Volatile and Persistent Memory for zkSNARKs via Algebraic Interactive Proofs

Alex Ozdemir Evan Laufer Dan Boneh
Stanford
aozdemir@cs.stanford.edu

***Abstract*—In verifiable outsourcing, an untrusted server runs an expensive computation and produces a succinct proof (called a SNARK) of the results. In many scenarios, the computation accesses a RAM that the server maintains a commitment to (persistent RAM) or that is initially zero (volatile RAM). But, SNARKs for such scenarios are limited by the high overheads associated with existing techniques for RAM checking. We develop new proofs about volatile, persistent, and sparse persistent RAM that reduce SNARK proving times. Our results include both asymptotic and concrete improvements—including a proving time reduction of up to 51.3× for persistent RAM. Along the way, we apply two tools that may be of independent interest. First, we generalize an existing construction to convert any algebraic interactive proof (AIP) into a SNARK. An AIP is a public-coin, non-succinct, interactive proof with a verifier that is an arithmetic circuit. Second, we apply Bézout's identity for polynomials to construct new AIPs for uniqueness and disjointness. These are useful for showing the independence of accesses to different addresses.**

1. Introduction

In verifiable outsourcing, a weak client outsources a computation to a powerful server [1, 2]. The server returns the result of the computation and a proof that the computation was done correctly. The proof is succinct: it is small and easy to verify.

Verifiable outsourcing is widely deployed in the context of decentralized systems, such as blockchains, where it is used to greatly reduce the amount of replicated effort to verify a computation. In this setting, an off-chain service performs a computation, such as processing a batch of transactions, and produces a succinct proof that the computation was done correctly. The result of the computation along with the succinct proof are posted on chain, and every validator in the network verifies the proof instead of redoing the computation from scratch. Having a succinct proof is important: its short size reduces total on-chain storage, and its fast verification time saves validator work.

In these deployments—and in many other applications of verifiable outsourcing—the outsourced computation reads from and writes to some persistent state. For example: a RAM that holds the account balances of every participant in a payment system. The computation operates on some initial memory state M, which is a map from a set of addresses $\{1, \ldots, N\}$ to values. To reduce storage and verification costs, the weak client (or blockchain) stores a succinct digest d of M, e.g., the root of a Merkle tree. The server stores all of M, and executes the computation to derive a new state M'. The server sends a new digest d' to the client, along with a proof that running the computation on a starting state consistent with d gives a final state consistent with d'.

The proof used here is called a *SNARK*, which we define in more detail in Section 2. In existing SNARKs, proof size and verification time are very small: for example, a few hundred bytes and a few milliseconds [3]. However, proving costs—both time and memory—are enormous: generally thousands of times higher than that of the original computation [4–6]. While costs can be defrayed with parallelism [7–12], they are still substantial, so optimizing SNARKs is an essential and active area of research [2–4, 13–25].

Most of the above research focuses on SNARKs for computations expressed as *arithmetic circuits*. Since many computations are not circuits, there has been significant research into extending SNARKs to richer computational models including control flow [4, 6, 9], persistent state [26–31], lookup tables [32–41], and RAM [6, 42–46]. These extensions aim to make SNARKs more practical, by improving their compatibility with existing software toolchains. However, they also introduce significant overhead.

We develop improved SNARKs for three kinds of RAM:

- The foregoing has discussed the case in which a SNARK proves the correctness of a computation that accesses a RAM of size N whose initial and final state are committed to with a digest. We call this a **persistent** RAM.
- Second, we consider a **sparse persistent** RAM, in which N is very large (e.g., $N \approx 2^{256}$), but at most C (e.g., $C \approx 2^{20}$) cells are non-zero at a time. This case naturally arises when addresses are hashes of unstructured identifiers (e.g., an email).
- Third, we consider a RAM that is initialized to zero for each computation, and whose final state is uncommitted. We call this a **volatile** RAM. This case is essential for SNARKs of RAM program execution.

We obtain significant asymptotic and concrete improvements. First, for volatile RAM, we give the first construction with proving cost independent of N. Second, for persistent RAM, our implementation reduces concrete proving costs

by up to 51.3× relative to the state of the art [30]. Finally, we estimate that our sparse persistent RAM reduces costs by up to 143× over approaches based on prior ideas [30].

Our improvements rely on three ideas. First, we build on a special kind of SNARK called a *commit-and-prove* (CP) SNARK, which allows for efficient commitments to (and proofs about) moderately sized arrays [47]. CP-SNARKs are promising because they do not require expensive cryptographic operations to be encoded as an arithmetic circuit—unlike state commitments for SNARKs based on Merkle trees or RSA accumulators [30]. However, a CP-SNARK does not immediately provide a RAM, because the array cannot be accessed at data-dependent locations.

Second, we give new *interactive* proofs to show that a sequence of RAM accesses is consistent with (committed) initial and final memory states. In doing this, one challenge that emerges is proving that accesses (or groups thereof) have unique addresses (or disjoint address sets). We show that using *interaction* and *randomness* gives better proofs for these properties, by leveraging Bézout's identity for polynomials. Our proofs are public-coin, non-succinct, and interactive. In them, the verifier's final test is an arithmetic circuit, so we call them *algebraic interactive proofs* (AIPs).

Third, we adapt a recent SNARK, Mirage [48], so that it can be used to compress an AIP into a SNARK. In this transformation, SNARK proving time depends on the size of the verification circuit for the AIP. Thus, by optimizing the AIP verifier, we reduce final proving times.

Our contributions are:

- We give new AIPs for the consistency of a sequence of RAM accesses, where the RAM is volatile (§4), persistent (§5), or sparse and persistent (§6). We define and prove security for all of our protocols.

  Our proofs apply Bézout's identity for univariate polynomials to prove various uniqueness and disjointness properties, which may be of independent interest.

- We construct CP-Mirage+: a generalization of Mirage that converts a (public-coin, multi-round, non-succinct, non-zero-knowledge) AIP into a CP-zkSNARK.[1] We define and prove security for our construction (§7).

- We implement (§8) and evaluate (§9) our constructions.

  For persistent RAM, our implementation performs best on midsize RAMs ($2^{10} \leq N \leq 2^{20}$), where it reduces proving time by up to 51.3×, relative to prior work.

  For volatile RAM, our construction is an asymptotic improvement: its proving costs are independent of N, while prior constructions scale as $(\log N)/(\log\log N)$. However, the concrete improvement for $N \leq 2^{60}$ is modest: at most a 32.9% proving time reduction.

  We also compare our sparse RAM against constructions based on prior ideas, via estimates of rank-1 constraint counts (a standard cost model; §3). For large RAMs of moderate capacity ($2^{10} \leq C \leq 2^{20}$), we predict an improvement of up to 143×.

[1] Our goal is efficiency, which requires only a CP-SNARK. Yet, CP-Mirage+ also guarantees zero-knowledge, making it a CP-zkSNARK.

In the rest of the paper, we discuss background and related work (§2), define an arithmetization for AIPs (§3), and then present our contributions (§4–§9).

2. Background and related work

Notation. Let $\mathbb{F}$ be a finite field of prime order p, (with $p \approx 2^{255}$), represented as $\{0, \ldots, p-1\}$. The order of $\omega \in \mathbb{F}$, denoted $|\omega|$, is the minimal natural $n > 0$ such that $\omega^n = 1$. Let $[i]$ denote $\{1, \ldots, i\}$ for $i \in \mathbb{N}$. For vectors $\mathbf{x}, \mathbf{y}$, let $(\mathbf{x}, \mathbf{y})$ denote their concatenation: $(x_1, \ldots, x_m, y_1, \ldots, y_n)$. For boolean b and any values t, f, let $\mathsf{ite}(b, t, f)$ be t if b is true and f otherwise ("if-then-else"). Let $\mathsf{to}_{\mathbb{F}}(b) = \mathsf{ite}(b, 1, 0) \in \mathbb{F}$. Let $\mathbb{F}^{n \times m}$ denote n by m matrices of elements from $\mathbb{F}$.

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of order p, generated by G_1, G_2, G_T, with the group operation $+$. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing: an efficient, non-trivial bi-linear map. For vector $\mathbf{x} \in \mathbb{F}^n$ and group element G let $\mathbf{x}G = (x_1 G, \ldots, x_n G)$.

Bézout's identity. Let $\mathbb{F}^{<n}[X]$ denote the set of univariate polynomials in X with coefficients in $\mathbb{F}$, of degree less than n. Let f, g, h be univariate polynomials in X of any degree. Bézout's identity states that there exist polynomials s, t such that $fs + gt = h$ if and only if h is divisible by $\gcd(f, g)$. Two corollaries follow. First, take $h = 1$; then suitable s, t exist if and only if f and g share no factors. Second, take $h = X^c$; then suitable $c \in \mathbb{N}, s, t$ exist if and only if powers of X are the only common factors of f, g.

Keyed hashing. The *coefficient hash* $H_c(\alpha \in \mathbb{F}, \mathbf{x} \in \mathbb{F}^n)$ for key α and data $\mathbf{x}$ is defined as $\sum_{i=1}^{n} \alpha^i x_i$. The *root hash hash* $H_r(\alpha, \mathbf{x})$ is defined as $\prod_{i=1}^{n} (\alpha - x_i)$. H_c is a *universal hash function* for vector inputs. That is, for all $\mathbf{x} \neq \mathbf{y}$, if α is sampled uniformly and independently, then the probability of a collision is at most $n/|\mathbb{F}|$. Meanwhile, H_r is a universal hash function for inputs as *multisets*. That is, if $\mathbf{x}$ and $\mathbf{y}$ differ as multisets then the probability of a collision is at most $n/|\mathbb{F}|$. Note that the collision resistence of these keyed hash functions requires keys sampled independently of the data.

Proofs. First, we introduce interactive proofs from the complexity perspective. An *interactive proof* (IP) [49, 50] is a protocol between a *prover* $\mathcal{P}$ and a PPT (probabilistic, polytime) *verifier* $\mathcal{V}$, by which $\mathcal{P}$ convinces $\mathcal{V}$ that an *instance* x is in a *language* $\mathcal{L}$. Two properties must hold:

- *completeness*: if $x \in \mathcal{L}$, then $\mathcal{V}$ accepts with overwhelming probability.
- *soundness*: if $x \notin \mathcal{L}$, then $\mathcal{V}$ accepts with negligible probability.

Cryptographic Proofs. Now, we introduce additional cryptographic properties for proofs of witness relations.

Let $\mathcal{P}$ and $\mathcal{V}$ be as before, and let $R(x, w)$ be a *witness relation* with public *instance* x (initially known to both

$\mathcal{P}$ and $\mathcal{V}$) and private *witness* w (initially known to only $\mathcal{P}$). Define the *instance language* for R: $\mathcal{L}_{x,R} = \{x : \exists w, R(x,w) = 1\}$, and regard $(\mathcal{P}, \mathcal{V})$ and an IP for $\mathcal{L}_{x,R}$. We say the IP is *complete* (resp. *sound*) for R if it is complete (resp. sound) for $\mathcal{L}_{x,R}$.

If soundness holds only against PPT $\mathcal{P}$, then the IP is called an *argument* rather than a proof. One can also require *knowledge soundness* [51], which informally means that whenever $\mathcal{V}$ accepts, $\mathcal{P}$ must "know" a witness w (slightly more formally, there exists a PPT *extractor* which can compute a valid w through interaction with $\mathcal{P}$). One can also require *zero-knowledge* [49], which informally means than the protocol conveys no information about w beyond its validity (slightly more formally, there exists a PPT *simulator* which can generate protocol transcripts—without using w—that are indistinguishable from real transcripts). A proof system is public-coin (also known as *Merlin-Arthur* [52]) if $\mathcal{V}$'s messages are uniformly random. A proof system is *non-interactive* if there is just one message (from $\mathcal{P}$ to $\mathcal{V}$); in this case, that message is called the *proof*.

Syntactically, a non-interactive proof system for relation class $\mathcal{R}$ (with pre-processing) is three PPT algorithms:

- $\mathsf{Setup}(R \in \mathcal{R}) \to (\mathsf{pk}, \mathsf{vk})$: sample proving and verifying keys
- $\mathsf{Prove}(\mathsf{pk}, x, w) \to \pi$: create a proof that $R(x,w)$ holds
- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \to \{0,1\}$: check a proof

It is *succinct* if the time and memory costs of Verify are at most $\mathsf{poly}(\log(|R|) + |x|)$, where $|x|$ is the size of x and $|R|$ is the time needed to check R directly. A *SNARK* [25] is a succinct, non-interactive argument that is knowledge sound. A zkSNARK is additionally zero-knowledge. There are many zkSNARK constructions, we will build on a construction of Groth [3] that was extend by Kosba et al. [48].

2.1. Related work

Memory checking has a long history [53], and there is much prior work on SNARKs with volatile [32, 34, 43–45, 54–57] and persistent [6, 42, 58] RAM. These constructions use tools such as Merkle trees, routing networks [59], lookup proofs [34], and RSA accumulators [60, 61]. We describe this prior work—and compare it to our constructions—in Sections 4.5 and 5.3.

One concurrent work [62] gives a PolyIOP construction for volatile RAM based on Bézout's identity (like ours; §4), but does not consider persistent or sparse persistent RAM. Another [63] studies persistent RAM using techniques quite different from ours (building on Caulk [64]).

A number of RAM-related works consider goals orthogonal to ours. For instance, VRAM focus on RAM with computation-independent Setup [65], and other works [32, 44, 45, 66] and projects [67–69] build on SNARKs for RAM to prove executions of RAM programs.

Further afield, others have studied RAM checking for interactive, non-succinct zero-knowledge proofs [70–74].

There is also significant work on SNARKs that access other data structures. A long line of work [32], with several recent additions [33–41], develops "lookup" proofs for SNARKs. These are essentially *read-only* memories. Others consider sets [75], multisets [31], maps [76], concurrent maps [29, 77], and relations [78].

Protostar [79] builds a folding scheme for special-sound protocols; this is closely related to our CP-zkSNARK for AIPs. We build on Mirage (a zkSNARK for MA[1] [48]) and on commit-and-prove SNARKs [47, 80].

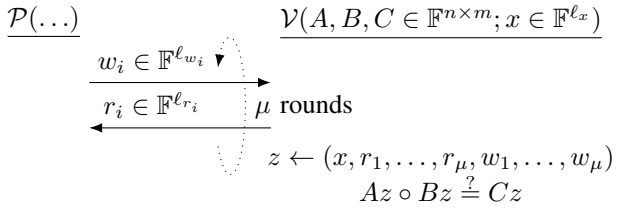


Figure 1: Our AIP arithmetization: I-R1CS. The parties exchanges field vectors in μ rounds, $\mathcal{V}$'s messages are random, and $\mathcal{V}$'s final test is a rank-1 constraint system (A, B, C).

3. AIPs and zkSNARKs

Our memory proofs will be *algebraic interactive proofs* (AIPs). In this section we define what an AIP is, and specify an AIP arithmetization (§3.1). Then, we state the key properties of our construction, called Mirage+, that is used to convert an AIP into a zkSNARK (§3.2). We present the Mirage+ construction in Section 7.

Algebraic Interactive Proofs. An AIP is a public-coin interactive proof in which all messages are vectors of elements from some field $\mathbb{F}$, and $\mathcal{V}$'s final test is a arithmetic circuit over $\mathbb{F}$. We will be interested in AIPs which (as IPs) are complete, sound, and knowledge-sound. We will **not** require our AIPs to be succinct or zero-knowledge.

3.1. I-R1CS

Our AIP arithmetization is called *I-R1CS* (interactive rank-1 constraint system). I-R1CS generalizes R1CS (a common arithmetization for NP [4, 19, 22]) to AIPs. In I-R1CS, $\mathcal{V}$'s final test is a rank-1 system. We illustrate I-R1CS in Figure 1 and define it here:

Definition 1 (I-R1CS). *An I-R1CS is a tuple* $\phi = (\mathbb{F}; \mu, n, m, \ell_x, \ell_{r_1}, \ldots, \ell_{r_\mu}, \ell_{w_1}, \ldots, \ell_{w_\mu} \in \mathbb{N}; A, B, C \in \mathbb{F}^{n\times m})$. *It defines a* μ*-round AIP over* $\mathbb{F}$*, where in round* $i \in [\mu]$*,* $\mathcal{P}$ *sends a message* $w_i \in \mathbb{F}^{\ell_{w_i}}$ *and receives a uniformly random response* $r_i \in \mathbb{F}^{\ell_{r_i}}$*. The last response must be empty:* $\ell_{r_\mu} = 0$*.* $\mathcal{V}$*'s test is defined by three matrices* $A, B, C \in \mathbb{F}^{n\times m}$ *where* n *is the number of constraints and* $m = \ell_x + \sum_i (\ell_{w_i} + \ell_{r_i})$ *is the number of variables.* $\mathcal{V}$ *accepts if* $Az \circ Bz = Cz$ *where* $z = (x, r_1, \ldots, r_\mu, w_1, \ldots, w_\mu)$.

Any AIP can be expressed as I-R1CS by having $\mathcal{P}$ send the results of all intermediate multiplications in $\mathcal{V}$'s circuit. In the resulting I-R1CS, n depends linearly on the number of non-linear multiplications in $\mathcal{V}$'s circuit. We call these

multiplications *constraints*, since that is what they become in the final I-R1CS. In future sections of this paper, our goal is to reduce the number of constraints.

3.2. A zkSNARK for I-R1CS

In Section 7, we construct Mirage+: a zkSNARK for I-R1CS. We also prove its security. Here, we summarise the key security and efficiency properties of our construction. The first is that Mirage+ transforms a complete and knowledge-sound AIP ϕ into a zkSNARK. We denote the application of Mirage+ to ϕ as $Mirage+_{\phi}$.

Theorem 1. *Let I-R1CS ϕ be complete and **knowledge-sound** for witness relation R. Then in the generic group model (GGM), $Mirage+_{\phi}$ is a zkSNARK for R.*

Since the AIP need not be succinct or zero-knowledge, it's easy to transform a sound AIP into a knowledge-sound AIP. Thus, Mirage+ can also transform a complete and **sound** AIP into a zkSNARK. So, it suffices to construct AIPs that are complete and sound.

Corollary 1. *Let R be a witness relation and let I-R1CS ϕ be complete and **sound** for language $\mathcal{L}_R = \{(x, w) : R(x, w) = 1\}$. Then there is a zkSNARK for R in the GGM.*

Proof. Note that in $\mathcal{L}_R$, (x, w) is the instance. Let ϕ' be ϕ, with w moved from the instance to the first message. Then, ϕ' is complete and knowledge-sound for R. Applying Mirage+ to ϕ' gives a zkSNARK for R. □

Efficiency. Mirage+ is very efficient. Verification uses $O(\ell_x + \mu)$ time and memory; concretely, just a few milliseconds (Sec. 9). Proofs are $\mu + 2$ elements of $\mathbb{G}_1$ and 1 element of $\mathbb{G}_2$; concretely, just a few hundred bytes (Sec. 9). Generally, proving costs are dominated by $O(n \log n)$ $\mathbb{F}$ operations for Fourier transforms and $O(n\lambda/(\log n))$ $\mathbb{G}_1$ operations for multi-scalar multiplications.

Shrinking Mirage+'s $\mathcal{P}$ cost generally requires shrinking n: the number of I-R1CS constraints. Interestingly, n is a measure of the *verifier* cost of the AIP that encoded by the I-R1CS. Thus, for the rest of this paper, our primary objective is to construct AIPs with **small $\mathcal{V}$ complexity**.

Commit and prove. We also construct CP-Mirage+: an efficient zkSNARK for witness relations where parts of the witness are committed. It uses the commit-and-prove technique [47] to avoid encoding the commitment scheme as I-R1CS constraints, which would be very expensive. The following theorem states the properties of CP-Mirage+; the full construction and interface are in Appendix B.

Theorem 2. *Let I-R1CS ϕ be complete and knowledge-sound for witness relation $R(x, (w_1, \ldots, w_k, w))$. Let* Com *be a Pedersen commitment scheme in $\mathbb{G}_1$. Then in the GGM, CP-Mirage+ is a zkSNARK for witness relation*

$$R_{com} = \{((x, c_1, \ldots, c_k), ((w_1, o_1), \ldots, (w_k, o_k), w)) : \bigwedge_{i=1}^{k} c_i = \mathsf{Com}(x_i, o_i) \wedge R(x, (w_1, \ldots, w_k, w))\}$$

Mirage+ and CP-Mirage+ can transform AIPs into zkSNARKs (with optionally committed witnesses). Thus, as we will see, we can build SNARKs for RAM consistency by building AIPs and then applying either Mirage+ (for volatile RAM) or CP-Mirage+ (for persistent RAM).

