Spartic: A Sparse Polynomial Commitment Scheme from Lattices

Ihyun Nam and Dan Boneh

Stanford University

November 29, 2024

Abstract

Sparse polynomial commitment schemes allow a prover to commit to a polynomial that evaluates to zero at most locations in its domain and later prove its evaluations. These schemes are a research interest for their ability to handle large, sparse structures efficiently unlike dense polynomial commitment schemes, and have enjoyed several constructions based on polynomials in finite fields. However, we are yet to see a lattice-based construction that works on polynomials defined over rings. In this paper we present Spartic, the first sparse polynomial commitment scheme from standard lattice assumptions with an optimal prover cost that is linear in the sparsity of the polynomial. Much like in the sparse polynomial commitment scheme used in the Lasso lookup arguments (Eurocrypt 2024), at the heart of our scheme is reducing a sparse polynomial to its unique dense representation and having the prover commit to it using any dense polynomial commitment scheme for lattices. Then in the evaluation phase, the prover proves evaluations of the sparse polynomial with respect to its dense representation using the offline memory checking technique of Spartan (CRYPTO 2020). The difficulty here is achieving negligible soundness error in the presence of complicating factors, like zero divisors, in rings that make tools like the linear sum-check protocol used in prior constructions not directly applicable. We adapt the core techniques of Spark to the ring domain.

1 Introduction

A polynomial commitment scheme [KZG10] is a cryptographic primitive that allows a prover to commit to a degree-bound polynomial $f \in R^{\leq d}[X_1, \cdots, X_\mu]$ over a ring R and μ variables and later convince a verifier of its evaluations in the form of $f(\mathbf{x}) = y$ for some public values $\mathbf{x} \in \mathcal{R}^\mu$ and $y \in \mathcal{R}$. A *sparse* polynomial $f' \in R^{\leq d}[X_1, \cdots, X_\mu]$ with sparsity $m \in \mathbb{N}$ evaluates to a non-zero value at at most m points in its domain. In other words, f' has at most m non-zero coefficients in the so-called multilinear Lagrange polynomial basis. A sparse polynomial commitment scheme commits to such polynomials efficiently by leveraging the polynomials' sparsity. Therefore, sparse polynomial commitment schemes have natural applications in constructing look-up arguments [STW24], succinct non-interactive arguments of knowledge [BDFG20, BSU24, GNS24], or multi-party computation [BHV+23, ABGK22].

Among several ways to build a polynomial commitment scheme, lattice- and hash-based approaches can achieve plausible quantum resistance, in preparation of the post-quantum world. Between those, lattice cryptography is deemed especially attractive as standard lattice assumptions are stronger than hash-based assumptions (i.e., security of hash functions). In the past few years, lattice-based polynomial commitment schemes have witnessed a progress in achieving concrete efficiency [NS24, HSS24, CMNW24, FMN23] from standard lattice assumptions. For instance, [NS24] introduced the Greyhound commitment scheme that achieved the first concretely efficient sublinear verifier time and small proof sizes, by using the LaBRADOR [BS22] proof system as a black box. This was an improvement from previous constructions [ACL+22, BCFL23, WW23] that could render succinctness of commitments and proofs and efficient verifier time only if the evaluation point was known before evaluation time.

Sparse polynomial commitment schemes are a related yet independent research interest, often pursued as part of a larger primitive that needs to optimally commit to sparse structures. For instance, Spark is used in the Spartan zkSNARK [Set19] to prove evaluations of sparse multilinear extensions over a finite field. Spark achieves optimal prover costs that depends only on the sparsity of the polynomial. While Spark assumes an honest prover at commitment time, [STW24] proves that Spark remains secure even without the assumption. This way, Spark in Lasso represents the first 'standard' commitment scheme for sparse multilinear polynomials. It is then generalized and used to build the Lasso lookup arguments.

However, at the intersection of these two research interests, there has been no known *sparse* polynomial commitment schemes built from *lattices*. Therefore we ask the following question: *Can we build sparse polynomial commitment schemes from lattices, and do so efficiently?* In this paper, we answer this in the affirmative by presenting Spartic, the first sparse polynomial commitment scheme from lattices with optimal prover costs that are linear in the sparsity of the committed polynomial. In the commit phase, we propose that we can make a black-box use of any dense lattice-based multilinear polynomial commitment scheme. Spartic may also use a lattice-based univariate polynomial commitment scheme, but with an additional overhead incurred by the general transformation [CBBZ23, ZXZS19, BCHO22] from a polynomial commitment scheme supporting univariate polynomials to one supporting multilinear polynomials. In the evaluation phase, we adapt several techniques from [Set19] and [STW24] to the ring domain. Spartic assumes an untrusted prover and has perfect completeness and knowledge soundness.

1.1 Technical Overview

Suppose an untrusted prover wishes to commit to an m-sparse multilinear polynomial $g \in \mathcal{R}_q$ defined over $\log N = 2 \log m$ variables. In our sparse polynomial commitment scheme, the prover does so by committing to the unique dense representation of g, which are a few $(\log m)$-variate multilinear polynomials. These polynomials can also be naturally transformed into a list of all of the monomials of g with a non-zero coefficient (and the corresponding coefficient). When the verifier requests an evaluation $g(\mathbf{r})$ of g at some $\mathbf{r} \in \mathcal{R}^\mu$, the prover runs a time-optimal algorithm from [Set19] to output $g(\mathbf{r})$. Then, the evaluation proof of our sparse polynomial commitment scheme amounts to proving a correct execution of the time-optimal algorithm using the committed dense polynomials. To do so, the prover uses an offline memory checking methods of [BEG+91] using an untrusted memory, that effectively forces the prover to commit to the execution trace of the algorithm. The verifier then checks that the time-optimal algorithm was correctly ran to output $g(\mathbf{r})$. The verifier has an oracle access to the memory reads returned from the untrusted memory and runs appropriate verification to check the returned memory values are correct. For any $c \in \mathbb{N}$, if c memories of size $N^{1/c}$ are used for memory checking, the cost of the prover is dominated by committing to $(3c + 1)$ many dense multilinear polynomials over $\log m$ many variables and c many dense multilinear polynomials over $\log(N^{1/c})$ many variables.

2 Preliminaries

Notation. Let λ denote the security parameter. For $n \in \mathbb{N}$, let $[n]$ be the set $\{1, 2, \cdots, n\}$. Let $\mathcal{R}$ be the ring $\mathbb{Z}/(X^d + 1)$ and $\mathcal{R}_q$ be the ring $\mathbb{Z}_q/(X^d + 1)$ for some power of two d and integer q. We use $\mathcal{R}^{\leq d}[X_1, \cdots, X_\mu]$ to mean the set of μ-variate polynomials in $\mathcal{R}$ defined over the variables $X_1, \cdots, X_\mu$ where the degree of each variable is at most d. Let B_μ denote the μ-dimensional boolean hypercube. If an element s is drawn randomly from a set S, we write either $s \xleftarrow{\$} S$ or $s \in_\$ S$, depending on the context. We write vectors with bold letters and individual elements of a vector using subscripts. For instance, the ith element of a vector $\mathbf{x}$ is denoted $\mathbf{x}_i$.

