



Zero-Knowledge RAM: Doubly Efficient and Black-Box

Yuval Ishai^{1,2} , Rafail Ostrovsky³ , and Akash Shah³

¹ Technion - Israel Institute of Technology, Haifa, Israel

² AWS, New York, USA

³ University of California Los Angeles, Los Angeles, USA

akashshah08@g.ucla.edu

Abstract. We consider the problem of verifying the computations of a RAM program on a committed database $x \in \{0, 1\}^n$. A recent work by Ishai, Ostrovsky, and Shah (Crypto 2023) obtained a *doubly efficient* solution, in which the communication and verifier's work are polylogarithmic in n and the prover's work is comparable to the (possibly sublinear) running time of the RAM program. This only makes a *black-box* use of a collision-resistant hash function, or alternatively can be implemented unconditionally and non-interactively in the random oracle model.

In the current work, we extend this prior work by providing an additional *zero-knowledge* guarantee and by supporting *database updates*. This gives the first doubly efficient zero-knowledge implementation of RAM programs that makes a black-box use of symmetric cryptography.

While the extra zero knowledge and updatable database features of our solution are orthogonal in scope, our means for achieving them are technically related: to verify with zero knowledge many computations on the same database, we rely on a database refreshing procedure that we also use to accommodate updates.

1 Introduction

Zero-knowledge (ZK) proof systems [8, 31] allow a prover $\mathcal{P}$ to convince a polynomial-time verifier $\mathcal{V}$ that a given statement is true, without revealing any additional information to the verifier. A bit more formally, consider an NP-language $\mathcal{L}$ defined by relation $\mathcal{R}(x, w)$. A ZK proof system allow $\mathcal{P}$, given (x, w) , to convince $\mathcal{V}$ of the statement: $\exists w$ s.t. $\mathcal{R}(x, w) = \text{accept}$, without leaking anything about the witness w to the verifier. We will focus on *computationally sound* proof systems, or *arguments*, in which soundness only holds against polynomial-time cheating provers.

In this work, we consider ZK arguments for *RAM programs* in the following offline-online setting. In the offline phase, $\mathcal{P}$ commits to a large database $\mathbf{x} \in \{0, 1\}^n$. Then, in the i th invocation of the online phase, it wishes to prove in ZK that $\mathbf{x} \in \mathcal{L}_i$, where the corresponding NP-relation $\mathcal{R}_i$ can be verified by a RAM program that runs in sublinear time $T \ll n$. This problem has been the focus of several prior works [14, 15, 19, 22, 26, 40, 52–54, 66] (see Sect. 1.4 for more related

works). We refer to this as a “zero-knowledge argument for RAM programs on a committed database,” or ZK-RAM for short.

Following the terminology of [28,30], we would like the above proof system to be *doubly efficient* in the following sense. The verifier’s work, including both communication and computation, should ideally be polylogarithmic in n in both the offline phase and each invocation of the online phase. The prover’s work in each invocation of the online phase should ideally be polynomial in T , which can be sublinear in n . We refer to this as *doubly efficient* ZK-RAM.

Standard succinct ZK proof systems from the literature, e.g. Kilian’s interactive construction based on a collision-resistant hash (CRH) [44] or Micali’s non-interactive construction based on a random oracle [51], fall short of being doubly efficient, requiring the prover’s work to be at least linear in n . A general technique for converting standard succinct ZK proofs into doubly efficient ones was presented in [54]. However, this came at the price of making a *non-black-box* use of the underlying cryptographic primitives. Such non-black-box constructions typically have poor concrete efficiency, and furthermore, they cannot yield an unconditional non-interactive solution in random oracle model. These limitations motivated several follow-up works that explored black-box alternatives.

One line of work focused on improving the *concrete efficiency* of ZK-RAM, typically making a black-box use of symmetric cryptography. The first result in this direction was given in [40], which obtained a black-box ZK-RAM construction with $\tilde{O}(T)$ online work and communication. While meeting our prover efficiency goals, this does not meet the polylogarithmic communication requirement. Several follow-up works, starting from [22,53], further improved the asymptotic and concrete efficiency of such protocols, still requiring $\Omega(T)$ communication. Another line of works [20,59,68,69] implies ZK-RAM constructions that make use of specific “public-key” assumptions related to bilinear maps and require both parties in the online protocol to access a structured reference string (SRS) that grows linearly with the size of the database n . See Sect. 1.4 for further discussion of related works.

To summarize, all prior ZK-RAM constructions either make a non-black-box use of symmetric cryptography, are not doubly efficient, or make use of public-key cryptography and a long SRS. This leaves the following question open:

Is there a doubly efficient ZK-RAM that only makes a black-box use of symmetric cryptography?