4. Volatile memory

In this section, we develop an improved AIP for *volatile* memory checking in a zkSNARK. First we define what an AIP for volatile memory is. Then, we explain our solution.

4.1. Problem and overview

An *access* acc is a tuple (a, v, w) with *address* $a \in \mathbb{F}$, *value* $v \in \mathbb{F}$, and *write* bit $w \in \{0, 1\} \subset \mathbb{F}$. A *transcript* tr is an access sequence $\mathsf{acc}_1, \ldots, \mathsf{acc}_A$. A memory *state* (of size $N \leq |\mathbb{F}|$) is a function $M : [N] \to \mathbb{F}$, where $M(a)$ is the value at address $a \in [N]$. (We will consider different representations of M, including as a vector in $\mathbb{F}^N$.) A transcript is valid with respect to initial state M_0 and final state M_A if:

$$\mathsf{valid}(\mathsf{tr}, M_0, M_A) \triangleq \exists M_1, \ldots, M_{A-1}, \forall i \in [A], \forall a \in [N], (M_{i-1}(a) = M_i(a)) \vee (w_i = 1 \wedge a = a_i \wedge M_i(a) = v_i)$$

This condition is called *read-over-write* (RoW), because it requires the values of future reads to agree with past writes.

Volatile memory captures the case where the initial state is zeroed and the final state is unspecified. (It is easy to generalize to various other structured initial states.) If $\mathbf{0}$ denotes the memory state that is zero everywhere, then the language of valid volatile transcripts is the set:

$$\mathcal{L}_{\mathsf{Vmem}} = \{\mathsf{tr} : \exists M_A, \mathsf{valid}(\mathsf{tr}, \mathbf{0}, M_A)\}$$

Now we explain our new AIP for $\mathcal{L}_{\mathsf{Vmem}}$, which is built on a *conditional uniqueness* proof. First, we explain the role of ordering proofs in volatile memory, and how to build a sufficient ordering proof from a conditional uniqueness proof (§4.2). Second, we design novel AIPs for uniqueness (§4.3) and conditional uniqueness (§4.4). Third, we compare the overall asymptotic cost of our volatile memory proof with those from prior work (§4.5).

4.2. Reducing to conditional uniqueness

Figure 2 illustrates the architecture of prior volatile memory proofs. The parties start with a t-order transcript tr. The high-level idea is to sort the transcript by address, and then check the address-order transcript (which is easier). First, $\mathcal{P}$ and $\mathcal{V}$ augment each access acc_i in the original transcript tr with a new "time" field $t_i = i$, for $i \in [A]$. Notice that tr is t-ordered. Then, $\mathcal{P}$ sends a new transcript tr' that should be lexicographically ordered by (a, t). Finally, $\mathcal{P}$ proves that tr' (1) is a permutation of tr, (2) is (a, t)-ordered and (3) respects RoW [43].

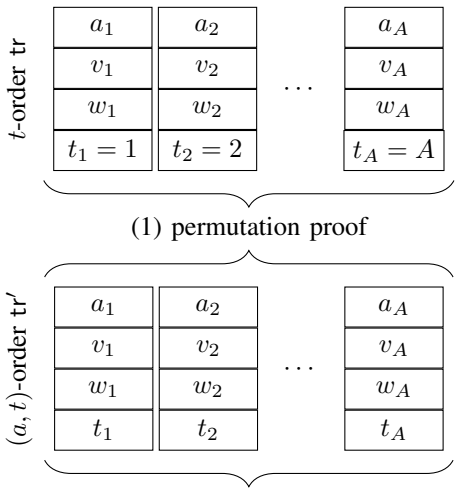


Figure 2: Prior work reduces a volatile memory proof to a (1) permutation proof, (2) ordering proof, and (3) read-over-write (RoW) proof for a transcript in (a, t)-order.

Prior work gives good permutation and RoW proofs for tr' [32, 43, 54]. In Appendix C, we present the state-of-the-art techniques for each of these, expressed as AIPs for suitable languages. Ultimately, the AIPs for both problems have perfect completeness, soundness error $O(A/|\mathbb{F}|)$, $\mathcal{V}$ complexity $O(A)$, and $\mathcal{P}$ complexity $\tilde{O}(A + \log N)$.

However, prior (a, t)-ordering proofs have some drawbacks. In particular, no prior (a, t)-ordering proof has $\mathcal{V}$ complexity $O(A)$—all have some dependence on N.

Our first insight is that (a, t)-ordering is not necessary for soundness (though it is sufficient). A sufficient and necessary condition is that tr' be a-grouped and t-ordered within those groups. That is, define g_i ("group start") to be 1 if $(i = 1 \vee a_{i-1} \neq a_i)$ or 0 otherwise. Then, it suffices for adjacent accesses to satisfy $g_i = 1 \vee t_i > t_{i-1}$ **and** for the sequence $(a_i : i \in [A], g_i = 1)$ to contain unique elements. Intuitively, this suffices because the address are independent of one another—so their order is irrelevant. We prove this sufficiency in Appendix C.1, by constructing and proving the soundness of Π_{Vmem}: an AIP for $\mathcal{L}_{\mathsf{Vmem}}$ which follows Figure 2 but only uses a-grouping.

So, it remains to build a conditional ordering and conditional uniqueness proof. The former is easy with existing tools (see App. C.1), but uniqueness requires new ideas.

4.3. Uniqueness proof

First, we consider the case when all $g_i = 1$: i.e., a proof that a sequence $\mathbf{a} = (a_1, \ldots, a_A) \in \mathbb{F}^A$ contains unique elements. That is, we give an AIP for $\mathcal{L}_{\mathsf{uniq}}$:

$$\mathcal{L}_{\mathsf{uniq}} = \{\mathbf{a} : \forall i \neq j, a_i \neq a_j\}$$

Naively enforcing pairwise disequalities has $\mathcal{V}$ complexity $\Theta(A^2)$. Our AIP will have $\mathcal{V}$ complexity $4A + O(1)$.

$\Pi_{\mathsf{uniq}}(a_1, \ldots, a_A)$

$\mathcal{P}(\cdots)$		$\mathcal{V}(\cdots)$
		$z(X) \triangleq \prod_i (X - a_i)$
		$z'(X) \triangleq \frac{d}{dX} z(X)$
$s, t \leftarrow \mathsf{UniqBez}(\mathbf{a})$	$s \in \mathbb{F}^{\leq A-2}[X]$, $t \in \mathbb{F}^{\leq A-1}[X]$ $\longrightarrow$	
		sample $\alpha \in \mathbb{F}$
		$z(\alpha)s(\alpha) + z'(\alpha)t(\alpha) \stackrel{?}{=} 1$

Protocol 1: Our uniqueness proof for the a_i. $\mathcal{V}$'s final test is implementable with $4A + O(1)$ constraints.

At a high-level, our protocol reduces the uniqueness of the a_i to a question about the multiplicity of a polynomial's roots. First, both $\mathcal{P}$ and $\mathcal{V}$ define the polynomial $z(X) = \prod_i (X - a_i)$. Notice that z's roots have multiplicity one iff the a_i are unique. Now, to show that the roots have multiplicity one, $\mathcal{P}$ can show that z' and z have no *common* roots. (Observe that if $z(X)$ contained a double root r, then it would have form $(X - r)^2 f(X)$ for some f, so $z'(X)$ would be $(X - r)(f'(X)(X - r) + 2f(X))$, which shares root r with z.) To show the absence of common roots, $\mathcal{P}$ uses Bezout's identity: $\mathcal{P}$ provides $s(X)$ and $t(X)$ such that $zs + z't = 1$ (Sec. 2).

Protocol 1 is our AIP. In it, $\mathcal{P}$ computes s, t using $\mathsf{UniqBez}$ and sends them to $\mathcal{V}$ as coefficients. Then, $\mathcal{V}$ samples $\alpha \in \mathbb{F}$ and tests $z(\alpha)s(\alpha) + z'(\alpha)t(\alpha) = 1$. $\mathcal{V}$ obtains $s(\alpha)$ and $t(\alpha)$ through Horner evaluation. For $z(\alpha)$ and $z'(\alpha)$ it uses the following recursion. For $j \in [A]$, define $z_j(X) = \prod_{i \leq j} (X - a_i)$. Then, $z_1(\alpha) = (\alpha - a_1)$, $z'_1(\alpha) = 1$, and for $j > 1$, we have:

$$z_j(\alpha) = z_{j-1}(\alpha) \times (\alpha - a_j)$$
$$z'_j(\alpha) = z_{j-1}(\alpha) + z'_{j-1}(\alpha) \times (\alpha - a_j)$$

Thus, $z(\alpha) = z_A(\alpha)$ and $z'(\alpha) = z'_A(\alpha)$ are computable with $2A + O(1)$ constraints. So, $\mathcal{V}$'s circuit has $4A + O(1)$ constraints in total.

Theorem 3. *The AIP Π_{uniq} for language $\mathcal{L}_{\mathsf{uniq}}$ has perfect completeness, soundness error $(2A - 2)/|\mathbb{F}|$, and $\mathcal{V}$ complexity $4A + O(1)$.*

Proof. Completeness follows immediately from Bézout's identity, and the $\mathcal{V}$ complexity has been analyzed above.

For soundness, consider the probability that $\mathcal{V}$ accepts if $\mathbf{a}$ contains duplicates. In this case, z contains at least one root with multiplicity at least two. Therefore, z and z' share this root. Therefore, $zs + z't$ cannot equal 1 (Sec. 2), so $zs + z't - 1$ is a non-zero polynomial (of degree $\leq 2A - 2$). Then, the $\mathcal{V}$ accepts (i.e., $z(\alpha)s(\alpha) + z'(\alpha)t(\alpha) = 1$) only when α is a root of said polynomial, which occurs with probability $\leq (2A - 2)/|\mathbb{F}|$, by the Schwatz-Zippel lemma [81–83]. □

$\mathsf{SimpleUniqBez}(a_1, \ldots, a_L) \to (s, t)$

$z(X) \leftarrow \mathsf{ProductTree}(X - a_1, \ldots, X - a_L)$
$\mathbf{b} \leftarrow \mathsf{Evaluate}(z'(X), \mathbf{a})$
$t(X) \leftarrow \mathsf{Interpolate}((b_1^{-1}, \ldots, b_L^{-1}), \mathbf{a})$
return $s \leftarrow (1 - z't)/z$ and t

Algorithm 1: A simple algorithm, SimpleUniqBez, for computing Bézout polynomials s, t for $z(X) = \prod_{i=1}^{L}(X - a_i)$ and its derivative. $\mathsf{ProductTree}(p_1, \ldots, p_L)$ computes $\prod_i p_i$, $\mathsf{Evaluate}(p, \mathbf{a})$ computes $p(a_1), \ldots, p(a_L)$, and $\mathsf{Interpolate}(\mathbf{y}, \mathbf{x})$ computes $f \in \mathbb{F}^{<L}[X]$ such that $f(x_i) = y_i$. The optimized UniqBez is in Appendix D.

Optimizing the prover. The prover requires coefficients for s and t. The natural way to get these is to compute coefficients for: z (using a product tree), then z', and then s and t (using the fast extended Euclidean algorithm—the FEEA) [84]. Computing each of z and s, t takes time $O(A \log^2 A)$,[2] but the FEEA is the concrete bottleneck, for two reasons. First, the FEEA's constants are large [84, Chapter 9]. Second, the FEEA is not divide-and-conquer, so parallelizing it is tricky and has higher overhead.

Instead, we give a simple, specialized, divide-and-conquer algorithm for computing the coefficients of s and t from $\mathbf{a}$. SimpleUniqBez (Alg. 1) is simplification of our algorithm. First, it computes z using a product tree. Second, it evaluates z' at each a_i. Third, it interpolates t as the polynomial whose evaluation at each a_i is $(z'(a_i))^{-1}$. Fourth, it computes s as $(1 - z't)/z$

To see that SimpleUniqBez is correct, consider $t(a_i)$. Per Bézout's identity, we have $z(a_i)s(a_i) + z'(a_i)t(a_i) = 1$. But, $z(a_i)$ is 0, so $t(a_i)$ is just the inverse of $z'(a_i)$. Thus, t can be interpolated from the $(z'(a_i))^{-1}$. Then, per Bézout's identity again, we have that z divides $1 - z't$ without remainder, and that the quotient is s.

SimpleUniqBez is efficient. The last step (computing s) takes $O(A \log A)$ time. The other steps are the bottleneck: all take $O(A \log^2 A)$ time. Also, they are all divide-and-conquer algorithms that are easy to parallelize [84].

We present the full UniqBez algorithm in Appendix D. It optimizes SimpleUniqBez by exploiting connections between the uses of ProductTree, Evaluate, and Interpolate. Specifically, in evaluating the third, one incidentally evaluates the first two. The use of ProductTree in Interpolate is unsurprising. However, the special role that evaluations of z' play in Interpolate is quite fortuitous. We prove the following theorem about UniqBez:

Theorem 4. *Given $a_1, \ldots, a_A \in \mathbb{F}$,* UniqBez *outputs $s, t \in \mathbb{F}[X]$ satisfying $sz + tz' = 1$ (where $z = \prod_i (X - a_i)$) in time at most $4.5A \log^2 A + O(A \log A)$.*[2]

[2]Throughout, we assume that $M_p(A)$ (abbreviated $M(A)$)—the number of $\mathbb{F}_p$ operations needed to multiply polynomials in $\mathbb{F}_p^{<A}[X]$—is $O(A \log A)$. In our fields of interest—which are smooth—the FFT gives this. In other fields, understanding $M_p(A)$ is an old (but still active) problem [85–87]. When giving a precise leading constant for algorithms that depend on $M(A)$, we assume $M(A) = A \log A$.

Reference	(a,t)-property	Overall $\mathcal{V}$ complexity
TinyRam [44]	(a,t)-order	$\Theta(A(\log A + \log N))$
Arya [32]	(a,t)-order	$\Theta_A\left(\frac{\log N}{\log\log N}\right), \Theta_N(A)$
Haböck [34]	(a,t)-order	$\Theta_A\left(\frac{\log N}{\log\log N}\right), \Theta_N(A)$
[55–57]	*other tools*	$\Theta(A + N)$
This work	a-group, t-order	$\Theta(A)$

Table 1: The $\mathcal{V}$ complexity of checking A accesses to a volatile RAM of size N. Prior works order accesses by address a using various techniques. Our approach *groups* accesses by a using a uniqueness proof.

Whereas, the FEEA requires time $22A \log^2 A + O(A \log A)$ in the worst case [84, Theorem 11.7].[2]

4.4. Conditional uniqueness

One can generalize Protocol 1 to be a *conditional* uniqueness proof. That is, a proof for the language:

$$\mathcal{L}_{\mathsf{c\text{-}uniq}} = \{(\mathbf{a} \in \mathbb{F}^A, \mathbf{g} \in \{0,1\}^A) : \\ \forall i \neq j, g_i = 0 \vee g_j = 0 \vee a_i \neq a_j\}$$

$\mathcal{P}$ simply omits a_i with $g_i = 0$ from the computation of z, z', s, t. Also, $\mathcal{V}$'s circuit uses a *conditional* recurrence:

$$z_j(\alpha) = z_{j-1}(\alpha) \times (g_j \times [\alpha - a_j - 1] + 1)$$
$$z'_j(\alpha) = g_j \times [z_{j-1}(\alpha) + z'_{j-1}(\alpha) \times (\alpha - a_j - 1)] + z'_{j-1}(\alpha)$$

Thus, z, z' are evaluable with a size $4A + O(1)$ circuit. The result is an AIP for $\mathcal{L}_{\mathsf{c\text{-}uniq}}$ with the same characteristics as Protocol 1, save that $\mathcal{V}$'s circuit now has size $6A + O(1)$.

4.5. Comparison to prior work

We have presented a novel AIP for conditional uniqueness with $\mathcal{V}$-cost $\Theta(A)$. As discussed, this gives an AIP for $\mathcal{L}_{\mathsf{Vmem}}$. Now, we compare our memory proof against prior works. As we will see, ours is the first which has cost independent of N: the address space size. In Table 1, we compare $\mathcal{V}$ complexity of the schemes. The approaches differ in (a) whether they group or order the addresses a and (b) how they do that.

TinyRAM [44] orders values through bit-splitting. Informally, $a_i \geq a_{i-1}$ reduces to $a_i - a_{i-1}$ being "small,"—fitting in $\lceil \log N \rceil$ bits. Ordering times requires $O(\log A)$ more bits, so each access requires $O(\log A + \log N)$ constraints.

To avoid this log overhead, Arya [32] range-checks though *lookup proofs*. Essentially, it argues that $a_i - a_{i-1}$ is in the set $\{0, \ldots, N\}$. Each lookup uses $O(1)$ constraints, but there is an $O(N)$ one-time overhead for the set, which is untenable for large N. One can defray this cost by combining lookups with bit-splitting [33, 35, 39, 41]. One splits the differences into medium-size "digits" that are each shown to be in a set $\{0, \ldots, k-1\}$. The net cost is $O(k + A \log_k(N))$ When k is set optimally, the asymptotics

of this approach become non-trivial. We analyze them in Appendix C, finding that the overhead now depends quasi-logarithmically on N (Table 1).[3] Recently, Haböck improved the concrete multiplicative costs of this approach [34].

Finally, other researchers [55–57] consider RAM proofs with per-cell initialization costs, for total cost $\Theta(A+N)$. These constructions perform poorly when N is large.

Our AIP has $\mathcal{V}$ cost independent of N. $\mathcal{V}$ cost is important because when the AIP is passed to Mirage+, this metric determines the cost of the concrete bottlenecks in zkSNARK proving: the MSMs and FFTs (Sec. 3.2). One limitation of our AIP is that asymptotically its $\mathcal{P}$ cost is $O(A \log^2 A)$, due to UniqBez. However, since AIP $\mathcal{V}$ complexity is the concrete bottleneck in zkSNARK proving, our scheme's concrete performance is still good. In Section 9, our experiments shows that it reduces concrete zkSNARK proving time by a much as 32.9%, for 60-bit addresses. This is good, but our persistent and sparse-persistent RAM schemes will yield far larger improvements (Sections 5, 6).

5. Persistent memory

Now we turn to *persistent memory*. Here, the problem is to show that a transcript tr is consistent with specified initial and final memory states. Thus, to build an AIP for:

$$\mathcal{L}_{\mathsf{Pmem}} = \{(\mathsf{tr}, M_0, M_A) : \mathsf{valid}(\mathsf{tr}, M_0, M_A)\}$$

Through CP-Mirage+ (with commitment scheme Com), we can transform an AIP for $\mathcal{L}_{\mathsf{Pmem}}$ into a zkSNARK for a witness relation in which M_0 and M_A are represented as Pedersen commitments c, c' with randomness o, o':

$$\begin{aligned} R_{\mathsf{Pmem}} = \{((\mathsf{tr}, c, c'), (M_0, M_A, o, o')) : \\ c = \mathsf{Com}(M_0, o) \wedge c' = \mathsf{Com}(M_A, o') \\ \wedge \mathsf{valid}(\mathsf{tr}, M_0, M_A)\} \end{aligned}$$

In our AIP for $\mathcal{L}_{\mathsf{Pmem}}$, the initial and final memory states are represented as vectors $\mathbf{m}, \mathbf{m}' \in \mathbb{F}^N$. That is, address $i \in [N]$ has initial value m_i and final value m'_i.

Figure 3 illustrates our high-level approach.[4] First, $\mathcal{P}$ sends a vector of "active" addresses $\mathbf{a}$—those that are touched by tr. They also send the initial and final values at those addresses $\mathbf{v}, \mathbf{v}' \in \mathbb{F}^A$. Then, $\mathcal{P}$ and $\mathcal{V}$ run two sub-protocols, which we explain in the rest of this section. First, to show that $(\mathbf{v}, \mathbf{v}', \mathbf{a})$ represent the active cells of $\mathbf{m}$ and $\mathbf{m}'$ (§5.1). Second, to show that $(\mathbf{v}, \mathbf{v}', \mathbf{a})$ is consistent with the transcript tr (§5.2). We conclude with a comparison to prior work (§5.3).

5.1. Selecting active cells

Our first sub-protocol proves the connection between $\mathbf{v}, \mathbf{v}', \mathbf{a}, \mathbf{m}$, and $\mathbf{m}'$. Each (a_i, v_i) pair should match some

[3] Recall that $O_A(f(N))$ allows the big-O constant to depend on A.
[4] Assume that $A \leq N$; lifting this requirement is straightforward.