2.1 Sum-checks and Multilinear Extensions over Rings

Definition 2.1 (Sampling Sets and Strong Sampling Sets [CCKP19])**.** For an arbitrary ring $\mathcal{R}$, a subset $\mathcal{C}$ of $\mathcal{R}$ is a sampling set if the difference of any two distinct elements in $\mathcal{C}$ is not a zero divisor. $\mathcal{C}$ is further a *strong* sampling set if the difference is also invertible.

Observe that in $\mathcal{R}_q$ with a strong sampling set, we are also guaranteed to have a (non-strong) sampling set, the most trivial example of which is simply the strong sampling set itself. [BC24] shows that the ring $\mathcal{R}_q$ has a strong sampling set $\mathcal{C}_{strong}$ of exponential size. The following lemma follows naturally.

Lemma 2.2. *Let $\mathcal{C}_{strong}$ be a strong sampling set in $\mathcal{R}_q$ with an exponential size and let $\mathcal{C}$ be some sampling set such that $\mathcal{C}_{strong} \subseteq \mathcal{C}$. Then, $\mathcal{C}$ also has an exponential size in $\mathcal{R}_q$.*

Definition 2.3 (Multilinear Extensions over Rings)**.** Let $\mathcal{R}$ be an arbitrary ring with zero 0 and identity 1. Given a function $f : \{0, 1\}^l \to \mathcal{R}$, we define the multilinear extension $\widetilde{f}$ of

f as

$$\widetilde{f}(\mathbf{x}) = \sum_{\mathbf{r}\in\{0,1\}^l} f(\mathbf{x}) \cdot \widetilde{eq}(\mathbf{r}, \mathbf{x})$$

where, for $\mathbf{x} = (x_1, \cdots, x_l)$ and $\mathbf{r} = (r_1, \cdots, r_l)$,

$$\widetilde{eq}(\mathbf{r}, \mathbf{x}) = \prod_{i=1}^{l} \big[(1 - r_i)(1 - x_i) + r_i x_i\big].$$

The set $\{\widetilde{eq}(\mathbf{r}, \mathbf{x}) : \mathbf{r} \in \{0,1\}^l\}$ are the Lagrange basis polynomials for l-variate multilinear polynomials.

Sparse polynomials. Given the definition of multilinear extensions over rings, we can define a sparse polynomial. The following two definitions are from [STW24].

Definition 2.4 (Dense Representation of a Polynomial)**.** For a multilinear polynomial $g \in \mathcal{R}$ in l variables, $\mathsf{DenseRepr}(g)$ is a list of tuples L such that for all $i \in \{0,1\}^l$, $(\mathsf{to\text{-}ring}(i), g(i)) \in L$ if and only if $g(i) \neq 0$, where $\mathsf{to\text{-}ring}$ is a canonical injection from $\{0,1\}^l$ to $\mathcal{R}$. $\mathsf{DenseRepr}(g)$ is unique to g because multilinear extensions over rings are unique.

Definition 2.5 (Sparse Polynomial)**.** A multilinear polynomial $g \in \mathcal{R}$ in l variables is a *sparse* polynomial if $|\mathsf{DenseRepr}(g)|$ is sublinear in $O(2^l)$. Otherwise, g is a *dense* polynomial.

Lemma 2.6 (Generalized Schwartz-Zippel [BCPS18])**.** *Let $f \in \mathcal{R}^{\leq d}[X_1, \cdots, X_\mu]$ be a μ-variate nonzero polynomial over a ring $\mathcal{R}$ with per-variable degree at most d. Let $\mathcal{C} \subseteq \mathcal{R}$ be a sampling set. Then we have $Pr_{\mathbf{r}\xleftarrow{\$}\mathcal{C}^\mu}[f(\mathbf{r}) = 0] \leq \frac{d\mu}{|\mathcal{C}|}$.*

Next, we recall that the sum-check protocol from [LFKN92] can be extended to work over a ring $\mathcal{R}$ using the generalized Schwartz-Zippel lemma.

Lemma 2.7 (Sum-check over Rings [CCKP19])**.** *Let $f \in \mathcal{R}^{\leq d}[X_1, \cdots, X_\mu]$ be a μ-variate nonzero polynomial over a ring $\mathcal{R}$ with per-variable degree at most d. Let $\mathcal{C}_{strong} \in \mathcal{R}$ be a strong sampling set. The protocol below checks that $s = \sum_{\mathbf{b}\in\{0,1\}^\mu} f(\mathbf{b})$ with soundness error $\frac{\mu d}{|\mathcal{C}_{strong}|}$.*

1. *In the i-th round for $1 \leq i < \mu$,*

   - *Upon receiving the challenges $r_1, \cdots, r_{i-1}$ from the previous rounds, the prover sends $d+1$ evaluations of the univariate polynomial h_i at $d+1$ points in $\mathcal{C}_{strong}$, where h_i is defined as*

   $$h_i(X) = \sum_{\mathbf{b}\in\{0,1\}^{\mu-i}} f(r_1, \cdots, r_{i-1}, X, \mathbf{b}) \in \mathcal{R}[X].$$

   - *Denote $h_0(r_0) = s$. The verifier checks that $h_i(0) + h_i(1) = h_{i-1}(r_{i-1})$, where $h_{i-1}(r_{i-1})$ is computed by Lagrange-interpolating the $d+1$ evaluations sent by the prover.*

- *Verifier sends a random challenge* $r_i \xleftarrow{\$} \mathcal{C}_{strong}$.

2. *Verifier checks that* $h_\mu(r_\mu) = f(r_1, \cdots, r_\mu)$.

2.2 Polynomial Commitment Scheme

We adapt the following definitions from [NS24].

Definition 2.8 (Polynomial Commitment Scheme)**.** The tuple of algorithms (Setup, Commit, Open, Eval) is a polynomial comitment scheme over a ring $\mathcal{R}$ with degree bound d if:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ on input a security parameter λ outputs public parameters pp,
- $\mathsf{Commit}(\mathsf{pp}, f) \to (C, \mathsf{st})$ on input public parameters pp and a message $f \in \mathcal{R}^{\leq d}$ outputs a commitment C and decommitment state st,
- $\mathsf{Open}(\mathsf{pp}, C, f, \mathsf{st}, c) \to \{0, 1\}$ on input public parameters pp, a commitment C, a message $f \in \mathcal{R}^{\leq d}$, a decommitment state st, and a relaxation factor $c \in \mathsf{SL}$ outputs 1 or 0 indicating whether C is a valid commitment to f under pp, and
- $\mathsf{Eval} := (\mathsf{Eval}.\mathcal{P}, \mathsf{Eval}.\mathcal{V})$ is a pair of probabilistic polynomial-time algorithms between $\mathsf{Eval}.\mathcal{P}(\mathsf{pp}, (C, x, y), (f, \mathsf{st}))$ the evaluation prover and $\mathsf{Eval}.\mathcal{V}(\mathsf{pp}, (C, x, y))$ the evaluation verifier.

