   - *(h)* $x_{\text{APR}}$ *is* $1 \quad 2^{-R} \ 1 + 2^{-d}$ *-distance,* 1*-overlapping, and* $2^{-(2+R+d)}(1 \quad 2^{-k})$ *-independent with* $T_{\text{Sampling}}(x_{\text{APR}}) \quad \mathbf{O}(|\mathbf{L}|)$

5. *Arithmetic complexity over* F *(cf. Remark B.1):*

   (a) **Verifier:** *The arithmetic complexity of* $V^{BAIR \to APR}(x; k; R)$ *is that which is required to compute the polynomials* $\mathbf{Z}_{B;\mathbf{j}}; E_{B;\mathbf{j}} \in F[\mathbf{x}]$ *for all* $1 \quad \mathbf{j} \quad w$ *(Appendix C.1.1), and in particular, it is at most* $2|B|^2$ *arithmetic operations*[27].

   (b) **Prover:** *The arithmetic complexity of* $P^{BAIR \to APR}(x; w; k; R)$ *is at most the accumulated arithmetic complexity of (i) the arithmetic complexity of* $V^{BAIR \to APR}(x; k; R)$*, (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* $2^{R+d+2}$ *shifts; in particular, each LDE can be computed in arithmetic complexity at most* $3 \quad (k + t) \quad 2^{k+t+R+d+2}$ *(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.12** (Algebraic placement and routing – permuted version)**.** *There are two algorithms, denoted* $V^{BPAIR \to APR}; P^{BPAIR \to APR}$*, such that for any* $k; R \in \mathbb{N}$*, with* $R > 1$*, and* BPAIR *instance* $x = (F; T; w; P_T; P \ ; B)$ *with* $T = 2^t \quad 1 > 4$*,* $\max\{\deg P_T; \deg P \ ; 2g \quad 2^d$*, and* $[F : F_2] > 2 + k + R + t + \lceil \log(t + 1) \rceil + d$*:*

1. *Perfect completeness:* $(x; w) \in \mathbf{R}_{BPAIR} \ ) \ (V^{BPAIR \to APR}(x; k; R); P^{BPAIR \to APR}(x; w; k; R)) \in \mathbf{R}_{APR}$
2. *Perfect soundness:* $x \in BPAIR \ ( \ V^{BPAIR \to APR}(x; k; R) \in APR$
3. *Knowledge extraction: There is an efficient knowledge extractor* $E^{BPAIR \to APR}$*, such that for every* $(V^{BPAIR \to APR}(x; k; R); w_{APR}) \in \mathbf{R}_{APR}$*, the extractor outputs a witness* $w \quad E^{BPAIR \to APR}(x; k; R; w_{APR})$ *such that* $(x; w) \in \mathbf{R}_{BPAIR}$
4. *Let* $x_{APR} = (F; T \ ; N \ ; \ ; \mathbf{L}; \mathbf{L}_{cmp}; \sim; \ _{cmp}) = V^{BPAIR \to APR}(x; k; R)$*:*

   (a) $|T| = 2(w + 1)$

   (b) $|N| \quad 11|T|$*, with same* 11 *neighbor polynomials for each witness index* $\in T$

   (c) $| \ | \quad 2|P_T| + |P \ | + 8w + 11$ *and* $T_{arith}( \ ) \quad 2T_{arith}(P_T) + T_{arith}(P \ ) + \mathbf{O}(w(|P_T| + |P \ |) + t)$

   (d) $\log_2 |\mathbf{L}| = 2 + k + R + t + \lceil \log(t + 1) \rceil + d$

   (e) $\log_2 |\mathbf{L}_{cmp}| = k + R + t + \lceil \log(t + 1) \rceil + d$

   (f) *The maximal rate* $_{max}$ *in* $\sim$ *satisfies* $_{max} \quad 2^{-(2+R+d)}$

   (g) $_{cmp} \quad 2^{-R}$

   (h) $x_{APR}$ *is* $1 \quad 2^{-R} \ 1 + 2^{-d}$ *-distance, 5-overlapping, and* $2^{-(2+R+d)}(1 \quad 2^{-k})$ *-independent with* $T_{Sampling}(x_{APR}) \quad \mathbf{O}(|\mathbf{L}|)$

5. *Arithmetic complexity over* F *(cf. Remark B.1):*

[27] For large sets of boundary constraints, one may modify the construction so that $\mathbf{Z}_{B;\mathbf{j}}$ is a linearized polynomial, in which case the arithmetic complexity decreases to $\mathbf{O}(|B| \log |B|)$; details omitted because typically $|B|$ is small.

*(a)* **Verifier:** *The arithmetic complexity of* $V^{BPAIR \to APR}(x; k; R)$ *is that which is required to compute the polynomials* $\mathbf{Z}_{B;j}; E_{B;j} \in F[\mathbf{x}]$ *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* $P^{BPAIR \to APR}(x; w; k; R)$ *is at most the accumulated arithmetic complexity of (i) the arithmetic complexity of* $V^{BPAIR \to APR}(x; k; R)$*, (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 $R_{APR}$. 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.13** (Binary RS proximity testing (RPT))**.** *Instances of the RS proximity testing problem (*RPT*) are triples* $x_{RS} = (F; \mathbf{S}; \ldots)$ *where* $\mathbf{S} \subseteq F$ *and* $\ldots \in [0; 1]$*. A witness* $w_{RS}$ *for* $x_{RS}$ *is a function* $w_{RS} : \mathbf{S} \to F$*, and we say it* satisfies $x_{RS}$ *iff and only if* $w_{RS} \in RS[F; \mathbf{S}; \ldots]$*. The relation* $R_{RPT}$ *is the set of pairs* $(x_{RS}; w_{RS})$ *such that* $w_{RS}$ *satisfies* $x_{RS}$*.*

*The* binary RPT *relation (*$R_{BRPT}$*) is one in which the instance satisfies:*

- $F$ *is a binary field*
- $\mathbf{S}$ *is an affine coset of an* $F_2$*-linear subspace of* $F$*, defined by a coset shift* $\mathbf{a}_0$ *and a basis* $(\mathbf{a}_1; \ldots; \mathbf{a}_k)$ *such that* $\mathbf{S} = \{\mathbf{a}_0 + \sum_{i=1}^{k} \alpha_i \mathbf{a}_i \mid \alpha_1, \ldots, \alpha_k \in F_2\}$
- $\ldots = 2^{-R}$ *for* $R \in \mathbb{N}^+$*.*

The following theorem describes the main properties of the ALI protocol. This protocol, described in Appendix D.1, takes an instance-witness pair of $R_{APR}$ and uses one round of interaction to reduce it to two instance-witness pairs of the $R_{BRPT}$ relation.

**Theorem B.14** (Algebraic linking IOP (ALI) properties)**.** *Let* $x := (F; T; N; \ldots; \mathbf{L}; \mathbf{L}_{cmp}; \sim; \ldots_{cmp})$ *be a* $\ldots$*-overlapping and* $\ldots$*-distance* APR *instance with* $\ldots > 0$*. The* ALI *protocol, applied to* $x$*, satisfies the following properties:*

1. **Protocol Schedule**

   *(a)* **Prover:** *On input* $(x; w^{(0)})$*, the output of* $P^{ALI}(x; w^{(0)})$ *is a single oracle* $O_{\text{assignment}}$ *comprised of* $|T| + 1$ *functions in* $F^{\mathbf{L}}$ *and a single function in* $F^{\mathbf{L}_{cmp}}$*;*

   *(b)* **Verifier:** *On input* $x$ *the output of* $V^{ALI}(x)$ *is a single message* $\mathbf{R}$ *comprised of* $2|T| + |\ldots|$ *random field elements and a pair* $x_{RS} = (F; \mathbf{L}; \ldots_{max}); x^{(0)}_{RS} = (F; \mathbf{L}_{cmp}; \ldots_{cmp})$ *of instances of the* BRPT *relation.*

(c) **Induced output:** *The oracle* $O_{\text{assignment}}$ *and verifier randomness define a pair of functions* $\mathbf{f}^{(0)} : \mathbf{L} \to F$ *and* $\mathbf{g}^{(0)} : \mathbf{L}_{\text{cmp}} \to F$*, such that each entry of* $\mathbf{f}^{(0)}$ *depends on* $|T| + 1$ *entries of* $O_{\text{assignment}}$ *and each entry of* $\mathbf{g}^{(0)}$ *depends on* $|N| + 1$ *entries of* $O_{\text{assignment}}$.

2. **Completeness** *If* $x$ *is satisfied by* $w$ *then for any randomness* $\mathbf{R}$ *we have* $\mathbf{f}^{(0)} \in RS[F; \mathbf{L}; \ell_{\max}]$ *and* $\mathbf{g}^{(0)} \in RS[F; \mathbf{L}_{\text{cmp}}; \ell_{\text{cmp}}]$.

3. **Soundness** *Let*

$$\alpha \triangleq \max\left\{4; 1 + \frac{|\mathbf{L}|}{|\mathbf{L}_{\text{cmp}}|}\right\} \quad (7)$$

*If* $x \notin \mathcal{L}(R)$ *then for every* $O_{\text{assignment}}$ *at least one of the following holds,*

(a) $\Pr_{\mathbf{R}}\left[\delta_{\mathbf{f}^{(0)}}(RS[F; \mathbf{L}; \ell_{\max}]) < \frac{\alpha}{2}\right] \leq \frac{1}{|F|}$

(b) $\Pr_{\mathbf{R}}\left[\delta_{\mathbf{g}^{(0)}}(RS[F; \mathbf{L}_{\text{cmp}}; \ell_{\text{cmp}}]) < \frac{\alpha}{2}\right] \leq \frac{1}{|F|}$

4. **Knowledge extraction** *Assume* $\ell_{\max} \geq \frac{1}{4}$. *Then there exists a Las Vegas (randomized) polynomial time algorithm* $E$ *for which the following holds. If* $w'$ *does not satisfy both conditions claimed in Items 3a and 3b above, then the output of* $E$ *on input* $(x; w')$ *is a witness* $w^{(0)}$ *that satisfies* $x$.

5. **Perfect Zero Knowledge** *If the instance* $x$ *is* $\mathcal{L}$*-independent, then the* ALI *protocol has perfect zero knowledge against any verifier that makes at most* $|\mathbf{L}|$ *queries to each* $\mathbf{w}_i$*. This holds for any choice of RS-IOPP sub-protocols used in step 2c of the protocol (cf. Appendix D.1).*

6. **Arithmetic complexity**

(a) **Prover:** *Assuming prover has* $w^{(0)}$ *satisfying* $x$*, prover arithmetic complexity is at most*

$$|T| \cdot |\mathbf{L}| \cdot (3 \log |\mathbf{L}| + 5) + 2|\mathbf{L}_{\text{cmp}}| \cdot (T_{\text{arith}}(\mathcal{C}) + |\mathcal{C}| + 1) \quad (8)$$

*where* $T_{\text{arith}}(\mathcal{C})$ *denotes the sum of the arithmetic complexities of the constraint set* $\mathcal{C}$.

(b) **Verifier:** $V^{\text{ALI}}$ *requires no arithmetic complexity, and involves only sampling* $2|T| + |\mathcal{C}|$ *uniformly random field elements from* $F$.

### 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 $\frac{1}{|F|}$, translating to a lower bound on soundness error of at least $1/|F|$. Repeating Step 1b for a number $\mathbf{k}$ of times, each with independent randomness $\mathbf{R}_1, \ldots, \mathbf{R}_{\mathbf{k}}$, reduces soundness error to $\frac{1}{|F|^k}$. We stress that for constant $\mathbf{k}$, 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 $\mathbf{O}(|\mathbf{L}| \log |\mathbf{L}|)$ operations (first summand of Equation (8)); subsequent steps are linear in $|\mathbf{L}|$ (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 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 $\mathbf{L} \subseteq F$ an affine space and $\mathbf{L}_0 \subseteq \mathbf{L}$ a subspace of size $|\mathbf{L}_0| = 2^{\ell}$, we define the $\mathbf{L}_0$-blockwise distance between $\mathbf{f}, \mathbf{g} : \mathbf{L} \to F$ be the fraction of cosets $\mathbf{S}$ of $\mathbf{L}_0$ in $\mathbf{L}$ on which $\mathbf{f}|_{\mathbf{S}} \neq \mathbf{g}|_{\mathbf{S}}$, and let $S(\mathbf{L}_0, \mathbf{L})$ denote the set of cosets of $\mathbf{L}_0$ that are contained in $\mathbf{L}$.

With this notation in mind, we say that $\mathbf{L}_0$ and $\mathbf{L}_0'$ are *good* for an instance $x = (F, T, N, \ell, \mathbf{L}, \mathbf{L}_{cmp}, \sim, \delta_{cmp})$, if (i) $\mathbf{L}_0 \subseteq \mathbf{L}$ and $\mathbf{L}_0' \subseteq \mathbf{L}_{cmp}$, and (ii) for every neighbor $\mathbf{N} \in N$ and subset $\hat{S} \subseteq S(\mathbf{L}_0', \mathbf{L}_{cmp})$, it holds that

$$\left| \left\{ \mathbf{S} \in S(\mathbf{L}_0, \mathbf{L}) \mid \mathbf{S} \setminus \mathbf{N} \subseteq \bigcup_{\hat{\mathbf{S}} \in \hat{S}} \hat{\mathbf{S}} \right\} \right| \geq \left| \bigcup_{\hat{\mathbf{S}} \in \hat{S}} \hat{\mathbf{S}} \right|.$$

Informally, this condition means that attempting to change the all entries of the union of cosets in $\hat{S}$ by changing entries of $w$ in the set $\mathbf{N} \cup \bigcup_{\hat{\mathbf{S}} \in \hat{S}} \hat{\mathbf{S}}$ will modify at least as many cosets of $S(\mathbf{L}_0, \mathbf{L})$ as there are *elements* in $\bigcup_{\hat{\mathbf{S}} \in \hat{S}} \hat{\mathbf{S}}$.

**Conjecture B.15.** *Suppose* $x = (F, T, N, \ell, \mathbf{L}, \mathbf{L}_{cmp}, \sim, \delta_{cmp})$ *is an unsatisfiable instance of* ALI *with* $\delta$*-distance, and let* $\mathbf{L}_0, \mathbf{L}_0'$ *be* good *for* $x$ *and satisfy* $|\mathbf{L}_0| = |\mathbf{L}_0'| = 2^{\ell}$. *Let* $\Delta^{(0)}(\mathbf{f}^{(0)})$ *denote the distance of* $\mathbf{f}^{(0)}$ *from* RS$[F, \mathbf{L}, \delta_{max}]$ *using the blockwise distance measure induced by* $\mathbf{L}_0$. *Similarly, let* $\Delta^{(0)}(\mathbf{g}^{(0)})$ *denote the distance of* $\mathbf{g}^{(0)}$ *from* RS$[F, \mathbf{L}_{cmp}, \delta_{cmp}]$ *using the blockwise distance measure induced by* $\mathbf{L}_0'$. *Notice* $\Delta^{(0)}(\mathbf{f}^{(0)})$ *and* $\Delta^{(0)}(\mathbf{g}^{(0)})$ *are random variables that depend on the verifier randomness* $\mathbf{R}$. *Then*

$$\Pr_{\mathbf{R}}\left[\Delta^{(0)}(\mathbf{f}^{(0)}) < \min\left(1 - \delta_{max}, \frac{2|\mathbf{L}_{cmp}|}{|\mathbf{L}|}\right) \cdot \Delta^{(0)}(\mathbf{g}^{(0)}) - \frac{1}{|F|}\right] \tag{9}$$

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.16.** *For every pseudo-prover* $P^*$ *and sufficiently large instance* $x = (F, T, N, \ell, \mathbf{L}, \mathbf{L}_{cmp}, \sim, \delta_{cmp})$, *we have*

$$\Pr\left[\Delta^{(0)}(\mathbf{f}^{(0)}) < 1 - \delta_{max} \wedge \Delta^{(0)}(\mathbf{g}^{(0)}) < 1 - \delta_{cmp} - \frac{1}{|F|}\right]. \tag{10}$$

*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.17** (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* $\mathbf{N} = |\mathbf{L}|$ *denotes blocklength (which equals Prover's input length for a fixed* RS$[F, \mathbf{L}, \cdot]$ *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* $\lceil \frac{(\log \mathbf{N}) - R}{2} \rceil$. *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* $6\mathbf{N}$ *arithmetic operations in* F*; proof length is less than* $\mathbf{N}/3$ *field elements and round complexity is at most* $\frac{\log \mathbf{N}}{2}$*;*
- **Verifier:** *for query-repetition parameter* $\ell \in \mathbb{N}^+$*, query complexity is* $2\ell \log \mathbf{N}$*; the verifier decision is computed using at most* $\ell \cdot 21 \log \mathbf{N}$ *arithmetic operations over* F
- **Soundness:** *There exists* $\alpha_0 \in \left(1 - \frac{1}{4}(1-\rho^3), 1 - \sqrt{\rho} + \frac{1}{\sqrt{\mathbf{N}}}\right)$ *such that every* $\mathbf{f}$ *that is* $\delta$*-far in relative Hamming distance from the code, is accepted with probability at most*

$$\mathrm{err}_{\mathrm{FRI}}(\delta) \stackrel{4}{=} \frac{3\mathbf{N}}{|F|} + (1 - \min\{\delta, \alpha_0\})^\ell. \tag{11}$$

  *where* $\ell \in \mathbb{N}^+$ *is the repetition parameter defined above.*
- **Parallelization:** *Each prover-message can be computed in* $\mathbf{O}(1)$ *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 $\alpha^{(0)}$, for any distance value $\alpha^{(0)}$, even when $\alpha^{(0)} \approx 1$. The following conjecture formalizes this; see [14] for a discussion of it.

**Conjecture B.18.** *Using the notation of Theorem B.17, for any* $\mathbf{f} : \mathbf{L} \to F$ *of blockwise distance* $\alpha^{(0)}$ *from* $\mathrm{RS}[F, \mathbf{L}, \mathbf{L}, |\mathbf{L}| = 2^{\mathbf{k}}]$ *and* $\delta \in [0, \alpha^{(0)}]$*, with probability at least*

$$1 - \frac{(\mathbf{k})^2}{\rho^2 |F|} \tag{12}$$

*over the randomness of the verifier during the* COMMIT *phase, the probability of rejection during the* QUERY *phase is at least* $1 - \alpha^{(0)}(1 - \delta)^\ell$.

## B.7 Proof of main theorems

In this section we prove Theorems 3.4 and 3.5 and Lemma B.5.

### 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.19** (Reduction to BAIR)**.** *Given* $\mathrm{L} \in \mathrm{NTimeSpace}(\mathbf{T}(\mathbf{n}), \mathbf{S}(\mathbf{n}))$*, there exists a reduction mapping an instance* x *of* L *of size* $\mathbf{n}$ *to an* $(\mathrm{n}, \mathrm{t}, \mathrm{d})$*-*BAIR *instance* $\mathrm{x}_{\mathrm{BAIR}} = (F, T, w, P, B)$ *in time in time* $\mathrm{poly}(\mathrm{n}, \mathbf{n})$ *(see Remark B.1). The resulting* $\mathrm{x}_{\mathrm{BAIR}}$ *satisfies the conditions of Lemma B.5, and furthermore*

$$w = \mathbf{S}(\mathbf{n}) + \mathbf{O}(1), \quad t = \log \mathbf{T}(\mathbf{n}) + \mathbf{O}(1), \quad |B| = \mathbf{n}, \quad T_{\mathrm{arith}}(P) = \mathbf{O}(\mathbf{S}(\mathbf{n})) \text{ and } d = \mathbf{O}(1). \tag{13}$$

*The asymptotic constants above are independent of* n*.*

*Given* $\mathrm{x}, \mathrm{x}_{\mathrm{BAIR}}$ *as above, and a witness* w *such that* $(\mathrm{x}, \mathrm{w}) \in \mathrm{R}_{\mathrm{L}}$*, a* BAIR *witness* $\mathrm{w}_{\mathrm{BAIR}}$ $(\mathrm{x}_{\mathrm{BAIR}}, \mathrm{w}_{\mathrm{BAIR}}) \in \mathrm{R}_{\mathrm{BAIR}}$ *can be computed in time* $\mathrm{poly}(\mathbf{T}(\mathbf{n}) \cdot \mathbf{S}(\mathbf{n}), \mathrm{n})$*. Vice versa, given* $\mathrm{x}, \mathrm{x}_{\mathrm{BAIR}}, \mathrm{w}_{\mathrm{BAIR}}$ *such that* $(\mathrm{x}_{\mathrm{BAIR}}, \mathrm{w}_{\mathrm{BAIR}}) \in \mathrm{R}_{\mathrm{BAIR}}$*, a witness* w *such that* $(\mathrm{x}, \mathrm{w}) \in \mathrm{R}_{\mathrm{L}}$ *can be computed (or extracted) in time* $\mathrm{poly}(\mathbf{T}(\mathbf{n}) \cdot \mathbf{S}(\mathbf{n}), \mathrm{n})$.

*Proof of Theorem 3.4.* Given instance x of L of size $\mathbf{n}$, denote $\mathbf{T} = \mathbf{T}(\mathbf{n}); \mathbf{S} = \mathbf{S}(\mathbf{n})$ and $= (\mathbf{n})$. Set R = 3 and let $= (1 \quad 2^{R}(1 + 2^{d}))$, for d given by Lemma B.19; notice is independent of n (which we have yet to specify). Let k be the smallest positive integer such that $t + k > 6 + \log + \log\log \frac{1}{1 \quad \frac{1}{10}}$, noticing $k \quad + \mathbf{O}(1)$. Now let $n = \quad + t + d + R + k + 3$, noticing $n \quad 2 \quad + t + \mathbf{O}(1)$. Apply Lemma B.19 and let $x_{BAIR}$ be the resulting $(n; t; d)$ instance. By construction, this instance satisfies the assumptions of Lemma B.5, so we apply the ZK-IOP of knowledge specified there to $x_{BAIR}$. Soundness, knowledge extraction and round complexity follow directly from Lemmas B.5 and B.19. 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. □

### B.7.2 Proof of Main Lemma B.5

*Proof of Lemma B.5.* We are given an instance-witness pair $(x_{BAIR}; w_{BAIR})$ for $R_{BAIR}$ where $x_{BAIR} = (F; T; w; P; B)$ satisfies the assumptions of the the lemma.

**Protocol description** Prover and verifier apply the following reductions:

1. the deterministic reduction of Theorem B.11: the prover executes $P^{BAIR \,!\, APR}(x_{BAIR}; w_{BAIR})$, leading to the witness $w_{APR}$; the verifier executes $V^{BAIR \,!\, APR}(x_{BAIR})$. By Theorem B.11 the resulting instance $x_{APR} = (F; T; N; \ ; \mathbf{L}; \mathbf{L}_{cmp}; \sim; \ _{cmp})$ has parameters as stated in Item 4 there; additionally, $x_{APR}$ is 1-overlapping, $2^{(2+R+d)}(1 \quad 2^{k}))$-independent and has -distance for $= (1 \quad 2^{R}(1 + 2^{d}))$;

2. the 1-round ALI protocol on $(x_{APR}; w_{APR})$: Prover sends the oracle $O_{assignment}$, then verifier sends public randomness that, with $O_{assignment}$, induces a a pair of functions $\mathbf{f}^{(0)} : \mathbf{L} \ ! \ F; \mathbf{g}^{(0)} : \mathbf{L}_{cmp} \ ! \ F$ that we denote by $w_{RS}; w^{0}_{RS}$, respectively. Notice that the parameter defined in Equation (7) equals 5 because $x_{APR}$ is 1-overlapping and $\frac{j\mathbf{L}j}{j\mathbf{L}_{cmp}j} = 4$

3. the FRI protocol is now applied to each the two witnesses; Verifier uses instances (i) $x_{RS} = (F; \mathbf{L}; \ _{max})$ for $w_{RS}$, and (ii) $x^{0}_{RS} = (F; \mathbf{L}_{cmp}; \ _{cmp})$ for $w^{0}_{RS}$. The repetition parameter in both cases is set to

$$\text{‘} = d\frac{+ 2}{\log \frac{1}{1 \quad =10}}e: \tag{14}$$

**Completeness** Perfect completeness follows directly from the perfect completeness of the various reductions: If $(x_{BAIR}; w_{BAIR}) \ 2 \ R_{BAIR}$ then by the completeness part of Theorem B.11 the pair $(x_{APR}; w_{APR}) \ 2 \ R_{APR}$; so, by the completeness of Theorem B.14 $(x_{RS}; w_{RS})$ and $(x^{0}_{RS}; w^{0}_{RS})$ belong to BRPT, hence, by the completeness of Theorem B.17 the verifier accepts both with probability 1.