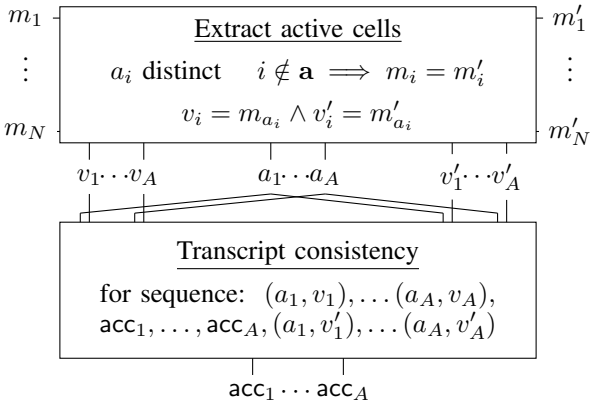


Figure 3: An overview of our persistent memory proof. $\mathbf{m}, \mathbf{m}'$ are the initial and final memory states. $\mathbf{a}$ includes all addresses accessed by $\mathsf{acc}_1, \ldots, \mathsf{acc}_A$. For all $i \in [A]$, v_i, v'_i are the initial and final values at a_i.

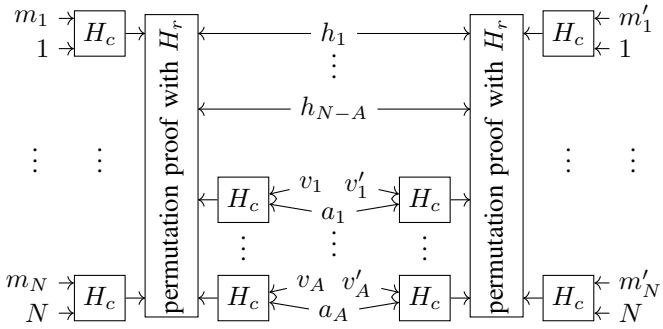


Figure 4: A visualization of Π_{active} (Protocol 2).

unique (i, m_i), and likewise for the a_i, v'_i, and m'_i. Further, every unmatched m_i should equal m'_i. More precisely:

$$\begin{aligned} \mathcal{L}_{\mathsf{active}} = \{(\mathbf{m}, \mathbf{m}' \in \mathbb{F}^N; \mathbf{v}, \mathbf{v}', \mathbf{a} \in \mathbb{F}^A) : \\ \textstyle\bigwedge_{i=1}^{A} (v_i = m_{a_i} \wedge v'_i = m'_{a_i}) \wedge \\ \textstyle\bigwedge_{i=1}^{N} (i \notin \mathbf{a} \implies m_i = m'_i) \wedge \\ \textstyle\bigwedge_{i=1}^{A} \bigwedge_{j=1}^{i-1} a_i \neq a_j\} \end{aligned}$$

Figure 4 illustrates our protocol Π_{active} for this language. The idea is to sample a key β and represent every value-address pair (v, a) by its hash $H_c(\beta, (v, a)) = v + \beta a$ (represented by the boxes with H_c). $\mathcal{P}$ sends the hashes $\mathbf{h} = (h_1, \ldots, h_{N-A})$ for *inactive* memory cells (top-center). Then, it suffices to show that $(\mathbf{h}, v_1 + \beta a_1, \ldots, v_A + \beta a_A)$ is a permutation of the initial memory state and $(\mathbf{h}, v'_1 + \beta a_1, \ldots, v'_A + \beta a_A)$ is a permutation of the final memory state. For this, we use the H_r hash to test multiset equality (the two vertical rectangles). Protocol 2 defines Π_{active} in full. We prove the following in Appendix A:

Theorem 5. *If $A \leq N$, the protocol Π_{active} (for language $\mathcal{L}_{\mathsf{active}}$) has perfect completeness, soundness error $\leq (N^2 + 2N + A)/|\mathbb{F}|$, and $\mathcal{V}$ complexity $3N + 2A + O(1)$.*

$\Pi_{\mathsf{active}}(\mathbf{m}, \mathbf{m}' \in \mathbb{F}^N; \mathbf{v}, \mathbf{v}', \mathbf{a} \in \mathbb{F}^A)$ for $\mathcal{L}_{\mathsf{active}}$

$\mathcal{P}(\cdots)$		$\mathcal{V}(\cdots)$
	β ←	sample $\beta \in \mathbb{F}$
$\mathbf{h} \leftarrow (m_i + i\beta)_{i \in [N], i \notin \mathbf{a}}$	$\mathbf{h} \in \mathbb{F}^{N-A}$ →	sample $\alpha \in \mathbb{F}$
		$c \leftarrow \prod_{i=1}^{N-A}(\alpha + h_i)$
		$\prod_{i=1}^{N}(\alpha + m_i + i\beta) \stackrel{?}{=} c \times \prod_{i=1}^{A}(\alpha + v_i + a_i\beta)$
		$\prod_{i=1}^{N}(\alpha + m'_i + i\beta) \stackrel{?}{=} c \times \prod_{i=1}^{A}(\alpha + v'_i + a_i\beta)$

Protocol 2: Extracting the active cells of a memory.

Why send h?. In Π_{active}, $\mathcal{P}$ sends the hashes h_i of inactive memory cells rather than their address-value pairs. In an alternative protocol for $\mathcal{L}_{\mathsf{active}}$, Π'_{active}, $\mathcal{P}$ sends these pairs, and $\mathcal{V}$ computes the h_i. This change increases $\mathcal{V}$ complexity by $N - A$, which is why we use Π_{active} instead of Π'_{active}. But, Π'_{active} has two advantages that might matter in other contexts. First, it has one fewer round, since the keys α and β can now be sampled simultaneously. Second, the soundness error of Π'_{active} is $N/|\mathbb{F}|$, which is better than our $O(N^2/|\mathbb{F}|)$ bound for Π_{active}. These advantages are not meaningful for us: CP-Mirage+ makes extra rounds very cheap, and its soundness error far exceeds that of Π_{active}.

5.2. Transcript validity

It remains to be shown that the transcript

$$(a_1, v_1), \ldots, (a_A, v_A), \mathsf{acc}_1, \ldots, \mathsf{acc}_A, (a_1, v'_1), \ldots, (a_A, v'_A)$$

is consistent. We give our full proof for this, Π_{Pmem}, in Appendix C.3. Here, we sketch our approach by describing how it differs from our volatile memory proof. First, we ensure that the initialization accesses (a_i, v_i) come first, and the finalization accesses (a_i, v'_i) come last. Second, we eliminate the uniqueness argument from the last section.

Initialization and finalization. To ensure that each a_i has initial value v_i and final value v'_i, it suffices to ensure that each address is initially written to by access (a_i, v_i) and finally read from access (a_i, v'_i). We ensure this by setting the time t associated with each access appropriately. As before, for $i \in [A]$, standard access acc_i is assigned time $t_i = i$. However, each initialization access (a_i, v_i) is assigned time $t_i = 0$ and finalization access (a_i, v'_i) is assigned time $t_i = A+1$. These times are respectively before and after all standard accesses.

Omitting uniqueness. Further, since the (a_i, v_i) must be the first accesses to each address, we set their "group start" (Sec. 4.2) g to be 1 and we set $g = 0$ for every other access. Now, the addresses with $g = 1$ are unique by construction. Thus, we can omit the last section's conditional uniqueness argument. Note that with this change, we set the g_i in the t-order transcript, and they are included in the permutation proof. Whereas, for volatile memory we computed the g's based on the a's in the (a, t)-order transcript.

Approach	Constraints	Other bottlenecks
Merkle	$600 \cdot A \log N$	
Linear scan	$3 \cdot N \cdot A$	
RSA [30]	$10\text{M} + 4\text{k} \cdot A$	$2048N$-bit RSA exp.
This work	$3 \cdot N + 41 \cdot A$	$2 \times \text{MSM}_{\mathbb{G}_1}(N)$

Table 2: Constraint counts for prior persistent memory approaches and ours. Merkle and RSA estimates are based on Ozdemir et al. [30], who require: ≈5M constraints for a Wesolowski [88] verifier, ≈2k constraints per RSA accumulator insert/delete, and ≈300 constraints per CR hash (Poseidon [89]). We compute the N-dependence of our approach directly, and the A-dependence through a regression on the results of Section 9.2.

5.3. Comparison to prior work

Prior persistent memory proofs for zkSNARKs use Merkle trees or RSA accumulators [30]. Table 2 summarizes the costs of these approaches and of ours.

In the Merkle approach, $\mathcal{V}$ checks one collision-resistant (CR) hash per read and per level of a $(\log N)$-depth tree. For a write, this is two hashes. Keyless hash functions are highly non-linear, so even multiplication-optimized hashes (e.g., Poseidon [89]), require ≈300 multiplications.

RSA accumulators give a very different persistent memory proof [30]. The $\mathcal{V}$ cost is independent of N, but their are three downsides. First, there is a large constant overhead (about 10M multiplications [30]) for verifying two Wesolowski proofs [88]. Second, each access requires many multiplications (about 2k [30]), to compute and accumulate a division-intractable hash [61]. Third, while the $\mathcal{V}$ complexity is independent of N, proving cost is not. $\mathcal{P}$ must additionally evaluate a $2048N$-bit exponentiation modulo the RSA modulus (which is concretely expensive) in order to compute its witness.

Our approach has $\mathcal{V}$ complexity $3N + 41A$. The N constant comes directly from Π_{active} and we measure the A constant in our experiments (Sec 9.2). Our experiments also see that this gives a significant concrete improvement for many N and A. It reduces constraint counts by up to 98.2× and overall proving times by up to 51.3×. Intuitively, the improvement is because both RSA and Merkle encode expensive cryptographic operations (e.g., Wesolowski's proof or CR hashing) in constraints, while we do not.

6. Sparse persistent memory

Now, we turn to our final problem: *sparse* persistent memory. The previous section considered a *dense* memory: each address $i \in [N]$ had its value explicitly represented by m_i in the memory state $\mathbf{m}$. Now, the address space will be nearly all of $\mathbb{F}$, but we will assume that at most C cells are non-zero. Thus, C is the *capacity* for a sparse memory.

The initial state is represented by $\mathbf{m}, \mathbf{a} \in \mathbb{F}^C$. Each non-zero a_i indicates that address a_i has value m_i. Thus, $0 \in \mathbb{F}$ is the only invalid address, and when $a_i = 0$, we require that

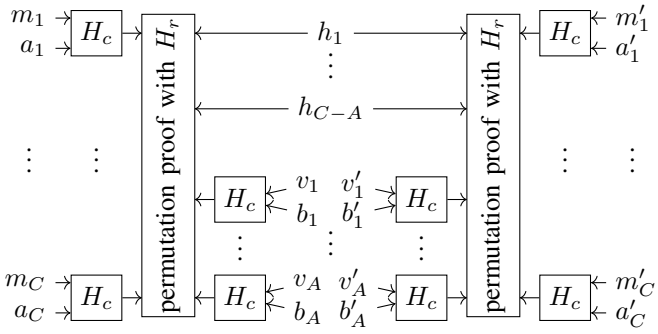


Figure 5: A visualization of our active-cell selection proof for sparse persistent memories. Two tests are not shown. First, for all $i \in [A]$, b_i must be zero, or $b_i = b'_i$; this is enforced as: $(b_i - b'_i)b_i = 0$. Second, the non-zero a'_i must be unique; we discuss how to ensure this in Section 6.2.

$m_i = 0$. Further, the non-zero a_i must be unique. The final state is represented by $\mathbf{m}', \mathbf{a}' \in \mathbb{F}^C$ with the same meaning.

First (§6.1), we summarize our approach and explain how constructing a sparse persistent memory proof reduces to proving the uniqueness of valid addresses. Second (§6.2), we explain how to prove the uniqueness of valid addresses. Third (§6.3), we compare against alternatives.

6.1. Overview

Our high-level protocol for sparse persistent memory is similar to the one for dense persistent memory. In the first round, $\mathcal{P}$ sends encodings of the initial and final states of *active* cells: those whose values might change. That is, $\mathcal{P}$ sends $\mathbf{b}, \mathbf{v}, \mathbf{b}', \mathbf{v}' \in \mathbb{F}^A$. The pair (b_i, v_i) represents a cell at address b_i with value v_i, *before* the accesses acc_i. The pair (b'_i, v'_i) represents a cell at address b'_i with value v'_i, *after* the accesses. We call a cell *valid* if its address is non-zero.

$\mathcal{P}$ must show a number of facts about the active cells. Figure 5 illustrates its proof, at a high-level. This proof follows Π_{active} very closely, so we will only discuss the differences. First, in Π_{active} value m_i lived at address i; now, it lives at address a_i (and likewise for m'_i, a'_i). This creates the main challenge of this section—proving the uniqueness of the valid a'_i—which we will address in Section 6.2.

Second, in Π_{active} the initial address b_i and final address b'_i of an active cell were always the same. Now, b'_i need only equal b_i if b_i is valid. If b_i is invalid, then b'_i *can* be valid: this corresponds to creating a new cell in the memory that is potentially non-zero. $\mathcal{V}$ enforces the conditional equality of b_i and b'_i with the equation $(b'_i - b_i)b_i = 0$.

Beyond active-cell selection, $\mathcal{P}$ must show that the active cells are consistent with the transcript. This amounts to proving the consistency of the following transcript: $(b'_1, v_1), \ldots, (b'_A, v_A), \mathsf{acc}_1, \ldots, \mathsf{acc}_A, (b'_1, v'_1), \ldots, (b'_A, v'_A)$. Note that the initialization accesses have addresses $\mathbf{b}'$, not $\mathbf{b}$. This is because $\mathbf{b}'$ contains both the pre-existing and newly created valid addresses, while $\mathbf{b}$ contains only the pre-existing ones. This consistency proof is done exactly as in Section 5.2.

$\Pi_{\mathsf{PuncUniq}}(a_1, \ldots, a_C)$ for $\mathcal{L}_{\mathsf{PuncUniq}}$

$\mathcal{P}(\cdots)$	$\mathcal{V}(\cdots)$
$S \leftarrow \{a_i\}_{i=1}^{C}$	$z(X) \triangleq \prod_i (X - a_i)$
$c \leftarrow C - \lvert S \rvert$	$z'(X) \triangleq \frac{d}{dX} z(X)$
$s, t \leftarrow \mathsf{UniqBez}(S)$	
$\xrightarrow{c, s, t}$	
	sample $\alpha \in \mathbb{F}$
	$z(\alpha)s(\alpha) + z'(\alpha)t(\alpha) \stackrel{?}{=} \alpha^c$

Protocol 3: Our punctured uniqueness proof for the a_i. $\mathcal{P}$'s messages are $c \leq C, s \in \mathbb{F}^{\leq C-2}[X], t \in \mathbb{F}^{\leq C-1}[X]$. $\mathcal{V}$'s test is as a size $4C + O(\log C)$ circuit; see the text.

6.2. Valid address uniqueness

How can $\mathcal{P}$ show the uniqueness of the (final) valid addresses? Naively, $\mathcal{P}$ could apply the proof Π_{uniq} (Sec. 4.3) to the a'_i. However, this approach is incorrect (incomplete) because it does not permit repetitions of the dummy address 0. Instead, $\mathcal{P}$ could provide an indicator sequence $\mathbf{c} \in \{0,1\}^C \subset \mathbb{F}^C$, where each $c_i = \mathsf{to}_{\mathbb{F}}(a'_i \neq 0)$. $\mathcal{P}$ would show that the c_i satisfy their definition, and then apply a *conditional* uniqueness argument to $\mathbf{a}'$ and $\mathbf{c}$ (Sec. 4.4). This approach is complete and sound. However, it's $\mathcal{V}$ complexity is $8C + O(1)$ (this is because the conditional uniqueness argument has cost $\approx 6C + O(1)$ and proving that the c_i satisfy their definition has cost $2C$ [90]).

In the sections below we give two other approaches that improve $\mathcal{V}$ complexity. First (§6.2.1), we give a protocol based on *punctured uniqueness*, with $\mathcal{V}$ complexity $4C + O(\log C)$. Second (§6.2.2), we give a protocol based on *punctured disjointness*, with $\mathcal{V}$ complexity $2C + O(A + \log C)$.

6.2.1. Punctured uniqueness. To inspire our punctured uniqueness proof, consider what goes wrong if we apply our (non-conditional) uniqueness proof (Sec. 4.3) to the a_i. Fist, $\mathcal{P}$ defines $z(X) = \prod_i (X - a_i)$ and computes the derivative z'. Now, $\mathcal{P}$ must show that $\gcd(z, z') = 1$ by providing s, t such that $zs + z't = 1$. But, if $\mathbf{a}$ contains $d \geq 2$ instances of 0, then X^{d-1} is a common factor of z and z'. So, $\mathcal{P}$ cannot complete the protocol.

Our idea is simple: have $\mathcal{P}$ show that $\gcd(z, z')$ has form X^c. This shows that 0 is the only common root of z and z', and is thus is the only duplicate root of z. Our proof is for the language $\mathcal{L}_{\mathsf{PuncUniq}}$, defined as follows:

$$\mathcal{L}_{\mathsf{PuncUniq}} = \{\mathbf{a} : \forall i \neq j,\ a_i \neq a_j \vee 0 = a_i = a_j\}$$

Protocol 3 shows Π_{PuncUniq}. First, $\mathcal{P}$ sends c: the number of *extra* 0s. $\mathcal{P}$ also sends Bézout coefficients s, t for the deduplicated list. $\mathcal{V}$ tests $zs + z't = X^c$ at a random α, evaluating $z(\alpha), s(\alpha), z'(\alpha), t(\alpha)$ as in Π_{uniq} (using $4C + O(1)$ multiplications) and α^c using $O(\log C)$ multiplications.

Theorem 6. *The protocol Π_{PuncUniq} (for language $\mathcal{L}_{\mathsf{PuncUniq}}$) has perfect completeness, soundness error $\leq$*

$\Pi_{\mathsf{PuncDisj}}(a_1, \ldots, a_C, d_1, \ldots, d_A)$ for $\mathcal{L}_{\mathsf{PuncDisj}}$

$\mathcal{P}(\cdots)$		$\mathcal{V}(\cdots)$
$c \leftarrow \left\lvert \{\!\{a_i\}\!\}_{i=1}^{C} \cap \{\!\{d_i\}\!\}_{i=1}^{A} \right\rvert$		$f(X) \triangleq \prod_{i=1}^{C}(X - a_i)$
$s, t \leftarrow \mathsf{DisjBez}(\mathbf{a}, \mathbf{d})$	$\xrightarrow{c, s, t}$	$g(X) \triangleq \prod_{i=1}^{A}(X - d_i)$
		sample $\alpha \in \mathbb{F}$
		$f(\alpha)s(\alpha) + g(\alpha)t(\alpha) \stackrel{?}{=} \alpha^c$

Protocol 4: Our punctured disjointness proof for the a_i and d_i. $\mathcal{P}$'s messages are $c \leq C, s \in \mathbb{F}^{\leq A-1}[X], t \in \mathbb{F}^{\leq N-1}[X]$. $\mathcal{V}$'s test is a size $2C + 2A + O(\log C)$ circuit; see the text.

$(C + A - 1)/|\mathbb{F}|$, *and* $\mathcal{V}$ *complexity* $4C + O(\log C)$. *(proved in Appendix A)*

This is already better than a conditional uniqueness proof. But, we can do even better by proving the uniqueness of the valid a'_i *assuming* the uniqueness of the valid a_i.

6.2.2. Punctured disjointness. Our first insight is that individual memory proofs do not need to prove valid address uniqueness from scratch. A persistent memory proof is used to allow a chain of zkSNARKs $\pi_1, \ldots, \pi_P$ to be provably consistent with a single RAM that is represented between proofs by a commitment. Thus, in each proof, the initial a_i come from a previous proof. So, if each proof assumes the uniqueness of the valid a_i and proves the uniqueness of the valid a'_i, induction shows that every proof produces a sequence a'_i whose valid addresses are unique.