Definition 2.9 (Perfect Completeness)**.** A polynomial commitment scheme PCS =(Setup, Commit, Open, Eval) is said to be *perfectly complete* if for every polynomial $g \in \mathcal{R}^{\leq d}$ and evaluation point $\mathbf{r} \in \mathcal{R}^\mu$, the following probability holds:

$$Pr\left[\begin{array}{c|c} \mathsf{Open}(\mathsf{pp}, C, f, \mathsf{st}, \mathsf{e}) = 0 \\ \vee b = 0 \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ C, \mathsf{st} \leftarrow \mathsf{Commit}(\mathsf{pp}, f) \\ \mathbb{x} := (C, x, f(x)), \quad \mathbb{w} := (f, \mathsf{st}) \\ (tr, b) \leftarrow \langle \mathsf{Eval}.\mathcal{P}(\mathsf{pp}, \mathbb{x}, \mathbb{w}), \mathsf{Eval}.\mathcal{V}(\mathsf{pp}, \mathbb{w}) \rangle \end{array}\right] = 1.$$

Definition 2.10 (Knowledge Soundness)**.** A polynomial commitment scheme PCS =(Setup, Commit, Open, Eval) is said to be *knowledge sound* with knowledge error ε if for all stateful probabilistic polynomial time (PPT) adversary $\mathcal{P}^*$, there exists an expected PPT extractor ϵ such that

$$Pr\left[\begin{array}{c|c} (\mathsf{Open}(\mathsf{pp}, C, f, \mathsf{st}, \mathsf{c}) \neq 1 \vee f(x) \neq y) \\ \vee b = 1 \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathbb{x} := (C, x, y), \mathsf{st}^* \leftarrow \mathcal{P}^*(\mathsf{pp}) \\ (tr, b) \leftarrow \langle \mathcal{P}^*(\mathsf{pp}, \mathbb{x}, \mathsf{st}^*), \mathcal{V}(\mathsf{pp}, \mathbb{x}) \rangle \\ (f, \mathsf{st}, c) \leftarrow \varepsilon^{\mathcal{P}^*}(\mathsf{pp}, \mathbb{x}) \end{array}\right] = \varepsilon(\lambda) + \mathsf{negl}(\lambda).$$

The extractor ε has a black-box oracle access to the (malicious) prover $\mathcal{P}^*$ and can rewind it to any point in the interaction.

2.3 Traversing the Boolean Hypercube

We want to define two linear functions next: $\mathcal{R}^\mu \to \mathcal{R}^\mu$ and prev: $\mathcal{R}^\mu \to \mathcal{R}^\mu$. Both functions traverse a μ-dimension boolean hypercube B_μ, and the two are inverses of each other. We

adopt the generator from [CBBZ23] and define it over $\mathcal{R}_{2^\mu}$ instead of a finite field. Also, we modify our generator to map $0^\mu \in \mathcal{R}^\mu$ to an arbitrary point on $B_\mu \backslash \{0\}^\mu$ instead of to itself, so that starting from *any* point on the boolean hypercube, *all* 2^μ points on the hypercube can be traversed.

A generator in $\mathcal{R}_{2^\mu}$. Fix some set $S \subseteq [\mu - 1]$. Let $\mathbf{x}_0$ be $0^\mu \in \mathcal{R}^\mu$ and let $\mathbf{x}_*$ be any arbitrary point on B_μ that is not $\mathbf{x}_0$. Denote $g_\mu(\mathbf{x}_*)$ as $\mathbf{x}_{*+1}$ (i.e., $\mathbf{x}_{*+1}$ is the point on B_μ that is next in 'sequence' in the generator after $\mathbf{x}_*$). For all $\mathbf{x} = (b_1, \cdots, b_\mu) \in B_\mu$ that is not $\mathbf{x}_0$ or $\mathbf{x}_*$, we define the generator function $g_\mu : B_\mu \to B_\mu$ as

$$g_\mu(b_1, \cdots, b_\mu) = (b_\mu, b'_1, \cdots, b'_{\mu-1}),$$

where $b'_i = b_i \oplus b_\mu$ if $i \in S$, and $b'_i = b_i$ otherwise. We define g_μ for the two special cases as follows:

$$g_\mu(\mathbf{x}_*) = \mathbf{x}_0; \text{ and}$$
$$g_\mu(\mathbf{x}_0) = \mathbf{x}_{*+1}.$$

Effectively, we have included $\mathbf{x}_0$ in the *traversal path* by g_μ by disconnecting two arbitrary consecutive points $\mathbf{x}_*$ and $\mathbf{x}_{*+1}$ in the path and making them the preimage and image, respectively, of $\mathbf{x}_0$ in g_μ.

Lemma 2.11. *For every $\mu \in \mathbb{N}$, let $g_\mu : B_\mu \to B_\mu$ be the generator defined above. For every $\mathbf{x} \in B_\mu$, it holds that $\{g_\mu^{(i)}(\mathbf{x})\}_{i \in [2^\mu - 1]} = B_\mu$, where $\{g_\mu^{(i)}(\cdot)\}$ denotes i many repeated applications of g_μ.*

Definition 2.12 (The next Function)**.** Given the result of Lemma 2.11, we define next: $\mathcal{R}_\mu \to \mathcal{R}_\mu$ as

$$\mathsf{next} := g_\mu(\cdot).$$

Inversely traversing the boolean hypercube. We introduce a new prev function that is the inverse of next. Specifically, prev: $\mathcal{R}^\mu \to \mathcal{R}^\mu$ is a linear function that traverses the entire μ-dimension boolean hypercube such that

$$\mathsf{prev}(\mathsf{next}(\mathbf{x})) = \mathsf{next}(\mathsf{prev}(\mathbf{x})) = \mathbf{x}$$

for all $\mathbf{x} \in \{0,1\}^\mu$. Furthermore, let $\mathsf{next}^i(\mathsf{prev}^j(\mathbf{x}))$ mean applying next i times and applying prev j times for any $i, j \in \mathbb{N}$ and in *any* order. Then for $i > j$

$$\mathsf{next}^i(\mathsf{prev}^j(\mathbf{x})) = \mathsf{next}^{i-j}(\mathbf{x})$$

and for $j > i$

$$\mathsf{next}^i(\mathsf{prev}^j(\mathbf{x})) = \mathsf{prev}^{j-i}(\mathbf{x}).$$

Inverse of the generator. We define the inverse of the generator g_μ from above with the aim of defining prev. Fix some set $S \subseteq [\mu - 1]$. Denote by $g_{\mu,S}$ the g_μ that operates on S. For all $\mathbf{x} = (b_1, \cdots, b_\mu) \in B_\mu$ that is not $\mathbf{x}_0$ or $\mathbf{x}_{*+1}$, the inverse of $g_{\mu,S}$ is defined as follows:

$$g_\mu^{-1}(b_1, \cdots, b_\mu) = (b'_2, \cdots, b'_{\mu-1}, b_1)$$

where $b'_i = b_i \oplus b_\mu$ if $(i-1) \in S$, and $b'_i = b_i$ otherwise. We define g_μ^{-1} for the two special cases as follows:

$$g_\mu^{-1}(\mathbf{x}_0) = \mathbf{x}_*; \text{ and}$$

$$g_\mu^{-1}(\mathbf{x}_{*+1}) = \mathbf{x}_0.$$

By inspection, we see that g_μ and g_μ^{-1} undo the traversal done by each other, provided that they operate on the same set $S \in [\mu - 1]$.

