



Succinct Arguments for RAM Programs via Projection Codes

Yuval Ishai¹(✉), Rafail Ostrovsky², and Akash Shah²

¹ Technion - Israel Institute of Technology, Haifa, Israel
yuvali@cs.technion.ac.il

² University of California Los Angeles, Los Angeles, USA

Abstract. Motivated by the goal of proving statements that involve small subsets of a big database, we introduce and study the notion of *projection codes*. A standard error-correcting code allows one to encode a message $\mathbf{x}$ into a codeword $\mathbf{X}$, such that even if a constant fraction of $\mathbf{X}$ is corrupted, the message $\mathbf{x}$ can still be recovered. A projection code extends this guarantee to any *subset* of the bits of $\mathbf{x}$. Concretely, for every projection of $\mathbf{x}$ to a subset $\mathbf{s}$ of its coordinates, there is a subset $\mathbf{S}$ of comparable size such that the projected encoding $\mathbf{X}|_{\mathbf{S}}$ forms a robust encoding of the projected message $\mathbf{x}|_{\mathbf{s}}$.

Our first main result is a construction of projection codes with a near-optimal increase in the length of $\mathbf{x}$ and size of $\mathbf{s}$. We then apply this to obtain our second main result: succinct arguments for the computation of a RAM program on a (big) committed database, where the communication and the run-time of both the prover and the verifier are close to optimal even when the RAM program run-time is much smaller than the database size. Our solution makes only a black-box use of a collision-resistant hash function, providing the first black-box alternative to previous non-black-box constructions with similar asymptotic efficiency.

1 Introduction

Succinct Arguments [46, 49] are computationally sound interactive proof systems that allow a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a given statement $\mathbf{x}$ is true, using an extremely short proof and very little computation overhead. A bit more formally, consider an NP-language $\mathcal{L}$ defined by a relation $\mathcal{R}(\mathbf{x}, w)$ that can be verified in $\text{DTIME}(T)$. Succinct arguments allow a polynomial-time $\mathcal{P}$, with input $(\mathbf{x}, w) \in \mathcal{R}$, to convince $\mathcal{V}$ that $\mathbf{x} \in \mathcal{L}$, where the communication and runtime of $\mathcal{V}$ are bounded by $\text{poly}(\lambda, \log T)$ and $\text{poly}(\lambda, |\mathbf{x}|, \log T)$ respectively, and where λ is a computational security parameter.

In this work, we consider a natural extension of succinct arguments in which $\mathcal{P}$ first commits to a large database $\mathbf{x} \in \{0, 1\}^n$, and later wishes to succinctly prove the correctness of the execution of RAM programs on $\mathbf{x}$ that run in *sub-linear time* $T \ll n$. We refer to this notion as “Succinct Argument for RAM programs on committed database” (or **SArg-RAM**, in short). More formally, **SArg-RAM** comprises of two phases, the *offline* and the *online* phase. In the offline phase, $\mathcal{P}$ commits to $\mathbf{x}$. In the online phase, $\mathcal{P}$ proves to $\mathcal{V}$ statements of

the form: $\exists w$ s.t. $\mathcal{R}(\mathbf{x}, w) = \text{accept}$, where $\mathcal{R}$ is an NP-relation that can be efficiently verified by a RAM program. The nontrivial efficiency goal is to respect the efficient verification requirements of standard succinct arguments, while ensuring that the online runtime of the prover is comparable to T , which is sublinear in n . The notion of **SArg-RAM** is very compelling and can be motivated by many applications that involve sublinear-time computations over databases. Such applications include proving statements about small subsets of entries, searching data structures, sequence matching (e.g., for genomic data or blockchains), statistical analysis, and more.

Indeed, notions similar to **SArg-RAM** have been considered in prior works. Examples include zero-knowledge¹ sets [20, 50], where $\mathcal{P}$ proves membership or non-membership assertions on the committed set, consistency proofs on committed databases [52], and sublinear zero-knowledge arguments for the execution of RAM programs on committed database [51].

A Generic Non-black-box Approach. Traditional approaches for constructing succinct arguments or their non-interactive variants called SNARKs² [15, 17, 30, 31, 37, 46, 49] typically rely on a circuit representation of the relation $\mathcal{R}$. Even when optimized for RAM programs [9, 10], they do not directly yield arguments with sublinear prover runtime. However, as already observed in [52] in the context of zero-knowledge proofs for data structures, there is a simple and general technique for converting any succinct “argument of knowledge” into a **SArg-RAM** with a low communication and computation overhead. In the offline phase, $\mathcal{P}$ succinctly commits to $\mathbf{x}$ using a *vector commitment* scheme, say based on a Merkle tree [48]. In the online phase, $\mathcal{P}$ then argues that it knows a way to open a subset of the committed entries of $\mathbf{x}$ which is consistent with some (short) accepting transcript. The downside of this approach is that it requires proofs for NP statements that involve the succinct commitment primitive, thus inherently making a *non-black-box* use of this primitive. Such non-black-box constructions are undesirable both in theory and in practice. In particular, they cannot be implemented unconditionally in generic models (such as the random oracle model), and typically involve a big concrete efficiency overhead. As a relevant example, non-black-box constructions of incrementally verifiable computation [16, 21, 57] (via recursive SNARKs) are much less efficient than black-box constructions of standard SNARKs. Indeed, there has been a large body of work on replacing non-black-box cryptographic constructions by black-box counterparts (see, e.g., [22, 24, 35, 36, 39, 42, 53, 54] and references therein).

The above state of affairs gives rise to the following question:

*Is there a **SArg-RAM** on committed databases that respects the runtime of the RAM program and only makes a black-box use of cryptography?*

¹ Zero-knowledge means no information beyond the validity of the statement is leaked to $\mathcal{V}$.

² A SNARK satisfies the additional *knowledge* property, which intuitively requires that if a prover can make the verifier accept, then it must know an NP-witness.

We answer this question in the affirmative, making a black-box use of any collision-resistant hash function. Alternatively, we get an *unconditional* non-interactive variant of this result in the random oracle model. Prior to our work, such black-box protocols were not known even if we only restrict the prover’s runtime and the communication, but not the verifier’s runtime.

1.1 Black-Box Succincts Argument for RAM Programs

We now give a more detailed overview of our results and techniques, starting with a more precise statement of the main feasibility result described above.

Definition 1 (SArg-RAM, informal). *An SArg-RAM allows a computationally bounded prover to succinctly commit to a database in an offline phase, such that in a later online phase it can prove the execution of RAM programs whose runtime is bounded by T .*

- **Completeness:** *If the statement about the execution of a RAM program on the committed database is true, $\mathcal{P}$ should always be able to convince $\mathcal{V}$.*
- **Soundness:** *If the statement is false, then any efficient malicious $\mathcal{P}^*$ can only make $\mathcal{V}$ accept with negligible probability.*
- **Efficiency:** *The online runtime of $\mathcal{P}$ should scale with T and not with the database size. Moreover, the protocol should be succinct in the sense that the communication, as well as the runtime of $\mathcal{V}$, are polylogarithmic in n .*

We obtain the following main result.

Theorem 1 (Black-box SArg-RAM, informal). *Assume a black-box access to a collision-resistant hash function. Then, for every $\alpha > 0$, there is $c_\alpha > 1$ for which there is an SArg-RAM with the following features:*

- *In the offline phase, $\mathcal{P}$ commits to an n -bit database $\mathbf{x}$ in time $\tilde{O}(n^{c_\alpha})$ and using $\text{polylog}(n)$ bits of communication;*
- *Each invocation of the online phase proves the validity of (a possibly different) NP-relation $R(\mathbf{x}, w)$ that can be verified by a RAM program in (possibly sublinear) time $T(n)$. $\mathcal{P}$ runs in time $\tilde{O}(T^{1+\alpha})$ and $\mathcal{V}$ in time $\text{polylog}(n)$.*

Alternatively, $\mathcal{P}$ ’s offline runtime can be improved to $\tilde{O}(n^{1+\alpha})$ at the expense of $\tilde{O}(T^2)$ online runtime, for any $\alpha > 0$.

At the heart of our construction for SArg-RAM is a new primitive that we call “projection codes,” which we describe in detail next.

1.2 Projection Codes

The main technical challenge in constructing a black-box SArg-RAM involves a conversion of a robust encoding of the database into a compact but robust encoding of a small subset of the database. This task is captured by the notion of projection codes. Consider an encoding scheme Enc that encodes an n -bit message $\mathbf{x}$ into an N -bit codeword $\mathbf{X}$. The guarantee we seek is that given any

sequence $\mathbf{s} \in [n]^k$ of k distinct message locations, there is a sequence $\mathbf{S} \in [N]^K$ of K codeword locations, such that $\mathbf{x}|_{\mathbf{s}}$ can be recovered from $\mathbf{X}|_{\mathbf{S}}$ even in the presence of a constant fraction of errors. Here, $\mathbf{x}|_{\mathbf{s}}$ denotes the sequence of bits in $\mathbf{x}$ restricted to indices in $\mathbf{s}$. Naturally, we would like N and K to be not much larger than n and k respectively. More formally:

Definition 2 (Projection code). Let $\text{Enc} : \{0, 1\}^n \rightarrow \{0, 1\}^N$ be an encoding function and $\text{Proj} : [n]^k \rightarrow [N]^K$ be a projection algorithm. We say that Enc is an (n, N, k, K, δ) -projection code if for every sequence $\mathbf{s} \in [n]^k$ of k distinct message locations, Proj outputs a sequence $\mathbf{S} \in [N]^K$ of K codeword locations such that there exists a subset decoder Dec with the following robustness guarantee. For every $\mathbf{x} \in \{0, 1\}^n$, and every $\tilde{\mathbf{X}}|_{\mathbf{S}}$ which is δ -close to $\mathbf{X}|_{\mathbf{S}}$ in relative Hamming distance, where $\mathbf{X} = \text{Enc}(\mathbf{x})$, we have $\text{Dec}(\tilde{\mathbf{X}}|_{\mathbf{S}}) = \mathbf{x}|_{\mathbf{s}}$.

We construct projection codes with the following (near-optimal) parameters.

Theorem 2 (Near-optimal projection codes, informal). For any $\alpha > 0$ and $k = k(n)$ there is an efficient $(n, O(n^{1+\alpha}), k, \tilde{O}(k), \Omega(1))$ -projection code.

In this construction, the runtimes of Enc and Proj are $\tilde{O}(n^{1+\alpha})$ and $\tilde{O}(k^2)$ respectively. The quadratic overhead of Proj can be avoided by an alternative (simpler) construction, at the expense of a bigger encoding size. Concretely, for any $\alpha > 0$, there is $c_\alpha > 1$ for which there is an efficient $(n, O(n^{c_\alpha}), k, O(k^{1+\alpha}), \Omega(1))$ projection code such that the runtimes of Enc and Proj are $\tilde{O}(n^{c_\alpha})$ and $\tilde{O}(k^{1+\alpha})$ respectively. These two families of projection codes correspond to the two possible tradeoffs between the online and offline runtime given by Theorem 1.

1.3 Related Work

Locally Decodable Codes. Our construction of projection codes is inspired by a construction of *multi-point locally decodable codes* from the work of Cramer, Xing, and Yuan [23]. A standard locally decodable code (LDC) [44, 59] allows a single bit of the encoded message to be decoded with high probability by only querying a small number of bits of a possibly corrupted codeword. Cramer et al. showed how to optimize the query complexity when decoding k bits simultaneously. Interestingly, while our construction of projection codes is technically very similar, projection codes are a conceptually different object, and we are not aware of a *general* construction of projection codes from multi-point LDCs. In particular, if an extra polylog factor is allowed, multi-point decoding can be trivially realized by invoking the decoder of a polylog-query LDC k times. This approach does not work for our purposes, since we need the projected codeword to be robust. Additionally, while LDCs are inherently randomized, projection codes are fully deterministic: for a set $\mathbf{s}$, the projected set $\mathbf{S}$ is fixed. As will be highlighted in Sect. 2, this requirement is used in the construction of SArg-RAM using projection codes.

SArg-RAM with Superlinear $T(n)$. The question of efficient succinct arguments for the execution of RAM programs was first explicitly studied by Ben-Sasson

et al. [9,11]. These works aim to minimize the overhead of arguing about the execution of a RAM program with runtime T on a statement of size n when $T \gg n$. In contrast, we focus on the case of RAM programs that take sublinear time (e.g., $T = O(n^{1/4})$) in the size of a *committed* database. Applying the results of [9,11] in this setting would lead to $\tilde{O}(n)$ online prover runtime.

Zero-Knowledge SArg-RAM. The question of *zero-knowledge* SArg-RAM was considered in [26,51]. Mohassel et al. [51] obtain such black-box protocols in which the online communication is $\tilde{O}(T)$. Franzese et al. [26] improve this by eliminating the polylogarithmic terms. In the above works, the online communication scales (at least) linearly with the RAM runtime T , whereas in our SArg-RAM it is polylogarithmic in n . One should note in this context that black-box SArg-RAM without zero knowledge are trivialized if we allow the online communication to scale linearly with the T . Indeed, the prover can just reveal the values read by the RAM program, including committed database bits and bits of the NP-witness. The verifier can then run the RAM program to check that the database is accepted. The question becomes nontrivial when one additionally requires the SArg-RAM to be zero knowledge.

2 Overview of Techniques

2.1 Projection Codes

We start by recalling the definition of standard Reed-Muller codes, on which our projection code constructions are based.

Reed-Muller encoding. Let $\mathbf{u} = (u_1, \dots, u_v)$ and let us denote the multivariate polynomial ring $\mathbb{F}_q[u_1, \dots, u_v]$ by $\mathbb{F}_q[\mathbf{u}]$. The Reed-Muller encoding is defined using a polynomial $p(\cdot) \in \mathbb{F}_q[\mathbf{u}]$ of total degree at most d over message space $\mathcal{M} = \mathbb{F}_q^n$ such that $n = \binom{v+d}{v}$. For $i \in [n]$, let $\mathbf{C}_i = (C_{i,1}, \dots, C_{i,v}) \in \mathbb{F}_q^v$ denote n distinct but fixed vectors in $\mathbb{F}_q^v$ such that the corresponding linear system has full-rank. Reed-Muller code $\mathcal{E}_{\text{RM}}$ with parameters (n, N, q, d, v) is defined with encoding Enc as below. For message $\mathbf{x} = (x_1, \dots, x_n)$, $\mathcal{E}_{\text{RM}}.\text{Enc}(\mathbf{x}) = \{(p(\mathbf{u}))_{\mathbf{u} \in \mathbb{F}_q^v} : p(\mathbf{u}) \in \mathbb{F}_q[\mathbf{u}], \text{degree of } p(\cdot) \leq d \text{ and } p(\mathbf{C}_i) = x_i, \forall i \in [n]\}$. The codeword length is $N = q^v$ and the message length satisfies $n = \binom{v+d}{v}$. We say that the codeword is *systematic* if the message symbols appear directly as the part of the codeword at pre-defined locations. Thus, Reed-Muller encoding is systematic.

A Simple Construction. We begin with a simple example for projection codes that uses the standard local decoding procedure for Reed-Muller codes [6,47].

Global codeword. Given a message $\mathbf{x}$ in $\mathbb{F}_q^n$, encode it using the systematic Reed-Muller encoding to obtain a codeword $\mathbf{X} \in \mathbb{F}_q^N$.

Projector. Given k message indices $s_1, \dots, s_k$, let $\mathbf{R}_1, \dots, \mathbf{R}_k$ be the corresponding codeword locations as defined in the systematic encoding procedure (i.e., $\mathbf{R}_i = \mathbf{C}_{s_i}$, defined above). Define v univariate polynomials $g_1(t), \dots, g_v(t)$, corresponding to each of the v variables $u_1, \dots, u_v$ satisfying the following: i) Each polynomial $g_i(t)$ is of degree at most $k-1$; ii) $\mathbf{g}(t) = (g_1(t), \dots, g_v(t))$ passes

through $\mathbf{R}_1, \dots, \mathbf{R}_k$, i.e., for each $j \in [k]$, $\mathbf{R}_j = \mathbf{g}(j)$. The projector outputs q locations obtained by evaluating $\mathbf{g}$ over its domain, namely $\mathbf{g}(i)$ for $i \in \mathbb{F}_q$.

Analysis. Restricting any multivariate polynomial of degree at most d to a univariate polynomial of degree at most $k - 1$ yields a univariate polynomial of degree at most $d \cdot (k - 1)$. Thus, when we project the global Reed-Muller code to the output of the projector, we end up with a Reed-Solomon code over $\mathbb{F}_q$ corresponding to a univariate polynomial of degree at most $d \cdot (k - 1)$. This is exactly our projection code with distance $q - d \cdot (k - 1)$.

Using the above approach, we can get projection codes of polynomial length that have a near-optimal projection size. However, the parameters it can achieve are limited in two ways. First, they cannot simultaneously achieve near-optimal encoding size and projection size, as in Theorem 2. Second, they do not yield good projection codes that are also locally testable and correctable with poly-logarithmic queries (see Sect. 3.2). This additional feature, while not essential, simplifies our main SArg-RAM construction. To get around these two limitations, we need to look for a more general approach.