**Soundness** Suppose $x_{BAIR} \ @ \ L_{BAIR}$. Then by the soundness of Theorem B.11 we also have $x_{APR} \ @ \ L_{APR}$. Hence, by the soundness of Theorem B.14 we have with all but probability $_{ALI} = 1=jFj \quad 2^{( \ +3)}$ have that at least one of $\mathbf{f}^{(0)}; \mathbf{g}^{(0)}$ is $^{(0)}$-far from the relevant RS code, where $^{(0)} = \frac{}{2} = \frac{}{10}$; assume $\mathbf{f}^{(0)}$ is it (the analysis for $\mathbf{g}^{(0)}$ is identical). Notice that $^{(0)} < \ _{0}$ for $\ _{0}$ as defined in Theorem B.17 because R 2. Therefore, by Theorem B.17, but for probability $err_{COMMIT} = \frac{3j\mathbf{L}j}{jFj} < 2^{\ +3}$ over the randomness of the

COMMIT phase, the probability of accepting $\mathbf{f}^{(0)}$ during the QUERY phase, conducted with repetition parameter $\ell$, is at most

$$\mathrm{err}_{\mathrm{QUERY}} = \left(1 - \frac{1}{10}\right)^{\ell} \cdot 2^{-(\kappa+2)}. \tag{15}$$

Summing up, the total probability of error is at most $\mathrm{err}_{\mathrm{ALI}} + \mathrm{err}_{\mathrm{COMMIT}} + \mathrm{err}_{\mathrm{QUERY}} \leq \mathrm{err}$, as claimed.

**Knowledge extraction** Let $\mathcal{P}$ 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 $\mathbf{p} > 4 \cdot \mathrm{err}$. Therefore, with probability at least $\mathbf{O}(1/(\mathbf{p} - \mathrm{err}))$, the oracle $O_{\mathrm{assignment}}$ provided by the Prover at the end of step 1 above, leads the verifier to accept with probability $> 2\mathrm{err}$. Fix such an $O_{\mathrm{assignment}}$. By the soundness of the FRI protocol as stated in Theorem B.17 and our setting of parameters, we conclude that that with probability strictly greater than $1/|\mathbb{F}|$ both $\Delta_H(\mathbf{f}^{(0)}, \mathrm{RS}[\mathbb{F}, \mathbf{L}, \rho]) < \frac{1}{2}$ and $\Delta_H(\mathbf{g}^{(0)}, \mathrm{RS}[\mathbb{F}, \mathbf{L}_{\mathrm{cmp}}, \rho_{\mathrm{cmp}}]) < \frac{1}{2}$. Noticing $\delta_{\max} = 2^{-R} \leq 1/4$, the knowledge extractor of Theorem B.14 outputs with high probability a witness $w_{\mathrm{APR}}$ that satisfies $x_{\mathrm{APR}}$. Hence, by the knowledge extraction property of Theorem B.11, Item 3, we conclude that a witness $w_{\mathrm{AIR}}$ can be extracted from $\mathcal{P}$ in time $\mathrm{poly}\left(\frac{T}{\mathbf{p} - \mathrm{err}}\right)$ 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 $r = \frac{\log|\mathbf{L}| - R}{2}$ and $2^{\eta} = 4$ queries to each oracle per test (there are $\ell$ many tests). Each query to $\mathbf{f}^{(0)}$ is simulated by making a single query to each of the $w$ members of $w_{\mathrm{APR}}$ and a single query to $\mathbf{f}_{\mathrm{mask}}$, and each query to $\mathbf{g}^{(0)}$ is simulated by making 3 queries to each member of $w_{\mathrm{APR}}$ and a single query to $\mathbf{g}_{\mathrm{mask}}$. Using the value of $\ell$ above, and the equalities $\dim(\mathbf{L}) = 2 + \dim(\mathbf{L}_{\mathrm{cmp}}) = k + R + t + d + 2$ from Theorem B.11, total query complexity is thus

$$\begin{aligned} q &= 2 \cdot \ell \cdot (4w + 2 + \dim(\mathbf{L}) - R + \dim(\mathbf{L}_{\mathrm{cmp}}) - R) \\ &= 2^{\eta+1} \cdot \ell \cdot (2w + k + t + d + 2) \\ &= 8\left\lceil \frac{\kappa + 2}{\log\frac{1}{1 - \rho - \eta/10}} \right\rceil (2w + k + t + d + 2) \end{aligned} \tag{16}$$

**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 $2|\mathbf{B}|^2$ arithmetic operations (see Footnote 27); (ii) the arithmetic complexity of the FRI verifier, at a cost of $21 \cdot \ell \cdot \log|\mathbf{L}|$, (iii) the cost of computing an entry of $\mathbf{f}^{(0)}$ from $w$, which costs $4(w + 1)$ operations per query (each test makes $2^{\eta} = 4$ such queries), and (vi) the computation of $\mathbf{g}^{(0)}$, which costs $4(|\mathbf{j}| + T_{\mathrm{arith}}(\mathbf{j}) + 1)$, where $T_{\mathrm{arith}}(\mathbf{j}) = T_{\mathrm{arith}}(\mathcal{P}) + |\mathbf{B}|$; recall that here we also make 4 queries per test. Summing up, total verifier arithmetic complexity is

$$\begin{aligned} t_{\mathcal{V}} &\leq 2|\mathbf{B}|^2 + \ell \cdot (16(w + |\mathbf{j}| + T_{\mathrm{arith}}(\mathbf{j})) + 2 \cdot 21(k + R + t + d + 2)) \\ &= 2|\mathbf{B}|^2 + \left\lceil \frac{\kappa + 2}{\log\frac{1}{1 - \rho - \eta/10}} \right\rceil (8(w + |\mathbf{j}| + T_{\mathrm{arith}}(\mathcal{P}) + |\mathbf{B}|) + 21(k + R + t + d + 2)) \end{aligned} \tag{17}$$

**Prover arithmetic complexity** Inspection of the prover arithmetic complexity shows that the components that dominate it are the $3w$ 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 $|\mathbf{L}|$. The computation of $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$ require $|\mathbf{L}|$ (4| | + 4w + $T_{\text{arith}}(P)$ + |B|) arithmetic operations and the two invocations of the FRI prover cost 2 6 $|\mathbf{L}|$ operations. This completes the analysis of prover complexity.

**Zero knowledge** By Theorem B.11, Item 4h, the instance $x_{\text{APR}}$ is -independent for = $2^{(2+R+d)}(1 - 2^{-k})$ and $|\mathbf{L}| = 2^{2+k+R+t+d}$. By Item 5 of Theorem B.14, if $|\mathbf{L}|$ is greater than the number of queries made into each member $\mathbf{w}$ of $O_{\text{assignment}}$ then the protocol has perfect zero knowledge. We have

$$|\mathbf{L}| = 2^{2+k+R+t+d-(2+R+d)}(1 - 2^{-k}) = 2^{k+t}(1 - 2^{-k}) \quad 2^{k+t-1}: \tag{18}$$

The last equality holds because k 1.

A single query to $\mathbf{f}^{(0)}$ requires a single query to $\mathbf{f}^{(0)}$ and a single query to $\mathbf{g}^{(0)}$ requires 3 queries to each $\mathbf{f}^{(0)}$. Each test of the FRI verifier makes 2 = 4 queries to either $\mathbf{f}^{(0)}$ or $\mathbf{g}^{(0)}$ and we use the repetition parameter $\ell$ defined above. Therefore, the total number of queries into each $\mathbf{w}$ is bounded 16 $\ell$. Combining this number with Equation (18) shows that for t 4 + log $\ell$ the protocol has perfect zero knowledge, thus, it holds for

$$k + t \quad 6 + \log \quad + \log\log \frac{1}{1 - \frac{1}{\sqrt{10}}}:$$

□

### B.7.3 Proof of Lemma B.6

*Proof of Lemma B.6.* The proof is essentially the same as that of Lemma B.5, with the following changes to the protocol description. We use a smaller field, as specified in the statement of Lemma B.6. In Step 3 of the protocol above, we repeat the FRI-COMMIT phase more than once on $\mathbf{f}^{(0)}, \mathbf{g}^{(0)}$, leading to several sets of oracles; let $\mathbf{s}$ denote the number of COMMIT repetitions (to be specified below). Additionally, we use different repetition parameters for $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$, and both are smaller than specified above. Finally, each QUERY invocation is applied to a randomly selected COMMIT oracle-set, out of $\mathbf{s}$.

Denoting by $\ell_{\mathbf{f}^{(0)}}, \ell_{\mathbf{g}^{(0)}}$ the repetition parameters for the FRI-QUERY phase for $\mathbf{f}^{(0)}, \mathbf{g}^{(0)}$, respectively, we set them (and $\mathbf{s}$) thus

$$\mathbf{s} = \left\lceil \frac{+2}{n \quad (10 + 2\log n)} \right\rceil; \qquad \ell_{\mathbf{f}^{(0)}} = \left\lceil \frac{+2}{R + d} \right\rceil; \qquad \ell_{\mathbf{g}^{(0)}} = \left\lceil \frac{+2}{R} \right\rceil \tag{19}$$

Since the modified protocol follows that of Lemma B.5, 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.5, 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

$$2 \quad \ell_{\mathbf{f}^{(0)}} + 3\ell_{\mathbf{g}^{(0)}} = 4 \quad \left\lceil \frac{+2}{R + d} \right\rceil + 3\left\lceil \frac{+2}{R} \right\rceil \quad 16 \quad \left\lceil \frac{+2}{R} \right\rceil:$$

Thus, this inequality holds when $k + t > 5 + \log\lceil \frac{+2}{R} \rceil$.

The computation of query and verifier follow directly from the definition of $\ell_{\mathbf{f}^{(0)}}; \ell_{\mathbf{g}^{(0)}}$ above, and gives

$$\begin{aligned} q &= 2 \quad \ell_{\mathbf{f}^{(0)}} \quad (w + 3 + k + t + d) + \ell_{\mathbf{g}^{(0)}} \quad (3w + 3 + k + t + d) \\ &= 4 \quad d\frac{+2}{R+d}e \quad (w + 3 + k + t + d) + d\frac{+2}{R}e \quad (3w + 3 + k + t + d) \\ &\quad 8d\frac{+2}{R}e\,(2w + 3 + k + t + d) \end{aligned}$$

$$tv^{F} \quad 2j\mathsf{B}j^{2} + d\frac{+2}{R}e \quad (16(w + j \quad j + T_{\text{arith}}(P) + 42 \quad (k + R + t + d + 2)))$$

Prover complexity accounts for the $\mathbf{s}$ repetitions of the FRI-COMMIT phase, and is

$$tp^{F} \qquad 9w(t + d + R + 3) + jPj + w + T_{\text{arith}}(P) + 12d\frac{+2}{n \quad (10 + 2\log n)}e \qquad 2^{t+d+R+3}$$

**Soundness error** Under Conjecture B.16, with all but probability $\text{err}_{\text{ALI}} = 1/jFj < 2^{\quad(+2)}$ we have that either $\mathbf{f}^{(0)}$ has distance at least $_{\mathbf{f}^{(0)}} = 1 \quad 2^{\quad(R+d)}$ from the code RS[F; $\mathbf{L}$; $_{\max}$], or $\mathbf{g}^{(0)}$ has distance at least $_{\mathbf{g}^{(0)}} = 1 \quad 2^{\quad R}$ from RS[F; $\mathbf{L}_{\text{cmp}}$; $_{\text{cmp}}$]; both $_{\mathbf{f}^{(0)}}; _{\mathbf{g}^{(0)}}$ are block-wise distances. Assume $_{\mathbf{g}^{(0)}} = 1 \quad 2^{\quad R}$ (the case of $_{\mathbf{f}^{(0)}} \quad 1 \quad _{\max}$ is argued similarly). Fixing $= 2^{\quad 10}$ (somewhat arbitrarily) and recalling $= 2$, Conjecture B.18 implies that with all but probability $\text{err}_{\text{COMMIT}} = \frac{2^{10}\mathbf{k}^2}{jFj}$ over the randomness of the verifier during the COMMIT phases, the ensuing QUERY phase, repeated $\ell_{\mathbf{g}^{(0)}}$ times, accepts with probability at most $\ell_{\mathbf{f}^{(0)}}^{\ell_{\mathbf{f}^{(0)}}} \ell_{\mathbf{g}^{(0)}}^{\ell_{\mathbf{g}^{(0)}}}$. Assuming $_{\mathbf{g}^{(0)}} \quad 1 \quad 2^{\quad R}$ and simplifying notation to $\mathbf{p} = \text{err}_{\text{COMMIT}}; = _{\mathbf{g}^{(0)}}$ and $\ell = \ell_{\mathbf{g}^{(0)}}$, by the law of conditional probability the error probability is at most

$$\sum_{\mathbf{i}=0}^{\mathbf{s}} \binom{\mathbf{s}}{\mathbf{i}} \mathbf{p}^{\mathbf{i}}(1)(1 \quad \mathbf{p})^{\mathbf{i}} \quad \frac{\mathbf{i}}{\mathbf{s}} + \frac{(\mathbf{s} \quad \mathbf{i})}{\mathbf{s}}^{\ell} \quad \sum_{\mathbf{i}=0}^{\mathbf{s}} (\mathbf{s}\mathbf{p})^{\mathbf{i}} \quad \frac{\mathbf{i} + (\mathbf{s} \quad \mathbf{i})}{\mathbf{s}}^{\ell} \tag{20}$$

We claim that each of the $\mathbf{s}$ terms on the right hand side above is bounded by $^{\ell}/_{\mathbf{s}}$, which is equivalent to claiming

$$\mathbf{s} \quad (\mathbf{s} \quad \mathbf{p})^{\mathbf{i}} \qquad \frac{\mathbf{s}}{\mathbf{i} + (\mathbf{s} \quad \mathbf{i})}^{\ell} :$$

By the assumption $\quad 2 [1/2; 1]$, the base of the exponent on the right hand side is at least $1/2$, so it suffices to show

$$\mathbf{s} \quad (\mathbf{s} \quad \mathbf{p})^{\mathbf{i}} \quad 2^{\ell}$$

taking logarithms, it suffices to show $\mathbf{i}\log\frac{1}{\mathbf{p}} \quad (\mathbf{i} + 1)\log\mathbf{s} \quad \ell$, which we shall do even for $\mathbf{i} = 1$. We have $\log\frac{1}{\mathbf{p}} \quad n \quad (10 + 2\log n)$ because $\mathbf{k} \quad n$, and $\log\mathbf{s} \quad 1 + \log \quad \log n$ thus, assuming

$$n \quad (12 + 4\log n) \quad d\frac{+2}{R}e$$

which holds for all $n \quad 60$ because $n >$ , we conclude that indeed each summand of Equation (20) is bounded by $\ell/\mathbf{s}$, hence $\text{err}_{\text{FRI}} \quad \ell \quad 2^{\quad(+2)}$. The total error probability of the protocol is thus at most $\text{err}_{\text{ALI}} + \text{err}_{\text{FRI}} < 2$ , as claimed. This completes the proof. □

### B.7.4 Proof of ZK-STIK Theorem 3.5

The proof below follows that of Theorem B.12, 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.5. Then we argue that asymptotic verifier complexity is strictly logarithmic, and prover complexity is strictly $\mathbf{O}(\mathbf{T}(\mathbf{n}) \log^2 \mathbf{T}(\mathbf{n}))$.

*Proof of Theorem 3.5.* The proof follows that of Lemma B.5 with the following different parameter choices:

- In Item 1 of the protocol description in the proof of Lemma B.5, both parties apply the deterministic reduction of Theorem B.12: Prover executes $P^{\text{BPAIR} \,!\, \text{APR}}(x_{\text{BPAIR}}; w_{\text{BPAIR}})$ leading to the witness $w_{\text{APR}}$ and verifier executes $V^{\text{BPAIR} \,!\, \text{APR}}(x_{\text{BPAIR}})$, leading to $x_{\text{APR}}$.
- The repetition parameter is set differently than in the proof of Lemma B.5, to account for the fact that $x_{\text{APR}}$ is 5-overlapping (not 1-overlapping, as above) and thus $\quad = 21$ (cf. Equation (7)) so $^{(0)} = \frac{}{42}$. This increases the repetition parameter $\text{‘}$ defined in Equation (14) to

$$\text{‘} = \mathrm{d}\frac{+2}{\log \frac{1}{1 \;\; \frac{}{42}}}\mathrm{e}$$

  so that $\text{err}_{\text{FRI}}(\;^{(0)})$ will be bounded by err=4 as stated in Equation (15).
- We also have $\dim(\mathbf{L}) = 2 + k + R + t + \mathrm{d}\log(t+1)\mathrm{e} + d$, i.e., the dimension of $\mathbf{L}$ is greater than that used in the proof of Lemma B.5 by an additive factor of $\mathrm{d}\log(t+1)\mathrm{e}$.
- Each query to $\mathbf{f}^{(0)}$ requires now $\mathrm{j}T\,\mathrm{j} = 2(w+1)+1$ queries to $\mathrm{O}_{\text{assignment}}$ (compared with $w+1$ in the proof above) and each query to $\mathbf{g}^{(0)}$ requires $11\mathrm{j}T\,\mathrm{j} = 22(w+1)+11$ queries to $\mathrm{O}_{\text{assignment}}$ rather than $3\mathrm{j}T\,\mathrm{j}$ 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 $4\frac{1}{3} = \frac{39}{9}$ increase in repetition parameter, and replacing w with $25w + 36$ in (16). This give

$$q = 156 \quad (\quad + 2) \quad (25w + t + d + \mathrm{d}\log(t+1)\mathrm{e} + 38) = \mathbf{O}(\quad t)\mathbf{:}$$

The last equality uses $w = \mathbf{O}(t)$ as stated in Lemma B.8.

**Verifier complexity** The arithmetic complexity of the verifier is dominated by the arithmetic complexity of the FRI verifier and the computation of entries of $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$, which should be repeated $\mathbf{O}(\text{‘}) = \mathbf{O}(\;)$ many times. Consequently, and using the bounds on $t; d = \mathbf{O}(1)$, $\mathrm{j}P_T[\;P\;\mathrm{j}; T_{\text{arith}}(P) = \mathbf{O}(\log \mathbf{T}(\mathbf{n}))$, and $\mathrm{j}B\mathrm{j} = \mathbf{O}(\mathbf{n})\mathrm{j}B\mathrm{j}$, we conclude verifier arithmetic complexity is $\mathbf{O}(\quad t)$ as well.

**Prover complexity** Asymptotic prover complexity is similar to that discussed in the proof of Lemma B.5, with two important differences. First, our prover needs to compute a routing on a back-to-back De Bruijn network, costing $\mathbf{O}(\mathbf{T}(\mathbf{n}) \log \mathbf{T}(\mathbf{n}))$ operations (see Theorem G.3); second, the dimension of $\mathbf{L}$ (and of $\mathbf{L}_{\text{cmp}}$) is larger by an additive factor of $\mathrm{d}\log(t+1)\mathrm{e}$. Therefore, the LDE computations are now over spaces of size $\mathbf{O}(\mathbf{T}(\mathbf{n}) \log \mathbf{T}(\mathbf{n}))$ and total arithmetic complexity of the prover is $\mathbf{O}(\mathbf{T}(\mathbf{n}) \log^2 \mathbf{T}(\mathbf{n}))$. □

## 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.6. 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.6, we set R = 3 and = 60, leading to security error of at most err 2 $^{60}$. The binary field we use has jFj = $2^{64}$, 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 8, and this is obtained for dataset sizes **n** $2^3$ because t = d**n** 62e. 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 $\rho_{\mathbf{f}^{(0)}}$ = 9 and $\rho_{\mathbf{g}^{(0)}}$ = 22, as follows from Equation (19).

# C Algebraic placement and routing (APR) reduction

In this section we prove Theorems B.11 and B.12, sequentially. The proof of Theorem B.11 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.11, given in Appendix C.2. Similarly, the proof of Theorem B.12 starts with a description of the reduction in Appendix C.3, followed by the proof of Theorem B.12 in Appendix C.4.

**Notation** In this section we view F as $F_2[g]/\mathbf{h}(g)$ for a primitive polynomial $\mathbf{h}$ (see Remark B.1). For $\mathbf{a}, \mathbf{b} \in F$ viewed as polynomials over $F_2$, the notation $\mathbf{a}\%\mathbf{b}$ represents the remainder of $\mathbf{a}$ divided by $\mathbf{b}$ in the ring of polynomials $F_2[g]$.

## 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:

- $\in F_2[g]$ is a primitive polynomial of degree t, notice $\in F$ because $t < n$
- $\mathbf{H}$ F is the space spanned by $g^{\mathbf{k}} \mid 0 \quad \mathbf{k} < t$
- $\mathbf{H}_0$ $\mathbf{H}$ is the subspace spanned by $g^{\mathbf{k}} \mid 0 \quad \mathbf{k} < t \quad 1$
- $\mathbf{H}_1$ $\mathbf{H}$ is the affine space $\mathbf{H}_1 \stackrel{4}{=} \mathbf{H}_0 + g^{t \ 1}$
- Given an affine space $\mathbf{S}$ F we define the vanishing polynomial $\mathbf{Z_S} \in F[\mathbf{x}]$ to be the monic polynomial of degree $|\mathbf{S}|$ such that $\forall \mathbf{s} \in \mathbf{S}, \mathbf{Z_S}(\mathbf{s}) = 0$
- For every $\mathbf{j} \in [w]$ we define $\mathbf{Z_{B;j}}; E_{\mathbf{B;j}} \in F[\mathbf{x}]$ by $\mathbf{Z_{B;j}}(\mathbf{x}) \stackrel{4}{=} \prod_{(\mathbf{i};\mathbf{j};\ ) \in B} \mathbf{x} \quad g^{\mathbf{i}}\%\ (g)$ and $E_{\mathbf{B;j}}$ is the polynomial of minimal degree such that for every $(\mathbf{i};\mathbf{j};\ ) \in B, E_{\mathbf{B;j}} \quad g^{\mathbf{i}}\%\ (g) =$
- Given a neighbor $(\ ;\mathbf{N}) \in N$ we denote by $(\ ;\hat{\mathbf{N}})$ the expression $(\ ;\hat{\mathbf{N}}) \stackrel{4}{=} X_{(\ ;\mathbf{N})}\ \mathbf{Z}_{\mathbf{B};\ (\mathbf{N}\ (X_{loc}))} + E_{\mathbf{B};\ (\mathbf{N}\ (X_{loc}))}$. We extend this notation further more, and denote by $(\ _1;\ _2;\dots;\ _{\mathbf{k}};\mathbf{N})$ the sequence $(\ _1;\hat{\mathbf{N}});(\ _2;\hat{\mathbf{N}});\dots;(\ _{\mathbf{k}};\hat{\mathbf{N}})$.