Lemma 2.13. *For every $\mu \in \mathbb{N}$ and any set $S \subseteq [\mu-1]$, let $g_\mu^{-1} : B_\mu \to B_\mu$ be the generator defined above. For every $\mathbf{x} \in B_\mu$, it holds that $\{g_\mu^{-1(i)}(\mathbf{x})\}_{i\in[2^\mu-1]} = B_\mu$, where $\{g_\mu^{-1(i)}(\cdot)\}$ denotes i many repeated applications of g_μ^{-1}. In other words, g_μ^{-1} is also a generator of B_μ.*

Definition 2.14 (The prev Function)**.** Given the result of Lemma 2.13, we define prev: $\mathcal{R}_\mu \to \mathcal{R}_\mu$ as

$$\mathsf{prev} := g_\mu^{-1}(\cdot).$$

3 Sparse Polynomial Commitment Scheme

We present our main result here: a sparse polynomial commitment scheme from lattices with optimal prover costs linear in the sparsity of the polynomial. We start by describing a time-optimal algorithm from Spartan that evaluates an m-sparse multilinear polynomial in $O(c \cdot m)$ time for some $c \in \mathbb{N}$ we will define shortly. This algorithm takes as input m Lagrange basis polynomials that express the unique dense representation of the sparse polynomial. We adapt this algorithm to the ring domain and use it in a black-box manner in Spartic.

A time-optimal algorithm for evaluating a multilinear polynomial of sparsity m. We make a black-box use of Spark's time-optimal algorithm for evaluating a sparse polynomial. Suppose we wish to evaluate a polynomial $g \in \mathcal{R}_q$ defined over $\log N = c \cdot \log m$ variables at a point $\mathbf{r} \in \mathcal{R}^{\log N}$. The high-level objective is to evaluate the m Lagrange basis polynomials with a non-zero coefficient at $\mathbf{r}$, multiply each result with the corresponding coefficient, and sum the results. To do so in time $O(c \cdot m)$, first decompose $\mathbf{r}$ into $c \in \mathbb{N}$ blocks so that $\mathbf{r} = (\mathbf{r}_1, \cdots, \mathbf{r}_c) \in (\mathcal{R}^{\log m})^c$. Then each of the m Lagrange basis polynomials evaluated at $\mathbf{r}$ is equal to the product of c many $(\log m)$-variate (smaller) Lagrange basis polynomials, the ith of which is evaluated at $\mathbf{r}_i$ $(1 \le i \le c)$. There are $2^{\log m} = m$ Lagrange basis polynomials defined over $\log m$ variables. Now, we can use any standard algorithm that evaluates all m Lagrange basis polynomials at all c points $\mathbf{r}_i$ $(1 \le i \le c)$ in $O(c \cdot m)$ time. The evaluation result for each $\mathbf{r}_i$ is written to a write-once memory M of size M. Later, the time-optimal algorithm can compute any given Lagrange basis polynomial at $\mathbf{r}$ by performing c lookups into M and multiplying together the returned values.

In $\mathcal{R}_q$, we must compute the Lagrange basis polynomials in a way that does not involve division by zero divisors. Otherwise, it becomes difficult to contain soundness error. As we defined in Definition 2.3, we use the standard multilinear interpolation functions

$$eq(\mathbf{r}, \mathbf{x}) := \prod_{i=1}^{\mu} \left[(1 - \mathbf{r}_i)(1 - \mathbf{x}_i) + \mathbf{r}_i \mathbf{x}_i\right]$$

1. $\mathsf{RS} \leftarrow \mathsf{RS} \cup \{(a, v, t)\}$,
2. store (v, next(t)) at address a in the untrusted memory, and
3. $\mathsf{WS} \leftarrow \mathsf{WS} \cup \{(a, v, \mathsf{next}(t))\}$.

Figure 1: Updating the internal states of a trusted checker during memory checking

as the Lagrange basis polynomials, where $\mathbf{r}_i \in \mathcal{R}^{\log m}$ is the ith sub-string of point $\mathbf{r}$ as decomposed in the time-optimal algorithm.

Offline memory checking. Given the algorithm above, proving the evaluation of a sparse polynomial is reduced to checking the contents of the memory cells written to over the course of the algorithm. Recall the offline memory checking algorithm of [BEG+91] where a trusted checker issues read and write operations to an untrusted memory. We use the version of this algorithm used in Lasso, where the memory is write-once. Let c be the number of memories of size $N^{1/c}$ used here.

Each memory cell maintains a counter whose *count* is incremented by the checker every time the cell is read. Additionally, the checker maintains a local state comprising two multisets: RS and WS. RS is initialized as $\{\}$, and WS is initialize to the set of tuples $(i, v_i, 0)$ for the count $i \in [N^{1/c}]$ and for v_i, the value stored at address i. After every read operation at address a in which the untrusted memory responds with the value-count pair (v, t), the checker updates its internal state as follows:

Lemma 3.1. *The domain of the counts is $\mathcal{R}$. There exists a natural one-to-one mapping between $\{0,1\}^n$ for a positive integer n and any constant in $\mathcal{R}$ that is not larger than 2^n. Let WS and RS denote the multisets maintained by the checker at the end of m read operations. Claim 2 in [STW24] proves that, if for every read operation, the untrusted memory truthfully returns the tuple last written to that location, then there exists a set S with cardinality M consisting of tuples of the form (k, v_k, t_k) for all $k \in [M]$ such that WS=RS $\cup$ S. Moreover, S is computable in time linear in M.*

Conversely, if the untrusted memory ever returns a value v' for a memory call $k \in [M]$ such that $v \neq v'$ where v is the initial value written to address k, then there does not exist any set S satisfying WS=RS $\cup$ S.

Proof. The proof closely resembles the proof for Claim 2 in [STW24]. If during all m read operations, the untrusted memory returned the correct initial value stored in the memory cell, then the set S exists as simply the current state of all memory cells i.e., M tuples of (address, value, count). To prove the converse case, assume for contradiction that the untrusted memory returns (a, v', t) during the ith read operation, for $v' \neq v$, and that there exists S such that $\mathsf{RS} \cup \mathsf{S} = \mathsf{WS}$ is true. The existence of S implies that $\mathsf{RS} \subseteq \mathsf{WS}$. Notice that the only way the untrusted checker can return (a, v', t) at the ith read operation is if $(a, v', \mathsf{prev}(t))$ was written in that memory cell in Step 3 (Figure 1) during the $(i-1)$th read operation. This means in Step 1 of the same read operation, 1, $(a, v', \mathsf{prev}(t))$ was read into RS. This, in turn, means that in Step 3 of the $(i-2)$ read operation, $(a, v', \mathsf{prev}^{(2)}(t))$ was added to WS, and so on. Since prev traverses the entire $(2^{\log N})$-dimension boolean hypercube starting from t, an induction-style argument here proves that the checker issued $2^{\log N}$ read

operations. However, this is a contradiction because the number of read operations is equal to the sparsity of the committed polynomial i.e., m, which is by definition of a sparse polynomial (significantly) less than the number of points on the boolean hypercube. □

3.1 Result for c=2

Our actual polynomial evaluation scheme is presented in Figure 2. Before that, we introduce the techniques that we will use to build the scheme. In this section, we present our main results in the special case when $c = 2$ and then in Section 3.2, we show how the result generalizes to any $c \in \mathbb{N}$.

Representing sparse polynomials with dense polynomials. The high-level idea of our sparse polynomial commitment scheme is to find the unique representation of a sparse polynomial as dense polynomials and commit to the dense polynomials using any (dense) polynomial commitment scheme for lattices.