A Generalized Construction. At a very high level, the idea is to encode the given codeword locations using a suitable encoding scheme in the projection.

Projector: Encode the codeword locations. Let the k codeword indices, say $\mathbf{R}_1, \dots, \mathbf{R}_k \in [N]$, correspond to the given k message indices based on the systematic encoding procedure. For each $i \in [k]$, parse $\mathbf{R}_i = (R_{i,1}, \dots, R_{i,v})$ as $\mathbb{F}_q^v \equiv [N]$. Then for each $j \in [v]$, encode $(R_{1,j}, \dots, R_{k,j}) \in \mathbb{F}_q^k$ into K symbols $(S_{1,j}, \dots, S_{K,j}) \in \mathbb{F}_q^K$ using an encoding scheme that will be appropriately chosen. For each $i \in [K]$, get v symbols $(S_{i,1}, \dots, S_{i,v}) \in \mathbb{F}_q^v$ and parse them as a codeword location $\mathbf{S}_i \in [N]$. The projector outputs these K locations, $(\mathbf{S}_1, \dots, \mathbf{S}_K) \in [N]^K$. Figure 1 illustrates the above process pictorially.

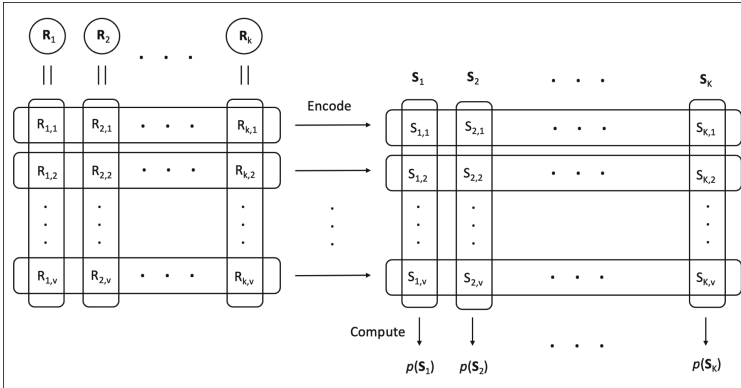


Fig. 1. Our projector in nutshell. Codeword indices $s_1, s_2, \dots, s_k$ correspond to the given k message indices and p is the v -variate polynomial of our Reed-Muller encoding of total degree d .

We now consider the codeword $\mathbf{Y} = (p(\mathbf{S}_1), \dots, p(\mathbf{S}_K)) \in \mathbb{F}_q^K$ formed by projecting the Reed-Muller codeword to these K locations, $(\mathbf{S}_1, \dots, \mathbf{S}_K)$. For $\mathbf{Y}$ to encode $\mathbf{x}|_{\mathbf{s}}$, we need the encoding scheme used in the projector to be a d -multiplicative linear code. Informally, a d -multiplicative linear code is a linear code that encodes k elements of $\mathbb{F}_q$ into K elements of $\mathbb{F}_q$ such that pointwise product (coordinate-wise multiplication) of up to d codewords in the code results in a new codeword (in a larger linear code); see Sect. 3.2 for a formal definition of d -multiplicative codes and Lemma 2 for a formal version of the above projection procedure using such codes.

In Sect. 4.2, we analyze the achievable parameters when the projector code is instantiated with either Reed-Muller codes or algebraic-geometric (AG) codes, as used in [23, 28]. These two options correspond to the two alternative parameters in Theorem 2. In a nutshell, the projector based on AG codes has near-optimal code length and projection size parameters but runs in time $\tilde{O}(k^2)$, whereas the RM-based projector can run in near-linear time $O(k^{1+\alpha})$ at the expense of a polynomial-size encoding.

We leave open the existence of projection codes with a linear-time encoding algorithm. Indeed, our approach heavily relies on multiplication-friendly codes, for which no linear-time instances are known. This should be contrasted with existing constructions of linear-time succinct arguments that can rely on standard linear-time encodable codes [18, 19, 33, 55, 58].

2.2 Holographic PCPs for RAM Programs

As an information-theoretic application of projection codes, which serves as a stepping stone for SArg-RAM, we obtain a RAM analogue of classical PCPs [2, 3] that we call a *holographic PCP for RAM programs* (HPCP-RAM).³ For ease of exposition, we first consider a simpler notion of a holographic PCP for *subsets* (HPCP-SS). In HPCP-SS, $\mathcal{P}$ encodes a database $\mathbf{x}$ in the offline phase using an encoding function E_1 to obtain $\mathbf{X} = E_1(\mathbf{x})$. In the online phase, for $\mathbf{s}$ encoded by another encoding function E_2 , $\mathcal{P}$ is interested to prove an NP-statement on the projected message $\mathbf{x}|_{\mathbf{s}}$. More formally, for an NP relation $\mathcal{R}$, $\mathcal{P}$ wants to generate a PCP proof for $\exists w$ s.t. $\mathcal{R}(\mathbf{x}|_{\mathbf{s}}, w) = \text{accept}$. Informally, an HPCP-SS construction should satisfy the following properties (cf. Definition 9):

- **Completeness:** If $\mathbf{x}|_{\mathbf{s}}$ satisfies the input statement, then $\mathcal{P}$ can generate a PCP proof which is always accepted by $\mathcal{V}$.
- **Soundness:** If $\mathbf{x}|_{\mathbf{s}}$ does not satisfy the input statement, then for any proof generated by $\mathcal{P}$, $\mathcal{V}$ should accept the proof only with negligible probability.

We would like the runtime of $\mathcal{P}$ in the online phase to be comparable to the size of the input set $\mathbf{s}$, independently of the size of $\mathbf{x}$, while $\mathcal{V}$ is fully succinct in that its runtime is $\text{polylog}(n)$. It is easy to discern that if there exists an

³ The term “holographic PCP” [4] refers to a PCP on an *encoded* input statement. This allows the verifier to only make a small number of queries to *both* the encoded input and the proof. Such PCPs can be constructed from *PCPs of proximity* [13].

encoding of $\mathbf{x}|_{\mathbf{s}}$, say $\mathbf{Y}$, which is proven to be consistent with global encoding, i.e., $\mathbf{X} = \mathbf{E}_1(\mathbf{x})$, $\mathcal{P}$ could generate a PCP proof with respect to codeword $\mathbf{Y}$. This is exactly the scenario in which projection codes are applicable.

In the offline phase, $\mathcal{P}$ encodes $\mathbf{x}$ using global encoder of projection codes. For the online phase, note that the projected codeword $\mathbf{X}|_{\mathbf{s}}$ forms an encoding of $\mathbf{x}|_{\mathbf{s}}$ and is part of the global codeword. Thus, $\mathcal{P}$ can generate a PCP proof with respect to the projected codeword. A deterministic projection procedure from $\mathbf{s}$ to $\mathbf{S}$ allows $\mathcal{P}$ to ascertain the encoded input for PCP without having the need to interact with $\mathcal{V}$ to sample randomness. Using PCPs, $\mathcal{P}$ then proves that projection of $\mathbf{s}$ is indeed $\mathbf{S}$, and the input statement on $\mathbf{x}|_{\mathbf{s}}$ (encoded with $\mathbf{X}|_{\mathbf{s}}$) is valid. Through this approach, we obtain a construction of HPCP-SS with the desired asymptotic complexity. HPCP-SS are easily extendable to HPCP-RAM. The idea is to view the bits accessed by the RAM program as the subset of bits on which we would like to prove an input statement. We obtain this extension based on a standard reduction of RAM programs to TM [38] (see Theorem 5).

Why aren't Standard LDCs Enough? For the purpose of the above HPCP-SS construction, it may appear that projection codes are an overkill and one could apply a standard locally decodable code (LDC) instead. However, that is not the case. Let the size of the subset $\mathbf{s}$ be $k \ll n$. The crucial point is that we need to apply a *short* holographic PCP, of length comparable to k , to a *robust* encoding of $\mathbf{x}|_{\mathbf{s}}$. If we implicitly encode each entry of $\mathbf{x}|_{\mathbf{s}}$ using the answers to a standard LDC, this encoding is not robust (since each entry is encoded separately). If we try to define an NP statement that involves the entire global encoding of $\mathbf{x}$, it is not clear how to make the PCP short.

2.3 Succinct Arguments for RAM Programs

As our final application, we construct a black-box SArg-RAM as follows. In the offline phase, $\mathcal{P}$ does the offline processing of $\mathbf{x}$ by encoding it using the global encoder of the projection code to obtain $\mathbf{X} \leftarrow \text{Enc}(\mathbf{x})$. $\mathcal{P}$ then uses Merkle hashing [48] to commit to $\mathbf{X}$, and sends short commitment to $\mathcal{V}$. The online phase is executed as in the Kilian's argument system [5, 12, 46].

In the online phase, $\mathcal{P}$ generates a PCP proof using HPCP-RAM construction and commits to the proof using Merkle hashing. $\mathcal{V}$ runs the verifier of HPCP-RAM, $\mathcal{V}'$. For every query of $\mathcal{V}'$, $\mathcal{V}$ forwards the query to $\mathcal{P}$, who opens the Merkle tree hashes corresponding to that query.

3 Preliminaries

We summarize the denotations of frequently used symbols in Table 1. For a vector $\mathbf{x} = (x_1, x_2, \dots, x_\ell)$, $|\mathbf{x}|$ denotes length of vector $\mathbf{x}$. For a positive integer ℓ , $[\ell] = \{1, \dots, \ell\}$. For a sequence $\mathbf{s} = (s_1, s_2, \dots, s_\ell)$ and $a \leq \ell$ -sized sequence $A = (j_1, j_2, \dots, j_a) \in [\ell]^a$, $\mathbf{s}|_A = (s_{j_1}, s_{j_2}, \dots, s_{j_a})$. Let $\mathbf{x} = (x_1, \dots, x_\ell)$ and $\mathbf{y} = (y_1, \dots, y_\ell)$, $\mathbf{x} \circ \mathbf{y}$ denotes concatenation of two sequences. We abuse this notation by using it for some scalar a . Unless stated explicitly, the size of the alphabet space of all the codewords that we use is q , i.e., $\mathbb{F}_q$ is the code alphabets

space. For $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^\ell$, $\mathbf{u} * \mathbf{v}$ denotes the pointwise product of the vectors defined as $(u_1 v_1, \dots, u_\ell v_\ell) \in \mathbb{F}_q^\ell$ and $w_H(\mathbf{u}, \mathbf{v})$ denotes the hamming distance between $\mathbf{u}$ and $\mathbf{v}$. We denote empty-string by $\emptyset$. We use $\log(\cdot)$ to denote logarithm to the base 2. Unless otherwise stated, we assume that all the algorithms in the computational setting implicitly take the security parameter λ as input.

Table 1. Table of Notations

Symbol	Denotation
$\mathcal{M}$	Message Space
$\mathbf{x}, \mathbf{y}$	Lower-case boldface letters denoting message vectors
n	Length of the global message
$\mathcal{C}$	Codeword Space
$\mathbf{X}, \mathbf{Y}$	Upper-case boldface letters denoting codeword vectors
N	Length of the global codeword
k	Size of the projected message sequence, i.e., $ \mathbf{s} $
$\mathbf{s}$	Sequence of distinct message locations to be projected ($\mathbf{s} \in [n]^k$)
K	Size of the projected codeword sequence, i.e., $ \mathbf{S} $
$\mathbf{S}$	Sequence of projected codeword locations ($\mathbf{S} \in [N]^K$)
δ	Fractional distance of projection code
λ	Computational security parameter

3.1 Model of Computation

We consider the Random Access Memory (RAM) model of computation and Turing Machines (TM). We denote a deterministic TM (resp. a non-deterministic TM) that on input of length n runs in time $T(n)$ (where $T : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$) by $\text{DTIME}(T(n))$ (resp. $\text{NTIME}(T(n))$) machines. If not mentioned explicitly, an algorithm is assumed to run in RAM model of computation.

We consider the standard concept of RAM model [1]. We adopt the definition of RAM model of computation from [32] which models it as an interactive TM communicating with Random Access Memory (RAM). We focus on RAM programs M that execute on a large RAM memory, $\mathbf{x}$, of size n .⁴

M accesses certain locations in $\mathbf{x}$ through a series of read/write instructions. Similar to [26, 29, 34], we consider a RAM program to be defined by a *next-instruction* computation unit. In this work, we define the RAM program by *next-instruction* TM Π . Π takes the current state $\mathbf{st}$ and an input bit $b \in \{0, 1\}$, and outputs the updated state $\mathbf{st}'$ along with the next instruction to execute. More formally, initialize $\mathbf{st} = \mathbf{start}$ and $b = 0$, and then until termination compute $(\mathbf{op}, \ell, b', \mathbf{st}') \leftarrow \Pi(\mathbf{st}, b)$, and set $\mathbf{st} = \mathbf{st}'$.

⁴ The RAM memory $\mathbf{x}$ comprises of the database as well as additional working space needed by M . The space reserved for database is instantiated with initial value of the database, i.e., before the first RAM program is executed. If operations are performed by RAM program over a bit in reserved working memory space, the first such operation is always a Write operation. This setting is similar to [29, 41, 51].

- If $\text{op} = \text{Stop}$, terminate with output b' .
- If $\text{op} = \text{Read}$, set $b = x_\ell$.
- If $\text{op} = \text{Write}$, set $x_\ell = b'$ and $b = b'$.

Thus, execution of RAM program M consists of executions of series of next-instruction TM Π initialized with start state until an instance of next-instruction TM outputs operation **Stop**. Here, we are interested in next-instruction TM Π whose computation time is $\text{polylog}n$.

3.2 Encoding Scheme

Definition 3 (Encoding Scheme). *An encoding scheme over $\mathbb{F}_q$ with message length n , codeword length N , relative distance δ , and relative decoding radius ρ , is defined by a pair of algorithms (Enc, Dec) defined below.*

1. $\text{Enc} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^N$. Enc maps a message $\mathbf{x} \in \mathbb{F}_q^n$ to codeword $\mathbf{X} \in \mathbb{F}_q^N$ such that for any $\mathbf{x} \neq \mathbf{x}'$, $w_H(\text{Enc}(\mathbf{x}), \text{Enc}(\mathbf{x}')) \geq \delta N$.
2. $\text{Dec} : \mathbb{F}_q^N \rightarrow \mathbb{F}_q^n$. Dec takes as input (possibly corrupted) codeword $\mathbf{X}' \in \mathbb{F}_q^N$. For $\mathbf{x} \in \mathbb{F}_q^n$, if $w_H(\text{Enc}(\mathbf{x}), \mathbf{X}') \leq \rho N$, where $\rho < \delta/2$, then Dec outputs $\mathbf{x}$.

A family of such encoding schemes $(\text{Enc}_n, \text{Dec}_n)$ with $q = \text{polylog}(n)$, $N = N(n)$ for some polynomial $N(\cdot)$, and fixed $\delta > 0$, is efficiently encodable (resp. decodable) if Enc (resp. Dec) runs in probabilistic polynomial time (PPT) in n . We refer to $\mathcal{E} = (\text{Enc}, \text{Dec})$ as an (n, N, δ) -encoding scheme with relative decoding radius ρ . Let $\mathcal{C} = \{\mathcal{E}.\text{Enc}(\mathbf{x}) \mid \forall \mathbf{x} \in \mathbb{F}_q^n\} \subseteq \mathbb{F}_q^N$ denote the codeword space of $\mathcal{E}$.

Locally Testable Codes. An encoding scheme is locally testable code with parameters: number of queries Q , relative distance δ' , and error bound ϵ (or (Q, δ', ϵ) -locally testable, in short) if there is a randomized local testing algorithm $\mathcal{A}$ (probabilistic TM) that takes as input ϵ (in binary) and is given oracle access to $\mathbf{X}$. $\mathcal{A}$ is required to run in $\text{polylog}(N/\epsilon)$ time and read at most Q locations in a codeword $\mathbf{X}$. At the end of the execution, $\mathcal{A}$ outputs either **accept/reject**. $\mathcal{A}$ satisfies the following properties:

- Completeness: For every $\mathbf{x} \in \mathbb{F}_q^n$, given oracle access to $\text{Enc}(\mathbf{x})$, $\mathcal{A}$ accepts with probability 1, i.e.,

$$\Pr[\mathcal{A}^{\text{Enc}(\mathbf{x})}(\epsilon) = \text{accept}] = 1.$$

- Soundness: For any $\mathbf{X}'$ which is more than δ' far from a valid codeword, $\mathcal{A}$ accepts $\mathbf{X}'$ with probability at most ϵ . That is, for any $\mathbf{X}'$ s.t. $\forall \mathbf{x} \in \mathbb{F}_q^n$, $w_H(\mathbf{X}', \text{Enc}(\mathbf{x})) \geq \delta' N$,

$$\Pr[\mathcal{A}^{\mathbf{X}'}(\epsilon) = \text{accept}] \leq \epsilon.$$

Locally Decodable Codes. An encoding scheme is locally decodable code with parameters: number of queries Q , relative distance δ' , and error bound ϵ (or (Q, δ', ϵ) -locally decodable, in short) if there is a randomized decoder $\mathcal{A}$ (probabilistic TM) that takes as input ϵ (in binary) and index $i \in [n]$, and is given oracle access to $\mathbf{X}$. $\mathcal{A}$ is required to run in $\text{polylog}(N/\epsilon)$ time and make at most Q many non-adaptive queries to codeword $\mathbf{X}$. At the end of the execution, $\mathcal{A}$ outputs $Y \in \mathbb{F}_q$. $\mathcal{A}$ satisfies the following properties:

- For every $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{F}_q^n$ and index $i \in [n]$, given oracle access to $\text{Enc}(\mathbf{x})$, $\mathcal{A}$ outputs x_i with probability 1, i.e.

$$\Pr[\mathcal{A}^{\text{Enc}(\mathbf{x})}(\epsilon, i) = x_i] = 1.$$

- For every message $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{F}_q^n$, index $i \in [n]$ and codeword $\mathbf{X}'$ such that $w_H(\text{Enc}(\mathbf{x}), \mathbf{X}') \leq \delta'N$,

$$\Pr[\mathcal{A}^{\mathbf{X}'}(\epsilon, i) = x_i] \geq 1 - \epsilon.$$

Locally correctable codes are defined similarly [59]. For a (Q, δ', ϵ) -locally correctable code, there exists a randomized algorithm $\mathcal{A}$ that given index i , makes at most Q -many oracle queries to $\mathbf{X}'$ that is at most $\delta'N$ far away from a valid codeword $\mathbf{X}$ and outputs X_i with probability at least $1 - \epsilon$.