Our second insight is that under these assumptions, *uniqueness* reduces to *disjointness*. First, define differences $d_i = b'_i - b_i$. Now, observe that the non-zero d_i are exactly the valid a'_i that are not in $\mathbf{a}$. So, to prove that the valid a'_i are unique, it suffices to assume the uniqueness of the valid a_i and show that they are disjoint with the non-zero d_i. That is, it suffices for $(\mathbf{a}, \mathbf{d})$ to be in the following language:

$$\mathcal{L}_{\mathsf{PuncDisj}} = \{(\mathbf{a} \in \mathbb{F}^C, \mathbf{d} \in \mathbb{F}^A) : \{a_i\}_{i=1}^{C} \cap \{d_i\}_{i=1}^{A} \subseteq \{0\}\}$$

Our third insight is that Bézout's identity gives a proof for $\mathcal{L}_{\mathsf{PuncDisj}}$. Π_{PuncDisj} (Protocol 4) is that proof. In it, $\mathcal{V}$ defines $f(X)$ to have roots $\mathbf{a}$ and $g(X)$ to have roots $\mathbf{d}$. Then, $\mathcal{P}$ provides $c \leq C, s \in \mathbb{F}^{<A}[X], t \in \mathbb{F}^{<C}[X]$ such that $fs + gt = X^c$. This polynomial equality can only hold if $\{a_i\} \cap \{d_i\} \subseteq \{0\}$. $\mathcal{V}$ tests it at a random α. $\mathcal{P}$ could compute s, t using the FEEA; but there is a natural adaptation of SimpleUniqBez, that we call DisjBez, that does the same thing using divide-and-conquer subroutines.

Theorem 7. *The protocol* Π_{PuncDisj} *(for* $\mathcal{L}_{\mathsf{PuncDisj}}$*) has perfect completeness, soundness error* $\leq (C + A - 1)/|\mathbb{F}|$*, and* $\mathcal{V}$ *complexity* $2C + 2A + O(\log C)$*. (proof in App. A)*

6.3. Comparison to alternatives

Now, we explain alternatives to our approach and present cost models for all options. Table 3 shows the cost models.

Approach	Constraints	Other bottlenecks
Merkle	$150\text{k} \cdot A$	
Linear scan	$3 \cdot C \cdot A$	
RSA [30]	$30\text{M} + 8\text{k} \cdot A$	$2048C$-bit RSA exps.
This work	$7 \cdot C + 51 \cdot A$	$2 \times \text{MSM}_{\mathbb{G}_1}(2C)$

Table 3: Constraint counts for prior persistent memory approaches and ours, under the same assumptions as Table 2. We explain our estimates for Merkle and RSA in the text.

We summarise our conclusions here, and explain the alternative approaches and cost models below.

For moderate C, our approach is better than one based on RSA or Merkle. The reason is just as in Section 5.3: we avoid collision-resistant hashing, division-intractable hashing, Wesolowski proofs, and multi-precision arithmetic. However, for very large C (say $C > 2^{25}$), Merkle seems best, because only its costs are C-independent.

Our approach. We can determine the C-dependence of our $\mathcal{V}$ complexity precisely. We start from the $3C$ cost of Π_{active}. Then, the H_c evaluations in Figure 5 add one multiplication each, and the Π_{PuncDisj} adds another $2C$. Thus, our final $\mathcal{V}$ complexity is $7C + O(\log C + A)$. The A-dependence is very similar to a dense RAM; we conservatively estimate it increases by 10. Finally, to handle address commitment, the extra MSMs of CP-Mirage+ grow slightly,

Merkle. *Sparse merkle trees* give a natural (but expensive) construction of sparse persistent RAM: a Merkle tree with $N = |\mathbb{F}| - 1$ leaves. Here, the $\log N$ factor of Table 2 is now $\approx$256, so $\mathcal{V}$ complexity is approximately 150k per access.

RSA. One could also build a sparse persistent RAM from RSA accumulators, by combining RSA-based RAM with batched RSA non-membership proofs [30, 61]. The approach would use two RSA accumulators: one for (m_i, a_i) pairs and the other for just the addresses a_i. RSA's constant overhead increases by 10M relative to Table 2 just to update the new address accumulator (two more Wesolowski proofs). Further, addresses are proved unique with batched non-membership proofs, which add another 10M constraints (this is 4 256-bit exponentiations, at $\approx$2.5M constraints each).

The additional (batched) accumulator operations also increase per-access costs. These costs should approximately double because their bottleneck (a multi-precision modular product of division intractable hashes) is done twice: one for (m_i, a_i) pairs and once for the a_i on their own.

7. Mirage+: A zkSNARK for AIPs

Now, we present Mirage+: our zkSNARK for AIPs expressed as I-R1CS. We first give I-Mirage+: a zkSARK (zero-knowledge succinct *interactive* argument of knowledge) for I-R1CS. It comprises a non-interactive Setup and two interactive algorithms $\mathcal{P}$ and $\mathcal{V}$ (Protocol 5).[5]

[5] $\mathcal{P}$ and $\mathcal{V}$ evaluate expressions of form gG_j (where G_j is a group generator and g is a polynomial in variables $\alpha, \beta, \ldots$) as *multi-scalar multiplications*. That is, they find $c_i \in \mathbb{F}$ such that $g = \sum_i c_i m_i$ (for monomials m_i in $\alpha, \beta, \ldots$) and evaluate $\sum_i c_i(m_i G_j)$, where each $m_i G_j$ is in pk or vk.

$\underline{\mathsf{Setup}(\phi) \to (\mathsf{pk}, \mathsf{vk})}$:

sample $\alpha, \beta, \gamma, \delta_1, \ldots, \delta_\mu, \tau \in \mathbb{F} \setminus \{0\}$

for $M \in \{A, B, C\}$, $i \in [m]$:

$f_{M,i}(X) \leftarrow \mathsf{Interp}(\omega, (M_{j,i})_{j=1}^n)$

for $i \in [m]$: $f_i(X) \leftarrow \beta f_{A,i}(X) + \alpha f_{B,i}(X) + f_{C,i}(X)$

$Y \leftarrow [\ell_x + \sum_{i=1}^{\mu} \ell_{r_i}]$

$z(X) \leftarrow \prod_{i=1}^{n}(X - \omega^i)$

for $i \in [\mu]$: $W_i \leftarrow \{k + |X| + \sum_{j<i} \ell_{w_j}\}_{k \in [\ell_{w_i}]}$

pk contains:

$\left(\alpha, \beta, \delta_1, \ldots, \delta_\mu, (\tau^i)_{i=0}^{n-1}, (\delta_\mu^{-1} \tau^i z(\tau))_{i=0}^{n-2}\right) \cdot G_1$

$(\delta_j^{-1} f_i(\tau))_{j \in [\mu], i \in W_j} \cdot G_1$; $\quad (\beta, \delta_\mu, (\tau^i)_{i=0}^{n-1}) \cdot G_2$

vk contains:

$(\alpha, (\gamma^{-1} f_i(\tau))_{i \in X}) \cdot G_1$; $\quad (\beta, \gamma, \delta_1, \ldots, \delta_\mu) \cdot G_2$

$\underline{\mathcal{P}(\mathsf{pk}, x, w)}$:

compute $f_i, f_{M,i}, z$ as in Setup.

for any $I \subseteq [m]$, define $f'_I(X) \triangleq \sum_{i \in I} z_i f_i(X)$

sample $\rho, \sigma, \kappa_1, \ldots, \kappa_{\mu-1} \in \mathbb{F}$

for round i in $[\mu - 1]$:

compute w_i from $(x, w, r_1, \ldots, r_{i-1})$

send to $\mathcal{V}$: $P_{C,i} \leftarrow (\delta_\mu \kappa_i + \delta_i^{-1} f'_{W_i}(\tau)) G_1$

receive r_i

compute w_μ and $z \leftarrow (x, r_1, \ldots, r_{\mu-1}, w_1, \ldots, w_\mu)$

for $M \in \{A, B, C\}$: $f''_M(X) \leftarrow \sum_i z_i f_{M,i}(X)$

$h(X) \leftarrow (f''_A \cdot f''_B - f''_C)/z$

define $\beta' \triangleq \beta + \sigma\delta_\mu + f''_B(\tau)$

send to $\mathcal{V}$:

$P_A \leftarrow (\alpha + \rho\delta_\mu + f''_A(\tau)) G_1$

$P_B \leftarrow \beta' G_2$

$P_{C,\mu} \leftarrow (\delta_\mu^{-1}(hz + f'_{W_\mu})(\tau) + \rho\beta' - \rho\sigma\delta) G_1$

$\quad -(\sum_{i=1}^{\mu-1} \kappa_i \delta_i) G_1 + \sigma P_A$

$\underline{\mathcal{V}(\mathsf{vk}, x)}$:

for round i in $[\mu - 1]$:

receive from $\mathcal{P}$: $P_{C,i}$; sample and send to $\mathcal{P}$: $r_i \in \mathbb{F}$

receive from $\mathcal{P}$: $P_A, P_B, P_{C,\mu}$

define f'_I (for $I \subseteq [m]$) as in $\mathcal{P}$

assert: $e(P_A, P_B) = e(\alpha G_1, \beta G_2)$

$\quad + e(\gamma^{-1} f'_Y(\tau) G_1, \gamma G_2) + \sum_{i=1}^{\mu} e(P_{C,i}, \delta_i G_2)$

Protocol 5: I-Mirage+: our zkSARK for I-R1CS. It is a slight generalization of Mirage, itself a generalization of Groth16. Differences from Groth16 are in blue. Let ω have order n and define $\mathsf{Interp}(\omega, \mathbf{v} \in \mathbb{F}^n)$ as the unique $f \in \mathbb{F}^{<n}[X]$ satisfying $f(\omega^i) = v_i$.

I-Mirage+ generalizes Mirage [48] and Groth16 [3]. We highlight the differences between I-Mirage+ and Groth16 in blue. We first replace Groth's P_C (which encodes the full witness) with $P_{C,1}, \ldots, P_{C,\mu}$ (which respectively encode messages $w_1, \ldots, w_\mu$). Second, for zero-knowledge, each $P_{C,i}$ must include a fresh blind κ_i, for $i < \mu$. Third, for completeness, we tweak $P_{C,\mu}$ (adding $-\sum_{i=1}^{\mu-1} \kappa_i \delta_i G_1$) to cancels these blinds in the verification equation. If μ is 1 or 2, then I-Mirage+ is equivalent to Groth's construction or Mirage, respectively. Theorem 8 states the security of I-Mirage+; the proof is in Appendix A.4.

Theorem 8. *Let I-R1CS ϕ be complete and **knowledge-sound** for witness relation R. Then in the generic group model (GGM), I-Mirage+ is a zkSARK for R.*

Non-interactivity. For constant μ, I-Mirage+ is a constant-round public-coin interactive argument. Thus, the Fiat-Shamir transform [91] (replacing $\mathcal{V}$'s messages with a hash of their view) gives a zkSNARK in the random oracle model [92]. Since a random oracle can be simulated in the GGM, this proves Theorem 1, from Section 3.2. We call the result of this transform **Mirage+**.

8. Implementation

Cryptosystem. We implement CP-Mirage+, starting from the `bellman` [93] Rust implementation of Groth's zk-SNARK [3]. Our patch (2.4k LOC and 1.3k LOC for tests) includes the linear subspace proof of Kiltz and Wee [94], the commitment linking proof of Campanelli, Fiore, and Querol [47], a generalization of `bellman`'s R1CS interface to I-R1CS, and CP-Mirage+ itself.

Compilation and RAM proofs. We implement our volatile and persistent RAM proofs as extensions to the CirC compiler infrastructure [95]. This is a non-trivial extension. First, we generalize CirC's Intermediate Representation (IR), to represent not just circuits (NP verifiers), but also public-coin IPs. Second, we adapt CirC's R1CS backend, to compile these IPs to I-R1CS. These changes are about 4k LOC, Rust.

CirC's Intermediate Representation (IR) can already express programs that manipulate memory. We implement a compiler pass that replaces IR memory operations with uses of our RAM proofs. This compiler pass essentially generates CirC IR that encodes our proofs. We also implement the proofs from TinyRAM [45] and Arya [32], for comparison. In sum, these changes are about 2k LOC.

Programming Interface. Implementing our proofs within CirC allows us to make their benefits very accessible to programmers. We do this my modifying one of CirC's input languages and one of its compiler driver binaries.

First, we add new keywords to CirC's Z# input language that trigger our new compiler pass. Figure 6 shows an example program that uses persistent memory via this syntax. The memory is the field array `mem`, which is an input to the program and is annotated as `committed`. Standard array access syntax is used for both reads and writes. The compiled CP-zkSNARK will include commitments to the initial and final states of this array. Memory accesses must be in-bounds, or the proof will fail. Volatile memory is available via a similar syntax, but volatile memories must have local scope and are annotated with `transcript`.

Second, we extend the CirC executable for compiling and using Z# programs. It already had subcommands for compiling a Z# instance to (I-)R1CS, as well as zkSNARK setup, proving, and verifying. We add subcommands for the new CP-zkSNARK operations (commitment setup, creation, and opening) and extend the existing subcommands to work

```
def main(committed field[8096] mem,
         private field idx) -> bool:
  field v = mem[idx]
  for field i in 0..10 do
    v = v * v
  endfor
  mem[idx] = v
  return true
```

Figure 6: A program with persistent memory. The memory is a field array declared with the `committed` keyword. This program reads an element, squares it 10 times, and writes it back. We give a more meaningful example in Section 9.4.

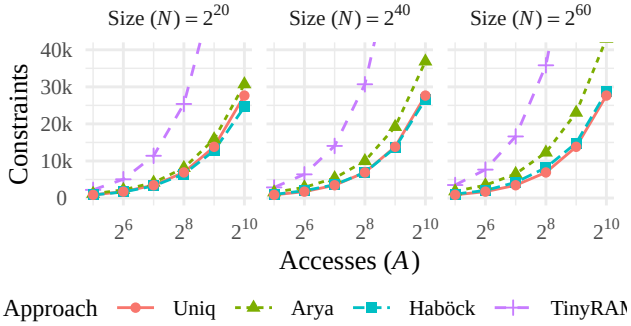


Figure 7: Constraint counts for volatile memory proofs.

for a CP-zkSNARK. In some, our modifications to Z# and CirC's compiler driver are about 1k LOC.

9. Evaluation

We evaluate our zkSNARKs for RAM against prior work. Our metrics are proving time and the number of rank-1 constraints in the I-R1CS, which is platform-independent and correlated with proving time (Sec. 7). In a few experiments, we also measure Setup time (expensive but run only once per application), $\mathcal{V}$ time (concretely small for all systems we benchmark) and proof size (also small).

9.1. Volatile memory

We compare our volatile memory proof, (§4; "Uniq"), against three baselines: TinyRAM [45], Arya [32], and Haböck [34] (Table 1). All proofs are about A accesses to a memory of size N, with no further computation.

Figure 7 shows constraint counts for $N \in \{2^{20}, 2^{40}, 2^{60}\}$ and $A \in \{2^5, \ldots, 2^{10}\}$. Uniq reduces constraints (relative to the best baseline) by up to 20%, at $A = 2^5$ and $N = 2^{60}$. This is the expected location for the optimum (large N, small A), since Arya and Haböck have worse N-scaling (Sec. 4.5). Uniq's concrete advantage is modest because the N-dependence of the baselines is modest, and because Uniq only optimizes one part (ordering) of the overall RAM proof.

Figure 8 compares proving times. Proving times increase less smoothly than constraint counts because they depend on

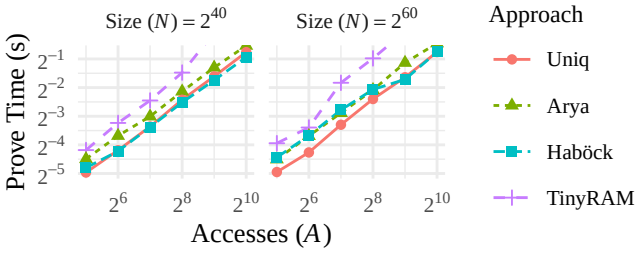


Figure 8: Proving times for volatile memory proofs.

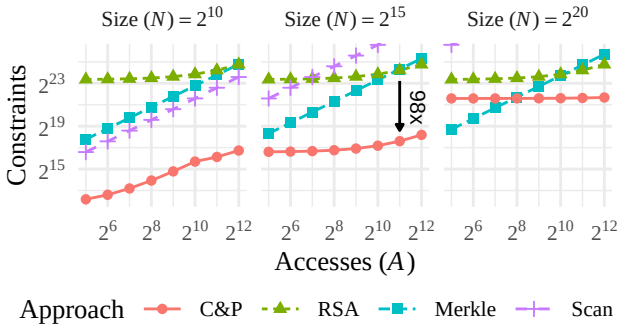


Figure 9: Constraint counts for persistent memory proofs.

Fourier domain sizes [4] and noise. Uniq reduces proving time by up to 32.9%. Importantly, for both proving time and constraint count, Haböck is sometimes better than Uniq, so the best choice for an application depends on N and A.

9.2. Persistent memory

We compare our persistent memory ("C&P"; §5) against Merkle trees [96] ("Merkle"), RSA accumulators [30] ("RSA"), and a naive linear scan ("Scan"). For "Merkle" and "RSA" we use the implementation of Ozdemir et al. [30].

Figure 9 shows constraint counts for $N \in \{2^{10}, 2^{15}, 2^{20}\}$ and $A \in \{2^5, \ldots, 2^{12}\}$. At $N = 2^{15}$, C&P reduces constraints by up to a factor of 98.2×, at $A = 2^{11}$ and $A = 2^{15}$. For smaller A or larger N, C&P's advantage shrinks relative to Merkle. For very small N, C&P's advantage shrinks relative to a naive scan. For large A, RSA's large constant overhead (Sec. 4.5) begins to amortize away, shrinking C&P's relative advantage. Of course, since C&P's per-A cost is lower than RSA's (the latter includes a division-intractable hash), C&P will be still be better for any A.

We also measure $\mathcal{P}$ time, $\mathcal{V}$ time, Setup time, and proof size. Figure 10 shows $\mathcal{P}$ time for $N \in \{2^{15}, 2^{20}\}$ and $A = 3000$.[6] We omit Scan, which is unreasonably expensive. For

[6] We omit non-cryptographic work in computing the AIP transcript ("Extra" in Table 4) from this plot for two reasons. First, this work is straightforward but tedious to optimize. Second, the baseline implementations do this more slowly than ours (Table 4), so including it would arguably be unfair. We do include all *cryptographic* work (i.e., computing the Wesolowski witness [97]), as this is already highly optimized.

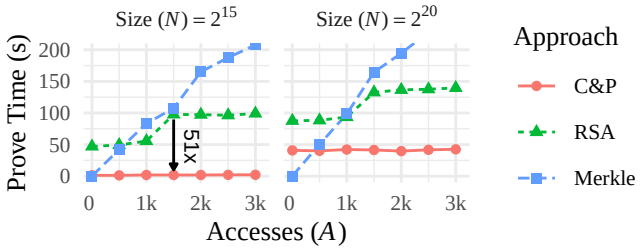


Figure 10: Proving times for persistent memory proofs.

Approach	log N	A	Setup (s)	$\mathcal{P}$ (s)	$\lvert\pi\rvert$ (B)	$\mathcal{V}$ (s)	Extra (s)
C&P	15	3000	28	2	500	0.009	2
C&P	20	3000	598	43	500	0.009	43
RSA	15	3000	1214	99	308	0.009	142
RSA	20	3000	1216	140	308	0.006	146
Merkle	15	3000	2087	208	308	0.007	693
Merkle	20	3000	2914	328	308	0.010	912

Table 4: Costs for a persistent memory of size $N = 2^{20}$.

a memory of size 2^{15}, C&P reduces $\mathcal{P}$ time by 51.3×, (at $A = 2^{11}$). This is slightly less than the constraint reduction, but still very significant. We think that the improvement in proving time is slightly degraded for two reasons. First is the extra commitment linking protocol used by CP-Mirage+ (Sec. 2). Second is the increased bit size of prover witness in C&P relative to RSA. In RSA, many witnesses are bits or other small values, but in C&P many witnesses depend on ≈255-bit (uniformly random) $\mathcal{V}$ challenges. This slows down the `bellman` library's MSMs and—by extension—proving.

Table 4 shows all metrics for this experiment. Setup and $\mathcal{P}$ time are both highly correlated with constraint counts. A C&P proof is slightly larger than Merkle or RSA, but is still only 500 bytes. $\mathcal{V}$ time is small for all systems.

9.3. Sparse persistent memory

We compare the estimated costs of our sparse persistent memory proof ("C&P"; §5) against the cost of alternate approached based on Merkle trees [96] ("Merkle"), RSA accumulators [30] ("RSA"), and a naive linear scan ("Scan"). In all cases, we compute the number of constraints based on our cost model in Table 3.

Figure 11 shows constraints for $C \in \{2^{10}, 2^{15}, 2^{20}\}$ and $A \in \{2^5, \ldots, 2^{12}\}$. C&P reduces constraints by up to a factor of 143×, at $C = 2^{15}$ and $A = 2^{12}$.