Recall that $\mathsf{eq}_s : \{0,1\}^s \to \{0,1\}^s$ takes as input two vectors of length s and outputs 1 if and only if the vectors are equal. Let D denote a $(2\log M)$-variate multilinear polynomial that evaluates to a non-zero value at at most m locations over $\{0,1\}^{2\log M}$ i.e., the boolean hypercube $B_{2\log m}$. Much like the technique in [STW24], we observe that D can be written as follows using multilinear extensions over rings.

Lemma 3.2. *For any $r \in R^{2\log M}$, interpret it as a tuple (r_x, r_y) in a natural manner, where $r_x, r_y \in R^{\log M}$. Then, using multilinear extensions over rings, we can evaluate D at (r_x, r_y) as*

$$D(r_x, r_y) = \sum_{(i,j)\in\{0,1\}^{\log M}\times\{0,1\}^{\log M} : D(i,j)\neq 0} D(r_x, r_y) \cdot \widetilde{eq}(i, r_x) \cdot \widetilde{eq}(j, r_y). \tag{1}$$

Lemma 3.3. *Let $\mathsf{to\text{-}ring}$ be a canonical injection from $\{0,1\}^n$ to $\mathcal{R}_q$, and let $\mathsf{to\text{-}bits}$ be its inverse. Given the polynomial D defined above, there exist three $(\log m)$-variate multilinear polynomials $\mathsf{row}, \mathsf{col}, \mathsf{val}$ such that the following holds for all $r_x, r_y \in \mathcal{R}^{\log M}$.*

$$D(r_x, r_y) = \sum_{k\in\{0,1\}^{\log m}} \mathsf{val}(k) \cdot \widetilde{eq}(\mathsf{to\text{-}bits}(\mathsf{row}(k)), r_x) \cdot \widetilde{eq}(\mathsf{to\text{-}bits}(\mathsf{col}(k)), r_y). \tag{2}$$

Proof. The proof closely resembles the proof for Claim 1 in [STW24]. Since D evaluates to a non-zero value at at most m points on $B_{2\log M}$, D can be represented uniquely with m tuples of the form $(i, j, D(i,j)) \in (\{0,1\}^{\log M}, \{0,1\}^{\log M}, \mathcal{R})$. Using $\mathsf{to\text{-}ring}$, we can represent the first two entries in each tuple as an element of $\mathcal{R}$. Then, we can make three vectors $R, C, V \in \mathcal{R}^m$ that encode these m tuples such that for the kth tuple, its $\mathsf{to\text{-}ring}(i)$ is stored in the kth location of R, $\mathsf{to\text{-}ring}(j)$ in the kth location of C, and $D(i,j)$ in the kth location of V. Let $\mathsf{row}, \mathsf{col}, \mathsf{val}$ be the unique multilinear extensions of R, C, V respectively. Then we can observe that Equation (2) holds by inspecting it together with Equation (1). □

Conceptually, for each of the m evaluations that are summed in Equation (1), the time-optimal algorithm for evaluating D fills up the first of its two memories with evaluations of $\widetilde{eq}(i, r_x)$ and the second memory with evaluations of $\widetilde{eq}(j, r_y)$ for $i, j \in \{0,1\}^{\log M}$. Then, the algorithm can compute Equation (2) via one lookup into each memory, to the respective memory cells with (binary) indices $\mathsf{to\text{-}bits}(\mathsf{row}(k))$ and $\mathsf{to\text{-}bits}(\mathsf{col}(k))$ for $k \in \{0,1\}^{\log m}$,

followed by two multiplications in $\mathcal{R}$. The algorithm then repeats this m times in $O(c \cdot m)$ time as claimed.

Counter polynomials. Consider the oracles E_{rx} and E_{ry} as the purported multilinear extensions of the values returned by all memory reads that the aforementioned time-optimal algorithm performed into its two memories. Notice that the returned values are *purported* because we assume an untrusted memory. Hence, we define three 'counter' polynomials that will aid the verifier in the evaluation proof of Figure 2 to check the correctness of the returned values i.e., the memory count of each memory cell. The counter polynomials encode truthful counts that would have been returned *had* the untrusted memory been honest in all its responses.

First, given the size M of memory and a list of m addresses involved in read operations, define two vectors $C_r \in \mathcal{R}^m$ and $C_f \in \mathcal{R}^M$ as follows. For $k \in [m]$, $C_r[k]$ stores the correct count returned by the kth read operation, and for $j \in [M]$, $C_f[j]$ stores the correct final count stored at the jth memory location. Computing these three vectors take $O(m)$ operations in $\mathcal{R}_q$.

Then, the counter polynomials are $\mathsf{read\_ts} = \widetilde{C_r}$, $\mathsf{write\_cts} = \widetilde{C_r} + 1$, and $\mathsf{final\_cts} = \widetilde{C_f}$, and they are unique for the memory size M and the list of m read addresses.

Lemma 3.4. *We reproduce Claim 3 in [STW24] here, adapted for the ring domain. Given a $(2 \log M)$-variate multilinear polynomial, suppose that* $\mathsf{row}, \mathsf{col}, \mathsf{val}$ *are multilinear polynomials committed to by the commit algorithm of a dense polynomial commitment scheme. Furthermore, suppose that*

$$(E_{rx}, E_{ry}, \mathsf{read\_ts_{row}}, \mathsf{final\_cts_{row}}, \mathsf{read\_ts_{col}}, \mathsf{final\_cts_{col}})$$

denote the additional polynomials sent by the prover at the beginning of the evaluation proof. For any $r_x \in \mathcal{R}^{\log M}$, suppose that

$$\forall k \in \{0,1\}^{\log m}, E_{rx}(k) = \widetilde{eq}(\mathsf{to\text{-}bits}(\mathsf{row}(k)), r_x). \tag{3}$$

Then the following holds: $\mathsf{WS} = \mathsf{RS} \cup \mathsf{S}$*, where*

- $\mathsf{WS} = \{(\mathsf{to\text{-}ring}(i), \widetilde{eq}(i, r_x), 0) : i \in \{0,1\}^{\log M}\} \cup \{(\mathsf{row}(k), E_{rx}(k), \mathsf{write\_cts_{row}}(k) = \mathsf{read\_ts_{row}}(k) + 1 : k \in \{0,1\}^{\log m}\}$
- $\mathsf{RS} = \{(\mathsf{row}(k), E_{rx}(k), \mathsf{read\_ts_{row}}(k)) : k \in \{0,1\}^{\log m}\}$*, and*
- $\mathsf{S} = \{(\mathsf{to\text{-}ring}(i), \widetilde{eq}(i, r_x), \mathsf{final\_cts_{row}}(i)) : i \in \{0,1\}^{\log M}\}$.

Meanwhile, if Equation (3) does not hold, then there is no set S *such that* $\mathsf{WS} = \mathsf{RS} \cup \mathsf{S}$*. Similarly, for any $r_y \in \mathcal{R}^{\log M}$, suppose that*

$$\forall k \in \{0,1\}^{\log m}, E_{ry}(k) = \widetilde{eq}(\mathsf{to\text{-}bits}(\mathsf{col}(k)), r_y).$$

Then the following holds: $\mathsf{WS} = \mathsf{RS} \cup \mathsf{S}$*, where*

- $\mathsf{WS} = \{(\mathsf{to\text{-}ring}(i), \widetilde{eq}(i, r_x), 0) : i \in \{0,1\}^{\log M}\} \cup \{(\mathsf{col}(k), E_{ry}(k), \mathsf{write\_cts_{col}}(k) = \mathsf{read\_ts_{col}}(k) + 1) : k \in \{0,1\}^{\log m}\}$
- $\mathsf{RS} = \{(\mathsf{col}(k), E_{ry}(k), \mathsf{read\_ts_{col}}(k)) : k \in \{0,1\}^{\log m}\}$*, and*
- $\mathsf{S} = \{(\mathsf{to\text{-}ring}(i), \widetilde{eq}(i, r_y), \mathsf{final\_cts_{col}}(i)) : i \in \{0,1\}^{\log M}\}$.