There exists RM code with parameters $q = \text{polylog}(n)$, $d = \text{polylog}(n)$, $v = \log(n)/c \log \log n$, and $N = O(n^{1+\alpha})$ for $c, \alpha > 0$ (cf. [40, Theorem 2.2]). Moreover, there exists $\epsilon = \text{negl}(n)$ and $Q = \text{polylog}(n)$ such that for any $0 < \delta < \frac{1}{2}$, the RM code with aforementioned parameters is (Q, δ, ϵ) -locally testable code [27, Theorem 7], [56, Theorem 9], and (Q, δ, ϵ) -locally decodable/ (Q, δ, ϵ) -locally correctable code [59, Proposition 2.6].

Multiplicative Linear Codes. Our projection code construction utilizes *multiplicative linear codes*. We recall the definition of multiplicative linear code that are often used for non-interactive computation of low-degree polynomials. This idea can be traced back to Ben-Or, Goldwasser, and Wigderson [7].

Definition 4 ($\mathcal{C}^{*d}$: d^{th} -power of $\mathcal{C}$). For an $\mathbb{F}_q$ -linear code $\mathcal{C} \subseteq \mathbb{F}_q^N$, the $\mathbb{F}_q$ -linear code $\mathcal{C}^{*d} \subseteq \mathbb{F}_q^N$, the d^{th} power of $\mathcal{C}$, is defined as the $\mathbb{F}_q$ -linear subspace generated by all terms of the form $\mathbf{C}_1 * \dots * \mathbf{C}_d$ with $\mathbf{C}_1, \dots, \mathbf{C}_d \in \mathcal{C}$.

Definition 5. An efficient (n, N, δ) -encoding scheme $\mathcal{E} = (\text{Enc}, \text{Dec})$ with code-word space $\mathcal{C} \subseteq \mathbb{F}_q^N$ is d -multiplicative linear, if there exist an efficient algorithm⁵ Dec^* (called special decoder) and a decoder for $\mathcal{C}^{*d}$, $\text{Dec}^{(d)}$, such that the following conditions are satisfied:

1. $\mathcal{C} \subseteq \mathbb{F}_q^N$ is a $\mathbb{F}_q$ -linear code and $\text{Dec}^* : \mathcal{C} \rightarrow \mathbb{F}_q^n$ is a surjective $\mathbb{F}_q$ -vector space morphism.

⁵ Polynomial time (possibly probabilistic) in (n, d) .

2. It is unital, i.e., $\mathbf{1} \in \mathcal{C}$ and $\text{Dec}^*(\mathbf{1}) = \mathbf{1}$.
3. The map Dec^* extends uniquely to an $\mathbb{F}_q$ -linear map $\text{Dec}^* : \mathcal{C}^{*d} \rightarrow \mathbb{F}_q^n$ such that the following holds:
 - Dec^* satisfies the multiplicative relation

$$\text{Dec}^*(\mathbf{X}_1 * \dots * \mathbf{X}_d) = \text{Dec}^*(\mathbf{X}_1) * \dots * \text{Dec}^*(\mathbf{X}_d) \in \mathbb{F}_q^n$$

for all $\mathbf{X}_1, \dots, \mathbf{X}_d \in \mathcal{C}$.

- $\mathcal{C}^{*d}$ has minimum distance at least δN .
4. Given $\mathbf{X}' \in \mathbb{F}_q^N$ that is at most δ -far from a codeword $\mathbf{X} \in \mathcal{C}^{*d}$, $\text{Dec}^{(d)}(\mathbf{X}') = \mathbf{X}$.

Succinctly, we refer $\mathcal{E}$ as (n, N, d, δ) -multiplicative linear encoding scheme.

3.3 Probabilistically Checkable Proofs of Proximity (PCPPs)

Informally, for an NP relation $\mathcal{R}$, probabilistic checkable proofs (PCPs) [3] are proofs that can be verified by reading only few bits of the proofs. In this work, we consider the notion of PCPs of Proximity (PCPPs) [14, 25] which we recall below. Similar to [14], we consider *pair languages* in which an input is of the form $(\mathbf{v}, \mathbf{x})$. For instance, $\mathbf{v}$ can be a description of a TM and $\mathbf{x}$ an input accepted by this TM. The first part of the input $\mathbf{v} \in \{0, 1\}^\ell$ is the *explicit* part, read entirely by the verifier, and the second part $\mathbf{x} \in \{0, 1\}^n$ is the *implicit* part to which the verifier has oracle access (we typically have $\ell \ll n$). For the application of PCPPs in our work (see Sect. 5), it will be convenient to associate with a PCPP a specific encoding function $\mathbf{E}$, where the verifier has oracle access not to $\mathbf{x}$ but to $\mathbf{X} = \mathbf{E}(\mathbf{x})$. The case of encoded implicit inputs $\mathbf{X}$ can be handled by simply defining an NP language of encoded instances. Then, a PCPP for NP (as defined in [14]) suffices.

Definition 6. (PCPs of Proximity (PCPPs).) Let $\mathcal{R}((\mathbf{v}, \mathbf{x}), w) \in \text{DTIME}(T(n))$ be an NP relation where $n = |\mathbf{x}|$ and $T : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$. A probabilistic proof system $(\mathcal{P}, \mathcal{V})$ (where $\mathcal{P}$ is a probabilistic RAM program and $\mathcal{V}$ is a probabilistic TM) with parameters: query complexity of $\mathcal{V}$ (Q), runtime of prover (t_p), runtime of verifier (t_v), proof length (ℓ_p), soundness error (ϵ), and proximity parameter (δ), is a PCPP system for $\mathcal{R}$ with respect to encoding function $\mathbf{E}$ if the following holds for every pair $(\mathbf{v}, \mathbf{x})$:

- **Input:** $\mathcal{P}$ receives $\mathbf{v} \in \{0, 1\}^\ell, \mathbf{x} \in \{0, 1\}^n$ and $w \in \{0, 1\}^\mu$ as input, and $\mathcal{V}$ receives $n \in \mathbb{N}, \mathbf{v} \in \{0, 1\}^\ell$ as input.
- **Input encoding:** $\mathcal{P}$ computes $\mathbf{X} = \mathbf{E}(\mathbf{x})$.
- **Prover's proof:** $\mathcal{P}$ generates a proof π (corresponding to statement $\mathcal{R}((\mathbf{v}, \mathbf{x}), w) = \text{accept}$) such that $|\pi| \leq \ell_p(n)$. We require runtime of $\mathcal{P}$ in proof generation to be bounded by $t_p(n)$.
- **Verifier's queries:** $\mathcal{V}$ makes $Q = Q(n)$ oracle queries $\mathbf{X}$ and π , and based on the responses outputs either **accept** or **reject**. We require the runtime of $\mathcal{V}$ to be within $t_v(n, \ell)$.
- **Completeness:** If $\mathcal{R}((\mathbf{v}, \mathbf{x}), w) = \text{accept}$, then when $\mathcal{V}$ queries the proof π generated by $\mathcal{P}$, it accepts with probability 1.

- *Soundness:* Suppose $\mathbf{v}, \mathbf{X}^*$ are such that for all $\mathbf{x}'$ for which there exists w such that $\mathcal{R}(\mathbf{v}, \mathbf{x}', w) = \text{accept}$, we have that $\mathbf{X}^*$ is δ -far from $\mathbf{E}(\mathbf{x}')$. Then, $\mathcal{V}$ accepts $(\mathbf{v}, \mathbf{X}^*)$ with probability at most $\epsilon = \epsilon(n)$.

Succinctly, we refer to such a system as a $(Q, t_p, t_v, \ell_p, \epsilon, \delta)$ -PCPP system.

4 Projection Codes

In this section, we first formalize our new primitive, i.e., *projection codes*. Later in the section, we provide efficient construction of projection codes inspired from techniques used in [23] to build multi-point locally decodable codes.

Definition 7. (Projection code.) For $k \leq n$, $K \leq N$, and $0 \leq \delta < 1$, a (binary) (n, N, k, K, δ) -Projection code is a triple of algorithms $\Pi = (\text{Enc}, \text{Proj}, \text{Dec})$ with the following syntax:

- $\text{Enc} : \{0, 1\}^n \rightarrow \{0, 1\}^N$ is an encoding algorithm mapping a message $\mathbf{x}$ to codeword $\mathbf{X}$.
- $\text{Proj} : [n]^k \rightarrow [N]^K$ is a projection algorithm mapping a sequence of distinct k message coordinates to a sequence of K codeword coordinates.
- $\text{Dec} : \{0, 1\}^K \rightarrow \{0, 1\}^k$ is a decoding algorithm, mapping a sequence of message coordinates and the corresponding codeword projection to a message projection.

We say that Π is an (n, N, k, K, δ) -projection code if for every sequence $\mathbf{s} \in [n]^k$ of k message locations there is a subset $\mathbf{S} \in [N]^K$ of K codeword locations and a subset decoder Dec with the following robustness guarantee. For every $\mathbf{x} \in \{0, 1\}^n$, and every $\tilde{\mathbf{X}}|_{\mathbf{S}}$ which is δ -close to $\mathbf{X}|_{\mathbf{S}}$ in relative Hamming distance, where $\mathbf{X} = \text{Enc}(\mathbf{x})$, we have $\text{Dec}(\tilde{\mathbf{X}}|_{\mathbf{S}}) = \mathbf{x}|_{\mathbf{s}}$.

In our paper, we typically consider a uniform family of projection codes defined by polynomial-time algorithms $(\text{Enc}, \text{Proj}, \text{Dec})$ that apply to any choice of message length n and projection size k . In such a case, we will view N as a function of n and K as a function of k and n .

For ease of exposition, we define the notion of projection codes over $\mathbb{F}_2$ keeping in mind the final application goal, i.e., to obtain succinct argument over a committed database of bits. From a coding theory perspective, we would like to stress that the notion of projection codes holds for any alphabet space. Indeed, our construction (see Fig. 2) is over $\mathbb{F}_q$.

4.1 Constructing Projection Codes

We construct projection codes for the entire parameter regime. Precisely, we first give a construction in which the execution time of Enc is $\text{poly}(n)$ but the execution time of Proj is quasi-linear in k . Second, we give a construction in which the execution time of Enc is quasi-linear in n but the runtime of Proj is quadratic in k . More formally, we prove the below theorem.

Theorem 3 (Projection Codes). *For any $\alpha > 0$, there exists $c_\alpha > 1$ such that there exists an efficient $(n, O(n^{c_\alpha}), k, O(k^{1+\alpha}), \Omega(1))$ projection code. The runtime of **Enc** and **Proj** is $\tilde{O}(n^{c_\alpha})$ and $\tilde{O}(k^{1+\alpha})$ respectively. Alternatively, for any $\alpha > 0$, there exists an efficient $(n, O(n^{1+\alpha}), k, \tilde{O}(k), \Omega(1))$ -projection code. The running time of **Enc** and **Proj** algorithms is $\tilde{O}(n^{1+\alpha})$ and $\tilde{O}(k^2)$ respectively.*

In order to obtain our result, we recall the following lemma on multiplicative linear code that is implicit in [23].

Lemma 1. *Suppose $\mathcal{E}_{\text{Mul}}$ with (codeword space, special decoder) $= (\mathcal{C}, \text{Dec}^*)$ is a $(k, K, d, 2\delta)$ -multiplicative linear encoding scheme over $\mathbb{F}_q$ and $\mathbf{C}_1, \dots, \mathbf{C}_v$ are any v codewords in $\mathcal{C}$. Then for any v -variate polynomial $p : \mathbb{F}_q^v \rightarrow \mathbb{F}_q$ of total degree at most d , we have*

$$p^{(K)}(\mathbf{C}_1, \dots, \mathbf{C}_v) \in \mathcal{C}^{*d}$$

$$\text{Dec}^*(p^{(K)}(\mathbf{C}_1, \dots, \mathbf{C}_v)) = p^{(k)}(\text{Dec}^*(\mathbf{C}_1), \dots, \text{Dec}^*(\mathbf{C}_v)),$$

where $\mathbf{C}_i = (C_{i,1}, \dots, C_{i,K})$ for $i \in [v]$. For $\mathbf{u}_1, \dots, \mathbf{u}_v \in \mathbb{F}_q^\ell$, where $\mathbf{u}_i = (u_{i,1}, \dots, u_{i,\ell})$ for each $i \in [v]$, we have $p^{(\ell)}((u_{1,1}, \dots, u_{1,\ell}), \dots, (u_{v,1}, \dots, u_{v,\ell}))$ defined as $(p(u_{1,1}, \dots, u_{v,1}), \dots, p(u_{1,\ell}, \dots, u_{v,\ell}))$.

Proof. Let M be the number of terms in v -variate polynomial $p(\mathbf{u}_1, \dots, \mathbf{u}_v)$, i.e., $p = \sum_{i=1}^M \beta_i \prod_{j=1}^v \mathbf{u}_j^{d_{i,j}}$, where $\beta_i \in \mathbb{F}_q$ and $d_{i,j} \in \mathbb{N}$ such that $\sum_{j=1}^v d_{i,j} \leq d$, for all $(i, j) \in [M] \times [v]$. Let $(\mathbf{C}_1, \dots, \mathbf{C}_v)$ be the v codewords in $\mathcal{C}$. For $i \in [v]$, let $\text{Dec}^*(\mathbf{C}_i) = \mathbf{m}_i = (m_{i,1}, \dots, m_{i,k})$.

$$\begin{aligned} \text{Dec}^*(p^{(K)}(\mathbf{C}_1, \dots, \mathbf{C}_v)) &= \text{Dec}^* \left(\sum_{i=1}^M \beta_i \prod_{j=1}^v C_{j,1}^{d_{i,j}}, \dots, \sum_{i=1}^M \beta_i \prod_{j=1}^v C_{j,K}^{d_{i,j}} \right) \\ &= \text{Dec}^* \left(\sum_{i=1}^M \beta_i \prod_{j=1}^v \mathbf{C}_j^{d_{i,j}} \right) \end{aligned} \quad (1)$$

$$= \sum_{i=1}^M \beta_i \text{Dec}^* \left(\prod_{j=1}^v \mathbf{C}_j^{d_{i,j}} \right) \quad (2)$$

$$= \sum_{i=1}^M \beta_i \prod_{j=1}^v \text{Dec}^* \left(\mathbf{C}_j^{d_{i,j}} \right) \quad (3)$$

$$= \sum_{i=1}^M \beta_i \prod_{j=1}^v \text{Dec}^* (\mathbf{C}_j)^{d_{i,j}} \quad (4)$$

$$\begin{aligned} &= \left(\sum_{i=1}^M \beta_i \prod_{j=1}^v m_{j,1}^{d_{i,j}}, \dots, \sum_{i=1}^M \beta_i \prod_{j=1}^v m_{j,k}^{d_{i,j}} \right) \\ &= (p(m_{1,1}, \dots, m_{v,1}), \dots, p(m_{1,k}, \dots, m_{v,k})) \\ &= p^{(k)}(\text{Dec}^*(\mathbf{C}_1), \dots, \text{Dec}^*(\mathbf{C}_v)). \end{aligned} \quad (5)$$

(2) follows from (1) as Dec^* is an $\mathbb{F}_q$ -linear map. (3) follows from (2) and (4) follows from (3) from property (3) of multiplicative linear codes. $\square$

Next, we give a projection code construction using systematic Reed-Muller code and multiplicative linear code. Our projection code construction $\mathcal{E}_{\text{Proj}}$ is formally described in Fig. 2. Notice that we denote the length of the global codeword (output of Enc) and projected codeword in our projection codes construction in bits, i.e., N' and K' denote the length in bits.