### C.1.2 Instance reduction

We now describe the (deterministic) verifier-side reduction $V^{BAIR\ !\ APR}$:

1. $T \stackrel{4}{=} [w]$
2. $N \stackrel{4}{=} \quad ;n^{id} \ ; \quad ;n_0^{cyc} \ ; \quad ;n_1^{cyc} \mid \quad \in T$ where (i) $n^{id}(\mathbf{x}) \stackrel{4}{=} \mathbf{x}$, (ii) $n_{\mathbf{b}}^{cyc}(\mathbf{x}) \stackrel{4}{=} g\mathbf{x} + \mathbf{b}$ for $\mathbf{b} \in \{0;1\}$.
3. For every $\mathbf{j} \in [w]$ denote by $\mathbf{q_j}(\mathbf{x};\mathbf{y}) \stackrel{4}{=} \mathbf{x}\mathbf{Z_{B;j}}(\mathbf{y}) + E_{\mathbf{B;j}}(\mathbf{y})$. Define for each $\mathbf{P} \in P$ the set $_{\mathbf{P}}$ by:

$$_{\mathbf{P}} \stackrel{4}{=} \begin{Bmatrix} \frac{X_{loc}(X_{loc}\ 1)}{\mathbf{Z}_{H_0}(X_{loc})} & \mathbf{P} & 1;\dots;\hat{w};n^{id} \ ; & 1;\dots;\hat{w};n_0^{cyc} & ; \\ \vdots & & & & \\ \frac{1}{\mathbf{Z}_{H_1}(X_{loc})} & \mathbf{P} & 1;\dots;\hat{w};n^{id} \ ; & 1;\dots;\hat{w};n_1^{cyc} & ; \end{Bmatrix}$$

Finally, we define $\stackrel{4}{=} \bigcup_{\mathbf{P} \in \mathrm{P}} \mathbf{P}$

4. Define $\mathbf{L}$ to be the affine space $\mathbf{L} \stackrel{4}{=} \mathrm{span}\ g^{1+k+R+t+d}\ (1+g)\ [\ g^{\mathbf{i}}\ j\ 0 \quad \mathbf{i} < 1+k+R+t+d\ + g^{1+k+R+t+d}$
5. Define $\mathbf{L}_{\mathrm{cmp}}$ to be the affine space $\mathbf{L}_{\mathrm{cmp}} \stackrel{4}{=} \mathrm{span}\ g^{\mathbf{i}}\ j\ 0 \quad \mathbf{i} < k+R+t+d\ + g^{1+k+R+t+d}$
6. For every $\mathbf{j} \in [w]$ define $_{\mathbf{j}}$ to be $_{\mathbf{j}} \stackrel{4}{=} \frac{2^{k+t}\ \deg(\mathbf{Z}_{B;j})}{j\mathbf{L}j}$
7. Define $_{\mathrm{cmp}} \stackrel{4}{=} \frac{1+2^{k+t+d}}{j\mathbf{L}_{\mathrm{cmp}}j}$

### C.1.3 Witness reduction

Finally, we describe the prover-side (randomized) reduction $P^{BAIR \to APR}$, denoting the (input) AIR witness members by $\mathbf{w}_{\mathbf{j}}^{AIR}$ and the (output) APR members by $w_{\mathbf{j}}^{APR}$. We show the construction of $w_{\mathbf{j}}^{APR}$ for every $\mathbf{j} \in [w]$ independently. Draw uniformly random a polynomial $\mathbf{Q_j} : F \to F$ of degree less than $2^{k+t}$ such that for every $\mathbf{i} \in [T]$ it satisfies $\mathbf{Q_j}\ g^{\mathbf{i}}\% \quad = \mathbf{w}_{\mathbf{j}}^{AIR}[\mathbf{i}]$. We define the APR witness by $w_{\mathbf{j}}^{APR}(\mathbf{x}) \stackrel{4}{=} \frac{\mathbf{Q}_j(\mathbf{x})\ E_{B;j}(\mathbf{x})}{\mathbf{Z}_{B;j}(\mathbf{x})}$, for every $\mathbf{x} \in \mathbf{L}$. Notice this division is well defined, as all the roots of $\mathbf{Z}_{B;\mathbf{j}}$ are in $\mathbf{H}$ by definition, and $\mathbf{H} \setminus \mathbf{L} = \ ;$ by construction.

## C.2 Proof of Theorem B.11 for space bounded computation

### C.2.1 Proof of completeness (Item 1)

*Proof.* We show that: (i) for all $\mathbf{j} \in [w]$, $w_{\mathbf{j}}^{APR} \in RS[F; \mathbf{L}; \ _{\mathbf{j}}]$, and (ii) $8 \quad 2 \quad ; \ _N \ \hat{w}^{APR} \in RS[F; \mathbf{L}_{\mathrm{cmp}}; \ _{\mathrm{cmp}}]$.

**Assignment code membership:** For every $\mathbf{j} \in [w]$, $\deg\ \mathbf{Q_j}\ E_{B;\mathbf{j}} < 2^{k+t}$, and by definition of satisfaction of AIR $\mathbf{Z}_{B;\mathbf{j}}\ j\ \mathbf{Q_j}\ E_{B;\mathbf{j}}$, thus $\frac{\mathbf{Q}_j\ E_{B;j}}{\mathbf{Z}_{B;j}}$ is a polynomial of degree less then $2^{k+t}\ \deg\ \mathbf{Z}_{B;\mathbf{j}}$, showing $w_{\mathbf{j}}^{APR} \in RS[F; \mathbf{L}; \ _{\mathbf{j}}]$ as required.

**Constraint code membership:** We notice first that for any $\mathbf{j} \in [w]$ and any neighbor $(\mathbf{j}; \mathbf{N}) \in N$, $(\mathbf{j}; \hat{\mathbf{N}}) = \mathbf{Q_j}\ \mathbf{N}$ (denoting by $\mathbf{f}\ \mathbf{g}$ the function $(\mathbf{f}\ \mathbf{g})(\mathbf{x}) \stackrel{4}{=} \mathbf{f}(\mathbf{g}(\mathbf{x}))$) thus it is sufficient to show that for any $\mathbf{P} \in P$ both rational functions:

$$\frac{\mathbf{x}(\mathbf{x} \quad 1)}{\mathbf{Z}_{\mathbf{H}_0}(\mathbf{x})}\ \mathbf{P}\ (\mathbf{Q}_1(\mathbf{x}); : : : ; \mathbf{Q}_w(\mathbf{x}); \mathbf{Q}_1(g\mathbf{x}); : : : ; \mathbf{Q}_w(g\mathbf{x})) \tag{21}$$

$$\frac{1}{\mathbf{Z}_{\mathbf{H}_1}(\mathbf{x})}\ \mathbf{P}\ (\mathbf{Q}_1(\mathbf{x}); : : : ; \mathbf{Q}_w(\mathbf{x}); \mathbf{Q}_1(g\mathbf{x} \quad ); : : : ; \mathbf{Q}_w(g\mathbf{x} \quad )) \tag{22}$$

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) $\mathbf{H} = g^{\mathbf{i}}\%\ j\ 0 < \mathbf{i} < T \quad 1\ [\ f0; 1g$, and (ii) $0; 1 \in \mathbf{H}_0$

For Equation (21) it is sufficient to show that $\mathbf{P}\left(\mathbf{el}\mathbf{Q}_1(\mathbf{x});\ldots;\mathbf{Q}_w(\mathbf{x});\mathbf{Q}_1(g\mathbf{x});\ldots;\mathbf{Q}_w(g\mathbf{x})\right) = 0$ for every $g^{\mathbf{i}} \in \mathbf{H}_0 \setminus \{0;1\}$. The claim follows by the fact that for any $g^{\mathbf{i}} \in \mathbf{H}_0$, we have $g^{\mathbf{i}+1}\% = g^{\mathbf{i}+1}$, thus by construction

$$\mathbf{P}\left(\mathbf{Q}_1\left(g^{\mathbf{i}}\right);\ldots;\mathbf{Q}_w\left(g^{\mathbf{i}}\right);\mathbf{Q}_1\left(g^{\mathbf{i}+1}\right);\ldots;\mathbf{Q}_w\left(g^{\mathbf{i}+1}\right)\right) \tag{23}$$

$$= \mathbf{P}\left(\mathbf{w}_1^{\mathrm{AIR}}[\mathbf{i}];\ldots;\mathbf{w}_w^{\mathrm{AIR}}[\mathbf{i}];\mathbf{w}_1^{\mathrm{AIR}}[\mathbf{i}+1];\ldots;\mathbf{w}_w^{\mathrm{AIR}}[\mathbf{i}+1]\right) = 0 \tag{24}$$

Where the last equation follows by definition of satisfaction in AIR.

For Equation (22) the claim follows similarly, by noticing $g\mathbf{x}\% = g\mathbf{x}\%$ for all $\mathbf{x} \in \mathbf{H}_1$. □

### C.2.2 Proof of soundness (Item 2)

*Proof.* Assume $V^{\mathrm{BAIR}\,!\,\mathrm{APR}}(x;k;R) \in \mathrm{APR}$ for $x = (F;T;w;P;B)$, and let $\vec{w}^{\mathrm{APR}}$ be a witness for it. We show $x \in \mathrm{BAIR}$ by constructing a witness $w^{\mathrm{BAIR}}$ for it. We define for every $\mathbf{j} \in [w]$ the mapping $\mathbf{w}_{\mathbf{j}}^{\mathrm{BAIR}}: [T] \to F$ by

$$\mathbf{w}_{\mathbf{j}}^{\mathrm{BAIR}}[\mathbf{i}] \stackrel{4}{=} w_{\mathbf{j}}^{\mathrm{APR}}\left(g^{\mathbf{i}}\%\right) - \mathbf{Z}_{B;\mathbf{j}}\left(g^{\mathbf{i}}\%\right) + E_{B;\mathbf{j}}\left(g^{\mathbf{i}}\%\right) \tag{25}$$

denoting by $w_{\mathbf{j}}^{\mathrm{APR}}$ as the low degree extension of the actual witness. We show $w^{\mathrm{BAIR}}$ satisfies $x$. Denote by $\mathbf{q}_{\mathbf{j}}(\mathbf{x};\mathbf{y}) \stackrel{4}{=} \mathbf{x} - \mathbf{Z}_{B;\mathbf{j}}(\mathbf{x}) + E_{B;\mathbf{j}}(\mathbf{x})$.

**Boundary constraints:** $w^{\mathrm{BAIR}}$ satisfies the boundary constraints $B$ by construction, as having $(\mathbf{i};\mathbf{j};\ ) \in B$ implies $\mathbf{Z}_{B;\mathbf{j}}\left(g^{\mathbf{i}}\%\right) = 0$ and $E_{B;\mathbf{j}}\left(g^{\mathbf{i}}\%\right) = $ , thus $\mathbf{w}_{\mathbf{j}}^{\mathrm{BAIR}}[\mathbf{i}] = $ as required.

**Consistency with $P$:** We show that for any $\mathbf{i} \in [T - 1]$ and for any $\mathbf{P} \in P$:

$$\mathbf{P}\left(\mathbf{w}_1^{\mathrm{BAIR}}[\mathbf{i}];\ldots;\mathbf{w}_w^{\mathrm{BAIR}}[\mathbf{i}];\mathbf{w}_1^{\mathrm{BAIR}}[\mathbf{i}+1];\ldots;\mathbf{w}_w^{\mathrm{BAIR}}[\mathbf{i}+1]\right) = 0$$

Assume this is not the case, thus there is some $\mathbf{i}$ and some $\mathbf{P} \in P$ for which the above does not hold, we show in such case $w^{\mathrm{APR}}$ can not be a witness of $V^{\mathrm{BAIR}\,!\,\mathrm{APR}}(x;k;R)$. Denote by $(F;T;N;\ ;\mathbf{L};\mathbf{L}_{\mathrm{cmp}};\sim;\ _{\mathrm{cmp}}) = V^{\mathrm{BAIR}\,!\,\mathrm{APR}}(x;k;R)$. Assume $w_{\mathbf{j}}^{\mathrm{APR}} \in \mathrm{RS}[F;\mathbf{L};\ _{\mathbf{j}}]$, and we show $\exists\ \in\ $ such that $_N\,\vec{w}^{\mathrm{APR}} \notin \mathrm{RS}[F;\mathbf{L}_{\mathrm{cmp}};\ _{\mathrm{cmp}}]$. Without loss of generality assume $g^{\mathbf{i}}\% \in \mathbf{H}_0 \setminus \{0;1\}$, we show the following constraint is not a member of $\mathrm{RS}[F;\mathbf{L}_{\mathrm{cmp}};\ _{\mathrm{cmp}}]$:

$$(\mathbf{x}) = \frac{\mathbf{x}(\mathbf{x}\ -\ 1)}{\mathbf{Z}_{\mathbf{H}_0}(\mathbf{x})}\ \mathbf{P}\begin{matrix}\mathbf{q}_1\left(w_1^{\mathrm{APR}}(\mathbf{x});\mathbf{x}\right);\ldots;\mathbf{q}_w\left(w_w^{\mathrm{APR}}(\mathbf{x});\mathbf{x}\right);\\ \mathbf{q}_1\left(w_1^{\mathrm{APR}}(g\mathbf{x});g\mathbf{x}\right);\ldots;\mathbf{q}_w\left(w_w^{\mathrm{APR}}(g\mathbf{x});g\mathbf{x}\right)\end{matrix}$$

Otherwise

$$(\mathbf{x})\ \mathbf{Z}_{\mathbf{H}_0}(\mathbf{x}) = \mathbf{x}(\mathbf{x}\ -\ 1)\ \mathbf{P}\begin{matrix}\mathbf{q}_1\left(w_1^{\mathrm{APR}}(\mathbf{x});\mathbf{x}\right);\ldots;\mathbf{q}_w\left(w_w^{\mathrm{APR}}(\mathbf{x});\mathbf{x}\right);\\ \mathbf{q}_1\left(w_1^{\mathrm{APR}}(g\mathbf{x});g\mathbf{x}\right);\ldots;\mathbf{q}_w\left(w_w^{\mathrm{APR}}(g\mathbf{x});g\mathbf{x}\right)\end{matrix} \tag{26}$$

for every $\mathbf{x} \in \mathbf{L}_{\mathrm{cmp}}$, where both sides, as codewords over $\mathbf{L}_{\mathrm{cmp}}$, are of rate at most

$$_{\mathrm{cmp}} + \frac{|\mathbf{H}_0|}{|\mathbf{L}_{\mathrm{cmp}}|}\ \frac{2^{k+t+d} + 2^t}{2^{k+R+t+d}}\ \ 2^{\ R}\ \ 1 + 2^{\ k\ \ d} < 1$$

But they don't agree over the entire field F, as for $g^{i\%}$ the left hand of the equation vanishes as $\mathbf{Z}_{\mathbf{H}_0}(g^{i\%}) = 0$, while the right hand does not vanish as

$$\mathbf{P}\begin{matrix} q_1\ w_1^{APR}\ g^{i}\ ;g^{i}\ ;\dots;q_w\ w_w^{APR}\ g^{i}\ ;g^{i}\ ; \\ q_1\ w_1^{APR}\ g^{i+1}\ ;g^{i+1}\ ;\dots;q_w\ w_w^{APR}\ g^{i+1}\ ;g^{i+1} \end{matrix} = \tag{27}$$

$$\mathbf{P}\ \mathbf{w}_1^{BAIR}[\mathbf{i}];\dots;\mathbf{w}_w^{BAIR}[\mathbf{i}];\mathbf{w}_1^{BAIR}[\mathbf{i}+1];\dots;\mathbf{w}_w^{BAIR}[\mathbf{i}+1] \neq 0 \tag{28}$$

The contradiction follows by showing both sides are polynomials of degree less then $_{cmp}|\mathbf{L}_{cmp}| + |\mathbf{H}_0|$. deg( $\mathbf{Z}_{\mathbf{H}_0}$) is low enough by assumption. We assumed $w_j^{APR} \in RS[F;\mathbf{L};\ _j]$, thus

$$\deg\ q_j\ \ w_j^{APR}(\mathbf{x});\mathbf{x} < 2^{k+t}$$

thus the right hand of Equation (26) is of degree at most

$$\deg < 2^{k+t+d} + 2 < 2^{k+t+d} + |\mathbf{H}_0| = {}_{cmp}|\mathbf{L}_{cmp}| + |\mathbf{H}_0|$$

concluding the proof. □

### 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. □

### C.2.4 Instance properties (Item 4h)

**Arithmetic complexity of :** 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 $\mathbf{V}$, the polynomial $\mathbf{Z}_{\mathbf{V}}$ vanishing over $\mathbf{V}$ is constructible in time poly(dim $\mathbf{V}$), and has exactly dim $\mathbf{V}$ nonzero coefficients, although its degree is $|\mathbf{V}|$. Moreover, for every constant $\mathbf{c} \in F$, the vanishing polynomial over $\mathbf{V} + \mathbf{c}$ is exactly $\mathbf{Z}_{\mathbf{V}+\mathbf{c}} = \mathbf{Z}_{\mathbf{V}} + \mathbf{Z}_{\mathbf{V}}(\mathbf{c})$. Given all this, we conclude that the values of $\mathbf{Z}_{\mathbf{H}_0}(X_{loc});\mathbf{Z}_{\mathbf{H}_1}(X_{loc})$ can be all computed using a single evaluation of $\mathbf{Z}_{\mathbf{H}_0}(X_{loc})$ and addition of a single constant value to compute $\mathbf{Z}_{\mathbf{H}_1}(X_{loc})$. In total, this single computation requires $3\mathbf{t}$ multiplications and additions in the field. Additional optimization is possible when representing $\mathbf{Z}_{\mathbf{H}_0}$ by a binary matrix, eliminating completely multiplications over the field from the evaluation process.

**$x_{APR}$ is $2^{(2+R+d)}(1 - 2^{k})$-independent:**

*Proof.* The witness reduction samples each $\mathbf{Q}_j$ from a $2^{k+t} - 2^{t}$ independent space, so by construction we notice $w_j^{APR}$ is sampled from a $2^{k+t} - 2^{t}$ independent space as well. The claim follows by noticing $2^{(2+R+d)}(1 - 2^{k})|\mathbf{L}| = 2^{k+t} - 2^{t}$. □

**$x_{APR}$ has $1 - 2^{R}(1 + 2^{d})$-distance:**

*Proof.* $_{cmp} - 2^{R} < 2^{R}(1 + 2^{d})$, and $_{max} - 2^{(2+R+d)} < 2^{R}(1 + 2^{d})$. It is sufficient to show the existence of the code $\mathbf{C}$ from Item 2. We define $\mathbf{C}$ to be the set of mappings $\mathbf{w}: \mathbf{L} \to F$ which are rational functions of the form $\mathbf{w}(\mathbf{x}) = \frac{\mathbf{p}(\mathbf{x})}{\mathbf{q}(\mathbf{x})}$ where $\mathbf{p}(\mathbf{x})$ is a polynomial of degree less than $2^{t+d}$ and $\mathbf{q}(\mathbf{x}) \triangleq \prod_{\mathbf{H}nf\{0;1\}^g}(\mathbf{x} - )$. $\mathbf{w}$ is well defined over $\mathbf{L}$, as none of the zeros of $\mathbf{q}(\mathbf{x})$ are in $\mathbf{L}$, and the distance of the

code $\mathbf{C}$ is the same as the distance of $\mathrm{RS}\left[\mathrm{F}; \mathbf{L}; \frac{2^{t+d}}{|\mathbf{L}|}\right] = \mathrm{RS}\left[\mathrm{F}; \mathbf{L}; 2^{(2+k+R)}\right]$, which is $1 - 2^{-(2+k+R)}$, and in particular grater than $1 - 2^{-R} \cdot 1 + 2^{-d}$. We conclude the proof by showing $\mathrm{RS}[\mathrm{F}; \mathbf{L}; \ell_{max}] \subseteq \mathbf{C}$ by noticing it is exactly the sub-code where $\mathbf{p}(\mathbf{x})$ is restricted to be of the form $\mathbf{p}(\mathbf{x}) = \mathbf{q}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x})$, where $\mathbf{v}$ is a polynomial of degree less than $2^{k+t}$. □

**$x_{APR}$ is 1-overlapping:**

*Proof.* Define the following 3 disjoint affine subspaces of $\mathbf{L}$:

- $\mathbf{S_x} \triangleq \mathbf{L}_{cmp} = \mathrm{span}\{g^{\mathbf{i}} \mid 0 \le \mathbf{i} < k + R + t + d\} + g^{1+k+R+t+d}$
- $\mathbf{S_{gx}} \triangleq \mathrm{span}\{g^{\mathbf{i}+1} \mid 0 \le \mathbf{i} < k + R + t + d\} + g^{2+k+R+t+d}$
- $\mathbf{S_{g\bar{x}}} \triangleq \mathrm{span}\{g^{\mathbf{i}+1} \mid 0 \le \mathbf{i} < k + R + t + d\} + 1 + g^{2+k+R+t+d}$

The claims follows by the observation that for any $\mathbf{N}(\mathbf{x}) \in \{\mathbf{x}; g\mathbf{x}; g\bar{\mathbf{x}}\}$, and for any $\mathbf{z} \in \mathbf{L}_{cmp}$, $\mathbf{N}(\mathbf{z}) \in \mathbf{S}_{\mathbf{N}(\mathbf{x})}$. □

## C.3 The APR reduction for general computation

### C.3.1 Common definitions

We expand the definition in Appendix C.1.1 $(\ ,\mathbf{H})$. For the sake of completeness we provide those definitions again, after providing the definitions introduced first in this section. The definitions of $\mathbf{Z}_{B;\ }, E_{B;}$ 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:

- $\in \mathrm{F}_2[g]$ is a primitive polynomial of degree $\lceil \log(t+1) \rceil$, notice $\ $ is a member of F
- $\mathbf{W} \subseteq \mathrm{F}$ is the space spanned by $\{g^{t+\mathbf{k}} \mid 0 \le \mathbf{k} < \lceil \log(t+1) \rceil\}$
  - $\mathbf{W}_0 \subseteq \mathbf{W}$ is the subspace spanned by $\{g^{t+\mathbf{k}} \mid 0 \le \mathbf{k} < \lceil \log(t+1) \rceil - 1\}$
  - $\mathbf{W}_1 \subseteq \mathbf{W}$ is the affine space $\mathbf{W}_1 \triangleq \mathbf{W}_0 + g^{t+\lceil \log(t+1) \rceil - 1}$
  - we denote be s the element $s \triangleq g^{t+1} \in \mathbf{W}$
- Given a neighbor $(\ ;\mathbf{N}) \in N$ we denote by $(\ ;\hat{\mathbf{N}})$ the expression $(\ ;\hat{\mathbf{N}}) \triangleq X_{(\ ;\mathbf{N})}\left[\mathbf{Z}_{B;\ }(\mathbf{N}(X_{loc})) + E_{B;\ }(\mathbf{N}(X_{loc}))\right]$. We extend this notation further more, and denote by $(\ _1;\ _2;\dots;\ _k;\mathbf{N})$ the sequence $(\ _1;\hat{\mathbf{N}});(\ _2;\hat{\mathbf{N}});\dots;(\ _k;\hat{\mathbf{N}})$.