Proof. Proof follows from Lemma 3.1. □

There is no direct way to prove the multiset relation $\mathsf{WS} = \mathsf{RS} \cup \mathsf{S}$ in the above lemma. Hence, we rely on the following probabilistic check. We use two collision-resistant hash functions $h_\gamma : \mathcal{R}^3 \to \mathcal{R}$ and $\mathcal{H}_{\tau,\gamma} : (\mathcal{R}^3)^* \to \mathcal{R}$ where $(\mathcal{R}^3)^*$ denotes a multiset with elements from $\mathcal{R}^3$ and $\tau, \gamma \in_\$ \mathcal{R}$. The hash functions are defined as follows.

$$h_\gamma(a, v, t) = a \cdot \gamma^2 + v \cdot \gamma + t \tag{4}$$

$$\mathcal{H}_{\tau,\gamma}(A) = \prod_{(a,v,t)\in A} (h_\gamma(a, v, t) - \tau) \tag{5}$$

Lemma 3.5. *Given two multisets A, B where each element is from $\mathcal{C}^3$ for a sampling set $\mathcal{C} \subseteq \mathcal{R}_q$, checking that A and B are permutations of one another is equivalent to checking the following, except for a soundness error of $3(\max(|A|, |B|))/|\mathcal{C}|$ over the choice of γ, τ from $\mathcal{R}_q$: $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$. That is, if $A = \sigma(B)$ for some permutation σ, then $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$ with probability 1 over randomly chosen values τ, γ, while if $A \neq \sigma(B)$, then $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$ with probability at most $O(\max(|A|, |B|))/|\mathcal{C}|)$. Furthermore, this probability is negligible if $\mathcal{C}$ is a sampling set of an exponential size, the existence of which in $\mathcal{R}_q$ is proven in Lemma 2.2.*

Proof. The statement $\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B)$ is equivalent to $\mathcal{H}_{\tau,\gamma}(A) - \mathcal{H}_{\tau,\gamma}(B) = 0$ where the LHS is some tri-variate multilinear polynomial $P \in \mathcal{R}^{\leq |A|+|B|}[A, V, T]$. It is easy to see by inspecting Equations (4) and (5) that P has per-variable degree at most $\max(|A|, |B|)$. Hence, the soundness error claim follows directly from the generalized Schwartz-Zippel lemma. That is, for any two multisets A and B over $\mathcal{R}_q$,

$$Pr_{\tau,\gamma}\{\mathcal{H}_{\tau,\gamma}(A) = \mathcal{H}_{\tau,\gamma}(B) | A \neq B\} \leq 3(\max(|A|, |B|))/|\mathcal{C}|.$$

□

We are now ready to introduce the actual commit and evaluation phases of our sparse polynomial commitment scheme.

Commit phase. To commit to D, the prover commits to to the three $(\log m)$-variate multilinear polynomials $\mathsf{row}, \mathsf{col}, \mathsf{val}$ using the commit function of any polynomial commitment scheme for dense multilinear polynomials in $\mathcal{R}_q$.

Evaluation phase. In the evaluation phase of Figure 2, the prover proves that it correctly ran the time-optimal algorithm at the beginning of Section 3 to evaluate D at the requested point $(r_x, r_y) \in (R^{\log M})^2$. Denote the prover by $\mathcal{P}$ and the verifier by $\mathcal{V}$.

1. $\mathcal{P} \to \mathcal{V}$: four $(\log m)$-variate multilinear polynomials $E_{rx}, E_{ry}, \mathsf{read\_ts_{row}}$, $\mathsf{read\_ts_{col}}$ and two $(\log M)$-variate multilinear polynomials $\mathsf{final\_cts_{row}}$, $\mathsf{final\_cts_{col}}$.
2. Notice that $D(r_x, r_y) = \sum_{k\in\{0,1\}^{\log m}} \mathsf{val}(k) \cdot E_{rx}(k) \cdot E_{ry}(k)$ assuming that
   - $\forall k \in \{0,1\}^{\log m}, E_{rx}(k) = \widetilde{eq}(\mathsf{to\text{-}bits(row(k))}, r_x)$, and **(Eq 1)**
   - $\forall k \in \{0,1\}^{\log m}, E_{ry}(k) = \widetilde{eq}(\mathsf{to\text{-}bits(col(k))}, r_y)$. **(Eq 2)**

   Hence, $\mathcal{V}$ and $\mathcal{P}$ apply the sum-check protocol over rings to reduce this check to checking that the following equations hold, where $r_z \in \mathcal{R}^{\log m}$ is chosen at random

by $\mathcal{V}$ over the course of the protocol, and $v_{E_{rx}}$ and $v_{E_{ry}}$ are values provided by $\mathcal{P}$ at the end of the sum-check protocol:

- $val(r_z) \stackrel{?}{=} v_{val}$,
- $E_{rx}(r_z) \stackrel{?}{=} v_{E_{rx}}$, and $E_{ry}(r_z) \stackrel{?}{=} v_{E_{ry}}$.