Very recently, Ishai, Ostrovsky, and Shah [42] constructed a doubly efficient arguments for RAM programs on committed database that meet the above requirements except for not being *zero-knowledge*, thus leaving the question open within the context of ZK-RAM. An additional limitation of the result from [42] is that it only considers the setting in which writes do not persist between executions of different program. However, in many applications (e.g. execution of update queries on SQL databases), it is essential to also allow efficient database updates. The question of constructing doubly efficient and black-box arguments for RAM programs on committed *updatable* databases is also left open.

In this work, we answer both of the above questions in the affirmative. Interestingly, our solution for the zero-knowledge problem builds on our solution for the updatability problem. We detail our contributions below.

1.1 Black-Box Doubly Efficient Updatable Arguments for RAM

We first provide a more detailed, albeit still informal, definition of updatable arguments for RAM (U-Arg-RAM). See Sect. 4 for the formal definition. Here and in the following, we let λ denote a computational security parameter.

Definition 1 (U-Arg-RAM, informal). *A U-Arg-RAM protocol involves efficient prover $\mathcal{P}$ and verifier $\mathcal{V}$. It is divided into an offline phase and an online phase. In the offline phase, $\mathcal{P}$ succinctly commits to a database $\mathbf{x}_0$. Subsequently, in each invocation of the online phase, $\mathcal{P}$ receives the current database $\mathbf{x}$, a RAM program $P = (\mathcal{R}, \mathcal{U})$ ($\mathcal{R}$ and $\mathcal{U}$ represent the NP relation and update algorithm respectively), and a witness w as input. Simultaneously, $\mathcal{V}$ is provided with RAM program P and the succinct commitment to the current database denoted by $\text{com}_{\mathbf{x}}$. $\mathcal{P}$ and $\mathcal{V}$ then engage in an interaction wherein $\mathcal{P}$ proves to $\mathcal{V}$ that $\mathcal{R}(\mathbf{x}, w) = \text{accept}$, and that execution of update algorithm $\mathcal{U}$ updates the database from state $\mathbf{x}$ to $\mathbf{x}'$. $\mathcal{P}$ sends the commitment $\text{com}'_{\mathbf{x}}$ to $\mathcal{V}$, which denotes commitment to the updated database state $\mathbf{x}'$.*

- **Completeness:** *If $\mathcal{R}(\mathbf{x}, w) = \text{accept}$, and the execution of $\mathcal{U}$ indeed updates the database from state $\mathbf{x}$ to $\mathbf{x}'$ such that $\text{com}'_{\mathbf{x}}$ is a commitment on $\mathbf{x}'$, then $\mathcal{P}$ should always be able to convince $\mathcal{V}$.*
- **Soundness:** *If $\nexists w$ s.t. $\mathcal{R}(\mathbf{x}, w) = \text{accept}$ or execution of $\mathcal{U}$ doesn't update the database state to $\mathbf{x}'$ such that $\mathcal{V}$ receives commitment on $\mathbf{x}'$, then any efficient malicious $\mathcal{P}^*$ can only make $\mathcal{V}$ accept with negligible probability.*

We obtain the following feasibility result for U-Arg-RAM.

Theorem 1 (Doubly efficient black-box U-Arg-RAM, informal). *Assume a black-box access to a collision-resistant hash function. For any $\alpha > 0$, there is a U-Arg-RAM with the following features:*

- *In the offline phase, $\mathcal{P}$ commits to an n -bit database $\mathbf{x}$ in time $O(n^{1+\alpha})$ and using $\text{poly}(\lambda, \log n)$ bits of communication;*
- *In each invocation of the online phase, $\mathcal{P}$ proves the validity of (a possibly different) execution of RAM program $P = (\mathcal{R}, \mathcal{U})$ that executes in (possibly sublinear) time $T(n)$. $\mathcal{P}$ runs in time $\tilde{O}(T^2)$ and $\mathcal{V}$ in time $\text{poly}(\lambda, \log n)$.*

1.2 Black-Box Doubly Efficient ZK Arguments for RAM Programs

U-ZK-RAM is a U-Arg-RAM scheme that satisfies zero-knowledge property in addition to the properties listed in Definition 1.

Definition 2 (U-ZK-RAM, informal). A U-Arg-RAM scheme is U-ZK-RAM if it satisfies the following additional property.

- **Zero-Knowledge:** For any efficient malicious verifier $\mathcal{V}^*$, there exists an efficient simulator that, when given the outputs of the RAM program, can simulate the view of $\mathcal{V}^*$ during its interaction with an honest $\mathcal{P}$.