Definition identical or similar to Appendix C.1.1:

- $\in \mathrm{F}_2[g]$ is a primitive polynomial of degree t, notice $\ $ is a member of F
- $\mathbf{H} \subseteq \mathrm{F}$ is the space spanned by $\{g^{\mathbf{k}} \mid 0 \le \mathbf{k} < t\}$
  - $\mathbf{H}_0 \subseteq \mathbf{H}$ is the subspace spanned by $\{g^{\mathbf{k}} \mid 0 \le \mathbf{k} < t - 1\}$

– $\mathbf{H}_1$ $\mathbf{H}$ is the affine space $\mathbf{H}_1 \stackrel{4}{=} \mathbf{H}_0 + g^{t\ 1}$

- For every $\ 2\ T$ we define $\mathbf{Z}_{B;}\ ;E_{B;}\ 2\ F[\mathbf{x}]$ by:
  - if $\ = (\mathbf{j};0)$ for $\mathbf{j}\ 2\ [w]$ then $\mathbf{Z}_{B;}\ (\mathbf{x}) \stackrel{4}{=} \prod_{(\mathbf{i};\mathbf{j};\ )2B} \mathbf{x}\quad g^{\mathbf{i}}\%\quad s$ and $E_{B;}$ is the polynomial of minimal degree such that for every $(\mathbf{i};\mathbf{j};\ )\ 2\ B, E_{B;}\quad g^{\mathbf{i}}\%\ + s\ =$
  - otherwise $\mathbf{Z}_{B;}\ = 1$ and $E_{B;}\ = 0$

### C.3.2 Instance reduction

We now describe $V^{BPAIR\ !\ APR}$:

1. $T \stackrel{4}{=} ([w] [ fctrlg)\quad f0;1g$
2. $N \stackrel{4}{=} N_P [ N_{routing}$, with $N_P ; N_{routing}$ defined as follows:

   **Definition of** $N_P$**:** $N_P \stackrel{4}{=}\ ;n^{id}\ ;\ ;n_0^{cyc}\ ;\ ;n_1^{cyc}\ j\ \ 2\ T$ where (i) $n^{id}(\mathbf{x}) \stackrel{4}{=} \mathbf{x}$, (ii) $n_{\mathbf{b}}^{cyc}(\mathbf{x}) \stackrel{4}{=} g\ \mathbf{x}\quad g^{t+1}\ +\mathbf{b}\ + g^{t+1}$ for $\mathbf{b}\ 2\ f0;1g$.

   **Definition of** $N_{routing}$**:** $N_{routing} \stackrel{4}{=} \{\ ;n_{\mathbf{r};\mathbf{c}}^{rout;\mathbf{b}}\ j\ \ 2\ T\ ;\mathbf{b};\mathbf{r};\mathbf{c}\ 2\ f0;1g\}$ where $n_{\mathbf{r};\mathbf{c}}^{rout;\mathbf{b}}(\mathbf{x}) \stackrel{4}{=} g\ \mathbf{x} + \mathbf{r}\ (g^t + 1) + \mathbf{c}\,(g^t\ ) + \mathbf{b}$
3. $\ $ is defined as $\ \stackrel{4}{=}\ _T [\ _P [\ _{routing}$ where $\ _T;\ _P\ ;\ _{routing}$ defined as follows:
   - $_T$ **definition:** Define for every $\mathbf{P}\ 2\ P_T$ define

     $$\mathbf{P} \stackrel{4}{=} \begin{Bmatrix} \frac{(X_{loc}\ \ s)(X_{loc}\ \ s\ \ 1)}{\mathbf{Z}_{H_0+s}(X_{loc})}\ \mathbf{P}\quad (1;0);\ldots;\widehat{}(w;0);n^{id}\ ;\ (1;0);\ldots;\widehat{}(w;0);n_0^{cyc}\quad ; \\ \vdots \\ \frac{1}{\mathbf{Z}_{H_1+s}(X_{loc})}\ \mathbf{P}\quad (1;0);\ldots;\widehat{}(w;0);n^{id}\ ;\ (1;0);\ldots;\widehat{}(w;0);n_1^{cyc}\quad ; \end{Bmatrix}$$

     Define $\ _T \stackrel{4}{=} \bigcup_{\mathbf{P}\,2\,P_T}\ \mathbf{P}$.
   - $_P$ **definition:** Define for every $\mathbf{P}\ 2\ P$ define

     $$\mathbf{P} \stackrel{4}{=} \frac{(X_{loc}\ \ s)\,(X_{loc}\ \ s\ \ 1)}{\mathbf{Z}_{\mathbf{H}+s}(X_{loc})}\ \mathbf{P}\quad (1;0);\ldots;\widehat{}(w;0);n^{id}\ ;\ (1;1);\ldots;\widehat{}(w;1);n^{id}$$

     Define $\ _T \stackrel{4}{=} \bigcup_{\mathbf{P}\,2\,P_T}\ \mathbf{P}$.

4. $_{\text{routing}}$ **definition:** We define the following polynomials used by out construction:

$$\mathbf{q}_{\text{eql}}(\mathbf{x};\mathbf{y}) \quad \stackrel{4}{=} \mathbf{x} \quad \mathbf{y} \tag{29}$$

$$\mathbf{q}_{\text{mv}}(\mathbf{x};\mathbf{y};\mathbf{z}) \quad \stackrel{4}{=} \mathbf{q}_{\text{eql}}(\mathbf{x};\mathbf{y})\mathbf{q}_{\text{eql}}(\mathbf{x};\mathbf{z}) \tag{30}$$

$$\mathbf{q}_{\text{cp}}(\mathbf{x};\mathbf{x}^{\emptyset};\mathbf{y};\mathbf{y}^{\emptyset};\mathbf{z};\mathbf{z}^{\emptyset}) \quad \stackrel{4}{=} \mathbf{q}_{\text{eql}}(\mathbf{x}^{\emptyset};\mathbf{z}^{\emptyset})\mathbf{q}_{\text{eql}}(\mathbf{x};\mathbf{y}) + \mathbf{q}_{\text{eql}}(\mathbf{x}^{\emptyset};\mathbf{y}^{\emptyset})\mathbf{q}_{\text{eql}}(\mathbf{x};\mathbf{z}) \tag{31}$$

$$\mathbf{S}_{0;0}(\mathbf{x}) \quad \stackrel{4}{=} \frac{\mathbf{Z}_{\mathbf{H}_0+\text{span}\{g^t\}}(\mathbf{x})}{\mathbf{Z}_{\mathbf{H}_0+\mathbf{W}_0}(\mathbf{x})} \tag{32}$$

$$\mathbf{S}_{1;0}(\mathbf{x}) \quad \stackrel{4}{=} \frac{\mathbf{Z}_{\mathbf{H}_1+\text{span}\{g^t\}}(\mathbf{x})}{\mathbf{Z}_{\mathbf{H}_1+\mathbf{W}_0}(\mathbf{x})} \tag{33}$$

$$\mathbf{S}_{0;1}(\mathbf{x}) \quad \stackrel{4}{=} \frac{1}{\mathbf{Z}_{\mathbf{H}_0+\mathbf{W}_1}(\mathbf{x})} \tag{34}$$

$$\mathbf{S}_{1;1}(\mathbf{x}) \quad \stackrel{4}{=} \frac{1}{\mathbf{Z}_{\mathbf{H}_1+\mathbf{W}_1}(\mathbf{x})} \tag{35}$$

and define the control constraints system by:

$$_{\text{ctrl}} \stackrel{4}{=} \begin{Bmatrix} \frac{1}{X_{\text{loc}}\ s}\mathbf{q}_{\text{eql}} \quad X_{((\text{ctrl};0);n^{\text{id}})};X_{((\text{ctrl};1);n^{\text{id}})} \\ \frac{1}{\mathbf{Z}_{H+s(X_{\text{loc}})}}\mathbf{q}_{\text{eql}} \quad X_{((\text{ctrl};0);n^{\text{id}})};X_{\text{loc}} \quad s \\ \frac{1}{\mathbf{Z}_{H+g^t(X_{\text{loc}})}}\mathbf{q}_{\text{eql}} \quad X_{((\text{ctrl};0);n^{\text{id}})};X_{((\text{ctrl};1);n^{\text{id}})} \end{Bmatrix} \tag{36}$$

$$\left[ \quad \mathbf{S}_{\mathbf{r};\mathbf{c}}(X_{\text{loc}}) \quad \mathbf{q}_{\text{mv}} \quad X_{((\text{ctrl};\mathbf{l});n^{\text{id}})};X_{((\text{ctrl};\mathbf{l});n^{\text{rout};0}_{r;c})};X_{((\text{ctrl};\mathbf{l});n^{\text{rout};1}_{r;c})} \quad j\ \mathbf{l};\mathbf{r};\mathbf{c} \in \{0;1\} \right] \tag{37}$$

The network flow copy constraints defined by:

$$_{\text{cp}} \stackrel{4}{=} \left\{ \mathbf{S}_{\mathbf{r};\mathbf{c}}(X_{\text{loc}})\,\mathbf{q}_{\text{cp}} \begin{pmatrix} (\mathbf{i};\mathbf{l})\hat{};n^{\text{id}} \\ (\mathbf{i};\mathbf{l})\hat{};n^{\text{rout};0}_{\mathbf{r};\mathbf{c}} \\ (\mathbf{i};\mathbf{l})\hat{};n^{\text{rout};1}_{\mathbf{r};\mathbf{c}} \end{pmatrix} ; \begin{matrix} X_{((\text{ctrl};\mathbf{l});n^{\text{id}})} \\ X_{((\text{ctrl};\mathbf{l});n^{\text{rout};0}_{r;c})} \\ X_{((\text{ctrl};\mathbf{l});n^{\text{rout};1}_{r;c})} \end{matrix} \quad j\ \mathbf{i} \in [w];\mathbf{l};\mathbf{r};\mathbf{c} \in \{0;1\} \right\} \tag{38}$$

We define $_{\text{routing}} \stackrel{4}{=} {}_{\text{ctrl}} [ {}_{\text{cp}}$.

5. Define $\mathbf{L}$ to be the affine space

$$\mathbf{L} \stackrel{4}{=} \text{span}\left( g^{1+k+R+t+\text{dlog}(t+1)e+d}\ (1+g) \quad [ \quad + g^{1+k+R+t+\text{dlog}(t+1)e+d} \right)_{g^{\mathbf{i}}\ j\ 0 \quad \mathbf{i} < 1+k+R+t+\text{dlog}(t+1)e+d}$$

6. Define $\mathbf{L}_{\text{cmp}}$ to be the affine space

$$\mathbf{L}_{\text{cmp}} \stackrel{4}{=} \text{span}\left( g^{\mathbf{i}}\ j\ 0 \quad \mathbf{i} < k+R+t+\text{dlog}(t+1)e+d \quad + g^{1+k+R+t+\text{dlog}(t+1)e+d} \right)$$

7. For every $1 \quad \mathbf{j} \quad w$ define $_{(\mathbf{j};0)}$ to be $_{(\mathbf{j};0)} \stackrel{4}{=} \frac{2^{k+t+\text{dlog}(t+1)e} \quad \deg(\mathbf{Z}_{B;j})}{j\mathbf{L}j}$; for any other $\quad 2\ T$ we define $\quad$ to be $\frac{2^{k+t+\text{dlog}(t+1)e}}{j\mathbf{L}j} = 2^{\ (2+R+d)}$

8. Define $_{\text{cmp}} \stackrel{4}{=} \frac{1+2^{k+t+\text{dlog}(t+1)e+d}}{j\mathbf{L}_{\text{cmp}}j}$

### C.3.3 Witness reduction

**General overview** Let $w = (\hat{w}; \ )$ be the witness. We construct all functions in

$$w_{APR} = w_{(1;0)}; w_{(1;1)}; \ldots; w_{(w;0)}; w_{(w;1)}; w_{(ctrl;0)}; w_{(ctrl;1)}$$

as follows:

1. For every $\in T$ we define a mapping $\mathbf{Q} : \mathbf{H} + \mathbf{W} \to F$
2. For every $\in T$ we define a mapping $\hat{\mathbf{Q}} : F \to F$ as the low degree extension of $\mathbf{Q}$
3. For every $\in T$ we draw a random polynomial $\mathbf{R} : F \to F$ of degree at most $2^{k+t+\lceil \log(t+1) \rceil}$ vanishing on $\mathbf{H} + \mathbf{W}$ and define $\mathbf{Q} \stackrel{4}{=} \mathbf{Q} + \mathbf{R}$
4. the function $w : \mathbf{L} \to F$ is the evaluation of $\frac{\mathbf{Q} \ E_B}{\mathbf{Z}_B}$ over $\mathbf{L}$

Given the above description, it is sufficient to describe the construction of $\mathbf{Q}$ for every $\in T$.

**Construction of $\mathbf{Q}_{(ctrl;0)}; \mathbf{Q}_{(ctrl;1)}$** Given $: [T] \to [T]$ we define the permutation $^{0}: F[g] \to F[g]$ by:

$$^{0}(0) = 0 \tag{39}$$

$$80 \quad \mathbf{i} < T : \ ^{0} g^{i}\% \ = g^{(i)}\% \tag{40}$$

$\mathbf{Q}_{(ctrl;0)}; \mathbf{Q}_{(ctrl;1)}$ are an affine embedding of back-to-back De Bruijn routing (Appendix G.2) of $^{0}$ where $\mathbf{Q}_{(ctrl;0)}(\mathbf{x} + s) = \mathbf{x}$ for every $\mathbf{x} \in \mathbf{H}$.

**Construction of $\mathbf{Q}_{(i;b)}$ mappings** Any other pair $\mathbf{Q}_{(i;0)}; \mathbf{Q}_{(i;1)}$ is intuitively an embedding of the same permutation $^{0}$ as well, and uses exactly the same routing as $\mathbf{Q}_{(ctrl;0)}; \mathbf{Q}_{(ctrl;1)}$. The sole difference is that the $\mathbf{Q}_{(i;0)}$ over $\mathbf{H} + s$ is not required to contain only distinct values, but instead satisfies $\mathbf{Q}_{(i;0)} \ g^{j}\% + s = \mathbf{w}_i[j]$.

**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 $l \ w$ 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.12 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 $\in T$, $w^{APR} \in RS[F; \mathbf{L}; \ ]$, and (ii) $8 \ \in \ ; \ _N \ \hat{w}^{APR} \in RS[F; \mathbf{L}_{cmp}; \ _{cmp}]$.

**Assignment code membership:** For every $\in T$, $\deg \mathbf{Q} \quad E_{B;} < 2^{k+t+\lceil \log(t+1) \rceil e}$, and by definition of satisfaction of BPAIR, we have $\mathbf{Z}_{B;} \mid \mathbf{Q} \quad E_{B;}$, thus $\frac{\mathbf{Q} \quad E_{B;}}{\mathbf{Z}_{B;}}$ is a polynomial of degree less then $2^{k+t+\lceil \log(t+1) \rceil e} \quad \deg \mathbf{Z}_{B;}$, showing $w^{APR} \in RS[F; \mathbf{L}; \ ]$ as required.

**Constraint code membership:** We notice first that for any $\in T$ and any neighbor $(\ ; \mathbf{N}) \in N, (\ ; \mathbf{N}) = \mathbf{Q} \quad \mathbf{N}$ is of degree less then $2^{k+t+\lceil \log(t+1) \rceil e}$. It is sufficient to show that every rational function $\in$ is a polynomial of degree at most $2^{k+t+\lceil \log(t+1) \rceil e+d}$. We notice that $= \frac{\mathbf{p}}{\mathbf{Z}_S}$ for some multivariate polynomial $\mathbf{p}$, and some set $\mathbf{S}$. For simplicity we denote by $\mathbf{g} \in F[\mathbf{x}]$ the low degree extension of $\mathbf{p}_N \ w^{APR}$. It is sufficient to show $\mathbf{g}$ is (i) of degree less then $2^{k+t+\lceil \log(t+1) \rceil e+d} + |\mathbf{S}|$, and (ii) vanishes over $\mathbf{S}$, thus in particular dividable by $\mathbf{Z}_\mathbf{S}$.

**The set** $_{P_T}$**:** for every $\mathbf{P} \in P_T$ denote by

$$\mathbf{P}_0(\mathbf{x}) \stackrel{4}{=} \mathbf{P} \quad \mathbf{Q}_{(1;0)} \quad n^{id}(\mathbf{x}); \ldots; \mathbf{Q}_{(w;0)} \quad n^{id}(\mathbf{x}); \mathbf{Q}_{(1;0)} \quad n_0^{cyc}(\mathbf{x}); \ldots; \mathbf{Q}_{(w;0)} \quad n_0^{cyc}(\mathbf{x}) \tag{41}$$

$$\mathbf{P}_1(\mathbf{x}) \stackrel{4}{=} \mathbf{P} \quad \mathbf{Q}_{(1;0)} \quad n^{id}(\mathbf{x}); \ldots; \mathbf{Q}_{(w;0)} \quad n^{id}(\mathbf{x}); \mathbf{Q}_{(1;0)} \quad n_1^{cyc}(\mathbf{x}); \ldots; \mathbf{Q}_{(w;0)} \quad n_1^{cyc}(\mathbf{x}) \tag{42}$$

$$\mathbf{P}_0'(\mathbf{x}) \stackrel{4}{=} \frac{(\mathbf{x} \quad s)(\mathbf{x} \quad s \quad 1)}{\mathbf{Z}_{\mathbf{H}_0+s}(\mathbf{x})} \quad \mathbf{P}_0(\mathbf{x}) \tag{43}$$

$$\mathbf{P}_1'(\mathbf{x}) \stackrel{4}{=} \frac{1}{\mathbf{Z}_{\mathbf{H}_1+s}(\mathbf{x})} \quad \mathbf{P}_1(\mathbf{x}) \tag{44}$$

We notice $\deg \mathbf{P}_0; \deg \mathbf{P}_1 < 2^{k+t+\lceil \log(t+1) \rceil e+d}$, thus it is sufficient to show that whenever the denominator vanishes, the enumerator vanishes as well. Assume $\mathbf{Z}_{\mathbf{H}_0+s}(\mathbf{x})$ or $\mathbf{Z}_{\mathbf{H}_1+s}(\mathbf{x})$ is zero, then $\mathbf{x} = \mathbf{y} + s$ for some $\mathbf{y} \in \mathbf{H}$. If $\mathbf{y} \in \{0; 1\}$ then $\mathbf{x} \in \mathbf{H}_0 + s$, thus only $\mathbf{Z}_{\mathbf{H}_0+s}(\mathbf{x}) = 0$, and $(\mathbf{x} \quad s)(\mathbf{x} \quad s \quad 1) = 0$ as well. Otherwise, there is some $\mathbf{j} \in [T \quad 1]$ such that $\mathbf{y} = g^{\mathbf{j}}\%$ , and by the witness construction (Appendix C.3.3) $\mathbf{Q}_{(i;0)} \quad g^{\mathbf{j}}\% \quad + s = \mathbf{w}_i[\mathbf{j}]$. In the case $g^{\mathbf{j}}\% \quad + s \in \mathbf{H}_0 + s$ we notice $n_0^{cyc} \quad g^{\mathbf{j}}\% \quad + s = g^{\mathbf{j}+1}\% \quad + s$. Similarly, in the case $g^{\mathbf{j}}\% \quad + s \in \mathbf{H}_1 + s$ it holds $n_1^{cyc} \quad g^{\mathbf{j}}\% \quad + s = g^{\mathbf{j}+1}\% \quad + s$. Finally, the claim follows by the assumption the BPAIR witness satisfies the instance, and in particular $\mathbf{P}(\mathbf{w}_1[\mathbf{j}]; \ldots; \mathbf{w}_w[\mathbf{j}]; \mathbf{w}_1[\mathbf{j}+1]; \ldots; \mathbf{w}_w[\mathbf{j}+1]) = 0$ for all $\mathbf{j} \in [T \quad 1]$.

**The set** $_P$ **:** Let $\mathbf{P} \in P$ be a polynomial, denote by

$$\mathbf{P}'(\mathbf{x}) \stackrel{4}{=} \mathbf{P} \quad \mathbf{Q}_{(1;0)}(\mathbf{x}); \ldots; \mathbf{Q}_{(w;0)}(\mathbf{x}); \mathbf{Q}_{(1;1)}(\mathbf{x}); \ldots; \mathbf{Q}_{(w;1)}(\mathbf{x}) \tag{45}$$

$$\mathbf{P}''(\mathbf{x}) \stackrel{4}{=} \frac{(\mathbf{x} \quad s)(\mathbf{x} \quad s \quad 1)}{\mathbf{Z}_{\mathbf{H}+s}(\mathbf{x})} \quad \mathbf{P}_0(\mathbf{x}) \tag{46}$$

We notice $\deg \mathbf{P}' < 2^{k+t+\lceil \log(t+1) \rceil e+d}$, thus it is sufficient to show that whenever the denominator vanishes, the enumerator vanishes as well. Let $\mathbf{x}$ be a member of $\mathbf{H} + s$. If $\mathbf{x} \in \{s; 1 + s\}$ then $(\mathbf{x} \quad s)(\mathbf{x} \quad s \quad 1) = 0$, otherwise there is some $\mathbf{j} \in [T \quad 1]$ such that $\mathbf{x} = g^{\mathbf{j}}\% \quad + s$, thus by construction $\mathbf{P}'(\mathbf{x}) = \mathbf{P}(\mathbf{w}_1[\mathbf{j}]; \ldots; \mathbf{w}_w[\mathbf{j}]; \mathbf{w}_1[\ (\mathbf{j})]; \ldots; \mathbf{w}_w[\ (\mathbf{j})])$, and in particular vanishes by definition of satisfaction in BPAIR.

**The set $_{\text{ctrl}}$:**

- For $(\mathbf{x}) = \frac{1}{\mathbf{Z}_{H+s}(\mathbf{x})} \mathbf{q}_{\text{eql}}\ \mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}); \mathbf{x}\ \ s$ we have $\deg\ \mathbf{q}_{\text{eql}}\ \mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}); \mathbf{x}\ \ s < 2^{k+t+\text{dlog}(t+1)\text{e}}$ and for every $\mathbf{x} \in \mathbf{H} + s$, it holds by construction (Appendix C.3) $\mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}) = \mathbf{x}\ \ s$.
- For $(\mathbf{x}) = \frac{1}{\mathbf{x}\ s} \mathbf{q}_{\text{eql}}\ \mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}); \mathbf{Q}_{(\text{ctrl};1)}(\mathbf{x})$ we have $\deg\ \mathbf{q}_{\text{eql}}\ \mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}); \mathbf{Q}_{(\text{ctrl};1)}(\mathbf{x}) < 2^{k+t+\text{dlog}(t+1)\text{e}}$, and $\mathbf{Q}_{(\text{ctrl};0)}(s) = \mathbf{Q}_{(\text{ctrl};1)}(s)$ because $^{0}(0) = 0$.
- For $(\mathbf{x}) = \frac{1}{\mathbf{Z}_{H+g^t}(\mathbf{x})} \mathbf{q}_{\text{eql}}\ \mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}); \mathbf{Q}_{(\text{ctrl};1)}(\mathbf{x})$ we have $\deg\ \mathbf{q}_{\text{eql}}\ \mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}); \mathbf{Q}_{(\text{ctrl};1)}(\mathbf{x}) < 2^{k+t+\text{dlog}(t+1)\text{e}}$, and for every $\mathbf{x} \in \mathbf{H} + g^t$ it holds $\mathbf{Q}_{(\text{ctrl};0)}(\mathbf{x}) = \mathbf{Q}_{(\text{ctrl};1)}(\mathbf{x})$, by the property of the back-to-back De Bruijn routing (Appendix G.2).
- For of the form $\mathbf{S}_{\mathbf{r};\mathbf{c}}(\mathbf{x})\ \mathbf{q}_{\text{mv}}\ \mathbf{Q}_{(\text{ctrl};\mathbf{l})}(\mathbf{x}); \mathbf{Q}_{(\text{ctrl};\mathbf{l})}\ n^{\text{rout};0}_{\mathbf{r};\mathbf{c}}(\mathbf{x}); \mathbf{Q}_{(\text{ctrl};\mathbf{l})}\ n^{\text{rout};1}_{\mathbf{r};\mathbf{c}}(\mathbf{x})$ : Low degree follows by the properties of the back-to-back De Bruijn routing (Appendix G.2).