3. $\mathcal{V}$: check if the above three equalities above hold with one oracle query to each of val, E_{rx}, and E_{ry}.
4. $\mathcal{V}$ checks **Eq 1** as follows.
   - 4-1. $\mathcal{V} \to \mathcal{P}$: $\tau, \gamma \in_{\$} \mathcal{R}$.
   - 4-2. $\mathcal{V} \leftrightarrow \mathcal{P}$: run a sum-check protocol to reduce the check that $\mathcal{H}_{\tau,\gamma}(\mathsf{WS}) = \mathcal{H}_{\tau,\gamma}(\mathsf{RS}) \cdot \mathcal{H}_{\tau,\gamma}(\mathsf{S})$, where WS, RS, S are defined in Lemma 3.4 and $\mathcal{H}$ is defined in Lemma 3.5, to checking if the following hold, where $r_M \in \mathcal{R}^{\log M}$ and $r_m \in \mathcal{R}^{\log M}$ are chosen at random by $\mathcal{V}$ over the course of the sum-check protocol:
     - $\widetilde{eq}(r_M, r_x) \stackrel{?}{=} v_{eq}$,
     - $E_{rx} \stackrel{?}{=} v_{E_{rx}}$,
     - $\mathsf{row}(r_m) \stackrel{?}{=} v_{\mathsf{row}}$, $\mathsf{read\_ts}_{\mathsf{row}}(r_m) \stackrel{?}{=} v_{\mathsf{read\_ts}_{\mathsf{row}}}$, and $\mathsf{final\_cts}_{\mathsf{row}}(r_M) \stackrel{?}{=} v_{\mathsf{final\_cts}_{\mathsf{row}}}$.
   - 4-3. $\mathcal{V}$: directly check if the first equality holds, which can be done with $O(\log M)$ ring operations, and check the remaining equations hold with one oracle query to each of row, E_{rx}, $\mathsf{read\_ts}_{\mathsf{row}}$, and $\mathsf{final\_cts}_{\mathsf{row}}$.
5. $\mathcal{V}$ checks **Eq 2** as follows.
   - 5-1. $\mathcal{V} \to \mathcal{P}$: $\tau', \gamma' \in_{\$} \mathcal{R}$.
   - 5-2. $\mathcal{V} \leftrightarrow \mathcal{P}$: run a sum-check protocol to reduce the check that $\mathcal{H}_{\tau',\gamma'}(\mathsf{WS}') = \mathcal{H}_{\tau',\gamma'}(\mathsf{RS}') \cdot \mathcal{H}_{\tau',\gamma'}(\mathsf{S}')$, where WS', RS', S' are defined in Lemma 3.4 and $\mathcal{H}$ is defined in Lemma 3.5, to checking if the following hold, where $r'_M \in \mathcal{R}^{\log M}$ and $r'_m \in \mathcal{R}^{\log M}$ are chosen at random by $\mathcal{V}$ over the course of the sum-check protocol:
     - $\widetilde{eq}(r'_M, r_y) \stackrel{?}{=} v'_{eq}$,
     - $E_{ry} \stackrel{?}{=} v_{E_{ry}}$,
     - $\mathsf{col}(r'_m) \stackrel{?}{=} v_{\mathsf{col}}$, $\mathsf{read\_ts}_{\mathsf{col}}(r'_m) \stackrel{?}{=} v_{\mathsf{read\_ts}_{\mathsf{col}}}$, and $\mathsf{final\_cts}_{\mathsf{col}}(r'_M) \stackrel{?}{=} v_{\mathsf{final\_cts}_{\mathsf{col}}}$.
   - 5-3. $\mathcal{V}$: directly check if the first equality holds, which can be done with $O(\log M)$ ring operations, and check the remaining equations hold with one oracle query to each of col, E_{ry}, $\mathsf{read\_ts}_{\mathsf{col}}$, and $\mathsf{final\_cts}_{\mathsf{col}}$.

Figure 2: Evaluation procedure for the Spartic sparse polynomial commitment scheme.

Lemma 3.6 (Perfect Completeness)**.** *Spartic has perfect completeness.*

Proof. Perfect completeness follows from the perfect completeness of the generalized sum-check protocol and the memory checking of [BEG+91], and the fact that the probabilistic multiset equality check of Lemma 3.5 verifies with probability 1 for multisets WS (respectively $\mathsf{WS'}$) and $\mathsf{RS} \cup \mathsf{S}$ (respectively $\mathsf{RS'} \cup \mathsf{S'}$) that are permutations of each other, if the prover is honest. □

Lemma 3.7 (Knowledge Soundness)**.** *Spartic has a negligible soundness error.*

Proof. The soundness error of multiset equality checks is $O(m)/|\mathcal{C}|$ for some sampling set $\mathcal{C}$ of $\mathcal{R}_q$. The soundness error of sum-check protocols over rings is $O(m)/|\mathcal{C}_{strong}|$ for some strong sampling set $\mathcal{C}_{strong}$ of $\mathcal{R}_q$. Without loss of generality, assume $\mathcal{C}_{strong}$ is such that $\mathcal{C}_{strong} \subseteq \mathcal{C}$. That is, $|\mathcal{C}_{strong}| \leq |\mathcal{C}|$. Using a standard union bound, we conclude that the soundness error for our sparse polynomial commitment scheme is at most $O(m)/|\mathcal{C}_{strong}|$. Using Lemma 2.2, we conclude $O(m)/|\mathcal{C}_{strong}|$ is indeed negligible. □

3.2 General Result for c>2

We can naturally extend Lemmas 3.2 and 3.3 to the general case when $c > 2$ memories are used for offline memory checking. Suppose D is a $(c \log M)$-variate polynomial.

Lemma 3.8. *For any $r \in R^{c \log M}$, interpret it as a tuple $(r_1, \cdots, r_c)$ in a natural manner, where $r_1, \cdots, r_c \in R^{\log M}$. Then, using multilinear extensions over rings, we can compute $D(r_1, \cdots, r_c)$ as*

$$\sum_{(x_1,\cdots,x_c)\in\{0,1\}^{\log M}\times\cdots\times\{0,1\}^{\log M}: D(x_1,\cdots,x_c)\neq 0} D(x_1, \cdots, x_c) \cdot \widetilde{eq}(x_1, r_1) \cdots \widetilde{eq}(x_c, r_c).$$

Lemma 3.9. *Let $\mathsf{to\text{-}ring}$ be a canonical injection from $\{0,1\}^n$ to $\mathcal{R}_q$, and let $\mathsf{to\text{-}bits}$ be its inverse. Given the polynomial D defined above, there exist $c+1$ $(\log m)$-variate multilinear polynomials $\mathsf{dim}_1, \cdots, \mathsf{dim}_c, \mathsf{val}$ such that the following holds for all $r_1, \cdots, r_c \in \mathcal{R}^{c \log m}$.*

$$D(r_1, \cdots, r_c) = \sum_{k\in\{0,1\}^{\log m}} \mathsf{val}(k) \cdot \widetilde{eq}(\mathsf{to\text{-}bits}(\mathsf{dim}_1(k)), r_1) \cdots \widetilde{eq}(\mathsf{to\text{-}bits}(\mathsf{dim}_c(k)), r_c). \quad (6)$$

Proof. It is easy to see how the proof for Lemma 3.3 generalizes to this case for any $c \in \mathbb{N}$. In particular, the proof for Lemma 3.3 is simply a special case in which dim_1 is row and dim_2 is col. □

Given Equations (4) and (5), we can define the evaluation proof of our sparse polynomial commitment scheme accordingly. The verifier now checks c different untrusted memories, rather than two, each of size $N^{1/c}$. That is, the nth memory $(1 \leq n \leq c)$ stores all evaluations of $\widetilde{eq}(x_c, r_c)$ as x_c ranges over $\{0,1\}^{\log M}$. Then, for each of the c memories checked, the prover commits to three $(\log m)$-variate multilinear polynomials and one $(\log M)$-variate polynomial.

Lemma 3.10 (Perfect Completeness in the General Case)**.** *Our sparse polynomial commitment scheme has perfect completeness when $c > 2$.*

Proof. Perfect completeness of the $c = 2$ case in Figure 2 extends to the general case. □

Lemma 3.11 (Knowledge Soundness in the General Case)**.** *Our sparse polynomial commitment scheme has a negligible soundness error when $c > 2$.*

Proof. The evaluation proof with c memories uses c multiset equality checks and $c+1$ sum-checks over rings. From the proof of Lemma 3.7, we see that this makes the soundness error in the general case $O(c \cdot m)/|\mathcal{C}_{strong}|$. Using Lemma 2.2, this probability is indeed negligible. □

We now formalize our results in the following theorem to standardize the commitment scheme for sparse multilinear polynomials in $\mathcal{R}_q$ with optimal prover costs.