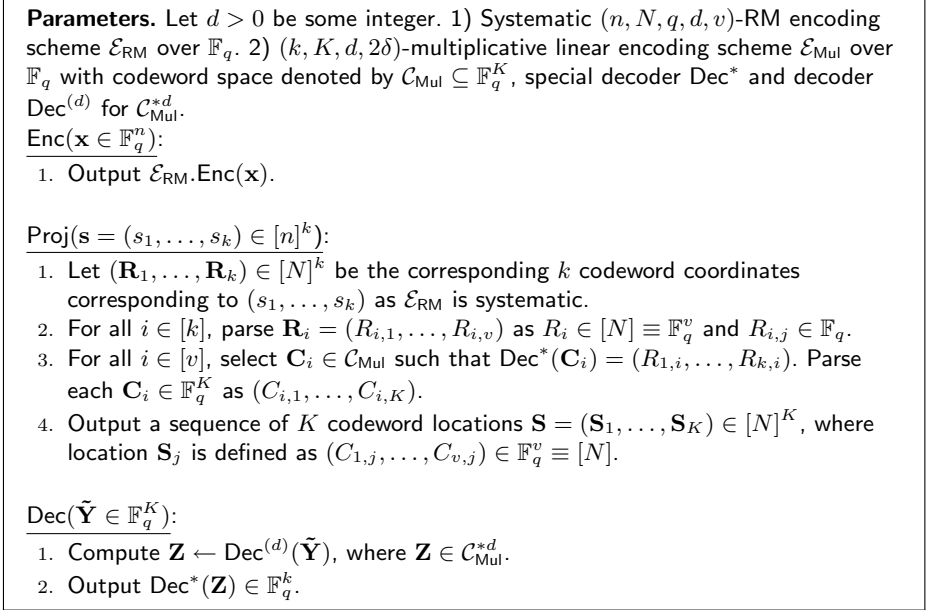


Fig. 2. $(n, N' = N \cdot \lceil \log(q) \rceil, k, K' = K \cdot \lceil \log(q) \rceil, \delta)$ -projection code construction $\mathcal{E}_{\text{Proj}}$

Lemma 2. *If there exists $(k, K, d, 2\delta)$ -multiplicative linear encoding scheme $\mathcal{E}_{\text{Mul}}$ over $\mathbb{F}_q$ then the systematic (n, N, q, d, v) -RM code over $\mathbb{F}_q$ results in $(n, N' = N \cdot \lceil \log(q) \rceil, k, K' = K \cdot \lceil \log(q) \rceil, \delta)$ -projection code $\mathcal{E}_{\text{Proj}}$ (Fig. 2) over $\mathbb{F}_q$.*

Proof. Let $\mathbf{x} \in \mathbb{F}_q^n$ be the message that is encoded by $\mathcal{E}_{\text{Proj}}.\text{Enc}$ into a projection code $\mathbf{X} \in \mathbb{F}_q^{N'}$. Let $p(\cdot)$ be the v -variate polynomial of degree at most d that is associated with the underlying Reed-Muller code. Given a sequence $\mathbf{s}$ of k distinct message coordinates, let $\mathbf{S}$ be the sequence of K codeword coordinates produced by $\mathcal{E}_{\text{Proj}}.\text{Proj}$. Let $\mathbf{Y} \leftarrow \mathbf{X}|_{\mathbf{S}}$ be the projected codeword.

Next, if we can show that the projected codeword $\mathbf{Y}$ belongs to $\mathcal{C}_{\text{Mul}}^{*d}$, then we can rely on the guarantees of the decoder $\text{Dec}^{(d)}(\cdot)$ of the multiplicative codes to recover $\mathbf{Y}$ from any corrupted $\tilde{\mathbf{Y}}$ as long as $\tilde{\mathbf{Y}}$ is obtained by changing at most δ -fraction of symbols in $\mathbf{Y}$. We may then finish the proof by proving that $\text{Dec}^*(\mathbf{Y})$ indeed equals to the projected message $\mathbf{x}|_{\mathbf{S}}$. Details follow.

Let $(\mathbf{C}_1, \dots, \mathbf{C}_v)$ be the v codewords in $\mathcal{C}_{\text{Mul}}$ computed within $\mathcal{E}_{\text{Proj.Proj}}$ and let $(\mathbf{S}_1, \dots, \mathbf{S}_K)$ be the output of $\mathcal{E}_{\text{Proj.Proj}}$.

$$\begin{aligned} \mathbf{Y} &= \mathbf{X}|_{\mathbf{S}} = (p(\mathbf{S}_1), \dots, p(\mathbf{S}_K)) \\ &= p^K(\mathbf{C}_1, \dots, \mathbf{C}_v). \end{aligned} \quad (6)$$

Thus, $\mathbf{Y} \in \mathcal{C}_{\text{Mul}}^{*d}$. As a result, by Lemma 1, we get

$$\begin{aligned} \text{Dec}^*(p^K(\mathbf{C}_1, \dots, \mathbf{C}_v)) &= p^k(\text{Dec}^*(\mathbf{C}_1), \dots, \text{Dec}^*(\mathbf{C}_v)) \\ &= p^k((\mathbf{R}_{1,1}, \dots, \mathbf{R}_{k,1}), \dots, (\mathbf{R}_{1,v}, \dots, \mathbf{R}_{k,v})) \\ &= (p(\mathbf{R}_1), \dots, p(\mathbf{R}_k)) \\ &= \mathbf{x}|_{\mathbf{S}}. \end{aligned} \quad (7)$$

Thus, $\mathcal{E}_{\text{Proj.Dec}}$ correctly decodes the desired k message symbols that appear at locations $s_1, \dots, s_k$. $\square$

Remark. In our construction, we use Reed-Muller encoding as the global encoder, i.e., $\text{Enc}(\cdot)$ is RM-code. Thus, $\text{Enc}(\cdot)$ is systematic, and there exists $Q = \text{polylog}(n)$, $\epsilon = \text{negl}(n)$ such that for $0 < \delta < \frac{1}{2}$, $\text{Enc}(\cdot)$ is (Q, δ, ϵ) -locally testable, (Q, δ, ϵ) -locally decodable, and (Q, δ, ϵ) -locally correctable code (See Sect. 3.2).

4.2 Instantiations

Case 1. Here, we instantiate Proj algorithm using Reed-Muller code as they satisfy the notion of d -multiplicativity over $\mathbb{F}_q$. Let the degree of the underlying polynomial in Reed-Muller encoding of Enc algorithm be denoted by d_E . Similarly, let d_P be the degree of polynomial in Reed-Muller encoding of Proj algorithm. Thus, the degree of polynomial of projected codeword is at most $d_E \cdot d_P$ which has to be less than $\sigma(q-1)$, for constant $\sigma > 0$ [59, Proposition 2.6]. Let v_E and v_P denote the number of variables in underlying polynomial of Reed-Muller encoding in Enc and Proj algorithm respectively. We require that $\binom{v_E + d_E}{v_E} \geq n$ and $\binom{v_P + d_P}{v_P} \geq k$. Observe that, $\binom{v_E + d_E}{v_E} \geq (d_E/v_E)^{v_E}$. Thus, we can set d_E and v_E such that $(d_E/v_E)^{v_E} \geq n$. Let $(d_E/v_E) = \log^{c_E}(n)$, for some constant $c_E > 1$. Let $a = \log(n)$. Thus, we can set $v_E = \log_a(n)/c_E$ and $d_E = \log^{c_E+1}(n)$. By a similar analysis, we obtain $v_P = \log_a(k)/c_P$ and $d_P = \log^{c_P+1}(n)$, for some constant $c_P > 0$. Now, $q \approx \log^{c_E+c_P+2}(n)$. We have $N = q^{v_E} = n^{1+(c_P+2)/c_E}$ and $K = q^{v_P} = k^{1+(c_E+2)/c_P}$. Setting $(c_E+2)/c_P = \alpha$ for some $\alpha > 0$, we obtain $c_\alpha = 1 + (c_P+2)/(\alpha c_P - 2) \approx 1 + 1/\alpha$ such that $N = n^{c_\alpha}$ and $K = k^{1+\alpha}$.

To encode a message $\mathbf{x} \in \mathbb{F}_q^n$ to codeword $\mathbf{X} \in \mathbb{F}_q^N$, the encoding time in Reed-Muller codes is $O(N \cdot \text{polylog} N) = \tilde{O}(N)$ [40, 45]. Plugging the parameters in Lemma 2, for $\alpha > 0$ there exists $c_\alpha > 1$ such that we have a $(n, O(n^{c_\alpha}), k, O(k^{1+\alpha}), \Omega(1))$ projection code with runtime of Enc and Proj being $\tilde{O}(n^{c_\alpha})$ and $\tilde{O}(k^{1+\alpha})$ respectively. This proves the first part of Theorem 3.

Case 2. For this case, we instantiate Proj algorithm using the multiplicative code construction of Cramer et al. [23, Proposition 2.5].

We require $\binom{v+d}{v} \geq n$ to encode n field elements using v -variate polynomials of degree at most d . Set $d = \log^{(1+\frac{1}{0.9\alpha})} n$, applying Stirling's approximation, we see that $v = \frac{0.9\alpha \log n}{\log \log n}$ suffices. Setting $q = 2d$, we get that $N = q^v = \left(2 \log^{(1+\frac{1}{0.9\alpha})} n\right)^{\frac{0.9\alpha \log n}{\log \log n}} = n^{1+0.9\alpha+o(1)}$. Thus $N = O(n^{1+\alpha})$ and $q = \text{poly log } n$ as α is constant > 0 . Now, we recall the construction of multiplicative code from Cramer et al. [23].

Lemma 3. *There is a $(k, K, d, \Omega(1))$ -multiplicative code $(\mathcal{C}, \text{Dec}^*)$ over $\mathbb{F}_q$ with decoder Dec for $\mathcal{C}^{*d}$ with $K = k \cdot q^2$ and $d = q/5$.*

Plugging the above in Lemma 2 yields a $(n, O(n^{1+\alpha}), k, \tilde{O}(k), \Omega(1))$ -projection code with the running time of Enc and Proj algorithms being $\tilde{O}(n^{1+\alpha})$ and $\tilde{O}(k^2)$ respectively. This proves the second part of Theorem 3.

5 Holographic PCPs for RAM Programs

For an NP-language $\mathcal{L}$, a PCP allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $x \in \mathcal{L}$ by having the verifier read just few bits of the proof. However, in certain applications $\mathcal{P}$ may be interested to prove a statement pertaining to only specific bits of the input $\mathbf{x}$ and not the entire input. This is especially the case in execution of RAM programs. For such scenarios, the cost of PCPs is prohibitive as it scales with the input length $|\mathbf{x}|$.

In this section, we are interested to construct *Holographic PCPs for RAM programs* (HPCP-RAM) in which the runtime of the prover scales only with the runtime of the RAM program (i.e., it is independent of the size of the memory $\mathbf{x}$) and the verifier is succinct, i.e., the runtime of verifier is polylog n , where $n = |\mathbf{x}|$. As an intermediate step, we introduce the notion of *holographic PCPs for subsets* (HPCP-SS) and provide its first construction using projection codes in Sect. 5.1. The notion of HPCP-SS is intricate and addresses the primary technical challenge that one faces in construction of HPCP-RAM.

In Sect. 5.2, we formalize the notion of HPCP-RAM and extend our HPCP-SS construction to obtain the first HPCP-RAM construction with the aforementioned computation complexity goal.

5.1 Holographic PCPs for Subsets

In obtaining HPCP-RAM, the first step is to prove statements that are consistent on a subset of bits of input $\mathbf{x}$. Looking ahead, these are the set of bits in $\mathbf{x}$ that are accessed by the RAM program of interest. Now, as we would like to prove statements about subsets of the message, we would like to also include the information about the location of the subset apart from the projection of the message, for greater generality. This is exactly achieved by the following notion.

Definition 8. (*Addressable projection $\mathbf{x}\|_{\mathbf{s}}$*) For any $\mathbf{x} = (x_1, \dots, x_n) \in \Sigma^n$ and any sequence $\mathbf{s} = (s_1, \dots, s_k) \in [n]^k$ of k -distinct locations in $\mathbf{x}$, we define addressable projection $\mathbf{x}\|_{\mathbf{s}}$ as the following sequence of k symbols $((s_1, x_{s_1}), \dots, (s_k, x_{s_k}))$ with each symbol belonging to $([n] \times \Sigma)$.

We consider an NP relation $\mathcal{R}$, where the input is split to two parts $\mathbf{v} \in \{0, 1\}^\ell$ and $\mathbf{x} \in \{0, 1\}^n$ such that $\ell \ll n$. The NP relation $\mathcal{R}$ takes as input⁶ $\mathbf{v}$, addressable projection of $\mathbf{x}$, i.e., $\mathbf{x}\|_{\mathbf{s}}$ and the witness $w \in \{0, 1\}^\mu$, where $|\mathbf{s}| = k$. Then, $\mathcal{R}(\mathbf{v}, \mathbf{x}\|_{\mathbf{s}}, w) \in \text{DTIME}(T(k) + T'(\ell))$, where $T, T' : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$. In this work, we are interested in the setting in which $\ell < k$, typically $\ell \leq \text{polylog} n$ and $\exists d_0$, s.t. $\text{DTIME}(T'(d)) \subseteq \text{DTIME}(T(d))$, for every $d > d_0 \in \mathbb{Z}^+$. Hence, $\mathcal{R}(\mathbf{v}, \mathbf{x}\|_{\mathbf{s}}, w) \in \text{DTIME}(T(k))$.

Definition 9. (*Holographic PCP for subsets*) A holographic PCP for subsets (HPCP-SS) for an NP relation $\mathcal{R} \in \text{DTIME}(T(k))$ with respect to $(Q_1, \delta_1, \epsilon_1)$ -locally-testable code $\mathbf{E}_1$ and $(Q_2, \delta_2, \epsilon_2)$ -locally-testable code $\mathbf{E}_2$ is a pair $(\mathcal{P}, \mathcal{V})$ (where $\mathcal{P}$ is a probabilistic RAM program and $\mathcal{V}$ is a probabilistic TM) with parameters: query complexity of the verifier (Q), offline proof computation complexity (t_{off}), offline proof size (p_{off}), online proof computation complexity (t_{on}), online proof size (p_{on}), runtime of $\mathcal{V}$ (t_v), proximity parameter (δ), and the soundness error of the HPCP system (ϵ), such that $(\mathcal{P}, \mathcal{V})$ satisfies the following requirements:

- **Offline Phase:**
 - **Inputs:** $\mathcal{P}$ receives $\mathbf{x} \in \{0, 1\}^n$.
 - **Database encoding:** $\mathcal{P}$ computes $\mathbf{X} = \mathbf{E}_1(\mathbf{x})$. We require that $|\mathbf{X}| \leq p_{\text{off}}(n)$ and is computed within time $t_{\text{off}}(n)$ by prover's RAM program.
- **Online Phase:**
 - **Inputs:** $\mathcal{P}$ receives $\mathbf{v} \in \{0, 1\}^\ell$, $\mathbf{s} \in [n]^k$, and a witness $w \in \{0, 1\}^\mu$ as input. $\mathcal{V}$ receives $k, n \in \mathbb{N}$, and $\mathbf{v} \in \{0, 1\}^\ell$ as input.
 - **Subset encoding:** $\mathcal{P}$ computes $\mathbf{T} = \mathbf{E}_2(\mathbf{s})$.
 - **Proof generation:** $\mathcal{P}$ computes π (for statement $\mathcal{R}(\mathbf{v}, \mathbf{x}\|_{\mathbf{s}}, w) = \text{accept}$) by running a RAM program. We require that $\mathbf{T}$ and π are of total length $\leq p_{\text{on}}(n, k)$ and are generated within time $t_{\text{on}}(n, k)$ by $\mathcal{P}$'s RAM program.
 - **Verification:** $\mathcal{V}$ makes $Q(n)$ oracle queries to $(\mathbf{X}, \mathbf{T}, \pi)$, and based on the responses outputs either **accept** or **reject**. We require the runtime of $\mathcal{V}$ to be within $t_v(n)$.
- **Completeness:** If $\mathcal{R}(\mathbf{v}, \mathbf{x}\|_{\mathbf{s}}, w) = \text{accept}$, then $\mathcal{V}$ with oracle access to the correct $(\mathbf{X}, \mathbf{T}, \pi)$ always accepts.
- **Soundness⁷:** Suppose $(\mathbf{v}, \mathbf{X}^*, \mathbf{T}^*)$ are such that for all $\mathbf{x}', \mathbf{s}'$ for which there exists w such that $\mathcal{R}(\mathbf{v}, \mathbf{x}'\|_{\mathbf{s}'}, w) = \text{accept}$, either $\mathbf{X}^*$ is δ_1 -far from $\mathbf{E}_1(\mathbf{x}')$ or $\mathbf{T}^*$ is δ_2 -far from $\mathbf{E}_2(\mathbf{s}')$. Then, $\mathcal{V}$ accepts $(\mathbf{v}, \mathbf{X}^*, \mathbf{T}^*)$ with probability at most $\epsilon = \epsilon(n)$. Here, δ is a function of δ_1 and δ_2 .

⁶ Jumping ahead, i) $\mathbf{v}$ denotes the description of the RAM program on input $\mathbf{x}$. ii) $\mathbf{s}$ denotes the sequence of memory locations accessed by the RAM program.