**The set $_{\text{cp}}$:** The low degree of every $\ 2\ _{\text{cp}}$ follows by construction, as all networks $\mathbf{Q}_{(\mathbf{j};0)}; \mathbf{Q}_{(\mathbf{j};1)}$ are routed exactly the same way as the embedded back-to-back De Bruijn routing in $\mathbf{Q}_{(\text{ctrl};1)}; \mathbf{Q}_{(\text{ctrl};1)}$. □

### C.4.2 Proof of soundness (Item 2)

*Proof.* Assume $V^{\text{BPAIR}\ !\ \text{APR}}(x; k; R) \in \text{APR}$ for $x = (F; T; w; P; B)$, and let $\hat{w}^{\text{APR}}$ be a witness for it. We show $x \in \text{BPAIR}$ by constructing a witness $w^{\text{BPAIR}} = (\hat{w}; \ )$ for it. We define:

- for every $\mathbf{j} \in [w]$ the mapping $\mathbf{w}^{\text{BPAIR}}_{\mathbf{j}} : [T] \ !\ F$ by

$$\mathbf{w}^{\text{BPAIR}}_{\mathbf{j}}[\mathbf{i}] \stackrel{4}{=} w^{\text{APR}}_{(\mathbf{j};0)}\ \ g^{\mathbf{i}}\%\ \ + s\ \ \mathbf{Z}_{B;(\mathbf{j};0)}\ \ g^{\mathbf{i}}\%\ \ + s\ + E_{B;\mathbf{j}}\ \ g^{\mathbf{i}}\%\ \ + s \tag{47}$$

  denoting by $w^{\text{APR}}_{(\mathbf{j};0)}$ the low degree extension of the actual witness.

- we define

$$(\mathbf{i}) = \mathbf{j}\ \ (\ )\ \ w^{\text{APR}}_{(\text{ctrl};1)}\ \ g^{\mathbf{i}}\%\ \ + s\ =\ g^{\mathbf{j}}\%\ \ + s \tag{48}$$

  for $\mathbf{i}; \mathbf{j} \in [T]$. Otherwise we say $(\mathbf{i}) = 1$.

We show $w^{\text{BPAIR}}$ satisfies x. Let $\mathbf{Q}(\mathbf{x}; \mathbf{y}) \stackrel{4}{=} \mathbf{x}\ \ \mathbf{Z}_{B;}(\mathbf{y}) + E_{B;}(\mathbf{y})$.

**Boundary constraints:** $w^{\text{BPAIR}}$ satisfies the boundary constraints B by construction, as having $(\mathbf{i}; \mathbf{j}; \ ) \in B$ implies $\mathbf{Z}_{B;(\mathbf{j};0)}\ \ g^{\mathbf{i}}\%\ \ + s\ = 0$ and $E_{B;(\mathbf{j};0)}\ \ g^{\mathbf{i}}\%\ \ + s\ =\ $, thus $\mathbf{w}^{\text{BPAIR}}_{\mathbf{j}}[\mathbf{i}] =\ $ as required.

**General technique for rest of soundness proof:** In what follows we show that in any case where is not a permutation, or at least one of $P_T, P$ is not satisfied by $w^{\mathbf{BPAIR}}$ then there exists $\ 2\ $ such that $_N\ \hat{w}^{\text{APR}} \ncong \text{RS}[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$. The general method we use is assuming by contradiction $_N\ \hat{w}^{\text{APR}} \in \text{RS}[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$, and denoting by $\ : F\ !\ F$ its low-degree extension. We notice the evaluation of every such is over $\mathbf{L}_{\text{cmp}}$ representable as some rational function, implied by definition of , $(\mathbf{x}) = \frac{\mathbf{p}(\mathbf{x})}{\mathbf{q}(\mathbf{x})}$ where (i) $\deg \mathbf{p} < 2^{k+t+\text{dlog}(t+1)\text{e}+d}$, (ii) $\deg \mathbf{q} < 2^{t+\text{dlog}(t+1)\text{e}}$, and (iii) there exists some $\mathbf{x}_0$ such that $\mathbf{q}(\mathbf{x}_0) = 0$ while $\mathbf{p}(\mathbf{x}_0) \neq 0$. In particular it must hold $\mathbf{q}(\mathbf{x})\ \ (\mathbf{x}) = \mathbf{p}(\mathbf{x})$ for all $\mathbf{x} \in \mathbf{L}_{\text{cmp}}$, but both equation sides are polynomial of degree less than $2^{k+t+\text{dlog}(t+1)\text{e}+d} + 2^{t+\text{dlog}(t+1)\text{e}} < |\mathbf{L}_{\text{cmp}}|$ that do not agree on $\mathbf{x}_0$, thus they can not agree on any set of size $|\mathbf{L}_{\text{cmp}}|$, contradicting their agreement over $\mathbf{L}_{\text{cmp}}$.

**Satisfaction of $P_T$:** Assume by contradiction there is $\mathbf{P} \in P_T$ and $\mathbf{j} \in [T-1]$ such that $\mathbf{P}(\mathbf{w}[\mathbf{j}];\mathbf{w}[\mathbf{j}+1]) \neq 0$. In case $g^{\mathbf{j}}\% \in \mathbf{H}_0$ the contradiction is achieved using $\mathbf{P}_0^{\emptyset} \in$ $_{P_T}$ (Equation (43)), otherwise $g^{\mathbf{j}}\% \in \mathbf{H}_1$ and the contradiction is achieved using $\mathbf{P}_1^{\emptyset} \in$ $_{P_T}$ (Equation (44)).

**is a permutation:** By Theorem G.5 it is sufficient to show $\mathbf{Q}_{(\mathrm{ctrl};0)};\mathbf{Q}_{(\mathrm{ctrl};0)}$ is an affine embedding of a back-to-back De Bruijn of degree t inducing a permutation $^{\emptyset}$ such that (i) the domain of $^{\emptyset}$ is $\mathbf{H}$, (ii) $^{\emptyset}(0) = 0$. By Definition G.4 it is sufficient to show:

1. For all $\mathbf{x} \in \mathbf{L}$ it holds $\mathbf{Q}_{(\mathrm{ctrl};0)}(\mathbf{x}+s) = \mathbf{x}$
2. For all $\mathbf{x} \in \mathbf{L}$ it holds $\mathbf{Q}_{(\mathrm{ctrl};0)}(\mathbf{x}+g^t) = \mathbf{Q}_{(\mathrm{ctrl};1)}(\mathbf{x}+g^t)$
3. For every $\mathbf{r};\mathbf{c};\mathbf{l} \in \{0;1g$, and every $\mathbf{x} \in \mathbf{H_r}+(\mathbf{W_c} \cap \{0;g^t g)$, $\mathbf{Q}_{(\mathrm{ctrl};\mathbf{l})}(\mathbf{x})$ equals to $\mathbf{Q}_{(\mathrm{ctrl};\mathbf{l})} \; n_{\mathbf{r};\mathbf{c}}^{\mathrm{rout};\mathbf{b}}(\mathbf{x})$ for some $\mathbf{b} \in \{0;1g$
4. $\mathbf{Q}_{(\mathrm{ctrl};0)}(s) = \mathbf{Q}_{(\mathrm{ctrl};1)}(s)$

We now now, using the general technique (Appendix C.4.2), that if any of the stated above constraints does not hold there is $\in$ such that $_N \; \mathrm{APR} \not\geq \mathrm{RS}[F;\mathbf{L}_{\mathrm{cmp}};\; _{\mathrm{cmp}}]$ by providing polynomials $\mathbf{p};\mathbf{q} \in F[\mathbf{x}]$ as required by the technique.

1. In case Item 1 does not hold, the contradiction is achieved using $\frac{\mathbf{q}_{\mathrm{eql}} \; X_{((\mathrm{ctrl};0);n^{\mathrm{id}})};X_{\mathrm{loc}} \; s}{\mathbf{Z}_{H+s}(X_{\mathrm{loc}})} \in$ $_{\mathrm{ctrl}}$
2. In case Item 2 does not hold, the contradiction is achieved using $\frac{\mathbf{q}_{\mathrm{eql}} \; X_{((\mathrm{ctrl};0);n^{\mathrm{id}})};X_{((\mathrm{ctrl};1);n^{\mathrm{id}})}}{\mathbf{Z}_{H+g^t}(X_{\mathrm{loc}})} \in$ $_{\mathrm{ctrl}}$
3. In case Item 3 does not hold, the contradiction is achieved using the corresponding polynomial of the form $\mathbf{S}_{\mathbf{r};\mathbf{c}}(X_{\mathrm{loc}}) \; \mathbf{q}_{\mathrm{mv}} \; X_{((\mathrm{ctrl};\mathbf{l});n^{\mathrm{id}})};X_{((\mathrm{ctrl};\mathbf{l});n_{r;c}^{\mathrm{rout};0})};X_{((\mathrm{ctrl};\mathbf{l});n_{r;c}^{\mathrm{rout};1})} \in$ $_{\mathrm{ctrl}}$
4. In case Item 4 does not hold, the contradiction is achieved using $\frac{\mathbf{q}_{\mathrm{eql}} \; X_{((\mathrm{ctrl};0);n^{\mathrm{id}})};X_{((\mathrm{ctrl};1);n^{\mathrm{id}})}}{X_{\mathrm{loc}} \; s} \in$ $_{\mathrm{ctrl}}$

**Satisfaction of P :** For every $\mathbf{j} \in [w]$, we notice the pair $\mathbf{Q}_{(\mathbf{j};0)};\mathbf{Q}_{(\mathbf{j};1)}$ is routed exactly the same as $\mathbf{Q}_{(\mathrm{ctrl};0)};\mathbf{Q}_{(\mathrm{ctrl};1)}$, as otherwise a contradiction is achievable using $_{\mathrm{cp}}$. Thus we conclude $\mathbf{Q}_{(\mathrm{ctrl};0)} \; g^{\mathbf{i}}\% + s = \mathbf{Q}_{(\mathrm{ctrl};1)} \; g^{(\mathbf{i})}\% + s$ for every $\mathbf{i} \in [T-1]$. Assuming there is $\mathbf{P} \in P$ and $\mathbf{i} \in [T-1]$ such that $\mathbf{P}(\mathbf{w}[\mathbf{i}];\mathbf{w}[(\mathbf{i})]) \neq 0$, then the contradiction is achieved using the corresponding polynomial in $_P$. □

### 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 strait forward, 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.

$x_{APR}$ **is 5-overlapping:**

*Proof.* Define the following 3 disjoint affine subspaces of $\mathbf{L}$:

- $\mathbf{S_x} \stackrel{4}{=} \mathbf{L}_{cmp} = \text{span}\ \{g^{\mathbf{i}}\}_{0 \le \mathbf{i} < k+R+t+d} + g^{1+k+R+t+d}$
- $\mathbf{S}_{g\mathbf{x}} \stackrel{4}{=} \text{span}\ \{g^{\mathbf{i}+1}\}_{0 \le \mathbf{i} < k+R+t+d} + g^{2+k+R+t+d}$
- $\mathbf{S}_{g\mathbf{x}+1} \stackrel{4}{=} \text{span}\ \{g^{\mathbf{i}+1}\}_{0 \le \mathbf{i} < k+R+t+d} + 1 + g^{2+k+R+t+d}$

We show there are at most 5 neighbor to each space, mapping elements of $\mathbf{L}_{cmp}$ to them. We notice that for every $\mathbf{x} \in \mathbf{L}_{cmp}$: (i) $\mathsf{n}^{\mathsf{id}}(\mathbf{x}) \in \mathbf{S_x}$, (ii) $\mathsf{n}_0^{\mathsf{cyc}}(\mathbf{x}); \mathsf{n}_{\mathbf{r};\mathbf{c}}^{\mathsf{rout};\mathbf{b}}(\mathbf{x}) \in \mathbf{S}_{g\mathbf{x}}$ whenever $\mathbf{b} + \mathbf{r} + \mathbf{c} = 0$, and (iii) $\mathsf{n}_1^{\mathsf{cyc}}(\mathbf{x}); \mathsf{n}_{\mathbf{r};\mathbf{c}}^{\mathsf{rout};\mathbf{b}}(\mathbf{x}) \in \mathbf{S}_{g\mathbf{x}+1}$ whenever $\mathbf{b} + \mathbf{r} + \mathbf{c} = 1$. 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 $(x_{APR}; w_{APR})$ as described in Appendix B.3 and Definition B.9, 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.13.

The ALI protocol is described in Appendix D.1. The main properties achieved by it were presented earlier in Theorem B.14, and we prove this Theorem in Appendix D.2. The section ends in Appendix D.3 with a justification for Conjectures B.15 and B.16 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 -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 -independent instance $x = x_{APR} = (F; T; N; ; \mathbf{L}; \mathbf{L}_{cmp}; \sim; _{cmp})$
- Prover has x and a witness $w = w_{APR} = \{w \in \mathbf{V}; \in T\}$ satisfying x, where each w was sampled uniformly from the $|\mathbf{L}|$-independent space $\mathbf{V}$ described in Definition B.10.

Protocol:

1. Prover samples uniformly and independently random
   - $\mathbf{f}_{mask} \in RS[F; \mathbf{L}; _{max}]$;
   - $\mathbf{g}_{mask} \in RS[F; \mathbf{L}_{cmp}; _{cmp}]$;

and sends to verifier the oracle $O_{\text{assignment}} \triangleq (w; \mathbf{f}_{\text{mask}}; \mathbf{g}_{\text{mask}})$

2. Verifier performs the following:
   (a) sample and send to prover a sequence $\mathbf{R}$ of $2|T| + |\quad|$ uniformly random F-elements containing:
      - $\mathbf{r}_{\ ;0}; \mathbf{r}_{\ ;1}$ for every $\quad \in T$
      - $\mathbf{r}_{\quad}$ for every $\quad \in$
   (b) define the *random constraint* $_R : F^V \to F$ by $_R(\ ) \triangleq \sum_{\quad \in} \mathbf{r}_{\quad}(\ )$
   (c) invoke the two following RS-IOPP sub-protocols, accepting if and only if both sub-protocols accept:
      i. verify proximity of the function $\mathbf{f}^{(0)} : \mathbf{L} \to F$ to the code $RS[F; \mathbf{L}; \ _{\max}]$ where

$$\forall \mathbf{x} \in \mathbf{L}; \quad \mathbf{f}^{(0)}(\mathbf{x}) \triangleq \mathbf{f}_{\text{mask}}(\mathbf{x}) + \sum_{\quad \in T} \mathbf{r}_{\ ;0} + \mathbf{r}_{\ ;1}\ \mathbf{x}^{|L|(\ _{\max}\ )}\ w(\mathbf{x}) \tag{49}$$

      ii. verify proximity of the function $\mathbf{g}^{(0)} : \mathbf{L}_{\text{cmp}} \to F$ to the code $RS[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$ where

$$\forall \mathbf{x} \in \mathbf{L}_{\text{cmp}}; \quad \mathbf{g}^{(0)}(\mathbf{x}) \triangleq \mathbf{g}_{\text{mask}}(\mathbf{x}) + \ _R(\ _{w;N}(\mathbf{x})) \tag{50}$$

## D.2 Proof of the ALI reduction Theorem B.14

Our proof follows the order of items stated in Theorem B.14. 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 $\mathbb{w}$ satisfies x according to Definition B.9. Then by definition of a -independent APR, the (random) witness w sampled by the prover in step 1 of the ALI satisfies x as well. This means that the following holds:

- $w \in RS[F; \mathbf{L}; \ ]$ for each $\quad \in T$ and $\mathbf{f}^{(0)}$ is in the linear span of w; thus $\mathbf{f}^{(0)} \in RS[F; \mathbf{L}; \ _{\max}]$.
- $_N[w] \in RS[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$ for each $\quad \in$ and $_{\mathbf{R}}$ is in the linear span of $\{\ _N[w]\}$, thus $_{\mathbf{R}} \in RS[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$ as well, concluding $\mathbf{g}^{(0)} \in RS[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$.

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* $\mathbf{S} \subseteq F$ *and* $\mathbf{d}; \mathbf{k} \in \mathbb{N}$ *satisfy* $\mathbf{d} + 2\mathbf{k} < |\mathbf{S}|$. *Given* $\mathbf{f} : \mathbf{S} \to F$ *define* $\hat{\mathbf{f}}(\mathbf{x}) \triangleq \mathbf{x}^{\mathbf{k}}\ \mathbf{f}(\mathbf{x})$. *Suppose that both* $\deg \text{interpolant}^{\mathbf{f}} < \mathbf{d} + \mathbf{k}$ *and* $\deg \text{interpolant}^{\hat{\mathbf{f}}} < \mathbf{d} + \mathbf{k}$. *Then we also have* $\deg \text{interpolant}^{\mathbf{f}} < \mathbf{d}$.

*Proof.* Let $\mathbf{P}(\mathbf{X}) = \text{interpolant}^{\mathbf{f}}$ and let $\mathbf{Q}(\mathbf{X}) \stackrel{4}{=} \mathbf{X}^{\mathbf{k}}\ \mathbf{P}(\mathbf{X})$; both $\mathbf{P}; \mathbf{Q}$ are viewed as members of $F[\mathbf{X}]$. By assumption $\deg(\mathbf{P}) < \mathbf{d} + \mathbf{k}$, so $\deg(\mathbf{Q}) < \mathbf{d} + 2\mathbf{k} < j\mathbf{S}j$. The multi-point evaluation of $\mathbf{Q}$ on domain $\mathbf{S}$ is precisely the function $\mathbf{f}^{\wedge}$. The uniqueness of the interpolant, along with the observation $\deg(\mathbf{Q}) < j\mathbf{S}j$, imply that $\text{interpolant}^{\mathbf{f}^{\wedge}} = \mathbf{Q}$. Therefore, the assumption $\deg(\text{interpolant}^{\mathbf{f}^{\wedge}}) < \mathbf{d} + \mathbf{k}$ gives $\deg(\mathbf{Q}) < \mathbf{d} + \mathbf{k}$. By construction $\deg(\mathbf{P}) = \deg(\mathbf{Q})\ \ \mathbf{k}$ 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* $\mathbf{C}\ \ F^{\mathbf{S}}$ *be an* F*-linear code of blocklength* jFj *and relative distance* . *Fix* $\mathbf{c} > 6=$ . *Suppose* $\mathbf{V}\ \ F^{\mathbf{S}}$ *satisfies*

$$\Pr_{\mathbf{v} 2 \text{span}(\mathbf{V})}\ \ {}_{H}(\mathbf{v}; \mathbf{C})\ \ \frac{1}{\mathbf{c}}\ > 1=jFj:$$

*Then there exists* $\mathbf{S}^{0}\ \ \mathbf{S}$ *of density* $(\mathbf{S}^{0}=\mathbf{S})\ \ 1\ \ \frac{2}{\mathbf{c}} > 1\ \ \frac{}{3}$ *such that* $\mathbf{V}\,j_{\mathbf{S}^{0}}\ \ \mathbf{C}j_{\mathbf{S}^{0}}$.

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* $\mathbf{C}\ \ F^{\mathbf{S}}$ *be a linear space. If* $\mathbf{f}_1; : : : ; \mathbf{f}_{\mathbf{k}}\ 2\ F^{\mathbf{S}}$ *are such that there exists* $\mathbf{f}_{\mathbf{i}}$ *that is* -*far from* $\mathbf{C}$ *in relative Hamming distance, then*

$$\Pr_{\mathbf{r}_1;:::;\mathbf{r}_k 2 F}\ \ {}_{H}\Big(\sum_{\mathbf{i}=1}^{\mathbf{k}} \mathbf{r}_{\mathbf{i}}\mathbf{f}_{\mathbf{i}}; \mathbf{C}\Big)\ \ =2\ \ 1=jFj$$

*Proof of Lemma D.2.* By Lemma D.3 we have

$$8\mathbf{v}\ 2\ \text{span}(\mathbf{V});\ \ {}_{H}(\mathbf{v}; \mathbf{C})\ \ \frac{2}{\mathbf{c}}: \tag{51}$$

Since $2=\mathbf{c} <\ \ =2$ by assumption, we conclude that the codeword of $\mathbf{C}$ that is closest to $\mathbf{v}\ 2\ \text{span}(\mathbf{V})$ is unique, denote it by $\mathbf{v}$. Define $\mathbf{S}_{\mathbf{v}} \stackrel{4}{=} f\mathbf{x}\ 2\ \mathbf{S}\ j\ \mathbf{v}(\mathbf{x})\ 6=\ \mathbf{v}(\mathbf{x})g$ and for $\mathbf{V}^{0}\ \ \text{span}(\mathbf{V})$ let $\mathbf{S}_{\mathbf{V}^{0}} = [_{\mathbf{v} 2 \mathbf{V}^{0}} \mathbf{S}_{\mathbf{v}}$. To prove Lemma D.2 it suffices to show

$$\mathbf{S}_{\text{span}(\mathbf{V})}=\mathbf{S}\ \ \frac{2}{\mathbf{c}} \tag{52}$$

by setting $\mathbf{S}^{0} = \mathbf{S}\ \text{n}\ \mathbf{S}_{\text{span}(\mathbf{V})}$.

We prove (52) by way of contradiction, namely, we show that $(\mathbf{S}_{\text{span}(\mathbf{V})}=\mathbf{S}) > \frac{2}{\mathbf{c}}$ and (51) together imply that (51) is false, so (52) holds.

Write $\mathbf{v} = \mathbf{v} + \mathbf{v}^{0}$ where $\mathbf{v}^{0}$ has relative Hamming weight $(\mathbf{S}_{\mathbf{v}^{0}}=\mathbf{S})$. Abusing notation, we identify $\mathbf{v}$ with $\mathbf{v}^{0}$ and henceforth assume the codeword closest to $\mathbf{v}$ is $\mathtt{0}$ and that $\mathbf{S}_{\mathbf{v}}$ denotes the support of $\mathbf{v}$, i.e., the set of its nonzero entries.