9.4. Application: Rollup

We implement a simple zk-Rollup [98] (a zkSNARK about A transactions in a payment system) using our persistent RAM. We use CirC's Z# input language. Figure 12 shows the Z# entry function for our rollup. There are two arguments. The first is a committed array of accounts. Each account is three scalars: the owner's public key (two

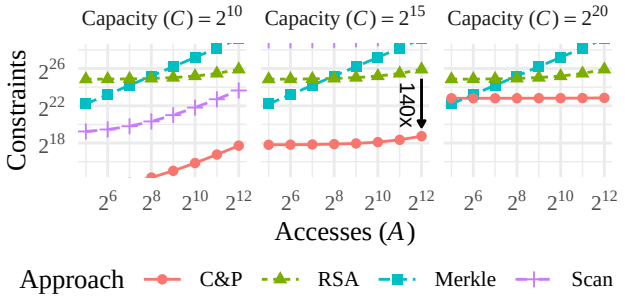


Figure 11: Modelled constraint counts for sparse persistent memory proofs.

```
def main(committed field[N * 3] accounts,
         private Tx[A] txs) -> bool:
  for field i in 0..A do
    field fromIdx = txs[i].fromIdx
    field toIdx = txs[i].toIdx
    Account from_ = read_account(accounts[3*
        fromIdx], accounts[3*fromIdx+1], accounts
        [3*fromIdx+2])
    Account to = read_account(accounts[3*toIdx],
        accounts[3*toIdx+1], accounts[3*toIdx+2])
    assert(from_.key == txs[i].from_)
    assert(to.key == txs[i].to)
    assert(verifyTxSig(txs[i]))
    assert(from_.amount >= txs[i].amount)
    assert(to.amount + txs[i].amount <=
        MAX_BALANCE)
    accounts[3*fromIdx + 2] = from_.amount - txs[i
        ].amount
    accounts[3*toIdx + 2] = to.amount + txs[i].
        amount
  endfor
  return true // only assertions matter
```

Figure 12: Our rollup's entry function. Each account is stored as three $\mathbb{F}$ elements: two for the public key, and one for the account balance. We check the signature and transfer.

field elements) and a balance (one field element) between 0 and `MAX_BALANCE`. The second is a (hidden) list of transactions. Each transfers some amount of money from one public key to another. It includes a signature and the indices of the relevant public keys in the array. The function `main` asserts the validity of all transactions and updates the account balances.

Figure 13 shows the constraint counts for the rollup as a function of A (the number of accesses) and N (the number of accounts). We benchmark the three persistent RAMs from the prior subsection. C&P's improvement is greatest when the number of accesses is high and the RAM size is high, but not too high. C&P's relative advantage here is less than in Figure 9 because the cost of signature verification (identical in all systems) is large. Despite that, C&P reduces constraint count (relative to prior state of the art) by up to 2.55×.

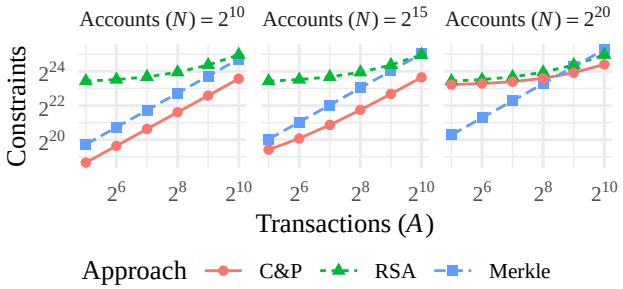


Figure 13: Constraint counts for different rollups.

10. Conclusion

We have presented new AIPs for the consistency of accesses to volatile, persistent, and sparse persistent memories. Our AIPs are based on proofs for different kinds of uniqueness and disjointness that rely on Bézout's identity for univariate polynomials. Our AIPs can be compiled into zkSNARKs using CP-Mirage+. We expect that proofs of uniqueness and disjointness, as well as our approach of compiling AIPs into zkSNARKs will have utility beyond memory-checking.

Acknowledgements. We thank Alessandro Chiesa, Anna P. Y. Woo, Binyi Chen, Lior Rotem, Paul Grubbs, Rohit Nema, Riad S. Wahby, Wilson Nguyen, and the anonymous reviewers for helpful conversations.

This work was funded in part by the Stanford Center for Automated Reasoning and the Stanford Center for Blockchain Research. This work was further supported by DARPA, the Simons Foundation, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[1] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them: from theoretical possibility to near practicality," *CACM*, 2015.

[2] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," *ACM STOC*, 2008.

[3] J. Groth, "On the size of pairing-based non-interactive arguments," *EUROCRYPT*, 2016.

[4] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *IEEE S&P*, 2013.

[5] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish, "Resolving the conflict between generality and plausibility in verified computation," *EuroSys*, 2013.

[6] R. S. Wahby, S. T. V. Setty, Z. Ren, A. J. Blumberg, and M. Walfish, "Efficient RAM and control flow in verifiable outsourced computation," *NDSS*, 2015.

[7] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica, "DIZK: A distributed zero knowledge proof system," *USENIX Security*, 2018.

[8] A. Chiesa, E. Tromer, and M. Virza, "Cluster computing in zero knowledge," *EUROCRYPT*, 2015.

[9] S. T. V. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality," *USENIX Security*, 2012.

[10] J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister, "Verifiable computation with massively parallel interactive proofs," *HotCloud*, 2012.

[11] W. Ma, Q. Xiong, X. Shi, X. Ma, H. Jin, H. Kuang, M. Gao, Y. Zhang, H. Shen, and W. Hu, "Gzkp: A gpu accelerated zero-knowledge proof system," *ASPLOS*, 2023.

[12] W. Nguyen, T. Datta, B. Chen, N. Tyagi, and D. Boneh, "Mangrove: A scalable framework for folding-based snarks," *CRYPTO*, 2024.

[13] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," *CRYPTO*, 2013.

[14] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," *ACM CCS*, 2017.

[15] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," *CRYPTO*, 2019.

[16] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," *CRYPTO*, 2019.

[17] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," *EUROCRYPT*, 2020.

[18] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge." 2019. https://eprint.iacr.org/2019/953.

[19] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," *CRYPTO*, 2020.

[20] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent SNARKs from DARK compilers," *EUROCRYPT*, 2020.

[21] J. Lee, "Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments," *TCC*, 2021.

[22] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. P. Ward, "Marlin: Preprocessing zkSNARKs with universal and updatable SRS," *EUROCRYPT*, 2020.

[23] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "HyperPlonk: Plonk with linear-time prover and high-degree custom gates," *EUROCRYPT*, 2023.

[24] A. Golovnev, J. Lee, S. T. V. Setty, J. Thaler, and R. S. Wahby, "Brakedown: Linear-time and field-agnostic SNARKs for R1CS," *CRYPTO*, 2023.

[25] N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinstein, and E. Tromer, "The hunting of the SNARK," *Journal of Cryptology*, vol. 30, pp. 989–1066, 2017.

[26] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, "ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data," *IEEE S&P*, 2015.

[27] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," *IEEE S&P*, 2015.

[28] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno, "Hash first, argue later: Adaptive verifiable computations on outsourced data," *ACM CCS*, 2016.

[29] S. Setty, S. Angel, T. Gupta, and J. Lee, "Proving the correct execution of concurrent services in zero-knowledge," *OSDI*, 2018.

[30] A. Ozdemir, R. S. Wahby, B. Whitehat, and D. Boneh, "Scaling verifiable computation using efficient set accumulators," *USENIX Security*, 2020.

[31] M. Campanelli, D. Fiore, S. Han, J. Kim, D. Kolonelos, and H. Oh, "Succinct zero-knowledge batch proofs for set accumulators," *ACM CCS*, 2022.

[32] J. Bootle, A. Cerulli, J. Groth, S. K. Jakobsen, and M. Maller, "Arya: Nearly linear-time zero-knowledge proofs for correct program execution," *ASIACRYPT*, 2018.

[33] A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin, "Caulk: Lookup arguments in sublinear time," *ACM CCS*, 2022.

[34] U. Haböck, "Multivariate lookups based on logarithmic derivatives." 2022. https://eprint.iacr.org/2022/1530.

[35] L. Eagen, D. Fiore, and A. Gabizon, "cq: Cached quotients for fast lookups." 2022. https://eprint.iacr.org/2022/1763.

[36] A. Gabizon and D. Khovratovich, "flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size." 2022. https://eprint.iacr.org/2022/1447.

[37] H. M. Ardevol, J. B. Melé, D. Lubarov, and J. L. Muñoz-Tapia, "RapidUp: Multi-domain permutation protocol for lookup tables." 2022. https://eprint.iacr.org/2022/1050.

[38] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables." 2020. https://eprint.iacr.org/2020/315.

[39] J. Posen and A. A. Kattis, "Caulk+: Table-independent lookup arguments." 2022. https://eprint.iacr.org/2022/957.

[40] A. Zapico, A. Gabizon, D. Khovratovich, M. Maller, and C. Ràfols, "Baloo: Nearly optimal lookup arguments." 2022. https://eprint.iacr.org/2022/1565.

[41] S. Setty, J. Thaler, and R. Wahby, "Unlocking the lookup singularity with lasso," *EUROCRYPT*, 2024.

[42] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," *SOSP*, 2013. Extended version: http://eprint.iacr.org/2013/356.

[43] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer, "Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: extended abstract," *ITCS*, 2013.

[44] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," *CRYPTO*, 2013.

[45] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," *USENIX Security*, 2014.

[46] A. E. Kosba, C. Papamanthou, and E. Shi, "xJsnark: A framework for efficient verifiable computation," *IEEE S&P*, 2018.
[47] M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular design and composition of succinct zero-knowledge proofs," *ACM CCS*, 2019.
[48] A. E. Kosba, D. Papadopoulos, C. Papamanthou, and D. Song, "MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs," *USENIX Security*, 2020.
[49] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems (extended abstract)," *ACM STOC*, 1985.
[50] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.
[51] M. Bellare and O. Goldreich, "On defining proofs of knowledge," *CRYPTO*, 1993.
[52] L. Babai, "Trading group theory for randomness," *ACM STOC*, 1985.
[53] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor, "Checking the correctness of memories," *FOCS*, 1991.
[54] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," *ACM CCS*, 2001.
[55] Y. Yang and D. Heath, "Two shuffles make a ram: Improved constant overhead zero knowledge ram.," *USENIX Security*, 2024.
[56] C. Delpech de Saint Guilhem, E. Orsini, T. Tanguy, and M. Verbauwhede, "Efficient proof of ram programs from any public-coin zero-knowledge system," *SCN*, 2022.
[57] N. Franzese, J. Katz, S. Lu, R. Ostrovsky, X. Wang, and C. Weng, "Constant-overhead zero-knowledge for ram programs," *CCS*, 2021.
[58] A. Ozdemir, R. S. Wahby, B. Whitehat, and D. Boneh, "Scaling verifiable computation using efficient set accumulators." 2019. https://eprint.iacr.org/2019/1494.
[59] A. Waksman, "A permutation network," *J. ACM*, vol. 15, no. 1, 1968.
[60] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," *CRYPTO*, 2002.
[61] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," *CRYPTO*, 2019.
[62] Y. Zhang, S.-F. Sun, R. Zhang, and D. Gu, "Polynomial IOPs for memory consistency checks in zero-knowledge virtual machines," *ASIACRYPT*, 2023.
[63] M. Dutta, C. Ganesh, S. Patranabis, S. Prakash, and N. Singh, "Batching-efficient RAM using updatable lookup arguments," 2024.
[64] A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin, "Caulk: Lookup arguments in sublinear time." 2022. https://eprint.iacr.org/2022/621.
[65] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vRAM: Faster verifiable RAM with program-independent preprocessing," *IEEE S&P*, 2018.
[66] L. Goldberg, S. Papini, and M. Riabzev, "Cairo – a Turing-complete STARK-friendly CPU architecture." 2021. https://eprint.iacr.org/2021/1063.
[67] "SP1 project.". https://github.com/succinctlabs/sp1.
[68] "RISC ZERO project.". https://www.risczero.com/.
[69] D. Marin, M. Abdalla, P. Govereau, J. Groth, S. Judson, K. Sosnin, G. V. Policharla, and Y. Zhang, "Nexus 1.0: Enabling verifiable computation," 2024. https://framerusercontent.com/assets/crHiRUJmrEGoUKkIwQmB6m04zEo.pdf.
[70] D. Heath and V. Kolesnikov, "A 2.1 KHz zero-knowledge processor with BubbleRAM," *ACM CCS*, 2020.
[71] D. Heath, Y. Yang, D. Devecsery, and V. Kolesnikov, "Zero knowledge for everything and everyone: Fast ZK processor with cached ORAM for ANSI C programs," *2021 IEEE S&P*, 2021.
[72] D. Heath and V. Kolesnikov, "PrORAM - fast $P(\log n)$ authenticated shares ZK ORAM," *ASIACRYPT*, 2021.
[73] N. Franzese, J. Katz, S. Lu, R. Ostrovsky, X. Wang, and C. Weng, "Constant-overhead zero-knowledge for RAM programs," *ACM CCS*, 2021.
[74] A. Goel, M. Hall-Andersen, and G. Kaptchuk, "Dora: Processor expressiveness is (nearly) free in zero-knowledge for ram programs," 2023.
[75] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos, "TRUESET: Faster verifiable set computations," *USENIX Security*, 2014.
[76] I. Tzialla, A. Kothapalli, B. Parno, and S. T. V. Setty, "Transparency dictionaries with succinct proofs of correct operation," 2022.
[77] J. Lee, K. Nikitin, and S. T. V. Setty, "Replicated state machines without replicated execution," *IEEE S&P*, 2020.
[78] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases," *IEEE S&P*, 2017.
[79] B. Bünz and B. Chen, "Protostar: Generic efficient accumulation/folding for special-sound protocols," *ASIACRYPT*, 2023.
[80] M. Campanelli, A. Faonio, D. Fiore, A. Querol, and H. Rodríguez, "Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions," *ASIACRYPT*, 2021.
[81] R. A. DeMillo and R. J. Lipton, "A probabilistic remark on algebraic program testing," *Inf. Process. Lett.*, vol. 7, no. 4, 1978.
[82] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM*, vol. 27, no. 4, 1980.
[83] R. Zippel, "Probabilistic algorithms for sparse polynomials," *International symposium on symbolic and algebraic manipulation*, 1979.
[84] J. von zur Gathen and J. Gerhard, *Modern computer algebra*. Cambridge university press, 3 ed., 2013.
[85] D. Harvey and J. van der Hoeven, "Integer multiplication in time $O(n \log n)$," *Annals of Mathematics*, vol. 193, no. 2, 2021.
[86] D. Harvey and J. van der Hoeven, "Polynomial multiplication over finite fields in time $O(n \log n)$," *JACM*, 2022.
[87] E. Ben-Sasson, D. Carmon, S. Kopparty, and D. Levit, "Elliptic curve fast fourier transform (ECFFT) Part I," *SODA*, 2023.
[88] B. Wesolowski, "Efficient verifiable delay functions," *Journal of Cryptology*, vol. 33, pp. 2113–2147, 2020.
[89] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," *USENIX Security*, 2021.
[90] B. Braun, "Compiling computations to constraints for verified computation," *UT Austin Honors Thesis HR-12-10*, 2012.
[91] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *CRYPTO*, 1987.
[92] T. Attema, R. Cramer, and L. Kohl, "A compressed Σ-protocol theory for lattices," *CRYPTO 2021, Part II*, 2021.
[93] Z. developers, "Bellman circuit library and zksnark.". https://github.com/zkcrypto/bellman.
[94] E. Kiltz and H. Wee, "Quasi-adaptive NIZK for linear subspaces revisited," *EUROCRYPT*, 2015.
[95] A. Ozdemir, F. Brown, and R. S. Wahby, "CirC: Compiler infrastructure for proof systems, software verification, and more," *IEEE S&P*, 2022.
[96] R. C. Merkle, "A digital signature based on a conventional encryption function," *CRYPTO*, 1988.
[97] B. Wesolowski, "Efficient verifiable delay functions," *EUROCRYPT*, 2019.
[98] B. Whitehat, "roll_up: Scale ethereum with SNARKs.". https://github.com/barryWhiteHat/roll_up.
[99] V. Shoup, "Lower bounds for discrete logarithms and related problems," *EUROCRYPT*, 1997.
[100] B. Braun, "Compiling computations to constraints for verified computation," 2012. UT Austin Honors Thesis HR-12-10.
[101] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," *EUROCRYPT*, 2012.
[102] V. E. Beneš, *Mathematical theory of connecting networks and telephone traffic*. Academic press, 1965.
[103] A. Arun, S. Setty, and J. Thaler, "Jolt: Snarks for virtual machines via lookups," *EUROCRYPT*, 2024.
[104] A. Hoorfar and M. Hassani, "Inequalities on the Lambert W function and hyperpower function," *J. Inequal. Pure and Appl. Math*, vol. 9, no. 2, pp. 5–9, 2008.

Appendix A. Deferred proofs

A.1. Theorem 5: Π_{active}

We restate and prove Theorem 5:

Theorem. *If $A \leq N$, the protocol Π_{active} (for language $\mathcal{L}_{\mathsf{active}}$) has perfect completeness, soundness error $\leq (N^2 + 2N + A)/|\mathbb{F}|$, and $\mathcal{V}$ complexity $3N + 2A + O(1)$.*

Proof. We prove completeness, then soundness, and then compute $\mathcal{V}$ complexity.

Completeness. Define $I = \{i \in [N] : i \in \mathbf{a}\}$ and define $J = [N] \setminus I$. Since the a_i are distinct, $|I| = A$ and $|J| = N - A$, so $\mathbf{h}$ has length $N - A$ as required. Now, we will show that $\mathcal{V}$'s first test passes. First, disjointness of I and J gives:

$$\prod_{i \in I \cup J} (\alpha + m_i + i\beta) = \prod_{i \in J} (\alpha + m_i + i\beta) \times \prod_{i \in I} (\alpha + m_i + i\beta)$$

Then, re-indexing the products and using $\mathbf{h}$ gives:

$$\prod_{i=1}^{N} (\alpha + m_i + i\beta) = \prod_{i=1}^{N-A} (\alpha + h_i) \times \prod_{i=1}^{A} (\alpha + v_i + a_i\beta)$$

which is $\mathcal{V}$'s first test. Applying this argument with $\mathbf{m}'$ and $\mathbf{v}'$ instead of $\mathbf{m}$ and $\mathbf{v}$ shows that $\mathcal{V}$'s other test succeeds too.

Soundness. Suppose that the instance is not in the language, and that $\mathcal{V}$ accepts. We will show that this is unlikely.

First, consider $\mathcal{V}$'s tests as equalities between polynomials in α. Per the fundamental theorem of algebra, these equalities hold except with probability $\leq 2N/|\mathbb{F}|$. So the factorizations must be equal; that is, these multisets must be equal:

$$\{\!\{m_i + i\beta\}\!\}_{i=1}^{N} = \{\!\{h_i\}\!\}_{i=1}^{N-A} \uplus \{\!\{v_i + a_i\beta\}\!\}_{i=1}^{A}$$
$$\{\!\{m'_i + i\beta\}\!\}_{i=1}^{N} = \{\!\{h_i\}\!\}_{i=1}^{N-A} \uplus \{\!\{v'_i + a_i\beta\}\!\}_{i=1}^{A}$$

Now, consider the left multisets. The (m_i, i) are all distinct, so the $m_i + i\beta$ must all be distinct as well, except with probability $\binom{N}{2}/|\mathbb{F}|$. Similarly for the (m'_i, i), except with probability $\binom{N}{2}/|\mathbb{F}|$. Thus our multiset equalities are set equalities and the unions are disjoint.

$$\{m_i + i\beta\}_{i=1}^{N} = \{h_i\}_{i=1}^{N-A} \uplus \{v_i + a_i\beta\}_{i=1}^{A}$$
$$\{m'_i + i\beta\}_{i=1}^{N} = \{h_i\}_{i=1}^{N-A} \uplus \{v'_i + a_i\beta\}_{i=1}^{A}$$

That implies the existence of injection $\pi : [A] \to [N]$ such that for all i, $m_{\pi(i)} + \pi(i)\beta = v_i + a_i\beta$. And similarly, the existence of injection $\pi' : [A] \to [N]$. Now, we have that for all i, $(m_{\pi(i)}, \pi(i)) = (v_i, a_i)$ and similarly for the primed terms, except with probability $2A/|\mathbb{F}|$.