⁷ One can aim to achieve the stronger notion of soundness, i.e., *knowledge soundness* (witness extractability) [10, Definition 12.35] by using PCPs with *knowledge soundness*. As it is not the focus of this work, we consider the simpler notion.

Succinctly, we refer to such system as an $(n, k, Q, t_{\text{off}}, p_{\text{off}}, t_{\text{on}}, p_{\text{on}}, t_v, \delta, \epsilon)$ -HPCP-SS.

Next, we will prove the following theorem.

Theorem 4 (HPCP-SS). *For any $\alpha > 0$, there exists $c_\alpha > 1$ such that there exists an $(n, k, Q = \text{polylog}(n), t_{\text{off}} = \tilde{O}(n^{c_\alpha}), p_{\text{off}} = O(n^{c_\alpha}), t_{\text{on}} = \tilde{O}(k^{1+\alpha}), p_{\text{on}} = O(k^{1+\alpha}), t_v = \text{polylog}(n), \delta = 1/\text{polylog}(n), \epsilon = \text{negl}(n))$ -HPCP-SS.*

Alternatively, for any $\alpha > 0$, there exists an $(n, k, Q = \text{polylog}(n), t_{\text{off}} = \tilde{O}(n^{1+\alpha}), p_{\text{off}} = O(n^{1+\alpha}), t_{\text{on}} = \tilde{O}(k^2), p_{\text{on}} = \tilde{O}(k), t_v = \text{polylog}(n), \delta = 1/\text{polylog}(n), \epsilon = \text{negl}(n))$ -HPCP-SS.

To prove Theorem 4, we recall two results from prior works. The first result is on an efficient reduction of RAM programs to TMs [38]. It states that for every non-deterministic computation M in RAM model that runs in time T , there exists a non-deterministic TM M' with runtime $\tilde{O}(T)$ that simulates M . Their result [38, Theorem 2] is stated formally in Theorem 5. Theorem 5 is reworded and uses different terminology than used in the original result [38, Theorem 2] in order to be consistent with the terminology used in our work.

In Theorem 5, the RAM program M is given access to $\mathbf{x}$ and w . It verifies the NP relation $\mathcal{R}(\mathbf{x}, w)$ that runs in time T . Let $\mathbf{v} \in \{0, 1\}^\ell$ denote the description of the RAM program, i.e., $\ell = |M|$ and $\mathbf{s}$ denote the sequence of locations accessed by M in $\mathbf{x}$. We define an NP-relation $\mathcal{R}'$ that takes as input $\mathbf{v}, \mathbf{x}|_{\mathbf{s}}$ and witness w' (comprising of w) s.t. for a fixed $\mathbf{v}$, $\mathcal{R}'$ is equivalent to $\mathcal{R}$. That is, for given input memory $\mathbf{x}$, witness w and w' , $\mathcal{R}'(\mathbf{v}, \mathbf{x}|_{\mathbf{s}}, w')$ outputs **accept** iff $\mathcal{R}(\mathbf{x}, w)$ outputs **accept**. Notice that, $\mathcal{R}$ is a RAM NP-relation, whereas $\mathcal{R}'$ is computed by TM. This allows us to use PCPPs to prove statements for NP relation $\mathcal{R}'$.

Theorem 5 ([38, Theorem 2]). *There exists a RAM program Conv and turing machine M' such that for every RAM program M (with description $\mathbf{v} \in \{0, 1\}^\ell$) verifying an NP relation $\mathcal{R}$ in time T ,*

- *The converter RAM program Conv with running time $\tilde{O}(T)$ takes as input description of RAM program M , i.e., $\mathbf{v}$, and witness w . It is given random-access to memory $\mathbf{x}$. Conv outputs a witness w' and a sequence $\mathbf{s} \in [n]^k$ that are the sequence of indices of bits in $\mathbf{x}$ accessed by RAM program M , where $k \leq T$.*
- *The Turing Machine M' that runs in time $\tilde{O}(T)$ for NP relation $\mathcal{R}'$. M' takes as input $\mathbf{v}, \mathbf{x}|_{\mathbf{s}}$ and witness w' .*
- *Completeness. If $\exists w$ s.t. $M(\mathbf{x}, w) = \text{accept}$, then $(w', \mathbf{s}) \leftarrow \text{Conv}(\mathbf{v}, \mathbf{x}, w)$ s.t. $M'(\mathbf{v}, \mathbf{x}|_{\mathbf{s}}, w') = \text{accept}$, and*
- *Soundness. If $\nexists w$ s.t. $M(\mathbf{x}, w) = \text{accept}$, then $\nexists (w', \mathbf{s}') \text{ s.t. } M'(\mathbf{v}, \mathbf{x}|_{\mathbf{s}'}, w') = \text{accept}$.*

Next, we recall the follow result on the existence of “quasi-optimal” PCPP proof system for any NP relation $\mathcal{R}$, implicit in [10].⁸

⁸ Theorem 1 in [10] states this for a PCP, but this can be extended to the PCPP case [8]. A similar PCPP with a slightly higher prover complexity appears in [13].

Theorem 6 (Quasilinear PCPPs). *Let $\mathcal{R}(\mathbf{v}, \mathbf{x}, w) \in \text{DTIME}(T(n))$ be an NP relation for a pair language, where $n = |\mathbf{x}|$ and $T : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$. Then there exists a PCPP system (P, V) for $\mathcal{R}$ with respect to an encoding function E , with the following parameters:*

- *Query Complexity:* $Q = \text{polylog}(T(n))$.
- *Prover Complexity:* $t_p = \tilde{O}((T(n)))$.
- *Verifier Complexity:* $t_v = \text{polylog}(T(n))$.
- *Proof Size:* $\ell_p = \tilde{O}((T(n)))$.
- *Proximity parameter:* $\delta = 1/\text{polylog}(T(n))$.
- *Soundness error:* $\epsilon = \text{negl}(n)$.

With the intention to motivate our construction for HPCP-SS, we first discuss simpler black-box approaches and observe why these approaches fail. In the offline phase, $\mathcal{P}$ can compute an encoding of input $\mathbf{x}$ denoted by $\mathbf{X}$. In the online phase, given k -length sequence $\mathbf{s} \in [n]^k$, the task of $\mathcal{P}$ is to prove an NP relation $\mathcal{R}$ on input $\mathbf{x}|_{\mathbf{s}}$. $\mathcal{P}$ computes an encoding of $\mathbf{s}$ which we denote by $\mathbf{T}$. $\mathcal{P}$ can now generate a PCPP with respect to encoded inputs $\mathbf{X}$ and $\mathbf{T}$. Unfortunately, the online runtime of $\mathcal{P}$ in this approach is proportional to the size of input $\mathbf{x}$, i.e., n . An alternative approach for $\mathcal{P}$ is to compute a fresh encoding $\mathbf{Y}$ of bits in $\mathbf{x}$ at coordinates in sequence $\mathbf{s}$, i.e., $\mathbf{Y}$ is an encoding of $\mathbf{x}|_{\mathbf{s}}$. $\mathcal{P}$ can now produce a PCPP with respect to encoded inputs $\mathbf{Y}$ and $\mathbf{T}$. This approach does achieve the desired complexity but does not satisfy *soundness* as $\mathcal{P}$ can encode an input inconsistent with $\mathbf{x}$ in the online phase. To remedy this situation, $\mathcal{P}$ is required to generate an additional proof to prove that $\mathbf{Y}$ is an encoding of an input that is consistent with $\mathbf{x}$, again resulting in $\tilde{O}(n)$ online computation load on $\mathcal{P}$.

Parameters.

1. $(n, N' = N \cdot \lceil \log(q) \rceil), k, K' = K \cdot \lceil \log(q) \rceil, \delta_P)$ -projection code $\Pi = (\text{Enc}, \text{Proj}, \text{Dec})$ such that the runtime of Enc and Proj algorithm is $O(t_{\text{Enc}}(n))$ and $O(t_{\text{Proj}}(n, k))$ respectively. Moreover, let Enc be $(Q_t^{(1)}, \delta_t^{(1)}, \epsilon_t^{(1)})$ -locally testable code and $(Q_c^{(1)}, \delta_c^{(1)}, \epsilon_c^{(1)})$ -locally correctable code.
2. $(Q_t^{(2)}, \delta_t^{(2)}, \epsilon_t^{(2)})$ -locally testable code $E_2 : [n]^k \rightarrow \{0, 1\}^{\Gamma_2}$ with encoding time $O(t_{E_2}(n, k))$.
3. $(Q_t^{(3)}, \delta_t^{(3)}, \epsilon_t^{(3)})$ -locally testable and $(Q_d^{(3)}, \delta_d^{(3)}, \epsilon_d^{(3)})$ -locally decodable code $E_3 : [N]^K \rightarrow \{0, 1\}^{\Gamma_3}$ with encoding time $O(t_{E_3}(N, K))$.
4. i) $(Q_p^{\text{vproj}}, t_p^{\text{vproj}}, t_v^{\text{vproj}}, \ell_p^{\text{vproj}}, \epsilon_p^{\text{vproj}}, \delta^{\text{vproj}})$ -PCPP proof system $(\mathcal{P}_{\text{vproj}}, \mathcal{V}_{\text{vproj}})$ for NP relation $\mathcal{R}$. ii) Description of RAM program M_{vproj} , i.e., $\mathbf{v}_{\text{vproj}}$.
5. $(Q', t'_p, t'_v, \ell'_p, \epsilon', \delta')$ -PCPP proof system $(\mathcal{P}', \mathcal{V}')$ for NP relation $\mathcal{R}$.

Fig. 3. Parameters of our HPCP-SS construction

As it turns out, this is exactly the issue that our projection code can resolve! In the offline phase, $\mathcal{P}$ computes the projection encoding (Enc) of $\mathbf{x}$ denoted by $\mathbf{X}$. In the online phase, $\mathcal{P}$ maps sequence of k message coordinates $\mathbf{s}$ to sequence of K codeword coordinates $\mathbf{S}$ using Proj. Using PCPP, $\mathcal{P}$ proves that the message

coordinates were projected correctly. By the robustness guarantee of projection code, we have that given oracle access to $\mathbf{X}$ and $\mathbf{S}$, one can obtain oracle access to the robust encoding of $\mathbf{x}|_{\mathbf{s}}$ with only polylog overhead. $\mathcal{P}$ can then use this encoding to produce a PCPP for NP relation $\mathcal{R}$. This way we obtain a HPCP-SS construction in which the online runtime of $\mathcal{P}$ depends on the size of $\mathbf{s}$, i.e., k , and is independent of the size of $\mathbf{x}$, i.e., n (ignoring polylog factors).

We formally describe the parameters used in our construction in Fig. 3. All the parameters are self-explanatory except PCPP proof system $(\mathcal{P}_{\text{vproj}}, \mathcal{V}_{\text{vproj}})$ for NP relation $\mathcal{R}_{\text{vproj}}$. The NP relation $\mathcal{R}_{\text{vproj}}$ takes as input $\mathbf{s}$ and $\mathbf{S}$ and outputs **accept** iff all indices in $\mathbf{s}$ are distinct and $\mathbf{S} = \text{Proj}(\mathbf{s})$. The runtime of Proj algorithm is $O(t_{\text{Proj}}(n, k))$. There exists a RAM program M_{vproj} that on input $(\mathbf{s}, \mathbf{S})$ and witness $w = \emptyset$, checks if all indices in $\mathbf{s}$ are distinct and $\mathbf{S} = \text{Proj}(\mathbf{s})$. The former check can be performed using sorting, thus, incurring at most $\tilde{O}(k)$ computation overhead. If the two checks are satisfied, M_{vproj} outputs **accept**, else it outputs **reject**. The computation complexity of M_{vproj} is $\tilde{O}(t_{\text{Proj}}(n, k))$. From Theorem 5, we have that there exists a converter RAM program Conv and a turing machine M' s.t. M' computes NP relation $\mathcal{R}'$ such that given description of M_{vproj} denoted by $\mathbf{v}_{\text{vproj}}$, $\mathcal{R}'$ is equivalent to $\mathcal{R}_{\text{vproj}}$. For input $\mathbf{v}_{\text{vproj}}$, the runtime of M'_{vproj} is $\tilde{O}(t_{\text{Proj}}(n, k))$. Thus, $\mathcal{R}'(\mathbf{v}_{\text{vproj}}, \cdot, \cdot) \in \text{DTIME}(\tilde{O}(t_{\text{Proj}}(n, k)))$. We denote $\mathcal{R}'(\mathbf{v}_{\text{vproj}}, \cdot, \cdot)$ by $\mathcal{R}'_{\text{vproj}}(\cdot, \cdot)$.

Offline Phase.

1. Input: $\mathcal{P}$ receives $\mathbf{x} \in \{0, 1\}^n$.
2. Database Encoding: $\mathcal{P}$ computes $\mathbf{X} = \Pi.\text{Enc}(\mathbf{x})$, i.e., $E_1(\cdot) = \Pi.\text{Enc}(\cdot)$.

Online Phase.

1. Input: $\mathcal{P}$ receives $\mathbf{s} \in [n]^k$, $\mathbf{v} \in \{0, 1\}^\ell$, and a witness $w \in \{0, 1\}^\mu$ as input.
2. Subset Encoding: $\mathcal{P}$ computes $\mathbf{T} = E_2(\mathbf{s})$.
3. Proof Generation: $\mathcal{P}$ performs the following computation to generate proof π .
 - i) Compute $\mathbf{S} \leftarrow \Pi.\text{Proj}(\mathbf{s})$ and set $\pi_{E_3} \leftarrow E_3(\mathbf{S})$.
 - ii) $(w_{\text{vproj}}, \mathbf{s}_{\text{vproj}}) \leftarrow \text{Conv}(\mathbf{v}_{\text{vproj}}, (\mathbf{s}, \mathbf{S}), \emptyset)$.
 - iii) Invoke $\mathcal{P}_{\text{vproj}}$ with input $(\mathbf{v}_{\text{vproj}}, (\mathbf{s}, \mathbf{S}))$, and witness w_{vproj} to obtain proof π_{vproj} .
 - iii) Invoke $\mathcal{P}'$ on input $(\mathbf{v}, \mathbf{x}|_{\mathbf{s}})$ and witness w to generate proof π' .
 - iv) $\mathcal{P}$ sets proof $\pi \leftarrow \pi_{E_3} \circ \pi_{\text{vproj}} \circ \pi'$.

Fig. 4. Prover $\mathcal{P}$ of our HPCP-SS construction

Remark. Observe that we use Theorem 5 only for relation $\mathcal{R}_{\text{vproj}}$ and not for our primary relation $\mathcal{R}$ as $\mathcal{R}_{\text{vproj}}$ is a RAM NP-relation whereas $\mathcal{R}$ is computable by a deterministic TM.

We present the computation of $\mathcal{P}$ and $\mathcal{V}$ in our HPCP-SS construction formally in Fig. 4 and 5 respectively. For the PCPP proof system $(\mathcal{P}_{\text{vproj}}, \mathcal{V}_{\text{vproj}})$ in our construction, instead of obtaining a single encoding for inputs $(\mathbf{s}, \mathbf{S})$, $\mathcal{V}_{\text{vproj}}$ receives separate encodings $\mathbf{T}$ (encoding of $\mathbf{s}$), and π_{E_3} (encoding of $\mathbf{S}$). However, observe that $\mathbf{T} \circ \pi_{E_3}$ is a valid encoding of input $(\mathbf{s}, \mathbf{S})$ but with the proximity

parameter of the overall encoding reduced compared to proximity parameter of individual encodings $\mathbf{T}$ and π_3 . Similarly, the verifier of $(\mathcal{P}', \mathcal{V}')$ PCPP system receives separate encoding for its individual input components. In our construction, we instantiate these individual encodings using schemes with appropriate parameters in order to obtain HPCP-SS scheme with desired parameters. See proof of Lemma 4 for a formal treatment of the aforementioned subtlety.

In step (2.iii), Fig. 5, we utilize the fact that the global codeword is locally-correctable. However, this is not a necessity, as we can generically compile any protocol that assumes an honest database encoding into one that does not by using an LDC (such as a Reed-Muller code) and PCPP for NP. The idea is that if an honest prover is supposed to commit to $\mathbf{X} = \Pi.\text{Enc}(\mathbf{x})$, the prover will commit to an LDC encoding of $\mathbf{X}$ denoted by $\mathbf{Y}$ and give a PCPP that $\mathbf{Y}$ is indeed close to the LDC-encoding of a valid $\mathbf{X}$. Then whenever we need to access $\mathbf{X}$ in the online phase, we emulate this by applying the local decoding algorithm of the LDC to $\mathbf{Y}$. With this approach, we can even use Reed-Solomon code for projection as described in Sect. 2. However, for simplicity and efficiency reasons we exploit the local correctability property of global codeword in our application.

Using Theorem 5 and Theorem 6, we prove Lemma 4 below. We obtain proof of Theorem 4 by using Theorem 3 in Lemma 4. In Lemma 4, let $t_t^{(1)}$ and $t_c^{(1)}$ respectively denote the time taken to check local-testability and local-correctability of code $\mathbf{E}_1$. Similarly, let $t_t^{(2)}$ denote the local-testability of code $\mathbf{E}_2$. Finally, let $t_t^{(3)}$ and $t_d^{(3)}$ denote the time taken for local-testability and local decoding of code $\mathbf{E}_3$ respectively.