Our main result is the following zero-knowledge variant of Theorem 1.

Theorem 2 (Doubly Efficient Black-box U-ZK-RAM, informal). Assume a black-box access to a collision-resistant hash function. For any $\alpha > 0$, there is a U-ZK-RAM with the following features:

- In the offline phase, $\mathcal{P}$ commits to an n -bit database $\mathbf{x}$ in time $O(n^{1+\alpha})$ and using $\text{poly}(\lambda, \log n)$ bits of communication;
- In each invocation of the online phase, $\mathcal{P}$ proves the validity of (a possibly different) execution of RAM program $P = (\mathcal{R}, \mathcal{U})$ that executes in (possibly sublinear) time $T(n)$. $\mathcal{P}$ runs in time $\tilde{O}(T^2)$ and $\mathcal{V}$ in time $\text{poly}(\lambda, \log n)$.

1.3 Locally Updatable Projection Codes

The work of [42] constructs black-box doubly efficient Arg-RAM, by building on a new coding primitive termed “*projection codes*.” A projection code has an encoding scheme Enc with the following guarantee. Suppose $\mathbf{x}$ is a message of length n , for which Enc generates the codeword $\mathbf{X}$ of length N ; then for any sequence $\mathbf{s} \in [n]^k$ of distinct message coordinates, there exists a sequence $\mathbf{S} \in [N]^K$ of K codeword coordinates 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 $\mathbf{s}$. Projection codes are useful in the context of doubly efficient proof systems because they can enable oracle access to a robust encoding of any sublinear-size subset of a big database.

In this work, to support persistent writes on $\mathbf{x}$, $\mathbf{X} = \text{Enc}(\mathbf{x})$ needs to be updated to an encoding of $\mathbf{x}'$. Here, $\mathbf{x}'$ represents state of database after performing the write operations on $\mathbf{x}$. Using projection codes would require recomputing the entire codeword resulting in the online runtime of $\mathcal{P}$ to scale with n .

This leads us to pose the following question: Can we construct projection codes that allow us to update t bits in the underlying message by modifying the codeword in at most $\mathcal{T}$ locations, where t is sublinear in n and $\mathcal{T}$ scales with t ?

This requirement violates the robustness requirement of the codewords, i.e., if we want an encoding scheme with relative distance δN , then changing one bit of the underlying message should require change in at least $2\delta N$ bits of the codeword. These are indeed two conflicting goals. Chandran et al. [13] encountered a similar challenge when developing a meaningful notion for locally updatable and locally decodable codes (LULDCs). Informally, LULDC enables decoding/updating a single bit within the encoded message by accessing/modifying

only a small number of bits of a potentially corrupted codeword. To address the challenge mentioned above, [13] introduced a slightly relaxed yet meaningful notion of robustness which permits a constant fraction of bits to be corrupted within the codeword, under the constraint that at most a δ fraction (for some constant δ) of the t ‘most-recently changed’ bits can be corrupted, where $1 \leq t \leq N$, where N is the length of the codeword. We observe that the distance metric corresponding to this notion of robustness can also be considered in the context of projection codes with *updatability* feature. Thus, *updatable* projection codes are feasible. In this work, we put forth the notion of “*Locally Updatable Projection Codes* (LUPCs)”. We provide an informal definition below (cf. Definition 10). Below, ‘*dist*’ denotes a distance metric.

Definition 3 (Locally Updatable Projection Codes (LUPCs), informal). LUPC comprises of algorithms (Enc, Update, Proj, Dec_E, Dec_P) described below.

- Enc maps a message $\mathbf{x} \in \{0, 1\}^n$ to a codeword $\mathbf{X} \in \{0, 1\}^N$.
- Given (possibly updated) codeword $\mathbf{X}$ and a sequence of updates $\mathbf{u}$, Update algorithm outputs an updated codeword $\mathbf{X}'$.
- Given sequence of distinct message coordinates $\mathbf{s} \in [n]^k$ and (possibly updated) codeword $\mathbf{X}'$, Proj algorithm outputs a sequence of codeword coordinates $\mathbf{S} \in [N]^K$.
- Given a codeword $\mathbf{X} \in \{0, 1\}^N$, Dec_E outputs message $\mathbf{x} \in \{0, 1\}^n$.
- Given a sequence of message coordinates $\mathbf{s} \in [n]^k$ and a projected codeword $\mathbf{Y} \in \{0, 1\}^K$, Dec_P outputs $\mathbf{y}$, a sequence of k (coordinate, bit) pairs.