Since π is an injection, this implies that the a_i are all distinct. Further, this implies that for all i, $v_i = m_{a_i}$ and $v'_i = m'_{a_i}$.

Next, from (3) and the definition of π, we have the existence of a bijection ρ from $[N] \setminus \mathrm{Im}(\pi)$ to $[N - A]$ such that $m_i + i\beta = h_{\rho(i)}$. And similarly, we have a ρ'. Let $\tau = \rho'^{-1} \circ \rho$. Then we have that for all $i \in \mathrm{Dom}(\rho)$, $m_i + i\beta = m'_{\tau(i)} + \tau(i)\beta$. Except with probability $(N - A)/|\mathbb{F}|$, this implies that $(m_i, i) = (m'_{\tau(i)}, \tau(i))$. That, in turn, implies that $m_i = m'_i$. On the other hand, if $i \notin \mathrm{Dom}(\pi)$, then it is in the image of π, so $i \in \mathbf{a}$. Either way, we have that $m_i = m'_i \vee i \in \mathbf{a}$.

Thus, we've shown that if $\mathcal{V}$'s tests succeed, then the instance is in the language, except with probability

$$\leq \frac{2N}{|\mathbb{F}|} + \frac{2}{|\mathbb{F}|}\binom{N}{2} + \frac{2A}{|\mathbb{F}|} + \frac{N-A}{|\mathbb{F}|} = \frac{N^2 + 2N + A}{|\mathbb{F}|}$$

$\mathcal{V}$ **Complexity.** Computing c requires $N - A + O(1)$ non-linear multiplications. Then, the first test requires $N + 2A + O(1)$. However, the second test requires only $N + A + O(1)$ new non-linear multiplications because all the $a_i\beta$ products were already computed for the first test. Thus, $\mathcal{V}$ complexity is $3N + 2A + O(1)$. □

A.2. Theorem 6: Π_{PuncUniq}

We restate and prove Theorem 6:

Theorem. *The protocol Π_{PuncUniq} (for language $\mathcal{L}_{\mathsf{PuncUniq}}$) has perfect completeness, soundness error $\leq (C + A - 1)/|\mathbb{F}|$, and $\mathcal{V}$ complexity $4C + O(\log C)$.*

Proof. We prove completeness, then soundness, then $\mathcal{V}$ complexity.

Completeness. Suppose $\mathbf{a} \in \mathcal{L}_{\mathsf{PuncUniq}}$. Let d be the number of occurrences of 0 in $\mathbf{a}$. Then, $c = \max(0, d - 1)$. Let $v(X) = \prod_{s \in S}(X - s)$. Then, $\gcd(v, v') = 1$, and UniqBez returns s, t such that $vs + v't = 1$ (per the correctness of UniqBez; Theorem 4). Further, since $z(X) = X^c v(X)$, we can show that $z'(X) = X^c v'(X)$. (Note that, since v may contain a factor X, showing this requires a little case-work.) Then, we have $zs + z't = X^c$. Then, $\mathcal{V}$'s test succeeds with probability 1.

Soundness. Suppose $\mathbf{a} \notin \mathcal{L}_{\mathsf{PuncUniq}}$. Then some $y \neq 0$ occurs in $\mathbf{a}$ at least twice. Then, $X - y$ divides $\gcd(z, z')$. So, there are no s, t such that $zs + z't$ is indivisible by $X - y$, and in particular $zs + z't$ cannot equal X^c. Now, consider the degree of the polynomial equation. The polynomials f, g have degrees C and A respectively. The polynomials s, t have degrees at most $A - 1$ and $C - 1$ respectively. Further, c can be at most C. Thus, the equation's degree is at most $C + A - 1$.

$\mathcal{V}$ **complexity.** Per the analysis of Π_{uniq}, z, z', s, t can be evaluated with $4C + O(1)$ multiplications. Then, repeated squaring can evaluate α^c in $O(\log C)$ multiplications. □

A.3. Theorem 7: Π_{PuncDisj}

We restate and prove Theorem 7:

Theorem. *The protocol Π_{PuncDisj} (for language $\mathcal{L}_{\mathsf{PuncDisj}}$) has perfect completeness, soundness error $\leq (C + A - 1)/|\mathbb{F}|$, and $\mathcal{V}$ complexity $2C + 2A + O(\log C)$.*

Proof. We prove completeness, then soundness, then $\mathcal{V}$ complexity.

Completeness. Suppose $(\mathbf{a}, \mathbf{b}) \in \mathcal{L}_{\mathsf{PuncDisj}}$. Then, I contains only 0s, and S' and T' are disjoint. Define $\tilde{f}(X) = \prod_{s \in S'}(X - s)$ and $\tilde{g}(X) = \prod_{t \in T'}(X - t)$. Then, s, t satisfy $s\tilde{f} + t\tilde{g} = 1$ (per the correctness of DisjBez). Define c as in the protocol. Then, $f = \tilde{f}X^c$ and $g = \tilde{g}X^c$. $sf + tg = X^c$. Then, $\mathcal{V}$'s test succeeds with probability 1.

Soundness. Suppose $(\mathbf{a}, \mathbf{b}) \notin \mathcal{L}_{\mathsf{PuncUniq}}$. Then some $y \neq 0$ occurs in $\mathbf{a}$ and $\mathbf{b}$. Then, $X - y$ divides $\gcd(z, z')$. So, there are no s, t such that $zs + z't$ is indivisible by $X - y$, and in particular $zs + z't$ cannot equal X^c. Now, consider the degree of the polynomial equation. The polynomials f, g have degrees C and A respectively. The polynomials s, t have degrees at most $A - 1$ and $C - 1$ respectively. Further, c can be at most C. Thus, the equation's degree is at most $C + A - 1$. So, the soundness error is at most $(C + A - 1)/|\mathbb{F}|$.

$\mathcal{V}$ **complexity.** Per their degrees, f, g, s, t can be evaluated in $2C + 2A - 2$ multiplications, and α^c requires $O(\log C)$ multiplications to compute. □

A.4. Theorem 8: I-Mirage+

We restate and prove Theorem 8:

Theorem. *Let I-R1CS ϕ be complete and knowledge-sound for witness relation R. Then in the generic group model, I-Mirage+ is a zkSARK for R.*

Proof. We prove completeness, zero-knowledge, and knowledge-soundness separately.

Completeness. We will show that verification succeeds with overwhelming probability when $\mathcal{P}$ follows the protocol. The completeness of ϕ as an AIP implies that (with overwhelming probability), $\mathcal{P}$ constructs z satisfying $Az \circ Bz = Cz$. Now, it suffices to show that verification equation for I-Mirage+ holds. Consider that equation:

$$\begin{aligned} e(P_A, P_B) &= e(\alpha G_1, \beta G_2) \\ &+ e(\gamma^{-1} f'_X(\tau) G_1, \gamma G_2) \\ &+ \textstyle\sum_{i=1}^{r} e(P_{C,i}, \delta_i G_2) \end{aligned}$$

Taking the logarithm of both sides with respect to the generator $e(G_1, G_2)$ gives:

$$\begin{aligned} &(\alpha + \rho\delta_\mu + f_A(\tau))(\beta + \sigma\delta_\mu + f_B(\tau)) \\ &= \alpha\beta + (\gamma^{-1} f'_X(\tau))\gamma + \textstyle\sum_{i=1}^{\mu-1}(\delta_\mu \kappa_i + \delta_i^{-1} f'_{W_i}(\tau))\delta_i \\ &+ (\sigma\alpha' + \rho\beta' - \rho\sigma\delta_\mu + \delta_\mu^{-1}(hz + f'_{W_\mu})(\tau) - \textstyle\sum_{i=1}^{\mu-1} \kappa_i \delta_i)\delta_\mu \end{aligned}$$

where $\alpha' = \alpha + \rho\delta_\mu + f_A(\tau)$. Expanding the products and the definitions of f'_{W_i}, f'_X gives:

$$f_A(\tau) f_B(\tau) = f_C(\tau) + h(\tau) z(\tau)$$

This follows if the following polynomial equality holds

$$f_A(X) f_B(X) = f_C(X) + h(X) z(X)$$

This holds if

$$z(X) \mid (f_A(X) f_B(X) - f_C(X))$$

This holds if the right is zero at all zeros of z; i.e.,

$$\forall i \in [n], f_A(\omega^i) f_B(\omega^i) = f_C(\omega^i)$$

$$\forall i \in [n], \left(\textstyle\sum_{j=1}^{m} z_j A_{j,i}\right)\left(\textstyle\sum_{j=1}^{m} z_j B_{j,i}\right) = \left(\textstyle\sum_{j=1}^{m} z_j C_{j,i}\right)$$

This follows from $Az \circ Bz = Cz$.

Zero-Knowledge. We construct a simulator $\mathcal{S}$ that, given x, outputs π (a $\mathcal{V}$ view for I-Mirage+), pk, and vk. First, $\mathcal{S}$ runs Setup, and saves the scalars $\alpha, \beta, \gamma, \delta_1, \ldots, \delta_r, \tau$. Second, $\mathcal{S}$ samples $r_1, \ldots, r_{\mu-1}$ uniformly. Third, $\mathcal{S}$ samples the group elements $P_A, P_B, P_{C,1}, \cdots, P_{C,\mu-1}$ by sampling their exponents $a, b, c_1, \ldots, c_{\mu-1}$. Fourth, $\mathcal{S}$ sets

$$P_{C,\mu} \leftarrow \delta_\mu^{-1}\left(ab - \alpha\beta - f'_X(\tau) - \sum_{i=1}^{\mu-1} c_i \delta_i\right) G_1$$

(where f'_X is defined as in Verify). Finally, $\mathcal{S}$ outputs $(r_1, \ldots, r_\mu, P_A, P_B, P_{C,1}, \ldots, P_{C,\mu})$, pk, vk.

Now, we show that the output of $\mathcal{S}$ is distributed identically as in I-Mirage+. First, pk, vk, and the r_i are constructed identically in both. Second, all group elements but $P_{C,\mu}$ are uniform and independent in both. Finally, in both, $P_{C,\mu}$ is the unique solution to the verification equation. So I-Mirage+ has perfect zero-knowledge.

Knowledge Soundness. First, we sketch the proof. Our analysis is similar to Groth's—without intermediate notions like split NILPs [3]. We also use the GGM to extract "witnesses" from $\mathcal{P}$'s group elements. In Groth, these witnesses are the solution to an R1CS. For us, they are AIP transcripts, that (via the AIP extractor) give an actual witness for R. Groth used the equivalence of Laurent polynomials to argue R1CS satisfaction; we will use it to argue AIP transcript validity. Just as in Groth, this equivalence requires Setup to hide the scalars that it samples. Our analysis is in the bilinear version of Shoup's GGM [99], with $|S|/|\mathbb{G}| = \Omega(2^\lambda)$, where S is the encoding space.

Let $\mathcal{V}$ be the verifier for an AIP that is knowledge-sound for $R(x, w)$. Thus, there is an extractor $\mathcal{E}$ that computes w from x and oracle access to a convincing prover $\mathcal{P}$. We define an extractor $\mathcal{E}'$ that, given x and oracle access to a I-Mirage+ prover $\mathcal{P}'$, computes w. Let $\mathcal{V}'$ be the verifier for I-Mirage+. $\mathcal{E}'$ interacts with $\mathcal{P}'$ and with $\mathcal{E}$ (to which it imitates $\mathcal{P}$). First, $\mathcal{E}'$ receives $P_{C,1}$ from $\mathcal{P}'$. Since $\mathcal{P}'$ is an adversary in the GGM, with an encoding space that is exponentially larger than the group, $\mathcal{P}'$ can only obtain valid group elements by interacting with the group oracle (except with negligible probability).[7] Thus, by observing the oracle queries of $\mathcal{P}'$, $\mathcal{E}'$ can compute scalar *explanations* of $P_{C,1}$ [3]. That is, it can compute scalars $C_{1,\alpha}, C_{1,\beta}, (C_{1,\delta_i})_{i\in[r]}, (C_{1,i})_{i=0}^{N-1}$, and scalar polynomials $A(\tau) \in \mathbb{F}^{<N}[\tau], A_z(\tau) \in \mathbb{F}^{<N-1}[\tau]$ such that

$$\begin{aligned} \log_{G_1}(P_{C,1}) &= \mathsf{lc}(C_{1,*}) \\ &= C_{1,\alpha}\alpha + C_{1,\beta}\beta + \textstyle\sum_{i=1}^{r} C_{1,\delta_i}\delta_i \\ &+ C(\tau) + C_z(\tau) z(\tau) \delta_\mu^{-1} \\ &+ \sum_{i\in X} C_{1,i}\gamma^{-1}(\beta f_{A,i}(\tau) + \alpha f_{B,i}(\tau) + f_{C,i}(\tau)) \\ &+ \sum_{j=1}^{\mu} \sum_{i\in W_j} C_{1,i}\delta_j^{-1}(\beta f_{A,i}(\tau) + \alpha f_{B,i}(\tau) + f_{C,i}(\tau)) \end{aligned}$$

In the future, we will refer to the collection of scalars and polynomials that represent group element P_x in terms of pk, vk, as $P_{x,*}$, and use the function $\mathsf{lc}(P_{x,*})$ to refer to the linear combination of exponents in the SRS according to such a collection. One proof component, P_B is a $\mathbb{G}_2$ element, so its explanation has form:

$$\log_{G_2}(P_B) = \mathsf{lc}(B_*) = B_\beta\beta + B_\gamma\gamma + \sum_{i=1}^{r} B_{\delta_i}\delta_i + B(\tau)$$

Now, $\mathcal{E}'$ sends $w_1 \leftarrow (C_{1,i})_{i\in W_1}$ as the first message to $\mathcal{E}$. When $\mathcal{E}$ replies with r_1, $\mathcal{E}'$ forwards it to $\mathcal{P}'$. $\mathcal{E}'$ handles future $P_{C,i}$, and r_i in the same way. (During this process, it gets explanations $C_{2,*}, \ldots, C_{r,*}$ too. It also gets explanations A_* and B_* for P_A and P_B. At the end, $\mathcal{E}'$ outputs the same w as $\mathcal{E}$.

[7] I-Mirage+ is also secure when the encoding space has the same size as the groups, but the proof is a bit more involved. One also has to argue that if $\mathcal{P}'$ uses any sampled group elements in producing its outputs, then the verification equation will fail.

Now, by the knowledge soundness of $\mathcal{V}$, we have that

$$\Pr[w \text{ valid}] \geq \Pr[\mathcal{E}' \text{ convinces } \mathcal{V}] - \mathsf{negl}$$

We must show that

$$\Pr[w \text{ valid}] \geq \Pr[\mathcal{P}' \text{ convinces } \mathcal{V}'] - \mathsf{negl}$$

Thus, it suffices to show that in the case that $\mathcal{P}'$ convinces $\mathcal{V}'$, $\mathcal{E}'$ convinces $\mathcal{V}$, except with negligible probability. In this case, the verification equation for I-Mirage+ holds, so we have equality of exponents:

$$\mathsf{lc}(A_*)\mathsf{lc}(B_*) = \alpha\beta + f'_X(\tau) + \textstyle\sum_{i=1}^{r} \delta_i \mathsf{lc}(C_{i,*}) \quad (1)$$

If this is true, then with the r_i and w_i that $\mathcal{E}'$ computes, we will show that $Az \circ Bz = Cz$ holds, where $z = (x, r_1, \ldots, r_{\mu-1}, w_1, \ldots, w_\mu)$. First, we observe that (except with negligible probability), Equation 1 must hold as a Laurent polynomial equation in over variables sampled during Setup. The reason for this is exactly the reason given in Groth's analysis [3, Lemma 1]: because pk and vk leak no information about Setup's scalars (except with negligible probability). We omit this reasoning here, since it essentially the same as that given by Groth, with Groth's δ replaced by the δ_i. So, we have that Equation 1 holds as a polynomial identity; it holds coefficient-by-coefficient.

By the equality of $\alpha\beta$ coefficients, we have $A_\alpha B_\beta = 1$. Then, we can multiply the A_* by B_β / A_α and divide the B_* by the same value to preserve Equation (1) and ensure that:

$$A_\alpha = 1 \wedge B_\beta = 1 \quad (2)$$

By the equality of $\alpha\gamma$ coefficients, (and Eq. 2) we have

$$A_\alpha B_\gamma = 0 \implies B_\gamma = 0 \quad (3)$$

By the equality of β^2 coefficients, (and Eq. 2) we have

$$A_\beta B_\beta = 0 \implies A_\beta = 0 \quad (4)$$

By the equality of coefficients of monomials involving $\beta^2 \delta_j^{-1}$, for all $j \in [r]$, we have

$$B_\beta \textstyle\sum_{i \in W_j} A_i f_{A,i}(\tau) = 0$$

Similarly, by the equality of monomials with $\beta^2\gamma^{-1}$, we have

$$B_\beta \textstyle\sum_{i \in X} A_i f_{A,i}(\tau) = 0$$

Putting these two together, we have

$$A_1 = 0 \wedge \cdots \wedge A_m = 0$$

From the coefficients of $\beta\delta_\mu^{-1}$, as polynomials in τ, we have

$$B_\beta A_z(\tau) z(\tau) = 0 \implies A_z(\tau) = 0$$

So, we have:

$$\mathsf{lc}(A_*) = \alpha + \textstyle\sum_{i=1}^{r} A_{\delta_i} \delta_i + A(\tau)$$
$$\mathsf{lc}(B_*) = \beta + \textstyle\sum_{i=1}^{r} B_{\delta_i} \delta_i + B(\tau)$$

We now consider the equality of terms involving $\delta_j \gamma^{-1} \tau^i$, for $j \in [r]$; their coefficients must be equal:

$$0 = \delta_j \textstyle\sum_{i \in X} C_{j,i} f_{C,i}(\tau)$$
$$\implies \forall j \in [r], \forall i \in X, C_{j,i} = 0$$

Now, the remaining $\alpha\tau^i$ terms give:

$$B(\tau) = \textstyle\sum_{i \in X} x_i f_{B,i}(\tau) + \sum_{j=1}^{\mu} \sum_{i \in W_j} C_{1,i} f_{B,i}(\tau)$$

which, with our definition of w_j and z is just:

$$B(\tau) = \textstyle\sum_{i=1}^{m} z_i f_{B,i}(\tau)$$

Similarly, the $\beta\tau^i$ terms give an expression for $A(\tau)$:

$$A(\tau) = \textstyle\sum_{i=1}^{m} z_i f_{A,i}(\tau)$$

And, (mod $z(\tau)$) the τ^i terms give:

$$\left(\sum_{i=1}^{m} z_i f_{A,i}(\tau)\right)\left(\sum_{i=1}^{m} z_i f_{B,i}(\tau)\right) \equiv \left(\sum_{i=1}^{m} z_i f_{C,i}(\tau)\right)$$

Which holds at ω^j (a root of z) for all $j \in [n]$:

$$\left(\sum_{i=1}^{m} z_i f_{A,i}(\omega^j)\right)\left(\sum_{i=1}^{m} z_i f_{B,i}(\omega^j)\right) = \left(\sum_{i=1}^{m} z_i f_{C,i}(\omega^j)\right)$$
$$\left(\sum_{i=1}^{m} z_i A_{j,i}\right)\left(\sum_{i=1}^{m} z_i B_{j,i}\right) = \left(\sum_{i=1}^{m} z_i C_{j,i}\right)$$
$$Az \circ Bz = Cz$$

This concludes our GGM proof. □

Appendix B. CP-Mirage+ construction

Let the relation R_{link} be

$$R_{\mathsf{link}} = \{((c_i, c'_i)_{i=1}^k; (w_i, o_i, o'_i)_{i=1}^k) : \bigwedge_{i=1}^{k} c_i = \mathsf{Com}(\mathsf{gens}, w_i, o_i) \wedge c'_i = \mathsf{Com}(\mathsf{gens}_i, w_i, o'_i)\}$$

$\mathsf{CP}_{\mathsf{link}}$ is a zkSNARK for R_{link} with the following interface

- $\mathsf{Setup}(\mathsf{gens}, (\mathsf{gens})_{i=1}^k) \to (\mathsf{pk}_{\mathsf{link}}, \mathsf{vk}_{\mathsf{link}})$: generate proving and verifying keys
- $\mathsf{Prove}(\mathsf{pk}_{\mathsf{link}}, (w_i, o_i, o'_i)_{i=1}^k) \to \pi_{\mathsf{link}}$: for $c_i = \mathsf{Com}(\mathsf{gens}, w_i, o_i)$ and $c'_i = \mathsf{Com}(\mathsf{gens}_i, w_i, o'_i)$, generate a proof that $R_{\mathsf{link}}((c_i, c'_i)_{i=1}^k; (w_i, o_i, o'_i)_{i=1}^k)$ holds
- $\mathsf{Verify}(\mathsf{vk}_{\mathsf{link}}, (c_i, c'_i)_{i=1}^k, \pi_{\mathsf{link}}) \to \{0, 1\}$: check the proof

We present the full construction of CP-Mirage+ in Protocol 6. The main idea is to use $\mathsf{CP}_{\mathsf{link}}$ to prove that each prover message $P_{C,i}$ is a commitment to the same witness as each c_i provided in the instance. The generators used by $\mathsf{CP}_{\mathsf{link}}$ are those used for the external commitments, and the implicit generators used to generate each $P_{C,i}$. The security of the scheme is straight-forward given the security of I-Mirage+ and $\mathsf{CP}_{\mathsf{link}}$.