Lemma 4. *Given the primitives listed in Fig. 3, for n -bit inputs, and k -sized sequences, there exist efficient encoding functions $\mathbf{E}_1 : \{0,1\}^n \rightarrow \mathbb{F}_q^N$ and $\mathbf{E}_2 : [n]^k \rightarrow \{0,1\}^{I_2}$ for which the following holds. For every NP relation $\mathcal{R} \in \text{DTIME}(T(k))$ there exists HPCP-SS for k -sized projections with respect to $\mathbf{E}_1$ and $\mathbf{E}_2$ satisfying the following parameters:*

1. *Query Complexity:* $Q = Q_t^{(1)} + Q_t^{(2)} + Q_t^{(3)} + Q^{\text{vproj}} + Q'(Q_d^{(3)} + Q_c^{(1)})$.
2. *Offline proof computation complexity:* $t_{\text{off}} = O(t_{\text{Enc}}(n))$.
3. *Offline proof size:* $p_{\text{off}} = N'$.
4. *Online proof computation complexity:* $t_{\text{on}} = \tilde{O}(t_{\text{Proj}}(n, k) + t_{\mathbf{E}_2}(n, k) + t_{\mathbf{E}_3}(N, K) + t_p^{\text{vproj}} + t'_p)$.
5. *Runtime of $\mathcal{V}$:* $t_v = \tilde{O}(t_t^{(1)} + t_t^{(2)} + t_t^{(3)} + t_v^{\text{vproj}} + Q' \cdot (t_d^{(3)} + t_c^{(1)}) + t'_v)$.
6. *Online proof size:* $\Gamma_2 + \Gamma_3 + \ell^{\text{vproj}} + \ell'$.
7. *Soundness error:* $\epsilon \leq \epsilon_t^{(1)} + \epsilon_t^{(2)} + \epsilon_t^{(3)} + Q' \epsilon_d^{(3)} + \epsilon^{\text{vproj}} + \epsilon'$.
8. *Proximity parameter:* δ (see proof for details).

Proof. Completeness. We need to show that for given $\mathbf{v}, \mathbf{x}$, and $\mathbf{s}$, if $\exists w$ s.t. $\mathcal{R}(\mathbf{v}, \mathbf{x} \|_{\mathbf{s}}, w) = \text{accept}$, then $\mathcal{V}$ always outputs **accept**. By completeness of local-testable codes $\Pi.\text{Enc}$, $\mathbf{E}_2$, and $\mathbf{E}_3$, if $\mathbf{X} = \Pi.\text{Enc}(\mathbf{x})$, $\mathbf{T} = \mathbf{E}_2(\mathbf{s})$ and $\pi_{\mathbf{E}_3} = \mathbf{E}_3(\mathbf{S})$, $\mathcal{V}$ doesn't output **reject** at step (3.i) of Fig. 5 with probability 1.

Now, let us show that $\mathcal{V}$ doesn't output **reject** in step (3.ii) of Fig. 5 with probability 1. If M_{vproj} outputs **accept** on input $\mathbf{s}$ and $\mathbf{S}$, then $\mathbf{S} = \Pi.\text{Proj}(\mathbf{s})$ (by definition of M_{vproj}). By the completeness property listed in Theorem 5, we have that $\exists w'_{\text{vproj}}$ such that TM M' on input $\mathbf{v}_{\text{vproj}}$, $\mathbf{s}$ and $\mathbf{S}$ outputs **accept**. From the completeness property of PCPP system $(\mathcal{P}^{\text{vproj}}, \mathcal{V}^{\text{vproj}})$ for NP relation $\mathcal{R}'$ (using TM M'), we have that $\mathcal{V}^{\text{vproj}}$ outputs **accept** with probability 1.

Online Phase.

1. **Input:** $\mathcal{V}$ receives $n, k \in \mathbb{N}$ and $\mathbf{v} \in \{0, 1\}^\ell$ as input.
2. **Verification:** $\mathcal{V}$ has oracle access to $\mathbf{X}$, $\mathbf{T}$, and $\pi = (\pi_{E_3} \circ \pi_{\text{vproj}} \circ \pi')$. $\mathcal{V}$ performs the following computation:
 - i) $\mathcal{V}$ performs at most $Q_t^{(1)}$, $Q_t^{(2)}$, and $Q_t^{(3)}$ -many queries to oracles $\mathbf{X}$, $\mathbf{T}$, and π_{E_3} to check its local-testability. $\mathcal{V}$ outputs **reject** if any of the test fails.
 - ii) $\mathcal{V}$ invokes $\mathcal{V}_{\text{vproj}}$ with input $\mathbf{v}_{\text{vproj}}$. For any oracle query of $\mathcal{V}_{\text{vproj}}$ to encoded inputs $\mathbf{T}/\pi_{E_3}$ or proof oracle π_{vproj} , $\mathcal{V}$ makes the corresponding query request to oracles $\mathbf{T}/\pi_{E_3}/\pi_{\text{vproj}}$ and forwards the response it receives to $\mathcal{V}_{\text{vproj}}$. If $\mathcal{V}_{\text{vproj}}$ returns **reject** then output **reject**.
 - iii) $\mathcal{V}$ invokes $\mathcal{V}'$ with input $\mathbf{v}$. $\mathcal{V}'$ requires oracle access to encoding of input $\mathbf{x}|_{\mathbf{s}}$, and proof π' . For any query of $\mathcal{V}'$ to proof π , $\mathcal{V}$ can simply make the corresponding query to proof π and forward the received response to $\mathcal{V}'$. For queries of $\mathcal{V}'$ to input oracle, if the query is to encoding of $\mathbf{s}$, i.e., $\mathbf{T}$, $\mathcal{V}$ makes the query to oracle $\mathbf{T}$ and sends the received response to $\mathcal{V}'$. Else, if the query is to encoding of input $\mathbf{x}$, i.e., $\mathbf{X}_{\mathbf{S}}$, $\mathcal{V}$ responds to the query as described next. Let $\mathcal{V}'$ request i^{th} bit in $\mathbf{X}_{\mathbf{S}}$. $\mathcal{V}$ determines $j \in [N]$ s.t. $(j-1) \cdot \lceil \log(q) \rceil \leq (i-1) < j \cdot \lceil \log(q) \rceil$. $\mathcal{V}$ makes $Q_d^{(3)}$ queries to locally decode π_{E_3} for index j to obtain j' . $\mathcal{V}$ performs $Q_c^{(1)}$ (locally-correctable) queries on codeword $\mathbf{X}$ to obtain the symbol ψ at index j' . Send bit at index $i' = (i-1) \bmod \lceil \log(q) \rceil + 1$ of ψ as response to $\mathcal{V}'$. Finally, if $\mathcal{V}'$ returns **reject** then output **reject**.
 - iv) $\mathcal{V}$ outputs **accept**.

Fig. 5. Verifier $\mathcal{V}$ of our HPCP-SS construction

Finally, it is easy to see that all the oracle queries of $\mathcal{V}'$ to π_{E_3} and π' are responded correctly. By the robustness property of projection code, we have that $\mathbf{X}|_{\mathbf{S}}$ is a robust encoding of input $\mathbf{x}|_{\mathbf{s}}$. For queries to encoding $\mathbf{X}|_{\mathbf{S}}$, we first determine the index $j \in [K]$ that contains the i^{th} bit (query of $\mathcal{V}'$) in $\mathbf{X}|_{\mathbf{S}}$. By completeness property of locally decodable code E_3 , we have $j' = S_j$. By the completeness property of locally-correctable code $\Pi.\text{Enc}$, we have $\psi = \mathbf{X}_{S_j}$. Thus, the i^{th} bit in $\mathbf{X}|_{\mathbf{S}}$ is indeed the bit i' of $\mathbf{X}_{S_j}$. Completeness of step (3.iii) then follows from completeness property of PCPP system $(\mathcal{P}', \mathcal{V}')$ for NP relation $\mathcal{R}$. Thus, $\mathcal{V}$ outputs **accept** if $\exists w$ s.t. $\mathcal{R}(\mathbf{v}, \mathbf{x}|_{\mathbf{s}}, w) = \text{accept}$ with probability 1.

Prover Complexity. In the offline phase, $\mathcal{P}$ computes $\Pi.\text{Enc}(\mathbf{x})$. Thus, $\ell_{\text{off}} = N'$ and $t_{\text{off}} = t_{\text{Enc}}(n)$. In the online phase, $\mathcal{P}$ in step (2) of Fig. 4 computes $\mathbf{T} = E_2(\mathbf{s})$ which takes $O(t_{E_2}(n, k))$ computation time. In step (3.i), computing

projection of sequence $\mathbf{s}$, i.e., $\mathbf{S}$, takes $O(t_{\text{Proj}}(n, k))$ computation time and computing encoding of $\mathbf{S}$ takes additional $O(t_{\text{E}_3}(N, K))$ time. $\mathcal{P}$ then invokes **Conv** in step (3.ii) that runs in $\tilde{O}(t_{\text{Proj}}(n, k))$ time. The computation cost of Step (3.iii) and (3.iv) is t_p^{vproj} and t'_p respectively. Thus, $t_{\text{on}} = \tilde{O}(t_{\text{Proj}}(n, k) + t_{\text{E}_2}(n, k) + t_{\text{E}_3}(N, K) + t_p^{\text{vproj}} + t'_p)$. Further, in the online phase $\mathcal{P}$ generates oracle access to $\mathbf{T}$, π_{E_3} , π_{vproj} , and π' . Thus, $\ell_{\text{on}} = \Gamma_2 + \Gamma_3 + \ell^{\text{vproj}} + \ell'$.

Verifier Complexity. The computation cost to locally test codewords $\mathbf{X}$, $\mathbf{T}$, and π_{E_3} in step (3.i) of Fig. 5 is $t_t^{(1)} + t_t^{(2)} + t_t^{(3)}$ and the number of queries required to test the same is at most $Q_t^{(1)} + Q_t^{(2)} + Q_t^{(3)}$. In step (3.ii), $\mathcal{V}$ invokes $\mathcal{V}^{\text{vproj}}$ that runs in time t_v^{vproj} and makes at most Q^{vproj} many queries in total to oracles $\mathbf{T}$, π_{E_3} , and π_{vproj} . Finally, $\mathcal{V}$ invokes $\mathcal{V}'$ that runs in t'_v time and makes Q' queries in total to oracles $\mathbf{T}$, $\mathbf{X}_{|\mathbf{S}}$, and π' . However, responding to a single query to encoding of input $\mathbf{x}_{|\mathbf{S}}$ requires $Q_d^{(3)}$ queries to oracle π_3 and $Q_c^{(2)}$ queries to oracle $\mathbf{X}$. Thus, in order to respond to a query of $\mathcal{V}'$, $\mathcal{V}$ makes at most $Q_d^{(3)} + Q_c^{(1)}$ oracle queries in worst-case. Thus, the overall computation and query cost incurred by $\mathcal{V}$ on invoking $\mathcal{V}'$ is $Q' \cdot (t_d^{(3)} + t_c^{(1)}) + t'_v$ and $Q' \cdot (Q_d^{(3)} + Q_c^{(1)})$ respectively. Hence, $t_v = \tilde{O}(t_t^{(1)} + t_t^{(2)} + t_t^{(3)} + t_v^{\text{vproj}} + Q' \cdot (t_d^{(3)} + t_c^{(1)}) + t'_v)$ and $Q = Q_t^{(1)} + Q_t^{(2)} + Q_t^{(3)} + Q^{\text{vproj}} + Q' \cdot (Q_d^{(3)} + Q_c^{(1)})$.

Soundness. In step (3.i) of Fig. 5, $\mathcal{V}$ accepts ill-formed codeword $\mathbf{X}$ or $\mathbf{T}$ or π_{E_3} with probability at most $\epsilon_t^{(1)} + \epsilon_t^{(2)} + \epsilon_t^{(3)}$. For the rest of the computation of $\mathcal{V}$, with probability $(1 - (\epsilon_t^{(1)} + \epsilon_t^{(2)} + \epsilon_t^{(3)}))$, we have $\mathbf{X} = \Pi.\text{Enc}(\mathbf{x}')$, $\mathbf{T} = \text{E}_2(\mathbf{s}')$, and $\pi_{\text{E}_3} = \text{E}_3(\mathbf{S}')$, for some $\mathbf{x}' \in \{0, 1\}^n$, $\mathbf{s}' \in [n]^k$, and $\mathbf{S}' \in [N]^K$.

From Theorem 5, if inputs $\mathbf{s}'$ and $\mathbf{S}'$ to M_{vproj} are such that $\mathbf{s}$ is not a sequence of distinct locations or $\mathbf{S}' \neq \Pi.\text{Proj}(\mathbf{s}')$, then $\#w_{\text{vproj}}$ such that TM M' outputs accept. Thus, $\mathcal{R}'$ on inputs $\mathbf{v}_{\text{vproj}}$, $\mathbf{s}'$ and $\mathbf{S}'$, outputs **reject**. Moreover, $\mathcal{V}^{\text{vproj}}$ receives access to input encodings $(\mathbf{T} \circ \pi_{\text{E}_3})$. $\mathbf{T}$ (resp. π_{E_3}) is a codeword of length Γ_2 (resp. Γ_3) with proximity parameter $\delta_t^{(2)}$ (resp. $\delta_t^{(3)}$). Thus, the proximity parameter of the overall codeword is,

$$\rho_1 = \min \left\{ \frac{\delta_t^{(2)} \cdot \Gamma_2}{\Gamma_2 + \Gamma_3}, \frac{\delta_t^{(3)} \cdot \Gamma_3}{\Gamma_2 + \Gamma_3} \right\}.$$

If $\delta^{\text{vproj}} = \rho_1$, then by soundness property of PCPP system $(\mathcal{P}^{\text{vproj}}, \mathcal{V}^{\text{vproj}})$, we have that $\mathcal{V}^{\text{vproj}}$ outputs accept with probability at most ϵ^{vproj} . Thus, for the rest of the computation, $\mathbf{S}' = \Pi.\text{Proj}(\mathbf{s}')$ with probability $1 - (\epsilon_t^{(1)} + \epsilon_t^{(2)} + \epsilon_t^{(3)} + \epsilon^{\text{vproj}})$.

Finally, $\mathcal{V}'$ of the $(\mathcal{P}', \mathcal{V}')$ receives oracle access to $\mathbf{T}$ and $\mathbf{X}_{|\mathbf{S}'}$. The queries to $\mathbf{T}$ are always responded correctly. Whereas each query to codeword $\mathbf{X}_{|\mathbf{S}'}$, requires local-decoding of codeword π_{E_3} which is incorrect with probability at most $\epsilon_d^{(3)}$. Thus, queries of $\mathcal{V}'$ are responded incorrectly with probability at most $Q' \epsilon_d^{(3)}$. The query for $\mathbf{X}_{j'}$ is answered on performing locally-correctable queries on codeword $\mathbf{X}$ which is incorrect with probability at most $\epsilon_c^{(1)}$. That is, $\mathcal{V}'$ receives response with respect to a codeword which is $\epsilon_c^{(1)}$ -far away from codeword $\mathbf{X}_{|\mathbf{S}'}$. Recall that

projection scheme Π has proximity parameter δ_P and the projected codeword has length $\Gamma_p = K \lceil \log_2(q) \rceil$. Therefore, we require that $\epsilon_c^{(1)} \leq \delta_P$. Thus, the proximity parameter of the overall codeword $(\mathbf{T} \circ \pi_{E_3} \circ \mathbf{X}|_{\mathbf{s}'})$ is,

$$\rho_2 = \min \left\{ \frac{\delta_t^{(2)} \cdot \Gamma_2}{\Gamma_2 + \Gamma_3 + \Gamma_p}, \frac{\delta_d^{(3)} \cdot \Gamma_3}{\Gamma_2 + \Gamma_3 + \Gamma_p}, \frac{\delta_p \cdot \Gamma_p}{\Gamma_2 + \Gamma_3 + \Gamma_p} \right\}.$$

Now, if the inputs $\mathbf{x}'$ and $\mathbf{s}'$ are such that $\nexists w$ s.t. $\mathcal{R}(\mathbf{v}, \mathbf{x}'|_{\mathbf{s}'}, w) = \text{accept}$ and proximity parameter of PCPP system $(\mathcal{P}', \mathcal{V}')$, i.e., $\delta' = \rho_2$, then by soundness property of PCPP system we have that $\mathcal{V}'$ accepts with probability at most ϵ' .

To conclude, our HPCP-SS system $(\mathcal{P}, \mathcal{V})$ has proximity parameter $\delta = \rho_2$ and soundness error (ϵ) is at most $(\epsilon_t^{(1)} + \epsilon_t^{(2)} + \epsilon_t^{(3)} + Q' \epsilon_d^{(3)} + \epsilon^{\text{vproj}} + \epsilon')$. $\square$

Using $(Q_t^{(2)} = \text{polylog}(n), \delta_t^{(2)} = \Omega(1), \epsilon_t^{(2)} = \text{negl}(n))$ -locally testable code E_2 with codeword length $\Gamma_2 = cK \log n$ and $(Q_t^{(3)} = \text{polylog}(n), \delta_t^{(3)} = \Omega(1), \epsilon_t^{(3)} = \text{negl}(n))$ -locally testable code and $(Q_d^{(3)} = \text{polylog}(n), \delta_d^{(3)} = \Omega(1), \epsilon_d^{(3)} = \text{negl}(n))$ -locally decodable code E_3 with codeword length $c'K \log n$, where $c = O(1)$ and $c' = O(1)$, and with result of Theorem 6 and Theorem 3, we obtain proof of Theorem 4.