Since (51) implies $(\mathbf{S}_{\mathbf{v}}=\mathbf{S}) <\ \ =3$, if (52) is false then there exists some $\mathbf{V}^{0}\ \ \mathbf{V}$ such that

$$(\mathbf{S}_{\mathbf{V}^{0}}=\mathbf{S})\ 2\ \ \frac{2}{\mathbf{c}}; \frac{4}{\mathbf{c}}\ \ \ \ \frac{2}{\mathbf{c}};\ \ \frac{2}{\mathbf{c}}$$

The containment follows because $4=\mathbf{c} < 2\ =3 <\ \ 2=\mathbf{c}$. By linearity of expectation, the expected support size of a random word in $\text{span}(\mathbf{V}^{0})$ is precisely $(1\ \ 1=jFj)j\mathbf{S}_{\mathbf{V}^{0}}j$ which is strictly greater than $(1\ \ 1=j\mathbf{S}_{\mathbf{V}^{0}}j)j\mathbf{S}_{\mathbf{V}^{0}}j$ because $j\mathbf{S}_{\mathbf{V}^{0}}j < j\mathbf{S}j\ \ jFj$. Thus, it must be the case that some $\mathbf{v}\ 2\ \text{span}(\mathbf{V}^{0})$ is fully supported on $\mathbf{S}_{\mathbf{V}^{0}}$ which means that the relative support size of $\mathbf{v}$ is in $\frac{2}{\mathbf{c}}; \ \ \frac{2}{\mathbf{c}}$ ; we conclude $\mathbf{v}$ has relative distance $(\mathbf{S}_{\mathbf{V}^{0}}=\mathbf{S}) > 2=\mathbf{c}$ from $\mathbf{C}$, contradicting (51) and completing the proof. □

**Soundness analysis**

*Proof of Item 3 of Theorem B.14.* We prove the contrapositive: If neither item 3a nor item 3b of Theorem B.14 hold, which means both of the following items hold:

1. $\Pr_{H}\left[\mathbf{f}^{(0)}; \mathrm{RS}[F; \mathbf{L}; \_{max}]\right] \quad \frac{\_}{2} > 1/|F|$
2. $\Pr_{H}\left[\mathbf{g}^{(0)}; \mathrm{RS}[F; \mathbf{L}_{cmp}; \_{cmp}]\right] \quad \frac{\_}{2} > 1/|F|$

Then x 2 APR. Details follow.

We apply Lemma D.2 to Item 1 above, while setting the constant **c** in that lemma to

$$\mathbf{c} \stackrel{4}{=} \frac{2}{\_} \tag{53}$$

The assumptions of Lemma D.2 hold because > 3 (cf. Equation (7)) and 1 $\_{max}$, hence **c** > 6/(1 $\_{max}$) as required by Lemma D.2. By that lemma we deduce the existence of a set **S** **L;** (**S**/**L**) $\frac{2}{\mathbf{c}}$ such that for all 2 T we have both

$$w\ |_{\mathbf{L}\cap\mathbf{S}} \in \mathrm{RS}[F; \mathbf{L}; \_{max}]|_{\mathbf{L}\cap\mathbf{S}} \text{ and } \mathbf{x}^{|\mathbf{L}|(\_{max})}\ w\ |_{\mathbf{L}\cap\mathbf{S}} \in \mathrm{RS}[F; \mathbf{L}; \_{max}]|_{\mathbf{L}\cap\mathbf{S}}: \tag{54}$$

Let **d** = deg $w\ |_{\mathbf{L}\cap\mathbf{S}}$ < $\_{max}$ |**L**| and **k** = |**L**| ( $\_{max}$ ). We have

$$\mathbf{d} + 2\mathbf{k} < |\mathbf{L}|(\_{max} + 2(\_{max})) < |\mathbf{L}|\ 3\ \_{max} < |\mathbf{L} \cap \mathbf{S}|:$$

The last inequality follows from $\_{max}$ 1/4 and |**S**| 2/**c** 1/4 (see Equations (7) and (53)). So by Claim D.1 we conclude

$$8\ \ 2\ T\ \ w\ |_{\mathbf{L}\cap\mathbf{S}} \in \mathrm{RS}[F; \mathbf{L}; \ ]|_{\mathbf{L}\cap\mathbf{S}}: \tag{55}$$

Let $\mathbf{H}_0 = \{\mathbf{x} \in \mathbf{L}_{cmp} \mid N(\mathbf{x}) \setminus \mathbf{S} \neq \emptyset\}$; a union bound gives $|\mathbf{H}_0|$ |**S**|. Let $\mathbf{w}^0$ be the low degree extension of $\mathbf{w}\ |_{\mathbf{L}\cap\mathbf{S}}$ to domain **L**, noticing $\mathbf{w}^0$ 2 RS[F; **L;** ]. Let $w^0 = \{\mathbf{w} \mid \ 2\ T\}$. Recall the assumption that x has -distance, and let **C** be the linear code of minimal distance that contains RS[F; $\mathbf{L}_{cmp}$; $\_{cmp}$] as required by Item 2). By Equation (55) and Item 2 we conclude $\_N[w^0]$ 2 **C** for each 2 . Thus, to complete our soundness analysis, we need only show that $\_N[w^0]$ 2 RS[F; $\mathbf{L}_{cmp}$; $\_{cmp}$] for each 2 .

Consider the set of functions

$$\_N\ w^0\ |\ \ 2\ \ \cup\ \{\mathbf{g}_{mask}\}$$

and let **V** denote the linear span of this set. Since, by assumption, $\_N[w^0]$ agrees with $\_N[w]$ on $\mathbf{L}_{cmp} \cap \mathbf{H}_0$, we conclude that $\mathbf{g}^{(0)}|_{\mathbf{L}_{cmp}\cap\mathbf{H}_0} \in \mathbf{V}_{\mathbf{L}_{cmp}\cap\mathbf{H}_0}$. Therefore, if there exists even one member of $\{\_N[w^0] \mid \ 2\ \}$ that does not belong to RS[F; $\mathbf{L}_{cmp}$; $\_{cmp}$], then Lemma D.3 implies

$$\Pr\left[\mathbf{g}^{(0)}; \mathrm{RS}[F; \mathbf{L}_{cmp}; \_{cmp}]\right] \quad \frac{1}{2}\ (\quad (\mathbf{H}_0/\mathbf{L}_{cmp})) \quad 1/|F|$$

which contradicts Item 2 stated at the beginning of this proof, because $\frac{\_}{2}$ $\frac{1}{2}$ ( (**H**$_0$/**L**$_{cmp}$)) by our choice of in Equation (7). Therefore we conclude that

$$8\ \ 2\ \ ;\ \ \_N\ w^0\ \in \mathrm{RS}[F; \mathbf{L}_{cmp}; \_{cmp}] \tag{56}$$

and hence $w^0$ satisfies x, completing the soundness analysis. □

**Remark D.4** (Potential for improvement of soundness analysis)**.** *The* 2 *loss in soundness, compared to distance , is due to two factors. Lemma D.3 "costs" a* 2 *factor, and the union bound used in the proof above incurs another* $1 + \frac{|\mathbf{L}|}{|\mathbf{L}_{cmp}|}$ *loss. It remains an interesting open problem to decide if either factor is actually required (cf. Conjectures B.15 and B.16).*

### D.2.3 Knowledge Extraction — Part 4

The proof of Item 4 of Theorem B.14 relies on the following lemma. After proving it, we complete the proof of knowledge extraction. Recall we assume x has -distance.

**Lemma D.5.** *Suppose there exists* $\mathbf{L}^{0}_{\text{cmp}}$ $\mathbf{L}_{\text{cmp}}$ *such that all of the following hold:*

1. - $\mathbf{L}^{0}_{\text{cmp}}/\mathbf{L}_{\text{cmp}}$ > 1
   - $_{N}[w]\,j_{\mathbf{L}^{0}_{\text{cmp}}}$ 2 RS[F; $\mathbf{L}_{\text{cmp}}$; $_{\text{cmp}}$]$j_{\mathbf{L}^{0}_{\text{cmp}}}$ *for every* 2
2. *For* $\mathbf{L}^{0} := N(\mathbf{L}^{0}_{\text{cmp}})$ *and each* 2 T *:*
   - $(\mathbf{L}^{0}/\mathbf{L})$ $^{p}$ —
   - w $j_{\mathbf{L}^{0}}$ 2 RS[F; $\mathbf{L}$; ]$j_{\mathbf{L}^{0}}$

*Then* x 2 APR *and the assignment* $w^{0} := \{w^{0}\ j\ \ 2\ T\}$ *where* $w^{0}$ *is the low-degree extension of* w $j_{\mathbf{L}^{0}}$ *to* $\mathbf{L}$ *witnesses* x 2 APR.

*Proof.* By definition of $w^{0}$ and by assumption 2 we conclude $w^{0}$ 2 RS[F; $\mathbf{L}$; ] for all 2 T. By assumption 1 we know there is some codeword $\mathbf{w}$ 2 RS[F; $\mathbf{L}_{\text{cmp}}$; $_{\text{cmp}}$] that is within distance of $_{N}[w^{0}]$, while $_{N}[w^{0}]$ is a codeword of $\mathbf{C}$ having relative distance at least , as x is a -distance instance, thus $_{N}[w^{0}]$ 2 RS[F; $\mathbf{L}$; ]. Finally we conclude $(x; w^{0})$ 2 $R_{\text{APR}}$ by definition of $R_{\text{APR}}$. □

**Lemma D.6.** *There exists a Las Vegas (randomized) polynomial time algorithm* E *that satisfies the following condition. Given as input a -distance instance* x *with* $_{\max}$ 1/4 *and* w $(F^{\mathbf{L}})^{T}$ *for which there exists* $\mathbf{L}^{0}_{\text{cmp}}$ *that satisfies the assumptions of Lemma D.5 with respect to* x *and* w*, the output of* E *on input* $(x; w)$ *is a witness* $w^{0}$ $(F^{\mathbf{L}})^{T}$ *that satisfies* x.

*Proof of Theorem B.14, 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.14 hold, then Equation (55) holds, and moreover, the low degree extension $w^{0}$ of wj $j_{\mathbf{L}\cap\mathbf{S}}$ (for $\mathbf{S}$ defined there) is a satisfying assignment. Thus, our extractor will find this $w^{0}$ (with high probability), using the polynomial time Guruswami-Sudan (GS) list-decoding algorithm [65]. Recall that for RS[F; $\mathbf{S}$; ]; $|\mathbf{S}| = \mathbf{n}$ and $\mathbf{f} : \mathbf{S} \to F$, the GS algorithm runs in time poly($\mathbf{n}$) and outputs a list $\mathbf{L_f}$ RS[F; $\mathbf{S}$; ]; $|\mathbf{L_f}| = \text{poly}(\mathbf{n})$ that contains every codeword that agrees with $\mathbf{f}$ on more than a $^{p}$ —fraction of entries. In other words, when given $\mathbf{f} : \mathbf{L} \to F$ that agrees with some codeword $\mathbf{w}$ 2 RS[F; $\mathbf{L}$; ] on more than a $^{p}$ —fraction of entries, the GS algorithm will return $\mathbf{w}$ as part of $\mathbf{L_f}$.

Notice that in our case, the assumption $_{\max}$ 1/4 along with the upper bound $|\mathbf{S}|$ $|\mathbf{L}|$/4, which follows from Equations (7) and (53), implies that the set of inputs on which $\mathbf{w}$ 2 w and $\mathbf{w}^{0}$ 2 $w^{0}$ is of size at least $^{p}$ —. We apply the following extractor.

E(x; w)

1. Sample uniformly random $\mathbf{a}$ 2 $F^{T}$ and compute $\mathbf{f} : \mathbf{L} \to F$ by $\mathbf{f}(\mathbf{x}) \overset{4}{=} \sum_{2T} \mathbf{a}\ \mathbf{w}\ (\mathbf{x})$
2. Let $\mathbf{L_f}$ be the output of the GS list-decoding algorithm on $\mathbf{f}$.
3. For each $\mathbf{g}$ 2 $\mathbf{L_f}$,
   (a) let $\mathbf{S_g} = \{\mathbf{x} \in \mathbf{L}_{\text{cmp}} \mid 8(\ ; \mathbf{N}) \in N; \mathbf{f}(\mathbf{N}(\mathbf{x})) = \mathbf{g}(\mathbf{N}(\mathbf{x}))\}$;
   (b) let $w^{0} = \{\mathbf{w}^{0}\}_{\ 2T}$ where $\mathbf{w}^{0}$ is the low-degree extension of $\mathbf{w}\ j_{N(\mathbf{S_g})}$;

(c) if $w^0$ satisfies x then output $w^0$ and return "success"

4. return "fail"

Clearly E runs in polynomial time in its input because $|\mathbf{L_f}| = \text{poly}(|\mathbf{L}|)$. Additionally, it is a one-sided error algorithm because if it returns "success" then $w^0$ indeed satisfies x. Thus it only remains to analyze the probability of failure, given by the following claim.

**Claim D.7.** *Suppose* $\mathbf{L}^0_{\text{cmp}}$ *satisfies all properties of Lemma D.5 and is of maximal size with respect to property 2 of that Lemma. Then*

$$\Pr_{\mathbf{a}}\left[\exists \mathbf{g} \in \mathbf{L_f}; \mathbf{S_g} = \mathbf{L}^0_{\text{cmp}}\right] \geq 1 - \frac{|\mathbf{L}_{\text{cmp}}|}{|F|}. \tag{57}$$

Assuming the claim, when examining $\mathbf{g} \in \mathbf{L_f}$ with $\mathbf{S_g} = \mathbf{L}^0_{\text{cmp}}$ we have $\mathbf{w}_{|N(\mathbf{S}_g)} \in \text{RS}[F; \mathbf{L}; \ ]_{|N(\mathbf{S}_g)}$ for any $\in T$ by assumption and the low degree extension $\mathbf{w}^0$ of $\mathbf{w}_{|N(\mathbf{S}_g)}$ witnesses $x \in \text{APR}$. This completes the proof of Lemma D.6 but for the proof of Claim D.7, which appears next. □

*Proof of Claim D.7.* The assumption that $\mathbf{L}^0_{\text{cmp}}$ is maximal with respect to property 2 of Lemma D.5 means that for each $\in \mathbf{L}_{\text{cmp}} \cap \mathbf{L}^0_{\text{cmp}}$ there exists $\in N(\ )$ and some $\in T$ such that $\mathbf{w}(\ )$ does not agree with the low-degree extension of $\mathbf{w}_{|N(\mathbf{L}^0_{\text{cmp}})}$ that we shall denote by $\mathbf{w}^0$. Let $\mathbf{e} : \mathbf{L} \to F$ be the "error function" related to $\mathbf{w}^0$, defined as $\mathbf{e}(\mathbf{x}) = \mathbf{w}^0(\mathbf{x}) - \mathbf{w}(\mathbf{x})$. We have

$$\Pr_{\mathbf{a}}\left[\mathbf{f}(\ ) = \sum_{\in T} \mathbf{a}\, \mathbf{w}^0(\ )\right] = \Pr_{\mathbf{a}}\left[\sum_{\in T} \mathbf{a}\, \mathbf{e}(\ ) = 0\right] = \frac{1}{|F|}$$

the last equality holds because by assumption $\mathbf{e}(\ ) \neq 0$. Applying a union bound to $\in \mathbf{L}_{\text{cmp}} \cap \mathbf{L}^0_{\text{cmp}}$ we conclude that (57) holds with probability at least $1 - \frac{|\mathbf{L}_{\text{cmp}}| \cdot |\mathbf{L}^0_{\text{cmp}}|}{|F|} \geq 1 - \frac{|\mathbf{L}_{\text{cmp}}|}{|F|}$, and this completes the proof. □

### 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 $\mathbf{R}$ 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 $P_{\mathbf{f}}; P_{\mathbf{g}}$ denote the two provers, respectively. Our STIK (and STARK) instantiates both $P_{\mathbf{f}}; P_{\mathbf{g}}$ 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 $O_{\text{assignment}}$ 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 $\mathbf{f}^{(0)} \in \text{RS}[F; \mathbf{L}; \ _{\max}]$ and $\mathbf{g}^{(0)} \in \text{RS}[F; \mathbf{L}_{\text{cmp}}; \ _{\text{cmp}}]$ and recording them. The simulator also instantiates the two RS-IOPP provers — $P_{\mathbf{f}}$ and $P_{\mathbf{g}}$ corresponding to steps Item 2(c)i, Item 2(c)ii of the ALI protocol — with $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$, respectively. It also invokes V and records the first message, which is the randomness $\mathbf{R}$ 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 $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$) are managed by Sim by invoking the corresponding IOPP prover(s) $P_{\mathbf{f}}; P_{\mathbf{g}}$.

To complete the description of Sim we need only explain how it answers queries to $O_{assignment} = (w; \mathbf{f}_{mask}; \mathbf{g}_{mask})$. Recall that w is a collection of functions, each with domain $\mathbf{L}$, and this is also the domain of $\mathbf{f}_{mask}$; the domain of $\mathbf{g}_{mask}$ is $\mathbf{L}_{cmp}$. As in [17, Section 6], Sim maintains a set of partial functions $w\ ; \mathbf{f}_{mask}; \mathbf{g}_{mask}$ (with the same domains as $w; \mathbf{f}_{mask}; \mathbf{g}_{mask}$); all these functions are initialized with values that indicate "undetermined". When a function in $O_{assignment}$ is queried, Sim answers with the value recorded in $w\ ; \mathbf{f}_{mask}; \mathbf{g}_{mask}$, if determined, and will otherwise determine it, i.e., change its value from to some element of F. The process by which undetermined values get determined is described next:

1. A query $\mathbf{x}_0 \ 2\ \mathbf{L}$ sent to a function $\mathbf{f}\ 2\ w\ [\ f\mathbf{f}_{mask}g$, is determined jointly for all functions $\mathbf{f}^{\emptyset}\ 2\ w\ [\ f\mathbf{f}_{mask}g$. Consider Equation (49). The term on the left hand side, $\mathbf{f}^{(0)}(\mathbf{x}_0)$, is already fixed by Sim. On the right hand side, the terms

$$\left\{ \mathbf{r}_{;0} + \mathbf{r}_{;1}\ \mathbf{x}_0^{j\mathbf{L}j(\ max\ )}\ j\quad 2\ T \right\} \tag{58}$$

   are all fixed. The only undetermined values are those of $\mathbf{f}_{mask}(\mathbf{x}_0)$ and $f w\ (\mathbf{x}_0)\ j\quad 2\ T\ g$. Thus, our simulator determines these undetermined values by sampling a uniformly random solution to the linear constraint imposed by $\mathbf{f}^{(0)}(\mathbf{x}_0)$ and Equation (58).

2. A query $\mathbf{y}_0\ 2\ \mathbf{L}_{cmp}$, sent to the function $\mathbf{g}_{mask}$, is determined thus. The left hand side of Equation (50) is already determined by Sim. For all $\mathbf{x}_0\ 2\ N(\mathbf{y}_0)$, our simulator determines the value of all $f w\ (\mathbf{x}_0)\ j\quad 2\ T\ g$ and $\mathbf{f}_{mask}(\mathbf{x}_0)$ using the process described in Item 1. After all such values $w\ (\mathbf{x}_0)$ are determined for $\mathbf{x}_0\ 2\ N(\mathbf{y}_0)$, notice that the rightmost summand of Equation (50) is also determined. Thus, Sim determines $\mathbf{g}_{mask}(\mathbf{y}_0)$ 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 x, is the same as the distribution on transcripts of the interaction between V and an honest prover holding a witness for x and operating as described in ALI. Notice the following facts about the distribution supplied by the honest prover:

1. Each w is sampled uniformly and independently from a $j\mathbf{L}j$-wise independent space;
2. $\mathbf{R}\ (\ _{w;N(\mathbf{y}_0)})$ is determined by $\mathbf{R}$ and $f w\ (\mathbf{x}_0)\ j\ \mathbf{x}_0\ 2\ N(\mathbf{y}_0);\quad 2\ T\ g$;
3. the pair $(\mathbf{f}_{mask}; \mathbf{g}_{mask})$ is sampled uniformly from $RS[F; \mathbf{L};\ _{max}]\quad RS[F; \mathbf{L}_{cmp};\ _{cmp}]$;
4. consequently, independently of $\mathbf{R}$ and w, the pair $(\mathbf{f}^{(0)}; \mathbf{g}^{(0)})$ is sampled uniformly from $RS[F; \mathbf{L};\ _{max}]$ $RS[F; \mathbf{L}_{cmp};\ _{cmp}]$

Item 4 above relies on the completeness property, which says that if w satisfies x then the rightmost summand of Equation (50) is a codeword of $RS[F; \mathbf{L}_{cmp};\ _{cmp}]$.

Consequently, for every fixing of the first verifier message $\mathbf{R}$, and for every subset $\mathbf{S}\quad \mathbf{L}; j\mathbf{S}j <$ $j\mathbf{L}j$, the distribution on $w j_{\mathbf{S}}; \mathbf{f}_{mask} j_{\mathbf{S}}; \mathbf{f}^{(0)} j_{\mathbf{S}}$ generated by the honest ALI prover is the *uniform distribution* on field elements satisfying the linear constraint of Equation (49) for each $\mathbf{x}\ 2\ \mathbf{S}$. By construction, the distribution supplied by Sim invoking V which makes these queries $\mathbf{S}$, is precisely the same distribution.

Next, assume the aforementioned $\mathbf{S}$ includes all $\mathbf{x}_0\ 2\ N(\mathbf{S}^{\emptyset})$, where $\mathbf{S}^{\emptyset}\quad \mathbf{L}_{cmp}$ is the set of queries made by V to $\mathbf{g}_{mask}$. 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 $\mathbf{g}^{(0)}|_{S^0}; \mathbf{g}_{\text{mask}}|_{S^0}$ is thus the uniform distribution on field elements satisfying the linear constraint of Equation (50) for every $\mathbf{y}_0 \in \mathbf{S}^0$. By construction, Sim produces the same distribution on $\mathbf{g}^{(0)}|_{S^0}; \mathbf{g}_{\text{mask}}|_{S^0}$.

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 $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$.

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.14.

**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 ($O_{\text{assignment}}$) even* before *sending the randomness* $\mathbf{R}$*. 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.14. Verifier complexity is as stated because the verifier's only action during the ALI protocol is to sample the randomness $\mathbf{R}$. Regarding prover complexity, we follow the steps of the protocol:

1. In step 1 the prover samples random functions $w; \mathbf{f}_{\text{mask}}; \mathbf{g}_{\text{mask}}$. For each member of w this requires $3 \cdot |\mathbf{L}| \log |\mathbf{L}|$ arithmetic operations, as stated in Item 4h of Theorem B.11, and Theorem B.2.

2. In the beginning of Item 2(c)i the prover computes $\mathbf{f}^{(0)}$, by performing point wise computation using Equation (49), at a cost of $2|\mathbf{L}||T| \cdot$ multiplications, and $2|\mathbf{L}|(|T| \cdot + 1)$ additions over F. The total arithmetic complexity of this part is less than $5|T| \cdot |\mathbf{L}|$.

3. In step 2(c)ii the prover computes $\mathbf{g}^{(0)}$ and (its proximity proof is discussed later), by performing a complete point wise computation using $|\mathbf{L}_{\text{cmp}}| \cdot (T_{\text{arith}}(\ ) + |\ |)$ multiplications, and $|\mathbf{L}_{\text{cmp}}| \cdot (T_{\text{arith}}(\ ) + |\ | + 1)$ additions over 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.15 and B.16, starting with the latter one.

**Pure pseudo-prover** Informally, Conjecture B.15 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 $O_{\text{assignment}} = (w\ ; \mathbf{f}_{\text{mask}}; \mathbf{g}_{\text{mask}})$; a pure pseudo-assignment satisfies

$$\text{span}(w\ ; \mathbf{f}_{\text{mask}}) \quad \text{RS}[F; \mathbf{L};\ _{\max}] \quad \text{span}(f\ (\ _{w\ ;N}) \mid \ \in\ ; \mathbf{g}_{\text{mask}}) \quad \text{RS}[F; \mathbf{L}_{\text{cmp}};\ _{\text{cmp}}]:$$

In words, a pure pseudo-assignment selects $O_{\text{assignment}}$ such that either each w is a codeword of the relevant code RS[F; **L**; ], in which case the resulting constraint polynomials will be, with very high probability, maximally far from RS[F; $\mathbf{L}_{\text{cmp}}$; cmp], or else P fixes the intended values of each 2 to correspond to a member **g** 2 RS[F; $\mathbf{L}_{\text{cmp}}$; cmp] and then find, for each **y** 2 $\mathbf{L}_{\text{cmp}}$, a setting for the values of w(N (**y**)) so that ( w ;N )(**y**) = **g** (**y**); in this case, the resulting w will likely be maximally far from RS[F; **L**; max].