$\underline{\mathsf{Setup}(\phi, \mathsf{gens}) \to (\mathsf{pk}, \mathsf{vk}) :}$
$(\mathsf{pk}', \mathsf{vk}') \leftarrow$ I-Mirage+.$\mathsf{Setup}(\phi)$
for $i \in [k]$: $\mathsf{gens}_i \leftarrow (G_1, (\delta_i^{-1} f_j(\tau))_{j \in W_i} \cdot G_1)_{j \in W_i}$
$(\mathsf{pk}_{\mathsf{link}}, \mathsf{vk}_{\mathsf{link}}) \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{Setup}(\mathsf{gens}, (\mathsf{gens}_i)_{i=1}^k)$
pk contains: $(\mathsf{pk}', \mathsf{pk}_{\mathsf{link}})$
vk contains: $(\mathsf{vk}', \mathsf{vk}_{\mathsf{link}})$
$\underline{\mathcal{P}(\mathsf{pk}, x, \mathsf{hint}):}$
$(\mathsf{pk}', \mathsf{pk}_{\mathsf{link}}) = \mathsf{pk}$
$(x', c_1, \ldots, c_k) = x$
Run I-Mirage+.$\mathcal{P}(\mathsf{pk}', x, \mathsf{hint})$
Let $P_{C,i}$, δ, and κ_i be from the I-Mirage+.$\mathcal{P}$
$\pi_{\mathsf{link}} \leftarrow \mathsf{CP}_{\mathsf{link}}.\mathsf{Prove}(\mathsf{pk}_{\mathsf{link}}, (w_i, o_i, \delta\kappa_i)_{i=1}^k)$
send to $\mathcal{V}$: π_{link}
$\underline{\mathcal{V}(\mathsf{vk}, x):}$
$(\mathsf{vk}', \mathsf{vk}_{\mathsf{link}}) = \mathsf{vk}$
$(x', c_1, \ldots, c_k) = x$
Run I-Mirage+.$\mathcal{V}(\mathsf{vk}', x)$
receive from $\mathcal{P}$:π_{link}
assert: $\mathsf{CP}_{\mathsf{link}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{link}}, (c_i, P_{C,i})_{i=1}^k, \pi_{\mathsf{link}}) = 1$

Protocol 6: CP-Mirage+: our efficient zkSARK for I-R1CS with an externally committed witness.

$\Pi_{\mathsf{Vmem}}(\mathsf{tr})$

$\underline{\mathcal{P}(\cdots)}$		$\underline{\mathcal{V}(\cdots)}$
include $t_i \leftarrow i$ in tr		include $t_i \leftarrow i$ in tr
$\mathsf{tr}' \leftarrow \mathsf{sort}_{(a,t)}(\mathsf{tr})$		
$\mathbf{g} \leftarrow (i = 1 \vee a'_{i-1} \neq a'_i)_{i=1}^A$		
$\mathbf{g}' \leftarrow (\mathsf{to}_{\mathbb{F}}(g_i))_{i=1}^A$	$\xrightarrow{\mathsf{tr}', \mathbf{g}'}$	
$(\mathbf{a}', \mathbf{t}', \mathbf{w}', \mathbf{v}') \leftarrow \mathsf{tr}'$		$(\mathbf{a}', \mathbf{t}', \mathbf{w}', \mathbf{v}') \leftarrow \mathsf{tr}'$
		$g'_1 \stackrel{?}{=} 1$
$\ldots\ldots \Pi_{\neq 0}((g'_2, \ldots, g'_A), (a'_2 - a'_1, \ldots, a'_A - a'_{A-1})) \ldots\ldots$		
$\ldots\ldots\ldots\ldots\ldots\ldots \Pi_{\mathsf{perm}}(\mathsf{tr}, \mathsf{tr}') \ldots\ldots\ldots\ldots\ldots\ldots$		
$\ldots\ldots\ldots\ldots\ldots \Pi_{\mathsf{c\text{-}ord}}(\mathbf{t}', \mathbf{1} - \mathbf{g}', 1, A) \ldots\ldots\ldots\ldots\ldots$		
$\ldots\ldots\ldots\ldots\ldots\ldots \Pi_{\mathsf{c\text{-}uniq}}(\mathbf{a}', \mathbf{g}') \ldots\ldots\ldots\ldots\ldots\ldots$		
$\ldots\ldots\ldots\ldots\ldots \Pi_{\mathsf{ordRoW}}(\mathbf{g}', \mathbf{v}', \mathbf{w}') \ldots\ldots\ldots\ldots\ldots$		

Protocol 7: Our volatile memory proof.

Appendix C. AIPs for memory

In this appendix, we describe our memory proofs in full, as AIPs. We do not give explicit I-R1CS instances for them. When compiled to I-R1CS, most of our proofs acquire one additional round (Sec. 7).

C.1. Volatile memory

Protocol 7 (Π_{Vmem}) is our volatile memory proof. It uses sub-proofs for non-zero testing, permutations, conditional ordering, conditional uniqueness, and an ordered read-over-write (RoW) property. Our contribution is the use of a conditional uniqueness proof and the construction of such a proof. We give the language for conditional uniqueness

$\Pi_{\neq 0}(\mathbf{g}, \mathbf{d} \in \mathbb{F}^n)$

$\underline{\mathcal{P}(\cdots)}$		$\underline{\mathcal{V}(\cdots)}$
$\mathbf{r} \leftarrow (\mathsf{ite}(d_i = 0, 0, d_i^{-1}))_{i=1}^n$	$\xrightarrow{\mathbf{r}}$	
		$\forall i \in [n] : r_i d_i \stackrel{?}{=} g_i$
		$\forall i \in [n] : (1 - g_i) d_i \stackrel{?}{=} 0$

Protocol 8: Proof that shows $\mathbf{g} \in \{0,1\}^n$ and that each g_i is 1 iff d_i is nonzero.

$\Pi_{\mathsf{perm}}(A, B \in \mathbb{F}^{n \times k})$

$\underline{\mathcal{P}(\cdots)}$	$\underline{\mathcal{V}(\cdots)}$
	sample $\alpha, \beta \in \mathbb{F}$
	$H_r(\alpha, (H_c(\beta, A_{i,\cdot}))_{i=1}^n) \stackrel{?}{=} H_r(\alpha, (H_c(\beta, B_{i,\cdot}))_{i=1}^n)$

Protocol 9: A proof that A's rows are a permutation of B's, using universal hashing. A_i denotes the i^{th} row of A.

and our construction in Section 4.4. The other sub-proofs and their languages are adaptations from prior work. The languages are:

$$\mathcal{L}_{\neq 0} = \{(\mathbf{g}, \mathbf{d} \in \mathbb{F}^n) : \bigwedge_{i=1}^n g_i \in \{0,1\} \wedge (g_i = 1 \implies d_i \neq 0)\}$$
$$\mathcal{L}_{\mathsf{perm}} = \{(A, B \in \mathbb{F}^{n \times k}) : \{A_i\}_{i=1}^n = \{B_i\}_{i=1}^n\}$$
$$\mathcal{L}_{\mathsf{c\text{-}ord}} = \{(\mathbf{d} \in \mathbb{F}^n, \mathbf{g} \in \{0,1\}^n) : \bigwedge_{i=2}^n (g_i = 1 \implies d_i > d_{i-1})\}$$
$$\mathcal{L}_{\mathsf{ordRoW}} = \{(\mathbf{g}, \mathbf{v}, \mathbf{w}) : \forall i \in [A], (w_i = 0) \implies (v_i = \mathsf{ite}(g_i = 0, v_{i-1}, 0)) \; v_0 = 0\}$$

Note that A_i denotes row i from matrix A.

The proofs themselves are $\Pi_{\neq 0}$ (Protocol 8), Π_{perm} (Protocol 9), $\Pi_{\mathsf{c\text{-}ord}}$ (Protocol 10), and Π_{ordRoW} (Protocol 11). $\Pi_{\neq 0}$ is due to Braun [100]. The Π_{perm} proof is a combination of universal hashing for sequences (H_c) and for multisets (H_r). H_r (also known as a "polynomial fingerprint") was first applied to cryptography by Neff [54], to cryptographic proof systems by Bayer and Groth [101], and to memory proofs by Bootle et al. [32]. The $\Pi_{\mathsf{c\text{-}ord}}$, Π_{ordRoW}, and Π_{range} (used by $\Pi_{\mathsf{c\text{-}ord}}$) are due to Ben-Sasson et al. [45]. However, while they instantiated Π_{range} with the bit-based $\Pi_{\mathsf{range,bit}}$, we instantiate it with $\Pi_{\mathsf{range,log}}$ [34]. We discuss both options in Appendix C.2.

Now, we state and prove the correctness of Π_{Vmem}.

Theorem 9. *The protocol Π_{Vmem} (for language $\mathcal{L}_{\mathsf{Vmem}}$) has perfect completeness and soundness error $\leq 3n/|\mathbb{F}| + n/(|\mathbb{F}| - n)$.*

$\Pi_{\mathsf{c\text{-}ord}}(\mathbf{x} \in [l,u]^n, \mathbf{g} \in \{0,1\}^n, l, u)$

$\mathcal{P}(\cdots)$	$\mathcal{V}(\cdots)$
$\Pi_{\mathsf{range}}((g_2(x_2 - x_1 - 1), \ldots, g_n(x_n - x_{n-1} - 1)), u - l - 1)$	

Protocol 10: Proof that $g_i = 1$ implies $x_i > x_{i-1}$. Assumes that all x_i are in a range $[l, u]$, with $(u - l - 1)n$ less than $\mathbb{F}$'s characteristic.

$\Pi_{\mathsf{ordRoW}}(\mathbf{g} \in \{0,1\}^A, \mathbf{v} \in \mathbb{F}^A, \mathbf{w} \in \{0,1\}^A)$

$\mathcal{P}(\cdots)$	$\mathcal{V}(\cdots)$
	$v_0 \leftarrow 0$
	$\forall i \in [A] : (1 - w_i)(v_i - (1 - g_i)v_{i-1}) \stackrel{?}{=} 0$

Protocol 11: Proof of read-over-write (RoW) semantics for an in-order memory transcript.

Proof. We assume the following lemmas about protocols from prior work:

- $\Pi_{\neq 0}$ has perfect completeness and soundness.
- Π_{perm} has perfect completeness and soundness error $\leq (k-1)n/|\mathbb{F}|$. (This follows from Schwartz-Zippel lemma [81–83].)
- $\Pi_{\mathsf{c\text{-}ord}}$ has perfect completeness and soundness error $\leq n/(|\mathbb{F}| - n)$
- Π_{ordRoW} has perfect completeness and soundness.

Completeness. First, observe that the definition of $\mathbf{g}'$ implies that $g_1 = 1$, so $\mathcal{V}$'s first test passes. Now, it suffices to show that all sub-proofs have arguments that are in their respective languages. Per the definition of $\mathbf{g}'$, $\Pi_{\neq 0}$'s argument is in $\mathcal{L}_{\neq 0}$. Per the construction of tr' as a sort of tr, it is a permutation of tr. Per the construction of tr' as an (a,t)-sort of tr and per the uniqueness of the t values, if for some i, $a_i = a_{i-1}$, then $t_i > t_{i-1}$. Thus,$(\mathbf{a}', \mathbf{g}') \in \mathcal{L}_{\mathsf{c\text{-}ord}}$. Finally, since tr respects read-over-write (i.e., $\mathsf{tr} \in \mathcal{L}_{\mathsf{Vmem}}$), its (a,t)-ordering tr', together with $\mathbf{g}'$, is in $\mathcal{L}_{\mathsf{ordRoW}}$.

Soundness. Suppose that tr does not respect read-over-write. Then, there exist $i < j$ such that (a) $a_i = a_j$, (b) $w_j = 0$, (c) $v_i \neq v_j$, and (d) there is no intermediate k between i and j such that $a_k = a_i$. Now, if tr' is not a permutation of tr', then Π_{perm} fails except with probability $3n/|\mathbb{F}|$ (note: we instantiate Π_{perm} with $k = 4$). Assume it is a permutation. Now, if $\Pi_{\mathsf{c\text{-}uniq}}$ and $\Pi_{\mathsf{c\text{-}ord}}$ do not fail, then tr' is a-grouped and within groups strictly t-ordered, except with probability $\leq n/(|\mathbb{F}| - n)$. Assume they are. Then, since there is no intermediate k, acc_i and acc_j are adjacent in tr', say the former is at index i'. Then we have $a'_{i'} = a'_{i'+1}$, so $g'_{i'+1} = 0$ (or $\Pi_{\neq 0}$ would fail). Also we have $w'_{i'+1} = 0$. And finally, we have $v'_{i'} = v'_{i'+1}$. These three facts imply that Π_{ordRoW} fails.

All of this was except with probability

$$\leq \frac{3n}{|\mathbb{F}|} + \frac{n}{|\mathbb{F}| - n}$$

so that is the soundness error of Π_{Vmem}. □

C.2. Prior volatile memory proofs

We now summarize how prior AIPs for memory can be created by replacing different sub-proofs in our protocols with alternatives. First, we discuss alternative permutation proofs. Then, we discuss alternative range proofs, which are used to show the ordering of tr'.

C.2.1. Permutation. TinyRAM [45] gives a different volatile memory proof, in part by using a different permutation proof that does not require $\mathcal{V}$ randomness. It is based on a routing network [102]. In it, the rows of A and B are connected by a network of crossbar switches which can be programmed to implement any permutation. $\mathcal{P}$ sends the switch settings. Then, $\mathcal{V}$ checks the permutation. TinyRAM specifically uses the Waksman network [59], which has size $\Theta_k(n \log n)$, engendering $\mathcal{V}$ cost $\Theta(kn \log n)$.

C.2.2. Ordering and ranges. We show that the permuted transcript tr' is *grouped* by a (and then ordered by t). Prior works show that it is *ordered* by (a,t). To show the micro-scale ordering by t, it suffices to use $\Pi_{\mathsf{c\text{-}ord}}$ (just as we do). To show the macro-scale ordering by a, prior works show that the a sequence is monotonic. That is, that $a_1, \ldots, a_A \in [N]$ is in the language:

$$\mathcal{L}_{\mathsf{mon}} = \{\mathbf{a} : \forall i \in [A-1],\ a_{i+1} \geq a_i\}$$

Since all the a_i are known to be in range $[N]$, and since $N < \mathrm{char}(\mathbb{F})$, this reduces to arguing that the differences $d_i = a_{i+1} - a_i \in \{0, \ldots, N\} \subset \mathbb{F}$. That is, to give range proof:

$$\mathcal{L}_{\mathsf{range}} = \{(\mathbf{d} \in \mathbb{F}^A, N) : \forall i \in [A],\ 0 \leq d_i \leq N\}$$

Alternatively, one can give a range proof for the interval $0 \leq d_i < n$, where n is any integer satisfying $Nn < \mathrm{char}(\mathbb{F})$ and $k > n$. By setting n to be the next power of two, Ben-Sasson et al. give a range-proof $\Pi_{\mathsf{range,bit}}$ that uses bit-splitting and has $\mathcal{V}$ complexity $\Theta(A \log N)$ [45]. This is the approached used in TinyRAM.

Another line of works give range proofs based on *lookup tables*. Bootle et al. give a range-proof $\Pi_{\mathsf{range,arya}}$ that implements a lookup table using a permutation proof> The total $\mathcal{V}$ cost is $3(N + A) + O(1)$ [32]. This is the approach used in Arya. This approach is great if N is small, but if $N \gg A$ (e.g., as when emulating a machine with a 64-bit address space) then it is abysmal.

Haböck gives a proof $\Pi_{\mathsf{range,log}}$ that improve the constants of $\Pi_{\mathsf{range,arya}}$ (in I-R1CS) to $N + A + O(1)$ [34], by using logarithmic derivatives. This is the approached used in the "Haböck" volatile memory proof that we benchmark. It is also the approached used in our implementation of $\Pi_{\mathsf{c\text{-}ord}}$.

To mitigate the large-N costs, one can hybridize lookup-tables and bit-decomposition. In this approach, $\mathcal{P}$ shows that d_i can be represented as $\log_k(N)$ digits in the range $\{0, \ldots, k-1\}$, and range-checks the digits using lookups The cost is $\Theta(k + A \log_k N)$ (from here on, we ignore rounding). As we will see, by setting k appropriately, one can get good asymptotic performance, even for large N. Our

experiments (Section 9.1) confirm that concrete performance is also good. In the rest of this paper, we assume use the names $\Pi_{\mathsf{range,arya}}$ and $\Pi_{\mathsf{range,log}}$ to refer to the hybridized versions of the protocols, with k set optimally.

As applied to tables of consecutive integers, this hybridization approach has been explored by recent works [33, 35, 36, 39], and generalized to tables with "Spark-only structure" [41] by Setty et al. There are applications to optimized zkSNARKs of virtual machine execution [103].

Theorem 10. *With k set optimally, $\Pi_{\mathsf{range,arya}}$ and $\Pi_{\mathsf{range,log}}$ have $\mathcal{V}$ complexity that is both $\Theta_A(\log N/(\log\log N))$ and $\Theta_N(A)$.*

Proof. First, we characterize the optimal k. Then, we compute the $\mathcal{V}$ complexity. We will reach a point where it is difficult to proceed with a simultaneous analysis in large N or large A, so we will settle for two different asymptotic bounds.

To find the optimal k, we annihilate the derivative:

$$\begin{aligned} 0 &= 1 - A(\ln N)(\ln k)^{-2}k^{-1} \\ k\ln^2 k &= A\ln N \\ k &= e^{2W_0\left((1/2)\sqrt{A\ln N}\right)} \end{aligned}$$

where $W_0(x)$ is the principal branch of the Lambert W function. So, our cost is:

$$e^{2W_0\left((1/2)\sqrt{A\ln N}\right)} + \frac{A(\ln N)}{2W_0\left((1/2)\sqrt{A\ln N}\right)}$$

This is a somewhat challenging expression to analyze asymptotically in two variables, as W_0's argument depends on both. And, while we have fairly simple asymptotic bounds on W_0 [104], its precise behavior is more complex. So, we settle for asymptotic bounds that depend on N or A but not both.

We begin with the fixed A case. The argument x to $W_0(x)$ is large, so we can apply a result of Hoorfar and Hassani: that $W_0(x) = \ln x - \ln\ln x + \Theta((\ln\ln x)/(\ln x))$ [104]. Since k is set optimally, it suffices to analyze either summand in the cost expression. We choose the second, and substitute the asymptotics for W_0:

$$\Theta_A\left(\frac{\ln N}{2\ln(\sqrt{\ln N}) + o(\ln(\sqrt{\ln N}))}\right) = \Theta_A\left(\frac{\ln N}{\ln\ln N}\right)$$

Now, we consider the other case: when N is fixed. It's easy to see that cost is $\Theta_N(A)$ in this case. □

C.3. Persistent memory

Protocol 12 is our persistent memory proof. It is very similar to our volatile memory proof, so we only explain the differences. First, both parties initially associate each access in tr with a time $t_i \leftarrow i$ *and* a flag $g_i \leftarrow 0$. Second, $\mathcal{P}$ produces a vector $\mathbf{a}$ of distinct addresses that include all those touched by tr, as well as the initial and final values at those addresses $\mathbf{v}$ and $\mathbf{v}'$. Third, both parties augment the transcript with initialization entries $(v_i, a_i, g_i = 1, w_i = 1, t_i = 0)$ and with finalization entries $(v'_i, a_i, g_i = 0, w_i = 0, t_i = A+1)$. Fourth, $\mathcal{P}$ sends the sorted transcript tr' to $\mathcal{V}$. Then, the parties engage in sub-protocol Π_{active}, to show that the active memory cells that are represented by $(\mathbf{v}, \mathbf{v}', \mathbf{a})$ are consistent with $(\mathbf{m}, \mathbf{m}')$. Finally, the parties use the same sub-protocols as in Π_{Vmem} to show that tr' (and by extension tr) respect read-over-write.