Remark. We include the length of $\mathbf{s}$, i.e., k as part of the parameters list of HPCP-SS system $(\mathcal{P}, \mathcal{V})$ in order to explicitly state the overall complexity of our protocol in terms of n and k . However, the parameters of the offline phase of our HPCP-SS construction are independent of k . In fact, the offline phase computation can be reused for varying length of $\mathbf{s}$. Thus, our HPCP-SS construction can be used as a building block in scenarios where support for varying sequence length is required such as in execution of RAM programs. Thus, in the rest of our paper we drop parameter k from the parameter list of HPCP-SS system.

5.2 Holographic PCPs for RAM Programs

In this section, we extend the notion of HPCP-SS to obtain Holographic PCPs for RAM programs (HPCP-RAM). We define the notion of HPCP-RAM below. In the online phase, both the parties obtain description of NP relation $\mathcal{R}$ that is given random access to memory $\mathbf{x}$ and takes as input a witness w . $\mathcal{P}$ is interested to prove $\mathcal{V}$ that $\exists w$ s.t. $\mathcal{R}(\mathbf{x}, w) = \text{accept}$. We denote the description of NP relation $\mathcal{R}$ by $P_{\mathcal{R}}$.

Definition 10. (*Holographic PCP for RAM programs*) A holographic PCP for RAM programs (HPCP-RAM) with respect to $(Q_1, \delta_1, \epsilon_1)$ -locally-testable code E_1 is executed between $\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}})$ and $\mathcal{V}$ (where $\mathcal{P}$ is a probabilistic RAM program and $\mathcal{V}$ is a probabilistic TM) with parameters: query complexity of the verifier (Q), offline proof computation complexity (t_{off}), offline proof size (p_{off}), online proof computation complexity (t_{on}), online proof size (p_{on}), runtime of $\mathcal{V}$ (t_v), proximity parameter (δ), and the soundness error of the HPCP system (ϵ), such that $(\mathcal{P}, \mathcal{V})$ satisfies the following requirements:

– **Offline Phase:**

- **Input:** $\mathcal{P}_{\text{off}}$ receives $\mathbf{x} \in \{0, 1\}^n$.
- **Database Encoding:** $\mathcal{P}_{\text{off}}$ computes $\mathbf{X} = E_1(\mathbf{x})$ and outputs state st . We require that $|\mathbf{X}| \leq p_{\text{off}}(n)$ and is computed within time $t_{\text{off}}(n)$ by prover's RAM program.

– **Online Phase:**

- **Inputs:** $\mathcal{P}_{\text{on}}$ receives st , $P_{\mathcal{R}} \in \{0, 1\}^\ell$ and a witness $w \in \{0, 1\}^\mu$ as input. $\mathcal{V}$ receives $n \in \mathbb{N}$, and $P_{\mathcal{R}} \in \{0, 1\}^\ell$ as input.
- **Proof Generation:** $\mathcal{P}_{\text{on}}$ generates a proof π (for statement $\mathcal{R}(\mathbf{x}, w) = \text{accept}$) which $\mathcal{V}$ gets oracle access to. We require that $|\pi| \leq p_{\text{on}}(n, T)$ and is generated within time $t_{\text{on}}(n, T)$, where T is the bound on the runtime of RAM program $\mathcal{R}$.
- **Verification:** $\mathcal{V}$ makes at most $Q(n)$ queries to $(\mathbf{X}, \pi)$, and based on the responses outputs either **accept** or **reject**. We require the runtime of $\mathcal{V}$ to be within $t_v(n)$.

– **Perfect Completeness:** If $\mathcal{R}(\mathbf{x}, w) = \text{accept}$, then $\mathcal{V}$ with oracle access to $(\mathbf{X}, \pi)$ always accepts.

– **ϵ -Soundness:** Suppose $\mathbf{X}^*$ is such that for all $\mathbf{x}'$ for which there exists w such that $\mathcal{R}(\mathbf{x}', w) = \text{accept}$, $\mathbf{X}^*$ is δ_1 -far from $E_1(\mathbf{x}')$. Then, $\mathcal{V}$ accepts $\mathbf{X}^*$ with probability at most $\epsilon = \epsilon(n)$. In this definition, δ denotes the proximity parameter of our HPCP-RAM system that depends on proximity parameter of encoding E_1 , i.e., δ_1 .

Succinctly, we refer to such system as an $(n, Q, t_{\text{off}}, p_{\text{off}}, t_{\text{on}}, p_{\text{on}}, t_v, \delta, \epsilon)$ -HPCP-RAM.

Let M be the RAM program that verifies NP relation $\mathcal{R}$ in time T and let $P_{\mathcal{R}}$ denote the description of M . Our HPCP-RAM construction is a straightforward extension of our HPCP-SS construction. The TM M' verifies NP relation $\mathcal{R}'$ that on first input fixed to $P_{\mathcal{R}}$ is equivalent NP relation $\mathcal{R}$. We describe our HPCP-RAM construction formally in Fig. 6. In Fig. 6, we denote the offline and online computations of $\mathcal{P}'$, the prover of HPCP-SS system, by $\mathcal{P}'_{\text{off}}$ and $\mathcal{P}'_{\text{on}}$ respectively. $\mathcal{P}'_{\text{off}}$ and $\mathcal{P}'_{\text{on}}$ share a state between them. In our construction, Conv and M' denote the converter RAM program and the TM of Theorem 5.

Lemma 5. *If there exists an $(n, Q', t'_{\text{off}}, p'_{\text{off}}, t'_{\text{on}}, p'_{\text{on}}, t'_v, \delta', \epsilon')$ -HPCP-SS system $(\mathcal{P}' = (\mathcal{P}'_{\text{off}}, \mathcal{P}'_{\text{on}}), \mathcal{V}')$ for NP relation $\mathcal{R}'$, then there exists an $(n, Q = Q' + \log n, t_{\text{off}} = t'_{\text{off}}, p_{\text{off}} = p'_{\text{off}}, t_{\text{on}} = t'_{\text{on}} + \tilde{O}(T), p_{\text{on}} = p'_{\text{on}} + \lceil \log n \rceil, t_v = t'_v + \lceil \log n \rceil, \delta = \delta', \epsilon = \epsilon')$ -HPCP-RAM for NP relation $\mathcal{R}$ (RAM program that runs in time at most T).*

Proof of Lemma 5 is deferred to the full version. Using Theorem 4 and 5, for a RAM NP relation $\mathcal{R}$ that runs in time at most T we have the following result.

Theorem 7 (HPCP-RAM). *For any $\alpha > 0$, there exists $c_\alpha > 1$ such that there exists an $(n, Q = \text{polylog}(n), t_{\text{off}} = \tilde{O}(n^{c_\alpha}), p_{\text{off}} = O(n^{c_\alpha}), t_{\text{on}} = \tilde{O}(T^{1+\alpha}), p_{\text{on}} = O(T^{1+\alpha}), t_v = \text{polylog}(n), \delta = 1/\text{polylog}(n), \epsilon = \text{negl}(n))$ -HPCP-RAM.*

Alternatively, for any $\alpha > 0$, there exists an $(n, Q = \text{polylog}(n), t_{\text{off}} = \tilde{O}(n^{1+\alpha}), p_{\text{off}} = O(n^{1+\alpha}), t_{\text{on}} = \tilde{O}(T^2), p_{\text{on}} = \tilde{O}(T), t_v = \text{polylog}(n), \delta = 1/\text{polylog}(n), \epsilon = \text{negl}(n))$ -HPCP-RAM.

Remark. More precisely, the verifier runtime is $O(\text{polylog}n + \ell)$, where $\ell = |P_{\mathcal{R}}|$ denotes the size of the program description. Since a RAM program (similarly to TM) is a uniform computational model that works on all input lengths n , the program length can be viewed as constant in n . Consequently, we omit ℓ relative to n when presenting asymptotic complexities in the rest of the paper.

Parameters.

1. Projection code $\Pi = (\text{Enc}, \text{Proj}, \text{Dec})$, where Enc is $(Q_1, \delta_1, \epsilon_1)$ -locally testable.
2. $(Q_2, \delta_2, \epsilon_2)$ -locally testable code E_2 .
3. $(n, Q', t'_{\text{off}}, p'_{\text{off}}, t'_{\text{on}}, p'_{\text{on}}, t'_v, \delta', \epsilon')$ -HPCP-SS system $(\mathcal{P}' = (\mathcal{P}'_{\text{off}}, \mathcal{P}'_{\text{on}}), \mathcal{V}')$ for NP relation $\mathcal{R}'$.

Instantiation.

1. Instantiate HPCP-SS system $(\mathcal{P}', \mathcal{V}')$ with encodings $(E_1(\cdot) = \Pi.\text{Enc}(\cdot), E_2(\cdot))$.

Offline Phase.

Computation of $\mathcal{P}_{\text{off}}$:

1. Input: $\mathcal{P}_{\text{off}}$ receives $\mathbf{x} \in \{0, 1\}^n$.
2. Database Encoding: $\mathcal{P}_{\text{off}}$ invokes $\mathcal{P}'_{\text{off}}$ with input $\mathbf{x}$. $\mathcal{P}'_{\text{off}}$ returns $\mathbf{X}$. $\mathcal{P}_{\text{off}}$ sets $\text{st} = \mathbf{x} \circ \mathbf{X}$.

Online Phase.

Computation of $\mathcal{P}_{\text{on}}$:

1. Input: $\mathcal{P}_{\text{on}}$ receives st , $P_{\mathcal{R}} \in \{0, 1\}^\ell$, and a witness w as input.
2. Proof Generation:
 - i) $\mathcal{P}_{\text{on}}$ computes $(w', \mathbf{s}) \leftarrow \text{Conv}(P_{\mathcal{R}}, \mathbf{x}, w)$ and sets $k \leftarrow |\mathbf{s}|$.
 - ii) $\mathcal{P}_{\text{on}}$ invokes $\mathcal{P}'_{\text{on}}$ with inputs $\mathbf{s}$, $P_{\mathcal{R}}$, and w' . $\mathcal{P}'_{\text{on}}$ returns encoding $\mathbf{T}$ and proof π' .
 - iii) $\mathcal{P}_{\text{on}}$ sets $\pi \leftarrow k \circ \mathbf{T} \circ \pi'$.

Computation of $\mathcal{V}$:

1. Input: $\mathcal{V}$ receives $n \in \mathbb{N}$, and $P_{\mathcal{R}} \in \{0, 1\}^\ell$ as input.
2. Verification: $\mathcal{V}$ has oracle access to $\mathbf{X}$, $\pi = k \circ \mathbf{T} \circ \pi'$. $\mathcal{V}$ performs the following computation:
 - i) $\mathcal{V}$ makes queries to oracle π to obtain k .
 - ii) $\mathcal{V}$ invokes $\mathcal{V}'$ with inputs $n, k, \ell \in \mathbb{N}$, and $P_{\mathcal{R}}$.
 - iii) For any query of $\mathcal{V}'$ to encoded inputs $\mathbf{X}/\mathbf{T}$ or proof oracle π' , $\mathcal{V}$ makes the corresponding query request to oracles $\mathbf{X}/\mathbf{T}/\pi'$ and forwards the response it receives to $\mathcal{V}'$.
 - iv) $\mathcal{V}$ outputs the output received from $\mathcal{V}'$.

Fig. 6. Our HPCP-RAM construction

6 Succinct Arguments for RAM Programs

In this section, we provide the first black-box construction for succinct arguments for RAM programs (SArg-RAM) over a succinctly committed database. Here, we first formalize the notion SArg-RAM. We use the standard definition of commitment scheme (cf. [43, Definition 5.13]).

Definition 11. (*Succinct Argument for RAM programs (SArg-RAM)*) A succinct argument for RAM programs (SArg-RAM) with respect to commitment scheme $\Pi_C = (\text{Gen}, \text{Commit}, \text{Verify})$ is executed between $\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}})$ and $\mathcal{V}$ (where $\mathcal{P}$ is a probabilistic RAM program and $\mathcal{V}$ is a probabilistic TM) with parameters: offline proof computation complexity (t_{off}), size of commitment in the offline phase (c_{off}), online proof computation complexity (t_{on}), online communication complexity (c_{on}), runtime of $\mathcal{V}$ (t_v), soundness error (γ), such that $(\mathcal{P}, \mathcal{V})$ satisfies the following requirements:

- **Trusted Setup** ($\text{Setup}(1^\lambda)$): $\text{ck} \xleftarrow{\$} \Pi_C.\text{Gen}(1^\lambda)$.
- **Offline Phase:**
 - **Input:** $\mathcal{P}_{\text{off}}$ receives λ , ck , and $\mathbf{x} \in \{0, 1\}^n$.
 - **Offline Commitment:** On receiving input, $\mathcal{P}_{\text{off}}$ outputs a state st and publishes commitment $\text{com} = \Pi_C.\text{Commit}(\text{ck}, \mathbf{x})$. We require $\text{com} \leq c_{\text{off}}(n, \lambda)$ and the runtime of $\mathcal{P}_{\text{off}}$ within $t_{\text{off}}(n, \lambda)$.
- **Online Phase:**
 - **Input:** $\mathcal{P}_{\text{on}}$ receives state st , RAM program $P_{\mathcal{R}} \in \{0, 1\}^\ell$ defining an NP-relation $\mathcal{R}$, and a witness $w \in \{0, 1\}^\mu$ as input. $\mathcal{V}$ receives $n, \lambda \in \mathbb{N}$, $P_{\mathcal{R}}$, ck , and com as input.
 - **Interaction:** $\mathcal{P}_{\text{on}}$ and $\mathcal{V}$ interact and $\mathcal{V}$ decides to **accept** or **reject** based on the interaction, and commitment com . We require the runtime of $\mathcal{P}_{\text{on}}$ and $\mathcal{V}$ to be respectively bounded by $t_{\text{on}}(n, \lambda, T)$ and $t_v(n, \lambda)$, and the communication cost to be within $c_{\text{on}}(n, \lambda)$, where T denotes the bound on the runtime of RAM program $\mathcal{R}$.
- **Completeness:** If $\mathcal{R}(\mathbf{x}, w) = \text{accept}$, then after the honest $\mathcal{P}$ and $\mathcal{V}$ execute the offline phase and then the online phase on inputs $(\mathbf{x}, w)$, $\mathcal{V}$ always accepts.
- **Soundness:** An SArg-RAM system is $\gamma(n, \lambda)$ -sound, if there exists PPT oracle machine EXT, called extractor, such that for every prover $\mathcal{P}^*$ and security parameter λ , equation (8) holds for the two experiments $\text{Real-Exp}_{(\mathcal{P}^*, \mathcal{V})}^{\text{SArg-RAM}}(\lambda)$ and $\text{Ideal}_{(\mathcal{P}^*, \text{EXT})}^{\text{SArg-RAM}}(\lambda)$ defined below.

$\text{Real-Exp}_{(\mathcal{P}^*, \mathcal{V})}^{\text{SArg-RAM}}(\lambda) :$

1. $\text{ck} \xleftarrow{\$} \text{Setup}(1^\lambda)$.
2. $(\text{st}^*, \text{com}^*, P_{\mathcal{R}^*}) \leftarrow \mathcal{P}^*(\text{ck}, \mathbf{x})$.
3. $\mathcal{P}^*(\text{ck}, \text{st}^*, P_{\mathcal{R}^*}, w/\emptyset)$ and $\mathcal{V}(\lambda, \text{ck}, P_{\mathcal{R}^*})$ interact in the online phase.
4. Return 1 if $\mathcal{V}$ accepts, else return 0.

$\text{Ideal}_{(\mathcal{P}^*, \text{EXT})}^{\text{SArg-RAM}}(\lambda) :$

1. $\text{ck} \xleftarrow{\$} \text{Setup}(1^\lambda)$.

2. $(\text{st}^*, \text{com}^*, P_{\mathcal{R}^*}) \leftarrow \mathcal{P}^*(\text{ck}, \mathbf{x})$.
3. $\mathbf{x}' \leftarrow \text{EXT}_{(\mathcal{P}^*, \text{EXT})}^{\mathcal{P}^*(\text{ck}, \text{st}^*, P_{\mathcal{R}^*}, w/\emptyset)}(\text{com}^*, P_{\mathcal{R}^*})$.
4. Return 1 if $(\exists w \text{ s.t. } \mathcal{R}^*(\mathbf{x}', w) = 1 \text{ AND } \text{com}^* = \Pi_C.\text{Commit}(\mathbf{x}')), \text{ else return } 0$.

$$\Pr[\text{Ideal}_{(\mathcal{P}^*, \text{EXT})}^{\text{SArg-RAM}}(\lambda) = 1] \geq \Pr[\text{Real-Exp}_{(\mathcal{P}^*, \mathcal{V})}^{\text{SArg-RAM}}(\lambda) = 1] - \gamma(n, \lambda). \quad (8)$$

- *Succinctness:* An SArg-RAM is succinct if $t_v(n, \lambda)$ and $c_{\text{on}}(n, \lambda)$ are $\text{poly}(\lambda, \log n)$.