**Mixed pseudo-prover** The "attack" imagined to support Conjecture B.16 is the following: on input x, the pseudo-prover starts with some w RS[F; **L**; max], which leads to f ( w ;N ) j 2 g being maximally far from RS[F; $\mathbf{L}_{\text{cmp}}$; cmp]. Next, for some fraction of the entries of $\mathbf{L}_{\text{cmp}}$, the attacker "reverse-engineers" a change to an -fraction of the entries of w as to make each member of f ( w ;N ) j 2 g -closer to low-degree. Assuming the spaces $\mathbf{L}_0; \mathbf{L}_0^{\emptyset}$ are good for x, as we do in Conjecture B.16, this modification "ruins" an 2 $\frac{j\mathbf{L}_{\text{cmp}}j}{\mathbf{L}}$-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 $F_{2^{64}}$, i.e., hgi = $F_{2^{64}}$. We assume throughout that field elements are represented according to a standard basis $(1; g; g^2; g^3; :::; g^{63})$, rather than a normal basis. We denote by $R_{(t)}$ 2 $F_{2^{64}}$ 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 $4\mathbf{n}$ elements and a key array of $4\mathbf{n}$ elements, such that each element resides in $F_{2^8}$. Rijndael executes in $\mathbf{n} + 6$ rounds, where each round consists of the following four steps (except for the last round that skips Step 3):

1. **SubBytes** - Given byte $\mathbf{x}$, compute $\mathbf{y} = \mathbf{M}\mathbf{x}^{-1} + \mathbf{b}$, where $\mathbf{M} \in F_{2^8}^{8\times 8}$, $\mathbf{b} \in F_{2^8}^{8\times 1}$ are constants.

2. **ShiftRows** - For $\mathbf{i} \in \{1;2;3;4\}$, perform $\mathbf{i}$ cyclic shifts (rightwards) of the $\mathbf{i}$th row of the plain-text matrix.

3. **MixColumns** - For $\mathbf{j} \in 1;2;\ldots;\mathbf{n}$, multiply the $\mathbf{j}$th column of the plain-text matrix by the following constant circulant MDS matrix:

$$\begin{bmatrix} P0[j]_{(t+1)} \\ P1[j]_{(t+1)} \\ P2[j]_{(t+1)} \\ P3[j]_{(t+1)} \end{bmatrix} = \begin{bmatrix} g_0 & g_1 & 1 & 1 \\ 1 & g_0 & g_1 & 1 \\ 1 & 1 & g_0 & g_1 \\ g_1 & 1 & 1 & g_0 \end{bmatrix} \begin{bmatrix} P0[j]_{(t)} \\ P1[j]_{(t)} \\ P2[j]_{(t)} \\ P3[j]_{(t)} \end{bmatrix}.$$

Here, $g_0$ is a specific field element of $F_{2^8}$ with $|\langle g_0 \rangle| = 51$, and $g_1 = g_0 + 1$ generates $F_{2^8}$. We note that MixColumns also can be defined as computing the following linear combinations:

$$\begin{aligned}
P0[j]_{(t+1)} &= g_0 \cdot P0[j]_{(t)} + g_1 \cdot P1[j]_{(t)} + P2[j]_{(t)} + P3[j]_{(t)} \\
P1[j]_{(t+1)} &= P0[j]_{(t)} + g_0 \cdot P1[j]_{(t)} + g_1 \cdot P2[j]_{(t)} + P3[j]_{(t)} \\
P2[j]_{(t+1)} &= P0[j]_{(t)} + P1[j]_{(t)} + g_0 \cdot P2[j]_{(t)} + g_1 \cdot P3[j]_{(t)} \\
P3[j]_{(t+1)} &= g_1 \cdot P0[j]_{(t)} + P1[j]_{(t)} + P2[j]_{(t)} + g_0 \cdot P3[j]_{(t)}
\end{aligned}$$

4. **AddRoundKey**:

- Key-Scheduler
  - Using $\mathrm{Rcon}(\mathbf{t}) = g_0^{\mathbf{t}-1}$, the first column in the new key-matrix is computed according to:

$$\begin{bmatrix} K00_{(t+1)} \\ K10_{(t+1)} \\ K20_{(t+1)} \\ K30_{(t+1)} \end{bmatrix} = \begin{bmatrix} \mathrm{SubBytes}(K14_{(t)}) + K00_{(t)} + \mathrm{Rcon}(\mathbf{t}) \\ \mathrm{SubBytes}(K24_{(t)}) + K10_{(t)} \\ \mathrm{SubBytes}(K34_{(t)}) + K20_{(t)} \\ \mathrm{SubBytes}(K04_{(t)}) + K30_{(t)} \end{bmatrix}.$$

  - 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 $F_{2^8}$ with field operations modulo the irreducible polynomial $\mathbf{x}^8 + \mathbf{x}^4 + \mathbf{x}^3 + \mathbf{x} + 1$, while the operations in our IOP system are over $F_{2^{64}}$, defined using a different primitive polynomial. The properties of finite fields entail that for

any field $F_{p^m}$ and $k|m$, there exists a subfield $F_{p^k}$. Therefore, there is an isomorphism between $F_{2^8}$ and a subfield of $F_{2^{64}}$.

We obtain such an isomorphism by mapping a primitive element of $F_{2^{64}}$ to a primitive element of the subfield with $\mathbf{F}' = F_{2^8}$, so that the mapping is implied by their powers. This is done by finding an element of $F_{2^{64}}$ with order $2^8 - 1$.

We then transform all constants needed for Rijndael to their representation in $\mathbf{F}'$, and perform all the field operations in $\mathbf{F}'$. Importantly, this enables an efficient constraint for the SubBytes step of Rijndael, since we can represent the field inverse via a single multiplication in $\mathbf{F}'$. Specifically, by using an auxiliary element $\mathbf{z} \in \mathbf{F}'$, the constraint $\mathbf{y} = \mathbf{x}^{-1}$ can be represented via $\mathbf{y} \cdot \mathbf{z} = \mathbf{x}$. By contrast, a naïve implementation of the inverse operation would require auxiliary elements $\{\mathbf{b_i}\}_{\mathbf{i}=0}^{7}$, booleanity constraints $\prod\{\mathbf{b_i}(\mathbf{b_i}+1)\}_{\mathbf{i}=0}^{7}$, and a polynomial of degree 8 with 256 summands.

The full SubBytes S-box is defined according to $\mathbf{x} \mapsto \mathbf{M} \cdot \mathbf{x}^{-1} + \mathbf{b}$, where $\mathbf{M} \in F_2^{8\times 8}$ and $\mathbf{b} \in F_2^{8\times 1}$ are constants. Adding the constant $\mathbf{b}$ is a simple field addition in $\mathbf{F}'$, whereas the multiplication by the constant matrix $\mathbf{M}$ can be represented using a linear transformation $\mathbf{T} : F_{2^n} \to F_{2^n}$. 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 $\mathbf{C}(\mathbf{x}) = \sum_{\mathbf{i}=0}^{7} \mathbf{c_i}\mathbf{x}^{2^i}$ by finding coefficients $\{\mathbf{c_i}\}_{\mathbf{i}=0}^{7}$ that satisfy $\mathbf{C}(\mathbf{a_i}) = \mathbf{b_i}$, where $(\mathbf{a}_0; \mathbf{a}_1; \ldots; \mathbf{a}_7)$ is a basis for the domain of $\mathbf{T}$ and $(\mathbf{b}_0; \mathbf{b}_1; \ldots; \mathbf{b}_7)$ is a basis for the range of $\mathbf{T}$.

While the degree of $\mathbf{C}(\mathbf{x})$ 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 $\mathbf{C}(\mathbf{x}) = \mathbf{C}_1(\mathbf{C}_2(\mathbf{C}_3(\mathbf{x})))$ with 3 auxiliary field elements, using an AIR such as $\{\mathbf{z}' + \mathbf{C}_3(\mathbf{x}); \mathbf{z}'' + \mathbf{C}_2(\mathbf{z}'); \mathbf{z} = \mathbf{C}_1(\mathbf{z}'')\}$. Per Section 2.3, this AIR is translated into a single constraint $(\mathbf{z}' = \mathbf{C}_3(\mathbf{z})) \wedge (\mathbf{z}'' = \mathbf{C}_2(\mathbf{z}')) \wedge (\mathbf{z} = \mathbf{C}_1(\mathbf{z}''))$, 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 $\{\mathbf{c_i}\}_{\mathbf{i}=0}^{7}$, rather than the polynomial composition $\mathbf{C}_1(\mathbf{C}_2(\mathbf{C}_3(\mathbf{x})))$.

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 $\mathbf{T}$).

| P00 | $P34_{(0)}$ | K00 | $K34_{(0)}$ | INV1 | $INV5_{(0)}$ | W11 | $W53_{(0)}$ | $F1_{(0)}$ | $F2_{(0)}$ | $RC_{(0)}$ | $INVRC_{(0)}$ | $STATE_{(0)}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ⋮ | | | | | | | | | | ⋮ | |
| P00 | $P34_{(t)}$ | K00 | $K34_{(t)}$ | INV1 | $INV5_{(t)}$ | W11 | $W53_{(t)}$ | $F1_{(t)}$ | $F2_{(t)}$ | $RC_{(t)}$ | $INVRC_{(t)}$ | $STATE_{(t)}$ |
| P00 | $P34_{(t+1)}$ | K00 | $K34_{(t+1)}$ | INV1 | $INV5_{(t+1)}$ | W11 | $W53_{(t+1)}$ | $F1_{(t+1)}$ | $F2_{(t+1)}$ | $RC_{(t+1)}$ | $INVRC_{(t+1)}$ | $STATE_{(t+1)}$ |
| | ⋮ | | | | | | | | | | ⋮ | |
| P00 | $P34_{(T)}$ | K00 | $K34_{(T)}$ | INV1 | $INV5_{(T)}$ | W11 | $W53_{(T)}$ | $F1_{(T)}$ | $F2_{(T)}$ | $RC_{(T)}$ | $INVRC_{(T)}$ | $STATE_{(T)}$ |

$$
\begin{aligned}
&(\texttt{INV1}_{(t)}\texttt{P00}_{(t)} + 1)(\texttt{P00}_{(t)} \wedge \texttt{INV1}_{(t)})\wedge \\
&(\texttt{W11}_{(t)} + \texttt{INV1}^4_{(t)}) \wedge (\texttt{W12}_{(t)} + \texttt{W11}^4_{(t)}) \wedge (\texttt{W13}_{(t)} + \texttt{W12}^4_{(t)})\wedge \\
&(\texttt{P00}_{(t+1)} + \mathbf{c}_0\ \texttt{INV1}_{(t)} + \mathbf{c}_1\ \texttt{INV1}^2_{(t)} + \mathbf{c}_2\ \texttt{W11}_{(t)} + \mathbf{c}_3\ \texttt{W11}^2_{(t)} \\
&\qquad + \mathbf{c}_4\ \texttt{W12}_{(t)} + \mathbf{c}_5\ \texttt{W12}^2_{(t)} + \mathbf{c}_6\ \texttt{W13}_{(t)} + \mathbf{c}_7\ \texttt{W13}^2_{(t)} + \mathbf{b})
\end{aligned}
$$

Figure 9: Constraints polynomial for the Rijndael-160 SubBytes S-box.

The 20 registers $\begin{bmatrix} \texttt{P00} & \texttt{P01} & \texttt{P02} & \texttt{P03} & \texttt{P04} \\ \texttt{P10} & \texttt{P11} & \texttt{P12} & \texttt{P13} & \texttt{P14} \\ \texttt{P20} & \texttt{P21} & \texttt{P22} & \texttt{P23} & \texttt{P24} \\ \texttt{P30} & \texttt{P31} & \texttt{P32} & \texttt{P33} & \texttt{P34} \end{bmatrix}$ contain the 160 bits of the plain-text.

The 20 registers $\begin{bmatrix} \texttt{K00} & \texttt{K01} & \texttt{K02} & \texttt{K03} & \texttt{K04} \\ \texttt{K10} & \texttt{K11} & \texttt{K12} & \texttt{K13} & \texttt{K14} \\ \texttt{K20} & \texttt{K21} & \texttt{K22} & \texttt{K23} & \texttt{K24} \\ \texttt{K30} & \texttt{K31} & \texttt{K32} & \texttt{K33} & \texttt{K34} \end{bmatrix}$ contain the 160 bits of the key.

Each of these 40 registers is a field element of $F_{2^{64}}$ that resides in the subfield $\mathbf{F}^0 = F_{2^8}$, 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 `INV1;INV2;INV3;INV4;INV5` are used primarily as the auxiliary field elements that compute the inverses for SubBytes.

For $\mathbf{i} \in \{1,\ldots,5\}$, the registers `Wi1;Wi2;Wi3` store the repeated quadrupling that are used to compute the powers of `INVi`: `Wi1 = INVi`$^4$`;Wi2 = Wi1`$^4$ `= INVi`$^{16}$`;Wi3 = Wi2`$^4$ `= Wi1`$^{16}$ `= INVi`$^{64}$. Our constraints will then also square these registers, for example `Wi3  Wi3 = INVi`$^{128}$.

The registers `F1;F2` 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 `F1;F2` to enforce the requirements of the current step.

The register `RC` is used to compute Rcon($\mathbf{i}$) in round $\mathbf{i}$ 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: $\text{hash}(\mathbf{B};\mathbf{K}) = \mathbf{E}_{\mathbf{K}}(\mathbf{B}) \oplus \mathbf{B}$, where $\mathbf{E}$ is the Rijndael-160 cipher in our case. The resulting $\text{hash}(\mathbf{B};\mathbf{K})$ is collision-resistant under the assumption that for any key $\mathbf{K}$ the function $\mathbf{E}_{\mathbf{K}}(\,)$ 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 $\mathbf{E_K}(\,)$ 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 $\mathbf{B}$ while computing the output of the Rijndael-160 cipher. As discussed, our implementation expands $\mathbf{B}$ into 20 registers that contain elements of $F_{2^{64}}$. Since each of these 20 registers holds only 8 bits of information, we can compress and save $\mathbf{B}$ in 3 registers that hold 64 bits of information.

The compression method is quite simple: by treating $F_{2^{64}}$ as an extension field of $\mathbf{F}^{\,0} = F_{2^8}$ of degree 8, we set a basis of the extension field and encode 8 registers of $\mathbf{B}$ into an element of $F_{2^{64}}$ 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 $\mathtt{B0} = \sum_{\mathbf{k}=0}^{7} \mathbf{p_k}\ g_0^{\mathbf{k}}$, where the values $\mathbf{p_k}$ are taken from the registers `Pij`. We encode `B2;B3` in the same manner, except that for `B3` 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 $\mathbf{i} \in \{0; 1; 2\}$, we decompress `Bi` 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 $\{\mathbf{p_k}\}_{\mathbf{k}=0}^{19}$ nondeterministically (with $\mathbf{p_k} = 0$ for $\mathbf{k} \in \{20; 21; 22; 23\}$), and enforcing the algebraic constraint $\mathtt{Bi} + \sum_{\mathbf{k}=0}^{7} \mathbf{p_{k+8i}}\ g_0^{\mathbf{k}} = 0$ for $\mathbf{i} \in \{0; 1; 2\}$. However, this linear combination is unique only if the coefficients $\mathbf{p_k}$ reside in $\mathbf{F}^{\,0}$, and therefore the verifier must also validate $\mathbf{p_k} \in \mathbf{F}^{\,0}$ for $\mathbf{k} \in \{0; \ldots; 19\}$. This is achieved using Fermat's little theorem, which states that for any finite field $\mathbf{F}$, $\forall \mathbf{x} \in \mathbf{F} : \mathbf{x}^{|\mathbf{F}|} = \mathbf{x}$. In our case the constraints are $[\,\{\mathbf{p_k}^{256} + \mathbf{p_k} = 0\}_{\mathbf{k}=0}^{19}$, and we again reduce their total degree to 8 by using repeated quadrupling.

The state machine thus requires 3 additional registers for every cycle:

| | | |
|---|---|---|
| $\mathtt{B0}_{(0)}$ | $\mathtt{B1}_{(0)}$ | $\mathtt{B2}_{(0)}$ |
| ⋮ | ⋮ | ⋮ |
| $\mathtt{B0}_{(t)}$ | $\mathtt{B1}_{(t)}$ | $\mathtt{B2}_{(t)}$ |
| $\mathtt{B0}_{(t+1)}$ | $\mathtt{B1}_{(t+1)}$ | $\mathtt{B2}_{(t+1)}$ |
| ⋮ | ⋮ | ⋮ |
| $\mathtt{B0}_{(T)}$ | $\mathtt{B1}_{(T)}$ | $\mathtt{B2}_{(T)}$ |

We show the crux of the algebraic constraints for compression and decompression in Figure 10.

Constraint polynomial for compression

$$\mathtt{B0}_{(\mathbf{t}+1)} + \mathtt{P00}_{(\mathbf{t})} + \mathbf{x}\ \mathtt{P10}_{(\mathbf{t})} + \mathbf{x}^2\ \mathtt{P20}_{(\mathbf{t})} + \mathbf{x}^3\ \mathtt{P30}_{(\mathbf{t})} + \mathbf{x}^4\ \mathtt{P01}_{(\mathbf{t})} + \mathbf{x}^5\ \mathtt{P11}_{(\mathbf{t})} + \mathbf{x}^6\ \mathtt{P21}_{(\mathbf{t})} + \mathbf{x}^7\ \mathtt{P31}_{(\mathbf{t})}$$

Constraint polynomial for Fermat's little theorem

$$(\mathtt{INV2}_{(\mathbf{t})} + \mathtt{INV1}_{(\mathbf{t})}^4) \wedge (\mathtt{INV3}_{(\mathbf{t})} + \mathtt{INV2}_{(\mathbf{t})}^4) \wedge (\mathtt{INV4}_{(\mathbf{t})} + \mathtt{INV3}_{(\mathbf{t})}^4) \wedge (\mathtt{INV1}_{(\mathbf{t})} + \mathtt{INV4}_{(\mathbf{t})}^4)$$

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 $\mathbf{N}$ 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 $\mathbf{N}$ 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 $\mathbf{F}^{\,0}$ (cf. Appendix E.2). 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 $\mathrm{D}(\mathbf{n})$ of $\mathbf{n}$ profiles requires $\mathbf{N} = 2\mathbf{n}$ chain elements, and Program 1 consists of logic that alternates between odd and even elements.

**Program 1** DNA profile match

**Explicit inputs:** $\mathbf{n}; \mathrm{cm}_{\mathbf{t}}; \mathrm{cm}_{\mathbf{p}}$
**Nondeterministic inputs:** $\{\mathtt{VAL}_{\mathbf{i};\mathbf{j}}\}_{\mathbf{i}\,2\,\{1;2\};\mathbf{j}\,2\,\{1;2;\ldots;20\}}; \{\mathbf{W}_{\mathbf{i}}\}_{\mathbf{i}\,2\,\{0;1;2;\ldots;N\}}$

I: **if** $\mathrm{cm}_{\mathbf{p}} \neq \mathrm{hash160}(\{\mathtt{VAL}_{1;\mathbf{j}}\}_{\mathbf{j}\,2\,\{1;2;\ldots;20\}}; \{\mathtt{VAL}_{2;\mathbf{j}}\}_{\mathbf{j}\,2\,\{1;2;\ldots;20\}})$ **then return** false **end if**
II: $\mathbf{k} \leftarrow 1$; flag $\leftarrow 0$; $\mathbf{h} \leftarrow 0$; $\mathbf{T} \leftarrow 0$; $\mathbf{N} \leftarrow 2\mathbf{n}$
III: **while** $\mathbf{k} \neq g^{\mathbf{N}}$ **do**
IV: Parse $(\mathbf{L}_1; \mathbf{R}_1; \mathbf{L}_2; \mathbf{R}_2; \ldots; \mathbf{L}_{10}; \mathbf{R}_{10}) = \mathbf{W}_{\mathbf{j}}$ . $\mathbf{j} = \log_g \mathbf{k}$
V: **if** flag $= 0$ **then**
VI: $\mathbf{T}_1 \leftarrow \mathrm{CheckPairs}(\mathtt{VAL}_{1;1}; \mathtt{VAL}_{1;2}; \mathbf{L}_1; \mathbf{R}_1; \mathtt{VAL}_{2;1}; \mathtt{VAL}_{2;2}; \mathbf{L}_2; \mathbf{R}_2; \ldots; \mathtt{VAL}_{10;1}; \mathtt{VAL}_{10;2}; \mathbf{L}_{10}; \mathbf{R}_{10})$
VII: **else**
VIII: $\mathbf{T}_2 \leftarrow \mathrm{CheckPairs}(\mathtt{VAL}_{11;1}; \mathtt{VAL}_{11;2}; \mathbf{L}_1; \mathbf{R}_1; \mathtt{VAL}_{12;1}; \mathtt{VAL}_{12;2}; \mathbf{L}_2; \mathbf{R}_2; \ldots; \mathtt{VAL}_{20;1}; \mathtt{VAL}_{20;2}; \mathbf{L}_{10}; \mathbf{R}_{10})$
IX: $\mathbf{T} \leftarrow \mathrm{MatchingResult}(\mathbf{T}_1; \mathbf{T}_2; \mathbf{T})$
X: **end if**
XI: $\mathbf{h} \leftarrow \mathrm{hash160}(\mathbf{h}; \mathbf{W}_{\mathbf{j}})$
XII: $\mathbf{k} \leftarrow g \cdot \mathbf{k}$
XIII: flag $\leftarrow$ flag $+ 1$
XIV: **end while**
XV: **if** $\mathrm{cm}_{\mathbf{t}} \neq \mathbf{h}$ **then return** false **else return** $\mathbf{T}$ **end if**

To validate that the prover does not skip over some prefix of the chain in the exhaustive search, the total number of hash invocations $\mathbf{N}$ 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 $\mathrm{F}_{2^{64}}$, thereby avoiding integer arithmetics.

The register $\mathbf{T}$ stores the output, and is verified in the last cycle via a boundary constraint. The output

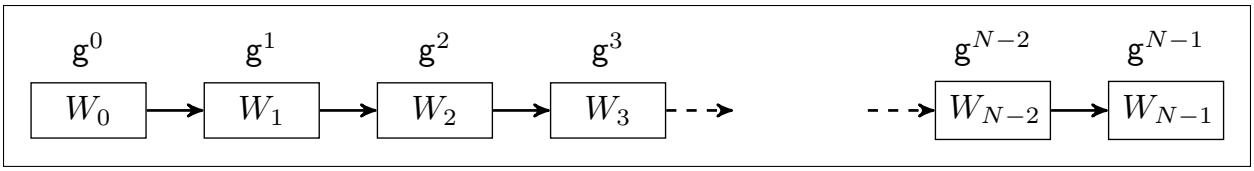


Figure 11: Illustration of the data structure used in the DNA profile match program.

can be either 1) "perfect match", meaning that an exact match between the input (i.e., the commitment $\mathrm{cm}_{\mathbf{p}}$ 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 $8\mathbf{j}\ 2\ \mathrm{f1;:::;20g}$ at least one STR value of its $\mathbf{j}$th pair matches a value of the $\mathbf{j}$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 $\mathtt{InnerMatch}_{(\mathbf{t})}$ and $\mathtt{Match}_{(\mathbf{t})}$ registers of Figure 15 correspond to $\mathbf{T}_1$ and $\mathbf{T}_2$ in Figure 13, and $\mathtt{Match}_{(\mathbf{t}+1)}$ corresponds to $\mathbf{T}$. In Figure 14, the registers $\mathtt{PAIR1}_{(\mathbf{t})}$ and $\mathtt{PAIR2}_{(\mathbf{t})}$ are compared to the $\mathbf{i}$th loci pair of the input. Each such comparison relies on 5 auxiliary registers: $\mathtt{INVTMP1}_{(\mathbf{t})};:::;\mathtt{INVTMP4}_{(\mathbf{t})}$ to compute field inverses, and $\mathtt{NewFlag}_{(\mathbf{t})}$ that represents the current inner matching (the register $\mathtt{LastFlag}_{(\mathbf{t})}$ is derived from the previous iteration).