Theorem 1. *[STW24] Suppose we have a polynomial commitment scheme for (dense) $(\log M)$-variate multilinear polynomials with the following parameters (where M is a positive integer and WLOG a power of 2):*

- *the size of the commitment is $\mathsf{c}(M)$ i.e., c is a function of M,*
- *the running time of the commit algorithm is $\mathsf{tc}(M)$,*
- *the running time of the prover to prove a polynomial evaluation is $\mathsf{tp}(M)$,*
- *the running time of the verifier to verify a polynomial evaluation is $\mathsf{tv}(M)$, and*
- *the proof size is $\mathsf{p}(M)$.*

Then, there exists a sparse polynomial commitment scheme for m-sparse multilinear polynomials over $2\log M = \log N$ variables with the following parameters:

- *the size of the commitment is $(3c+1) \cdot \mathsf{c}(m) + c \cdot \mathsf{c}(M)$,*
- *the running time of the commit algorithm is $O(c \cdot (\mathsf{tc}(m) + \mathsf{tc}(M)))$,*
- *the running time of the prover to prove a polynomial evaluation is $O(c \cdot (\mathsf{tp}(m) + \mathsf{tc}(M)))$,*
- *the running time of the verifier to verify a polynomial evaluation is $O(c \cdot (\mathsf{tv}(m) + \mathsf{tv}(M)))$, and*
- *the proof size is $O(c \cdot (\mathsf{p}(m)) + \mathsf{p}(M)))$.*

Substituting 2 for c in the above Theorem gives an equivalent theorem for the special case in Section 3.1.

Example costs. The generic transformation method in Appendix B of [CBBZ23] allows committing to a $(\log M)$-variate multilinear polynomial as a M-variate univariate polynomial. Consider using Greyhound to commit to this M-variate univariate polynomial with degree bound M. Then, Spartic achieves the following parameters:

- the size of the commitment is $(3c+1) \cdot O_\lambda(1) + c \cdot O_\lambda(1)$ i.e., $4c+1$ many $\mathbb{Z}_q$ elements,
- the running time of the prover to prove a polynomial evaluation is $O(c \cdot (m + \log M))$,

- the running time of the verifier to verify a polynomial evaluation is $O(c \cdot (\sqrt{m} + \sqrt{M}))$, and
- the proof size is $O(c \cdot (\mathsf{polylog}(m) + \mathsf{polylog}(M)))$.

For concrete costs, we refer the readers to Table 1 (proof sizes) and Table 2 (running times) of [NS24]. Furthermore, dense polynomial commitment schemes, including Greyhound, that support batch evaluation of proofs can do the multiset equality check of Lemma 3.5 once across all c memories. These schemes can enjoy prover and verifier costs that do not scale with the number of polynomials to evaluate i.e., lose the c in the last three bullet points of Theorem 1.

4 Conclusion

We presented Spartic, the first sparse polynomial commitment scheme for lattices. Our scheme achieves prover runtime that depends only on the sparsity of the committed polynomial, which is a key benefit of sparse polynomial commitment schemes. This is done by first transforming the sparse polynomial into its dense representation with respect to Lagrange basis polynomials and proving the correct execution of a time-optimal algorithm for evaluating multilinear polynomials. The evaluation proof is done efficiently using the generalized sum-check protocol for rings.

We also note without proving that we can build Lasso-style lookup arguments from standard lattice assumptions by replacing Spark with Spartic in Lasso. This can be achieved by having an untrusted prover commit to a vector $a \in \mathcal{R}^m$ and proving that all entries of a reside in some predetermined table $t \in \mathcal{R}^n$. Express the table as a highly-structured (in a precise sense that Lasso defines) vector of size N. Then, the lookup argument is equivalent to computing the inner product of an m-sparse vector of size N with the table vector. Because Lasso supports indexing tables by points on the boolean hypercube, it can be naturally composed with Spartic.

Acknowledgements. TODO.

References

[ABGK22] Nitin Agrawal, James Bell, Adrià Gascón, and Matt J. Kusner. Mpc-friendly commitments for publicly verifiable covert security, 2022.

[ACL+22] Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri AravindaKrishnan Thyagarajan. Lattice-based snarks: Publicly verifiable, preprocessing, and recursively composable. In *Advances in Cryptology – CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Proceedings*, pages 102–132, Germany, October 2022. Springer. International Cryptology Conference, CRYPTO ; Conference date: 13-08-2022 Through 18-08-2022.

[BC24] Dan Boneh and Binyi Chen. LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Paper 2024/257, 2024.

[BCFL23] David Balbás, Dario Catalano, Dario Fiore, and Russell W.F. Lai. Chainable functional commitments for unbounded-depth circuits. In Guy Rothblum and

Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 363–393, Germany, 2023. Springer.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. Cryptology ePrint Archive, Paper 2022/420, 2022.

[BCPS18] Anurag Bishnoi, Pete L. Clark, Aditya Potukuchi, and John R. Schmitt. On zeros of a polynomial in a finite grid. *Combinatorics, Probability and Computing*, 27(3):310–333, 2018.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. *IACR Cryptol. ePrint Arch.*, 2020:81, 2020.

[BEG+91] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. In *Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 90–99, 1991.

[BHV+23] Rishabh Bhadauria, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Wenxuan Wu, and Yupeng Zhang. Private polynomial commitments and applications to mpc. In *Public-Key Cryptography – PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part II*, page 127–158, Berlin, Heidelberg, 2023. Springer-Verlag.

[BS22] Ward Beullens and Gregor Seiler. LaBRADOR: Compact proofs for R1CS from module-SIS. Cryptology ePrint Archive, Paper 2022/1341, 2022.

[BSU24] Alexandre Belling, Azam Soleimanian, and Bogdan Ursu. Vortex: A list polynomial commitment and its application to arguments of knowledge. Cryptology ePrint Archive, Paper 2024/185, 2024.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part II*, page 499–530, Berlin, Heidelberg, 2023. Springer-Verlag.

[CCKP19] Shuo Chen, Jung Hee Cheon, Dongwoo Kim, and Daejun Park. Verifiable computing for approximate computation. *Cryptology ePrint Archive*, 2019.

[CMNW24] Valerio Cini, Giulio Malavolta, Ngoc Khanh Nguyen, and Hoeteck Wee. Polynomial commitments from lattices: Post-quantum security, fast verification and transparent setup. Cryptology ePrint Archive, Paper 2024/281, 2024.

[FMN23] Giacomo Fenzi, Hossein Moghaddas, and Ngoc Khanh Nguyen. Lattice-based polynomial commitments: Towards asymptotic and concrete efficiency. Cryptology ePrint Archive, Paper 2023/846, 2023.

[GNS24] Chaya Ganesh, Vineet Nair, and Ashish Sharma. Dual polynomial commitment schemes and applications to commit-and-prove SNARKs. Cryptology ePrint Archive, Paper 2024/943, 2024.

[HSS24] Intak Hwang, Jinyeong Seo, and Yongsoo Song. Concretely efficient lattice-based polynomial commitment from standard assumptions. Cryptology ePrint Archive, Paper 2024/306, 2024.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size com-

mitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992.

[NS24] Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: Fast polynomial commitments fromlattices. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part X*, page 243–275, Berlin, Heidelberg, 2024. Springer-Verlag.

[Set19] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550, 2019.

[STW24] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking thelookup singularity withlasso. In *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part VI*, page 180–209, Berlin, Heidelberg, 2024. Springer-Verlag.

[WW23] Hoeteck Wee and David J. Wu. Succinct vector, polynomial, and functional commitments from lattices. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, page 385–416, Berlin, Heidelberg, 2023. Springer-Verlag.

[ZXZS19] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. Cryptology ePrint Archive, Paper 2019/1482, 2019.