$\Pi_{\mathsf{Pmem}}(\mathbf{m}, \mathbf{m}' \in \mathbb{F}^N; \mathsf{tr})$

$\mathcal{P}(\cdots)$		$\mathcal{V}(\cdots)$
..... include $t_i \leftarrow i, g_i \leftarrow 0$ in tr		
$\mathbf{a} \leftarrow A$ addrs covering tr		
$\mathbf{v} \leftarrow (m_{a_i})_{i=1}^A$		
$\mathbf{v}' \leftarrow (m'_{a_i})_{i=1}^A$	$\xrightarrow{\mathbf{v}, \mathbf{v}', \mathbf{a}}$	
..... $\tilde{\mathsf{tr}} \leftarrow \mathsf{tr} \parallel (\mathbf{v}, \mathbf{a}, \mathbf{g} = \mathbf{1}, \mathbf{w} = \mathbf{1}, \mathbf{t} = \mathbf{0}) \parallel (\mathbf{v}', \mathbf{a}, \mathbf{g} = \mathbf{0}, \mathbf{w} = \mathbf{0}, \mathbf{t} = A+1)$		
$\mathsf{tr}' \leftarrow \mathsf{sort}_{(a,t)}(\tilde{\mathsf{tr}})$	$\xrightarrow{\mathsf{tr}'}$	
..... $(\mathbf{v}', \mathbf{a}', \mathbf{g}', \mathbf{w}', \mathbf{t}') \leftarrow \mathsf{tr}'$		
..... $\Pi_{\mathsf{active}}(\mathbf{m}, \mathbf{m}', \mathbf{v}, \mathbf{v}', \mathbf{a})$		
..... $\Pi_{\mathsf{perm}}(\tilde{\mathsf{tr}}, \mathsf{tr}')$		
..... $\Pi_{\mathsf{c\text{-}ord}}(\mathbf{t}', \mathbf{1} - \mathbf{g}', 0, A+1)$		
..... $\Pi_{\mathsf{c\text{-}uniq}}(\mathbf{a}', \mathbf{g}')$		
..... $\Pi_{\mathsf{ordRoW}}(\mathbf{g}', \mathbf{v}', \mathbf{w}')$		

Protocol 12: Our persistent memory proof.

Theorem 11. *The protocol Π_{Pmem} (for language $\mathcal{L}_{\mathsf{Pmem}}$) has perfect completeness and soundness error $\frac{N^2+6N+A}{|\mathbb{F}|} + \frac{N}{|\mathbb{F}|-N}$*

Proof. We address completeness and then soundness.

Completeness. First, we argue that Π_{active} succeeds. Per the construction of $\mathbf{v}$ and $\mathbf{v}'$, for all $i \in [A]$, $v_{a_i} = m_i$ and $v'_{a_i} = m'_i$. Further, since all $i \notin \mathbf{a}$ do not appear in tr, $m_i = m'_i$. Finally, the elements of $\mathbf{a}$ are unique by construction. Thus, $(\mathbf{m}, \mathbf{m}', \mathbf{v}, \mathbf{v}', \mathbf{a}) \in \mathcal{L}_{\mathsf{active}}$, so Π_{active} (which is perfectly complete) succeeds.

Now, it suffices to show that $\tilde{\mathsf{tr}} \in \mathcal{L}_{\mathsf{Vmem}}$ (excluding the g flags), and that tr' in Π_{Pmem} has g flags that have the same values as those in Π_{Vmem}. If these conditions are met, then the completeness of Π_{Vmem} implies that of Π_{Pmem}. First, observe that after sorting by (a, t), the first access to each address has form $(v_i, b_i, g = 1, w = 1, t = 0)$ in $\tilde{\mathsf{tr}}$, since these addresses have minimal t among all the addresses in $\tilde{\mathsf{tr}}$. Second, observe that every other access in tr' is a standard address or a finalization address, so it has $g = 0$. Thus our second condition is met. The first condition follows from the fact that $\mathsf{valid}(\mathbf{m}, \mathbf{m}', \mathsf{tr})$ holds.

Soundness. Suppose that $\mathsf{valid}(\mathbf{m}, \mathbf{m}', \mathsf{tr})$ does not hold. Then, for some address $a \in [N]$, one of the following must hold:

1) The first access to a in tr is a read whose value disagrees with $\mathbf{m}$. There are two sub-cases:
   a) a is equal to some $a_i \in \mathbf{a}$ with $v_i = m_{a_i}$, or

b) not.

2) There are accesses to a indexed by $i < j$ with no intermediate access to a such that $w_j = 0$ and $v_i \neq v_j$.
3) the last access to a in tr has a value that disagrees with $\mathbf{m}'$. There are two sub-cases:
   a) a is equal to some $a_i \in \mathbf{a}$ with $v'_i = m'_{a_i}$, or
   b) not.

If any of cases 1a, 2, or 3a hold, then the soundness of Π_{Vmem} implies that $\mathcal{V}$ rejects except with probability

$$\leq \frac{4N}{|\mathbb{F}|} + \frac{N}{|\mathbb{F}| - N}$$

This bound is slightly looser than Theorem 9 because Π_{perm} is applied to 5-tuples here, instead of to 4-tuples as in Π_{Vmem}.

In case 1b or 3b, we have that $(\mathbf{m}, \mathbf{m}', \mathbf{v}, \mathbf{v}', \mathbf{a})$ is not in $\mathcal{L}_{\mathsf{active}}$. Thus, Theorem 5 implies that $\mathcal{V}$ rejects except with probability $\leq (N^2 + 2N + A)/|\mathbb{F}|$.

Thus, $\mathcal{V}$ rejects except with probability at most

$$\frac{N^2 + 6N + A}{|\mathbb{F}|} + \frac{N}{|\mathbb{F}| - N}$$

□

C.4. Sparse persistent memory

Protocol 13 is our sparse persistent memory proof, building on Protocol 14 for extracting the active cells from the memory. It is very similar to Protocol 12, so we explain only the differences. First, the vectors $\mathbf{a}, \mathbf{a}'$ now denote part of the persistent storage, while $\mathbf{b}'$ denotes the "active" addresses. Second, the active addresses $\mathbf{b}'$ are computed slightly differently: $\mathbf{b}'$ must now contain elements that are a sub(multi)set of those in $\mathbf{a}'$, including all the addresses that appear in tr. Protocol 13 continues by building an augmented transcript $\tilde{\mathsf{tr}}$ that contains initialization and finalization entries for each address, as in Π_{Pmem}. It then runs the same collection of sub-protocols that Π_{Pmem} uses, save that it runs $\Pi_{\mathsf{s\text{-}active}}$ in place of Π_{active}.

Checking the consistency of the $(\mathbf{m}, \mathbf{m}', \mathbf{a}, \mathbf{a}', \mathbf{v}, \mathbf{v}', \mathbf{b}')$ tuple is deferred to the sub-protocol $\Pi_{\mathsf{s\text{-}active}}$ which was explained at a high level in Section 6. It is detailed as Protocol 14. At a high level, it shows that (a) each (b'_i, v'_i) is equal to some (a'_i, m'_i), (b) each (b'_i, v_i) is equal to some (a_i, m_i) or a_i is a dummy, and (c) the a'_i are all unique or dummies, assuming that the a_i are. More precisely, it proves membership in the following language:

$$\begin{aligned}
\mathcal{L}_{\mathsf{s\text{-}active}} = \{ (\mathbf{m}, \mathbf{m}', \mathbf{a}, \mathbf{a}' \in \mathbb{F}^N; \mathbf{v}, \mathbf{v}', \mathbf{b}' \in \mathbb{F}^A) : \\
\exists\, I \subseteq [C], |I| = C - A, \exists\, \mathbf{b} \in \mathbb{F}^A, \\
\textstyle\bigwedge_{i=1}^{A} (b_i = 0 \vee b_i = b'_i) \wedge \\
\{\!\{(a_i, m_i)\}\!\}_{i=1}^{C} = \{\!\{(a_i, m_i)\}\!\}_{i \in I} \uplus \{\!\{(b_i, v_i)\}\!\}_{i=1}^{A} \wedge \\
\{\!\{(a'_i, m'_i)\}\!\}_{i=1}^{C} = \{\!\{(a'_i, m'_i)\}\!\}_{i \in I} \uplus \{\!\{(b'_i, v'_i)\}\!\}_{i=1}^{A} \wedge \\
\{a_i\}_{i=1}^{C} \cap \{b'_i - b_i\}_{i=1}^{A} \subseteq \{0\} \}
\end{aligned}$$

$\Pi_{\mathsf{SPmem}}(\mathbf{m}, \mathbf{m}', \mathbf{a}, \mathbf{a}' \in \mathbb{F}^C; \mathsf{tr})$

$\mathcal{P}(\cdots)$		$\mathcal{V}(\cdots)$
..... include $t_i \leftarrow i$ in tr		
$\mathbf{b}' \leftarrow A$ addr.s covering tr		
$\mathbf{v} \leftarrow (m_{b'_i})_{i=1}^{A}$		
$\mathbf{v}' \leftarrow (m'_{b'_i})_{i=1}^{A}$	$\mathbf{v}, \mathbf{v}', \mathbf{b}'$ →	
$\tilde{\mathsf{tr}} \leftarrow \mathsf{tr} \parallel (\mathbf{v}, \mathbf{b}', \mathbf{g} = \mathbf{1}, \mathbf{w} = \mathbf{1}, \mathbf{t} = \mathbf{0}) \parallel (\mathbf{v}', \mathbf{b}', \mathbf{g} = \mathbf{0}, \mathbf{w} = \mathbf{0}, \mathbf{t} = A + 1)$		
$\mathsf{tr}' \leftarrow \mathsf{sort}_{(a,t)}(\tilde{\mathsf{tr}})$	tr' →	
..... $(\mathbf{v}', \mathbf{b}', \mathbf{g}', \mathbf{w}', \mathbf{t}') \leftarrow \mathsf{tr}'$		
..... $\Pi_{\mathsf{s\text{-}active}}(\mathbf{m}, \mathbf{m}', \mathbf{a}, \mathbf{a}', \mathbf{v}, \mathbf{v}', \mathbf{b}')$		
..... $\Pi_{\mathsf{perm}}(\tilde{\mathsf{tr}}, \mathsf{tr}')$		
..... $\Pi_{\mathsf{c\text{-}ord}}(\mathbf{t}', \mathbf{1} - \mathbf{g}', 0, A + 1)$		
..... $\Pi_{\mathsf{c\text{-}uniq}}(\mathbf{b}', \mathbf{g}')$		
..... $\Pi_{\mathsf{ordRoW}}(\mathbf{g}', \mathbf{v}', \mathbf{w}')$		

Protocol 13: Our sparse persistent memory proof.

$\Pi_{\mathsf{s\text{-}active}}(\mathbf{m}, \mathbf{m}', \mathbf{a}, \mathbf{a}' \in \mathbb{F}^C; \mathbf{v}, \mathbf{v}', \mathbf{b}' \in \mathbb{F}^A)$

$\mathcal{P}(\cdots)$		$\mathcal{V}(\cdots)$
find suitable $\mathbf{j} \in [C]^A$		
$\mathbf{b} \leftarrow (a_{j_i} : i \in [A])$	$\mathbf{b}$ →	
	← β	sample $\beta \in \mathbb{F}$
$\mathbf{h} \leftarrow (m_i + a_i \beta)_{i \in [C], i \notin \mathbf{b}'}$	$\mathbf{h} \in \mathbb{F}^{C-A}$ →	sample $\alpha \in \mathbb{F}$
..... $\Pi_{\mathsf{PuncDisj}}(\mathbf{a}, \mathbf{b}' - \mathbf{b})$		
		$\forall i \in [A] : (b'_i - b_i) b_i \stackrel{?}{=} 0$
		$c \leftarrow \prod_{i=1}^{C-A} (\alpha + h_i)$
		$\prod_{i=1}^{C} (\alpha + m_i + a_i \beta) \stackrel{?}{=} c \times \prod_{i=1}^{A} (\alpha + v_i + b_i \beta)$
		$\prod_{i=1}^{C} (\alpha + m'_i + a_i \beta) \stackrel{?}{=} c \times \prod_{i=1}^{A} (\alpha + v'_i + b'_i \beta)$

Protocol 14: Extracting the active cells of a sparse memory.

In this protocol, $\mathcal{P}$ starts by computing a persistent cell index j_i for each active cell i. $\mathbf{j}$ must contain distinct elements, for each $i \in [A]$, $a'_{j_i} = a'_{j_i}$, and $\mathbf{j}$ should be lexicographically minimal subject to these conditions. $\mathcal{P}$ can compute such a $\mathbf{j}$ quickly by setting each j_i to the small elements of $[C]$ satisfying $b'_i = a'_{j_i}$ that is not in $\mathbf{j}$ already. Then,

Theorem 12. *The protocol $\Pi_{\mathsf{s\text{-}active}}$ (for language $\mathcal{L}_{\mathsf{s\text{-}active}}$) has perfect completeness and soundness error $(C^2 + 3C + 2A - 1)/|\mathbb{F}|$*

Proof. We address completeness and then soundness.

Completeness. Define I to be $[C]$ with the items of $\mathbf{j}$ removed. Per membership in $\mathcal{L}_{\mathsf{s\text{-}active}}$, I and $\mathbf{b}$ exist. Moreover, one can show that $\mathbf{b}$ in this proof is equal to $\mathbf{b}$ in the protocol. The last condition of membership immediately implies that Π_{PuncDisj} succeeds. Also, the first condition of

membership ($b_i = 0 \vee b'_i = b_i$ for all $i \in [A]$) implies that $\mathcal{V}$'s first test succeeds. Let J be the set of $\mathbf{j}$'s contents, and $I = [C] \setminus J$.

Consider the multisets

$$\{\!\{m_i + a_i\beta\}\!\}_{i\in[C]} = \{\!\{m_i + a_i\beta\}\!\}_{i\in I} \uplus \{\!\{m_i + a_i\beta\}\!\}_{i\in J}$$

So, we have

$$\prod_{i\in[C]} (\alpha + m_i + a_i\beta) = \prod_{i\in I} (\alpha + m_i + a_i\beta) \times \prod_{i\in J} (\alpha + m_i + a_i\beta)$$

. The factors in the second product are exactly the $\alpha + h_i$, as computed by $\mathcal{P}$, so we have

$$\prod_{i\in[C]} (\alpha + m_i + a_i\beta) = c \times \prod_{i\in J} (\alpha + m_i + a_i\beta)$$

Then, by the definition of the $\mathbf{b}$ and $\mathbf{b}$, we have

$$\prod_{i\in[C]} (\alpha + m_i + a_i\beta) = c \times \prod_{i\in J} (\alpha + v_i + b_i\beta)$$

and $\mathcal{V}$'s second test succeeds. And, a similar argument shows that $\mathcal{V}$'s third test succeeds.

Soundness. Suppose the $\mathcal{V}$ accepts. We will argue that the instance is in the language. Because of $\mathcal{V}$'s last tree tests, β is independent of $\mathbf{m}, \mathbf{m}', \mathbf{a}, \mathbf{a}', \mathbf{v}, \mathbf{v}', \mathbf{b}, \mathbf{b}'$, and α is independent of all those and $\mathbf{h}$ too, we have two multiset equations:

$$\{\!\{(a_i, m_i)\}\!\}_{i\in[C]} = \{\!\{(a_i, m_i)\}\!\}_{i\in I} \uplus \{\!\{(b_i, v_i)\}\!\}_{i\in[A]}$$
$$\{\!\{(a'_i, m'_i)\}\!\}_{i\in[C]} = \{\!\{(a_i, m_i)\}\!\}_{i\in I} \uplus \{\!\{(b'_i, v'_i)\}\!\}_{i\in[A]}$$

for some $I \subseteq [C]$, except with probability $(C^2+2C+A)/|\mathbb{F}|$ by the same argument as in the proof of Theorem 5, with C in place of N. This satisfies the middle two conditions of $\mathcal{L}_{\text{s-active}}$. The first condition is ensured by $\mathcal{V}$'s first test, and the last condition is ensured by Π_{PuncDisj}, except with probability (Thm. 7): $(C + A - 1)/|\mathbb{F}|$ Thus, the overall soundness error is

$$\frac{C^2 + 3C + 2A - 1}{|\mathbb{F}|}$$

□

Theorem 13. *The protocol Π_{Pmem} (for language $\mathcal{L}_{\mathsf{Pmem}}$) has perfect completeness and soundness error $O((C^2 + A)/|\mathbb{F}|)$*

Proof. The proof of this theorem is similar to the proof of Theorem 11. □

Appendix D. Computing Bézout polynomials

Figure 2 gives our algorithm UniqBez. Like SimpleUniqBez, its runtime is $O(M(L)\log L)$ and its is easy to parallelize. In addition, we implement one optimization not shown. In the first two loops, we precompute all $\mathsf{rev}(p_{j,i}) \bmod X^{2^{\ell-j+1}}$ [84, Ex. 10.9], where rev reverses the coefficients of a polynomial. These values can be used to accelerate the modular reductions in the third loop. Now, we restate and prove Theorem 4.

UniqBez$(x_1, \ldots, x_{2^\ell})$

for $i \in [2^\ell]$:
  $p_{\ell,i} \leftarrow X - x_i$
for $j \in \{\ell, \ldots, 1\}, i \in [2^{j-1}]$:
  $p_{j-1,i} \leftarrow p_{j,2i} \cdot p_{j,2i+1}$
$q_{0,0} \leftarrow p'_{0,0}$
for $j \in [\ell], i \in [2^{j-1}]$:
  $q_{j,2i} \leftarrow q_{j-1,i} \bmod p_{j,2i}$
  $q_{j,2i+1} \leftarrow q_{j-1,i} \bmod p_{j,2i+1}$
for $i \in [2^\ell]$: $t_{\ell,i} \leftarrow (q_{\ell,i})^{-2}$
for $j \in \{\ell, \ldots, 1\}, i \in [2^{j-1}]$:
  $p_{j-1,i} \leftarrow t_{j,2i} \cdot p_{j,2i+1} + t_{j,2i+1} \cdot p_{j,2i}$
return $s \leftarrow (1 - p'_{0,0} \cdot t_{0,0})/p_{0,0}, t \leftarrow t_{0,0}$

Algorithm 2: Our algorithm for computing Bézout polynomials s, t for $z(X) = \prod_{i=1}^{2^\ell}(X - x_i)$ and its derivative.

Theorem. *Given $a_1, \ldots, a_A \in \mathbb{F}$,* UniqBez *outputs $s, t \in \mathbb{F}[X]$ satisfying $sz + tz' = 1$ (where $z = \prod_i (X - a_i)$) in time at most $4.5A\log^2 A + O(A \log A)$.*[2]

Proof. Let $z(X) = \prod_i (X - x_i)$. First, we prove UniqBez returns s, t satisfying $zs + z't = 1$. First, observe that $p_{0,0} = z$. Next, observe that $q_{\ell,i}$ is a scalar and is $z'(x_i)$. Next, observe that for each x_i, $1 = z(x_i)s(x_i) + z'(x_i)t(x_i) = z'(x_i)t(x_i)$. Thus, $t(x_i) = (z'(x_i))^{-1}$.

Now, observe that $t(X) = \sum_{i=1}^{L} (t(x_i))^2 \prod_{j\neq i}(X - x_i)$. This is because $t(x_i)\prod_{j\neq i}(X - x_i)$ is the l

Now, we analyze the runtime of UniqBez. As is the convention in computer algebra, we quantify runtime as the number of scalar operations, and we use the function $M(L)$ to denote the (asymptotic) cost of multiplying two polynomials of degree $< L$; this is $L \log L$. We will see that UniqBez has runtime $4.5M(L)\log L + O(L \log L)$. To see this, observe that UniqBez is very similar to the fast interpolation algorithm given by Gathen and Gerhard [84, Alg. 10.11]. The only difference is that UniqBez uses the $(q_{\ell,i})^{-1}$ as both the Lagrange coefficients *and the target evaluations*, instead of having a separate list $\mathbf{v} \in \mathbb{F}^L$ of target evaluations. Thus, the runtime of UniqBez is the same as the runtime of interpolation.[8] And that runtime is $4.5M(L)\log L + O(L \log L)$ [84, Ex. 10.9]. □

[8] UniqBez is very slightly faster, because it replaces $O(n)$ multiplications between Lagrange coefficients and the v_i with $O(n)$ squarings—but this is a low-order difference.