Since the prover (but not the verifier) knows the decommitment of $\mathrm{cm}_{\mathbf{p}}$, 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 $\mathrm{cm}_{\mathbf{p}}$. 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 $\mathrm{cm}_{\mathbf{t}}$). 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.

| | | | | |
|---|---|---|---|---|
| $\mathtt{L0}_{(0)}$ | $\mathtt{L1}_{(0)}$ | $\mathtt{L2}_{(0)}$ | $\mathtt{L3}_{(0)}$ | $\mathtt{L4}_{(0)}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $\mathtt{L0}_{(t)}$ | $\mathtt{L1}_{(t)}$ | $\mathtt{L2}_{(t)}$ | $\mathtt{L3}_{(t)}$ | $\mathtt{L4}_{(t)}$ |
| $\mathtt{L0}_{(t+1)}$ | $\mathtt{L1}_{(t+1)}$ | $\mathtt{L2}_{(t+1)}$ | $\mathtt{L3}_{(t+1)}$ | $\mathtt{L4}_{(t+1)}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| $\mathtt{L0}_{(T)}$ | $\mathtt{L1}_{(T)}$ | $\mathtt{L2}_{(T)}$ | $\mathtt{L3}_{(T)}$ | $\mathtt{L4}_{(T)}$ |

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 $\mathbf{k} = g^{\mathbf{j}}$, 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 $81\ \ 62\ \ \mathbf{N} = 5022\ \ \mathbf{N}$.

CheckPairs($VAL_{1;1}$; $VAL_{1;2}$; $\mathbf{L}_1$; $\mathbf{R}_1$; $VAL_{2;1}$; $VAL_{2;2}$; $\mathbf{L}_2$; $\mathbf{R}_2$; : : : ; $VAL_{10;1}$; $VAL_{10;2}$; $\mathbf{L}_{10}$; $\mathbf{R}_{10}$)

```
I:    t    2
II:   for j  2 f1; : : : ; 10g do
III:      if (VAL_{j;1} = L_j) AND (VAL_{j;2} = R_j) then Continue end if
IV:       if (VAL_{j;1} = R_j) AND (VAL_{j;2} = L_j) then Continue end if
V:        if (VAL_{j;1} = L_j) OR (VAL_{j;1} = R_j) OR (VAL_{j;2} = L_j) OR (VAL_{j;2} = R_j) then
VI:           t    1
VII:      else
VIII:         return 0
IX:       end if
X:    end for
XI:   return t
```

Figure 12: CheckPairs subroutine of Program 1.

MatchingResult($\mathbf{T}_1$; $\mathbf{T}_2$; $\mathbf{T}$)

```
I:    if T_1 = T_2 = 2 then return 2 end if
II:   if (T_1 = 0) OR (T_2 = 0) then return T end if
III:  if T = 0 then return 1 end if
```

Figure 13: MatchingResult subroutine of Program 1.

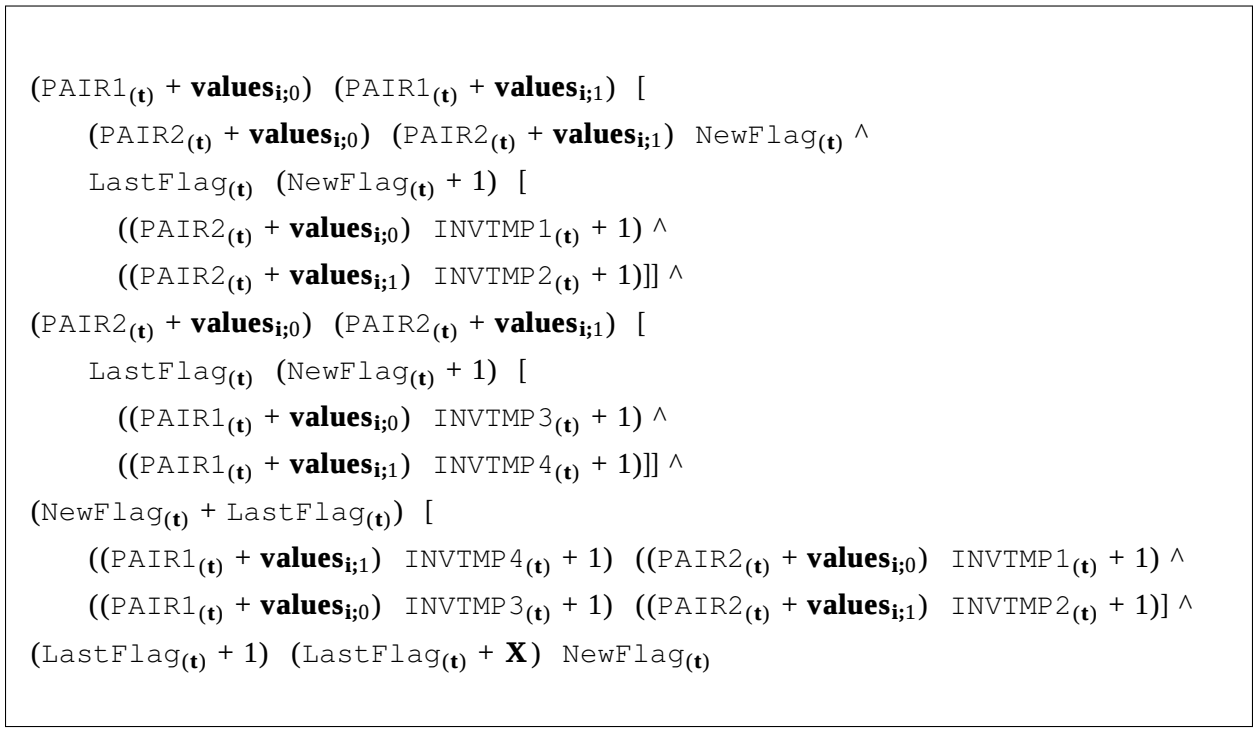


Figure 14: Algebraic constraints for one pair in the CheckPairs subroutine.

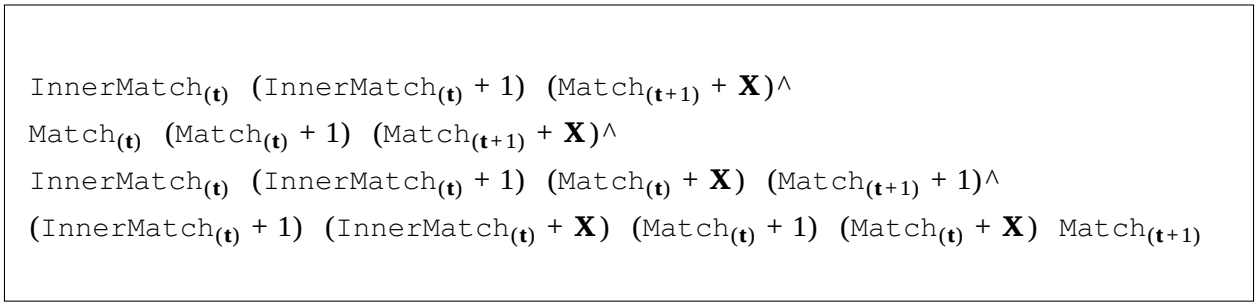


Figure 15: Algebraic constraints for the MatchingResult subroutine.

# F The AIR of the SHA2 hash function

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 $\mathbf{i}$th bit from $2\ F_{2^t}$ for any $\mathbf{i} < \mathbf{t}$; this system uses only a pair of constraints of degree 2 (notice the number of constraints and their degree is indepenedent of $\mathbf{t}$); 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 $\mathbf{y}\ 2\ F_{2^m}$ is defined as $\mathrm{Tr}_{\mathbf{m}|2}(\mathbf{y}) \ , \ \sum_{\mathbf{i}=0}^{\mathbf{m}\ 1} \mathbf{y}^{2^i}$.

**Proposition 1.** For any $F_2$-linear function $\mathbf{f} : F_2^{\mathbf{m}} \ ! \ F_2$, there exists a field element $_{\mathbf{f}}\ 2\ F_{2^m}$ such that $8\mathbf{y}\ 2\ F_{2^m} : \mathbf{f}(\mathbf{y}) = \mathrm{Tr}_{\mathbf{m}|2}(\ _{\mathbf{f}}\ \mathbf{y})$.

**Proposition 2.** For every $\mathbf{c}\ 2\ F_{2^m}$, the equation $\mathbf{y}^2 + \mathbf{y} = \mathbf{c}$ has solutions in $F_{2^m}$ if and only if $\mathrm{Tr}_{\mathbf{m}|2}(\mathbf{c}) = 0$. Notice that if $\mathbf{y}_0$ is a solution of $\mathbf{y}(\mathbf{y} + 1) = \mathbf{c}$ then $\mathbf{y}_0 + 1$ is the other solution, since the field characteristic is 2.

**Definition 1.** Let isZero( $;\mathbf{w};\mathbf{v}$) be the polynomial isZero( $;\mathbf{w};\mathbf{v}$) , $\mathbf{w}^2 + \mathbf{w} + \ \mathbf{v}$.

**Lemma 1.** The $\mathbf{i}^{\text{th}}$ coefficient in the standard basis representation of $\mathbf{y}\ 2\ F_{2^m}$ is 0 if and only if there exists $\mathbf{w}\ 2\ F_{2^m}$ such that $0 = \mathrm{isZero}(\ _{\mathbf{i}};\mathbf{w};\mathbf{y})$, where $_{\mathbf{i}}\ 2\ F_{2^m}$ is some field element that depends only on $\mathbf{i}$.

**Proof.** The function $\mathbf{f_i} : F_2^{\mathbf{m}} \ ! \ F_2$, $\mathbf{f_i}(\mathbf{y}) = \begin{matrix} 0; & \mathbf{i}^{\text{th}} \text{ coefficient of } \mathbf{y} \text{ is } 0 \\ 1; & \mathbf{i}^{\text{th}} \text{ coefficient of } \mathbf{y} \text{ is } 1 \end{matrix}$ is $F_2$-linear. It follows from Proposition 1 and Proposition 2 that the required $_{\mathbf{i}}$ exists. □

Furthermore, it is straightforward to pre-compute the constant $_{\mathbf{i}}$ 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.12. 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* $\mathbf{n}$ *is a directed graph* $\mathbf{G} = (\mathbf{V};\mathbf{E})$ *over the vertices* $\mathbf{V} \overset{4}{=} \{0;:::;\mathbf{n}\} \quad \{0;1\}^{\mathbf{n}}$ *and edges*

$$\mathbf{E} \overset{4}{=} \{[(\mathbf{i};\mathbf{w});(\mathbf{i}+1;\mathit{csr}(\mathbf{w}))];[(\mathbf{i};\mathbf{w});(\mathbf{i}+1;\mathit{csr}(\mathbf{w}) \quad 1)] \mid 0 \quad \mathbf{i} < \mathbf{n};\mathbf{w}\ 2\ \{0;1\}^{\mathbf{n}}\}$$

*where* $\mathit{csr}$: $\{0;1\}^{\mathbf{n}} \ ! \ \{0;1\}^{\mathbf{n}}$ *is the cyclic shift-right operation, and* $[\mathbf{x} \quad 1]$: $\{0;1\}^{\mathbf{n}} \ ! \ \{0;1\}^{\mathbf{n}}$ *flips the leftmost bit in* $\mathbf{x}$*, and does not change any other bit. Let* $C$ *be a set of colors. We say* $: \mathbf{V} \ ! \ C$ *is a*

*coloring of* $\mathbf{G}$ *if for every vertex* $\mathbf{v} = (\mathbf{i};\mathbf{w})$ *with* $\mathbf{i} < \mathbf{n}$*, there is a successor* $\mathbf{u}$ *of* $\mathbf{v}$ *(i.e.* $\mathbf{vu} \in \mathbf{E}$*) such that* $(\mathbf{v}) = (\mathbf{u})$.

**Definition G.2** (back-to-back De Bruijn routing)**.** *A* back-to-back De Bruijn routing *of dimension* $\mathbf{n}$ *is a pair of coloring functions* $= (_0; _1)$ *of the De Bruijn butterfly network of dimension* $\mathbf{n}$*, such that for all* $\mathbf{w} \in \{0;1\}^{\mathbf{n}}$ *(i)* $_0(\mathbf{n};\mathbf{w}) = _1(\mathbf{n};\mathbf{w})$*, and (ii) for any* $\mathbf{w}' \neq \mathbf{w}$*,* $_0(0;\mathbf{w}) \neq _0(0;\mathbf{w}')$*. In case* $|C| = 2^{\mathbf{n}}$ *we define the function induced by* $,$ $: C \to C$ *to be* $(_0(0;\mathbf{w})) = _1(0;\mathbf{w})$.

**Theorem G.3** (back-to-back De Bruijn is rearrangeable)**.** *For every set of colors* $C$ *of size* $2^{\mathbf{n}}$*, the set of functions induced by back-to-back De Bruijn routings of dimension* $\mathbf{n}$ *and colors from* $C$ *is exactly the set of permutations over* $C$*. Moreover, given a permutation* $: C \to C$*, there is an algorithm constructing a back-to-back De Bruijn routing* $$ *inducing* $$ *with time complexity* $\mathbf{O}(\mathbf{n}\ 2^{\mathbf{n}})$.

*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 . 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. □

**Procedure 2** Auxiliary method for routing

**Inputs:** two vectors $\mathbf{u};\mathbf{v}$ of length $2^{\mathbf{n}}$ indexed by $\{0;1\}^{\mathbf{n}}$ with distinct entries, where $\mathbf{u}$ is a permutation of $\mathbf{v}$
**Output:** vectors $\mathbf{v}^0;\mathbf{v}^1;\mathbf{u}^0;\mathbf{u}^1$ each of length $2^{\mathbf{n}-1}$ indexed by $\{0;1\}^{\mathbf{n}-1}$ with (i) each element of $\mathbf{v}$ is in $\mathbf{v}^0$ or $\mathbf{v}^1$, (ii) for every $\mathbf{w} \in \{0;1\}^{\mathbf{n}-1}$ and $\mathbf{i};\mathbf{b} \in \{0;1\}$, $\mathbf{v}_{\mathbf{b}\,\mathbf{w}} = \mathbf{v}^{\mathbf{i}}_{\mathbf{x}}$ or $\mathbf{u}_{\mathbf{b}\,\mathbf{w}} = \mathbf{u}^{\mathbf{i}}_{\mathbf{x}}$ only if $\mathbf{x} = \mathbf{w}$, and (iii) $\mathbf{u}^{\mathbf{i}}$ is a permutation of $\mathbf{v}^{\mathbf{i}}$.

I: **function** SPLIT($\mathbf{n};\mathbf{u};\mathbf{v}$)
II: initialize $\mathbf{v}^0;\mathbf{v}^1;\mathbf{u}^0;\mathbf{u}^1$ as partial mappings
III: **while** $\mathbf{v}^0;\mathbf{v}^1;\mathbf{u}^0;\mathbf{u}^1$ not yet well defined **do**
IV: **if** $\exists \mathbf{w} \in \{0;1\}^{\mathbf{n}-1},\mathbf{b};\mathbf{i} \in \{0;1\}$ such that $\mathbf{v}_{\mathbf{b}\,\mathbf{w}} \in \mathbf{u}^{\mathbf{i}}$ but $\mathbf{v}_{\mathbf{b}\,\mathbf{w}} \notin \mathbf{v}^{\mathbf{i}}$ **then**
V: $\mathbf{v}^{\mathbf{i}}_{\mathbf{w}} \leftarrow \mathbf{v}_{\mathbf{b}\,\mathbf{w}}$
VI: $\mathbf{v}^{\mathbf{i}}_{\mathbf{w}} \leftarrow \mathbf{v}_{\mathbf{b}\,\mathbf{w}}$
VII: continue
VIII: **else if** $\exists \mathbf{w} \in \{0;1\}^{\mathbf{n}-1},\mathbf{b};\mathbf{i} \in \{0;1\}$ such that $\mathbf{u}_{\mathbf{b}\,\mathbf{w}} \in \mathbf{v}^{\mathbf{i}}$ but $\mathbf{u}_{\mathbf{b}\,\mathbf{w}} \notin \mathbf{u}^{\mathbf{i}}$ **then**
IX: $\mathbf{u}^{\mathbf{i}}_{\mathbf{w}} \leftarrow \mathbf{u}_{\mathbf{b}\,\mathbf{w}}$
X: $\mathbf{u}^{\mathbf{i}}_{\mathbf{w}} \leftarrow \mathbf{u}_{\mathbf{b}\,\mathbf{w}}$
XI: **else**
XII: pick arbitrary $\mathbf{w} \in \{0;1\}^{\mathbf{n}-1}$ such that $\mathbf{v}_{0\,\mathbf{w}} \notin \mathbf{v}^0_{\mathbf{w}};\mathbf{v}^1_{\mathbf{w}}$
XIII: $\mathbf{v}^{\mathbf{i}}_{\mathbf{w}} \leftarrow \mathbf{v}_{\mathbf{i}\,\mathbf{w}}$ for $\mathbf{i} \in \{0;1\}$
XIV: **end if**
XV: **end while**
XVI: **return** $\mathbf{v}^0;\mathbf{v}^1;\mathbf{u}^0;\mathbf{u}^1$
XVII: **end function**

## G.2 Affine embedding of back-to-back De Bruijn routing

**Definition G.4** (Alegbraic De Bruijn)**.** *Let* $\mathrm{F}$ *be a field of characteristic* 2 *with primitive element* $\mathrm{g}$*, and fix some integer* $\mathrm{t}$*. Let* $\lceil \log(\mathrm{t}+1) \rceil$ *be a primitive binary polynomial of degree* $\lceil \log(\mathrm{t}+1) \rceil$*, Define* $\mathbf{H} \stackrel{4}{=}$

**Program 3** Routing permutation over back-to-back De Bruijn

**Inputs:** two vectors $\mathbf{u}; \mathbf{v}$ of length $2^{\mathbf{n}}$ indexed by $\{0,1\}^{\mathbf{n}}$ and distinct entries, where $\mathbf{u}$ is a permutation of $\mathbf{v}$
**Output:** back-to-back De Bruijn routing , represented as a pair of $2^{\mathbf{n}}$ $(\mathbf{n}+1)$ tables, with induced function satisfying $(\mathbf{v}_{\mathbf{w}}) = \mathbf{u}_{\mathbf{w}}$ for all $\mathbf{w}$

I: **function** ROUTE_DE BRUIJN($\mathbf{n}; \mathbf{u}; \mathbf{v}$)
II: . Recursion stopping condition
III: **if** $\mathbf{n} = 1$ **then**
IV: **if** $\mathbf{v}_0 = \mathbf{u}_0$ **then**
V: **return** $_0 = \ _1 = \begin{matrix} \mathbf{v}_0 & \mathbf{v}_0 \\ \mathbf{v}_1 & \mathbf{v}_1 \end{matrix}$
VI: **else**
VII: **return** $_0 = \begin{matrix} \mathbf{v}_0 & \mathbf{v}_0 \\ \mathbf{v}_1 & \mathbf{v}_1 \end{matrix}$ , $_1 = \begin{matrix} \mathbf{v}_1 & \mathbf{v}_0 \\ \mathbf{v}_0 & \mathbf{v}_1 \end{matrix}$
VIII: **end if**
IX: **end if**
X: . General case
XI: $(\mathbf{v}^0; \mathbf{v}^1; \mathbf{u}^0; \mathbf{u}^1)$ SPLIT($\mathbf{n}; \mathbf{u}; \mathbf{v}$)
XII: $^0$ ROUTE_DE BRUIJN($\mathbf{n}$ $1; \mathbf{u}^0; \mathbf{v}^0$)
XIII: $^1$ ROUTE_DE BRUIJN($\mathbf{n}$ $1; \mathbf{u}^1; \mathbf{v}^1$)
XIV: **for all** $\mathbf{w}$ $2$ $\{0,1\}^{\mathbf{n}}$ **do**
XV: $_0(0; \mathbf{w})$ $\mathbf{v}_{\mathbf{w}}$
XVI: $_1(0; \mathbf{w})$ $\mathbf{u}_{\mathbf{w}}$
XVII: **end for**
XVIII: **for all** $(\mathbf{i}; \mathbf{w})$ $2$ $[\mathbf{n}]$ $\{0,1\}^{\mathbf{n}}$ $^1$ **do**
XIX: represent $\mathbf{w}$ by $\mathbf{w} = \mathbf{x}$ $\mathbf{y}$ where $|\mathbf{y}| = \mathbf{i}$ $1$
XX: $_0(\mathbf{i}; \mathbf{x}$ $0$ $\mathbf{y})$ $^0_0(\mathbf{i}$ $1; \mathbf{w})$
XXI: $_0(\mathbf{i}; \mathbf{x}$ $1$ $\mathbf{y})$ $^1_0(\mathbf{i}$ $1; \mathbf{w})$
XXII: $_1(\mathbf{i}; \mathbf{x}$ $0$ $\mathbf{y})$ $^0_1(\mathbf{i}$ $1; \mathbf{w})$
XXIII: $_1(\mathbf{i}; \mathbf{x}$ $1$ $\mathbf{y})$ $^1_1(\mathbf{i}$ $1; \mathbf{w})$
XXIV: **end for**
XXV: **end function**

span $g^i \mid 0 \le i < t$, *and* $W \stackrel{4}{=} \text{span}\ g^{t+i} \mid 0 \le i < \lceil \log(t+1) \rceil$. *A pair of mappings* $\pi_0, \pi_1 : H \times W \to F$ *is an affine embedding of back-to-back De Bruijn of degree* t *if:*

- *for every* $x \neq x' \in H$, $\pi_0(x + g^{t+1}) \neq \pi_0(x' + g^{t+1})$
- *for every* $x \in H$, $\pi_0(x + g^t) = \pi_1(x + g^t)$
- *for every* $b \in \{0,1\}$, $x = \sum x_i g^i \in \text{span}\ g^i \mid 0 \le i < t$ *and every non zero* $y = \sum y_i g^i \in \text{span}\ g^i \mid 0 \le i < \lceil \log(t+1) \rceil$, *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* $\pi_b(z)$ *equals to one of:*
  - $\pi_b(g \cdot z + r \cdot (g^t + 1) + c \cdot (g^t))$
  - $\pi_b(g \cdot z + r \cdot (g^t + 1) + c \cdot (g^t) + 1)$

*Let* $C$ *be the image of* $\pi$ *over* $H$. *The function induced by* $\pi_0, \pi_1$ *is a mapping* $\pi : C \to F$ *mapping defined by* $\pi_0(x + g^{t+1}) \mapsto \pi_1(x + g^{t+1})$ *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* F.