Succinctly, we refer to such system as an $(n, t_{\text{off}}, c_{\text{off}}, t_{\text{on}}, c_{\text{on}}, t_v, \gamma)$ -SArg-RAM.

We present our SArg-RAM construction (see Fig. 7) that uses standard techniques [5, 12, 46] on top of our construction of HPCP-RAM. Our construction makes use of well-known construct *Merkle Trees* (MT) [48]. $\text{MT.Gen}(\cdot)$ denotes the algorithm to generate commitment key ck for merkle trees, $\text{MT.ComputeTree}(\text{ck}, \mathbf{x})$ computes merkle tree D_{Tree} on input $\mathbf{x}$, $\text{MT.GetRoot}(D_{\text{Tree}})$ returns root of D_{Tree} , $\text{MT.GetPath}(D_{\text{Tree}}, i)$ returns path in D_{Tree} corresponding to index i , MT.VerifyPath verifies if path for index i is consistent with the root of D_{Tree} . We direct readers to full-version of the paper for a formal description.

Lemma 6. *Given the following two primitives:*

- A family of collision-resistant hash functions $\mathcal{H}$.
- $(n, Q', t'_{\text{off}}, p'_{\text{off}}, t'_{\text{on}}, p'_{\text{on}}, t'_v, \delta', \epsilon')$ -HPCP-RAM system $(\mathcal{P}' = (P'_{\text{off}}, P'_{\text{on}}), \mathcal{V}')$ with respect to $(Q_1, \delta_1, \epsilon_1)$ -locally-testable code E_1 .

There exists an $(n, t_{\text{off}}, c_{\text{off}}, t_{\text{on}}, c_{\text{on}}, t_v)$ -SArg-RAM system $(\mathcal{P} = (P_{\text{off}}, P_{\text{on}}), \mathcal{V})$, satisfying the following parameters:

1. $t_{\text{off}} = t'_{\text{off}} + \tilde{O}(p'_{\text{off}} \log(p'_{\text{off}}))$.
2. $c_{\text{off}} = \lambda$.
3. $t_{\text{on}} = t'_{\text{on}} + \tilde{O}(p'_{\text{on}} \log(p'_{\text{on}}) + (Q_1 + Q') \log(p'_{\text{off}}))$.
4. $c_{\text{on}} = \tilde{O}((Q_1 + Q') \log(p'_{\text{off}}))$.
5. $t_v = t'_v + \tilde{O}((Q_1 + Q') \log(p'_{\text{off}}))$.
6. $\gamma = \text{negl}(\lambda) + \epsilon'$.

Complete proof of Lemma 6 is deferred to the full-version. Using Theorem 7 with the above lemma, we obtain the following result:

Theorem 8 (SArg-RAM). *If there exists a family of collision-resistant hash functions, then*

- For $\alpha > 0$ there exists a $c_\alpha > 1$ such that there exists an $(\tilde{O}(n^{c_\alpha}), \lambda, \tilde{O}(T^{1+\alpha}), \text{poly}(\lambda, \log(n)), \text{poly}(\lambda, \log(n)), \text{negl}(\lambda))$ -SArg-RAM, Or
- For $\alpha > 0$, there exists an $(\tilde{O}(n^{1+\alpha}), \lambda, \tilde{O}(T^2), \text{poly}(\lambda, \log(n)), \text{poly}(\lambda, \log(n)), \text{negl}(\lambda))$ -SArg-RAM.

Parameters.

1. A family of compressible functions $\mathcal{H} = \{h_\alpha : \{0, 1\}^{2\lambda} \rightarrow \{0, 1\}^\lambda\}_{\alpha \in \{0, 1\}^{\text{poly}(\lambda)}}$ to instantiate merkle tree MT.
2. $(n, Q', t'_{\text{off}}, p'_{\text{off}}, t'_{\text{on}}, p'_{\text{on}}, t'_v, \delta', \epsilon')$ -HPCP-RAM system $(\mathcal{P}' = (\mathcal{P}'_{\text{off}}, \mathcal{P}'_{\text{on}}), \mathcal{V}')$ with respect to $(Q_1, \delta_1, \epsilon_1)$ -locally-testable code E_1 .

Setup(1^λ):

1. $\text{ck} \xleftarrow{\$} \text{MT.Gen}(1^\lambda)$.

Offline Phase.Computation of $\mathcal{P}_{\text{off}}$:

1. Input: $\mathcal{P}_{\text{off}}$ receives λ , ck , and $\mathbf{x} \in \{0, 1\}^n$.
2. Offline Commitment: $\mathcal{P}_{\text{off}}$ invokes $\mathcal{P}'_{\text{off}}$ on input $\mathbf{x}$. $\mathcal{P}'_{\text{off}}$ return $\mathbf{X}$. $\mathcal{P}$ computes $D_{\text{Tree}}^{(x)} \leftarrow \text{MT.ComputeTree}(\text{ck}, \mathbf{X})$ and $_{,x} \leftarrow \text{MT.GetRoot}(D_{\text{Tree}}^{(x)})$. $\mathcal{P}_{\text{off}}$ sets $\text{st} = \mathbf{x} \circ \mathbf{X} \circ D_{\text{Tree}}^{(x)}$ and publishes $_{,x}$.

Online Phase.Computation of $\mathcal{P}_{\text{on}}$:

1. Input: $\mathcal{P}_{\text{on}}$ receives state $\text{st} = \mathbf{x} \circ \mathbf{X} \circ D_{\text{Tree}}^{(x)}$, RAM program $P_{\mathcal{R}} \in \{0, 1\}^\ell$ defining an NP-relation $\mathcal{R}$, and a witness $w \in \{0, 1\}^\mu$ as input.
2. Interaction:
 - i) $\mathcal{P}_{\text{on}}$ invokes $\mathcal{P}'_{\text{on}}$ with inputs $\text{st}' = \mathbf{X}$, $P_{\mathcal{R}}$, and w . $\mathcal{P}'_{\text{on}}$ returns proof π' .
 - ii) $\mathcal{P}_{\text{on}}$ computes $D_{\text{Tree}}^{(\pi)} \leftarrow \text{MT.ComputeTree}(\text{ck}, \pi)$ and $_{,\pi} \leftarrow \text{MT.GetRoot}(D_{\text{Tree}}^{(\pi)})$.
 - iii) $\mathcal{P}_{\text{on}}$ sends $_{,\pi}$ to $\mathcal{V}$.
 - iv) On receiving request for i^{th} block of $\mathbf{X}$ (resp. π) from $\mathcal{V}$, $\mathcal{P}_{\text{on}}$ computes and sends $(v_x, \text{path}_x) \leftarrow \text{MT.GetPath}(D_{\text{Tree}}^{(x)}, i)$ (resp. $(v_\pi, \text{path}_\pi) \leftarrow \text{MT.GetPath}(D_{\text{Tree}}^{(\pi)}, i)$) to $\mathcal{V}$.

Computation of $\mathcal{V}$:

1. Input: $\mathcal{V}$ receives $n, \lambda \in \mathbb{N}$, $P_{\mathcal{R}}$, ck , and $_{,x}$ as input.
2. Interaction:
 - i) $\mathcal{V}$ receives $_{,\pi}$ from $\mathcal{P}_{\text{on}}$.
 - ii) $\mathcal{V}$ generates at most Q_1 -many queries $j_1, j_2, \dots, j_r$ ($r \leq Q_1$) to check local-testability of codeword committed to $_{,x}$. For each $\ell \in [r]$, $\mathcal{V}$ computes $i_\ell = \lfloor (j_\ell - 1)/\lambda \rfloor + 1$. $\mathcal{V}$ queries $\mathcal{P}_{\text{on}}$ for block i_ℓ of $\mathbf{X}$. $\mathcal{V}$ receives $(v_{x,\ell}, \text{path}_{x,\ell})$ as response from $\mathcal{P}_{\text{on}}$. $\mathcal{V}$ computes $\text{MT.VerifyPath}(\text{ck}, _{,x}, i, v_{x,\ell}, \text{path}_{x,\ell})$. If MT.VerifyPath outputs reject, $\mathcal{V}$ outputs reject and aborts. Else, $\mathcal{V}$ sets a_ℓ to bit at index $((j_\ell - 1) \bmod \lambda + 1)$ of $v_{x,\ell}$. Given responses $a_1, \dots, a_r$ to queries $j_1, \dots, j_r$, $\mathcal{V}$ outputs reject and aborts, if local-testability fails.
 - iii) $\mathcal{V}$ invokes $\mathcal{V}'$ with inputs n and $P_{\mathcal{R}}$.
 - iv) On receiving query for bit j of $\mathbf{X}$ (resp. π) from $\mathcal{V}'$, $\mathcal{V}$ computes $i = \lfloor (j - 1)/\lambda \rfloor + 1$. $\mathcal{V}$ queries $\mathcal{P}_{\text{on}}$ for block i of $\mathbf{X}$ (resp. π). $\mathcal{V}$ receives (v_x, path_x) (resp. (v_π, path_π)) as response from $\mathcal{P}_{\text{on}}$. $\mathcal{V}$ computes $\text{MT.VerifyPath}(\text{ck}, _{,x}, i, v_x, \text{path}_x)$ (resp. $\text{MT.VerifyPath}(\text{ck}, _{,\pi}, i, v_\pi, \text{path}_\pi)$). If MT.VerifyPath outputs reject, $\mathcal{V}$ outputs reject and aborts. Else, $\mathcal{V}$ sends bit at index $((j - 1) \bmod \lambda + 1)$ of v_x (resp. v_π) to $\mathcal{V}'$.
 - v) $\mathcal{V}$ outputs the output received from $\mathcal{V}'$.

Fig. 7. Our SArg-RAM construction

Acknowledgments. We would like to thank Ashutosh Kumar for his contributions to this research, and the anonymous Crypto reviewers for their helpful comments. Y. Ishai was supported in part by ERC Project NTSC (742754), BSF grant 2018393, ISF grant 2774/20, and a Google Faculty Research Award. R. Ostrovsky was supported in part by DARPA under Cooperative Agreement HR0011-20-2-0025, the Algorand Centers of Excellence programme managed by Algorand Foundation, NSF grants CNS-2001096 and CCF-2220450, US-Israel BSF grant 2015782, Amazon Faculty Award, Cisco Research Award and Sunday Group. Any views, opinions, findings, conclusions or recommendations contained herein are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, the Department of Defense, the Algorand Foundation, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright annotation therein.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)
2. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. *J. ACM (JACM)* **45**(3), 501–555 (1998)
3. Arora, S., Safra, S.: Probabilistic checking of proofs: a new characterization of NP. *J. ACM (JACM)* **45**(1), 70–122 (1998)
4. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in poly-logarithmic time. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 5–8 May 1991, New Orleans, Louisiana, USA. ACM (1991)
5. Barak, B., Goldreich, O.: Universal arguments and their applications. In: Proceedings 17th IEEE Annual Conference on Computational Complexity, pp. 194–203. IEEE (2002)
6. Beaver, D., Feigenbaum, J.: Hiding instances in multioracle queries. In: Choffrut, C., Lengauer, T. (eds.) STACS 1990. LNCS, vol. 415, pp. 37–48. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52282-4_30
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC '88, pp. 1–10. Association for Computing Machinery, New York, NY, USA (1988)
8. Ben-Sasson, E., Chiesa, A.: Personal communication (2023)
9. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In: Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, 9–12 January 2013, pp. 401–414. ACM (2013)
10. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: On the concrete efficiency of probabilistically-checkable proofs. In: STOC. ACM (2013)
11. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6
12. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_2

13. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.: Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.* **36**(4), 889–974 (2006)
14. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Short PCPs verifiable in polylogarithmic time. In: 20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11–15 June 2005, San Jose, CA, USA, pp. 120–134. IEEE Computer Society (2005)
15. Bitansky, N., et al.: The hunting of the SNARK. *J. Cryptol.* **30**(4), 989–1066 (2017)
16. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, pp. 111–120 (2013)
17. Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_18
18. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 336–365. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_12
19. Bootle, J., Chiesa, A., Liu, S.: Zero-knowledge IOPs with linear-time prover and polylogarithmic-time verifier. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13276, pp. 275–304. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07085-3_10
20. Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial commitments with applications to zero-knowledge sets. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 422–439. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_25
21. Chen, M., Chiesa, A., Gur, T., O’Connor, J., Spooner, N.: Proof-carrying data from arithmetized random oracles. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023. LNCS, vol. 14005, pp. 379–404. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30617-4_13
22. Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: Simple, black-box constructions of adaptively secure protocols. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 387–402. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_23
23. Cramer, R., Xing, C., Yuan, C.: Efficient multi-point local decoding of reed-muller codes via interleaved codex. *IEEE Trans. Inf. Theory* **66**(1), 263–272 (2019)
24. Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_23
25. Dinur, I.: The PCP theorem by gap amplification. *J. ACM (JACM)* **54**(3), 12-es (2007)
26. Franzese, N., Katz, J., Lu, S., Ostrovsky, R., Wang, X., Weng, C.: Constant-overhead zero-knowledge for ram programs. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 178–191 (2021)
27. Friedl, K., Sudan, M.: Some improvements to total degree tests. In: ISTCS. IEEE Computer Society (1995)
28. Garcia, A., Stichtenoth, H.: A tower of Artin-Schreier extensions of function fields attaining the Drinfeld-Vladut bound. *Invent. Math.* **121**(1), 211–222 (1995)

29. Garg, S., Steve, L., Ostrovsky, R.: Black-box garbled RAM. In: FOCS. IEEE Computer Society (2015)
30. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
31. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Fortnow, L., Vadhan, S.P. (eds.) Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6–8 June 2011, pp. 99–108. ACM (2011)
32. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. *J. ACM* **43**(3), 431–473 (1996)
33. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: linear-time and post-quantum snarks for R1CS. IACR Cryptology ePrint Archive (2021)
34. Gordon, S.D., et al.: Secure two-party computation in sublinear (amortized) time. In: CCS, pp. 513–524. ACM (2012)
35. Goyal, V.: Constant round non-malleable protocols using one way functions. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, pp. 695–704 (2011)
36. Goyal, V., Ostrovsky, R., Scafuro, A., Visconti, I.: Black-box non-black-box zero knowledge. In: Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, pp. 515–524 (2014)
37. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_19
38. Gurevich, Y., Shelah, S.: Nearly linear time. In: Meyer, A.R., Taitlin, M.A. (eds.) Logic at Botik 1989. LNCS, vol. 363, pp. 108–118. Springer, Heidelberg (1989). https://doi.org/10.1007/3-540-51237-3_10
39. Haitner, I., Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: Black-box constructions of protocols for secure computation. *SIAM J. Comput.* **40**(2), 225–266 (2011)
40. Hamlin, A., Ostrovsky, R., Weiss, M., Wichs, D.: Private anonymous data access. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 244–273. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_9
41. Hu, Z., Mohassel, P., Rosulek, M.: Efficient zero-knowledge proofs of non-algebraic statements with sublinear amortized cost. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 150–169. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_8
42. Ishai, Y., Khurana, D., Sahai, A., Srinivasan, A.: On the round complexity of black-box secure MPC. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 214–243. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_8
43. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, 2nd edn. CRC Press, Boca Raton (2014)
44. Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, pp. 80–86 (2000)
45. Kedlaya, K.S., Umans, C.: Fast modular composition in any characteristic. In: 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, 25–28 October 2008, Philadelphia, PA, USA. IEEE Computer Society (2008)

46. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, pp. 723–732 (1992)
47. Lipton, R.J.: Efficient checking of computations. In: Choffrut, C., Lengauer, T. (eds.) STACS 1990. LNCS, vol. 415, pp. 207–215. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-52282-4_44
48. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
49. Micali, S.: CS proofs (extended abstracts). In: FOCS. IEEE Computer Society (1994)
50. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: FOCS. IEEE Computer Society (2003)
51. Mohassel, P., Rosulek, M., Scafuro, A.: Sublinear zero-knowledge arguments for RAM programs. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 501–531. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_18
52. Ostrovsky, R., Rackoff, C., Smith, A.: Efficient consistency proofs for generalized queries on a committed database. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 1041–1053. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27836-8_87
53. Ostrovsky, R., Rao, V., Scafuro, A., Visconti, I.: Revisiting lower and upper bounds for selective decommitments. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 559–578. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_31
54. Pass, R., Wee, H.: Black-box constructions of two-party protocols from one-way functions. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 403–418. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_24
55. Ron-Zewi, N., Rothblum, R.D.: Proving as fast as computing: succinct arguments with constant prover overhead. In: Leonardi, S., Gupta, A. (eds.) STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, 20–24 June 2022, pp. 1353–1363. ACM (2022)
56. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. (1996)
57. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_1
58. Xie, T., Zhang, Y., Song, D.: Orion: zero knowledge proof with linear prover time, In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022. LNCS, vol. 13510, pp. 299–328. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15985-5_11
59. Yekhanin, S.: Locally decodable codes. Found. Trends® Theor. Comput. Sci. **6**(3), 139–255 (2012)