Let $\tilde{\mathbf{X}}$ be a codeword such that $\text{dist}(\tilde{\mathbf{X}}, \mathbf{X}) \leq (\delta_E/2)N$ for $0 \leq \delta_E < 1$, where $\mathbf{X}$ is an encoding of $\mathbf{x}$ obtained as output of Enc/Update algorithm. For a sequence of updates $\mathbf{u}$, let $\mathbf{x}'$ denote the updated message obtained on making updates $\mathbf{u}$ on message $\mathbf{x}$. Then, $\text{Dec}_E(\tilde{\mathbf{X}}) = \mathbf{x}$ and $\text{Dec}_E(\text{Update}(\tilde{\mathbf{X}}, \mathbf{u})) = \mathbf{x}'$. Moreover, given $\mathbf{S} \leftarrow \text{Proj}(\tilde{\mathbf{X}}, \mathbf{s})$ and if $\tilde{\mathbf{X}}|_{\mathbf{S}}$ is $(\delta_P/2)$ -close to $\mathbf{X}|_{\mathbf{S}}$ in relative hamming distance, then we have $\text{Dec}_P(\mathbf{s}, \tilde{\mathbf{X}}|_{\mathbf{S}}) = \mathbf{x}|_{\mathbf{s}}$, where $\mathbf{x}|_{\mathbf{s}}$ denotes the sequence of pairs of (coordinate, bit) in $\mathbf{x}$ restricted to indices in $\mathbf{s}$.

Theorem 3 (LUPC, informal). For any $\alpha > 0$, $\exists$ LUPC $\Pi = (\text{Enc}, \text{Update}, \text{Proj}, \text{Dec}_E, \text{Dec}_P)$ such that $N = O(n^{1+\alpha})$ and runtime of $\Pi.\text{Enc}$ is $\tilde{O}(n^{1+\alpha})$. Moreover, for any $t = t(n)$ -length sequence of updates, the runtime of $\Pi.\text{Update}$ is $\tilde{O}(t \cdot n^\alpha)$, and for any sequence of $k = k(n)$ message coordinates, $\mathbf{S}$ is of length $\tilde{O}(k)$ and runtime of $\Pi.\text{Proj}$ is $\tilde{O}(k^2)$.

Why are LULDCs not Enough? While our construction of LUPC is inspired by the construction of LULDC proposed in [13], LULDCs alone do not seem sufficient for constructing LUPC. Consequently, they are not applicable in constructing U-Arg-RAM and U-ZK-RAM. This limitation parallels the reason why locally decodable codes (LDCs) are not adequate for constructing projection codes, as noted by [42]. Specifically, the codeword obtained by concatenating bits of the codeword at indices probed by LDC queries for each individual index in $\mathbf{s}$ does not form a robust encoding of the projected message. Robust encoding is essential to apply a *short* holographic PCP proof.

1.4 Related Work

We would like to emphasize that, prior to our work, even without zero-knowledge and updatability, only the construction of Ishai et al. [42] achieved our goal: namely, being doubly efficient and making black-box use of symmetric primitives. Other lines of work fall short by either making a non-black-box use of cryptography, not meeting the double efficiency requirement, or using public-key cryptography. We discuss each category separately.

Non-black-box Use of Cryptography. Following the early work on succinct zero-knowledge arguments for data structure problems [54], there are several recent works (e.g. [9, 46–48]) that build incrementally verifier computation [63] for general NP languages, which in turn can be used to construct ZK-RAM. However, these approaches make a non-black-box use of the underlying cryptographic primitives, and thus do not meet our requirement.

Linear Prover Runtime. The standard notion of Zero-Knowledge Succinct Non-Interactive Arguments (zkSNARGs) [7, 51] enable the prover to generate short proofs that verify NP computations with significantly lower complexity than classical NP verification. There is a long line of works on optimizing the concrete proof length and prover runtime of zkSNARGs, originating from [25, 33, 34, 58], and a more recent line of works consider more fine-tuned optimizations that apply to verifying executions of RAM programs [45, 64, 70]. However, when applied to general ZK-RAM, the prover’s runtime in these protocols scales at least linearly in the database size, failing to meet our efficiency goals.

ZK-RAM with $\approx T$ Communication. There is a line of recent works that aim to improve the concrete efficiency of ZK-RAM with communication and verifier computation costs that scale with the runtime T of the RAM program [2, 22, 26, 38, 65, 66]. This improves over general-purpose zkSNARGs when T is sublinear in the database size, and offers good concrete efficiency for many practical use cases of ZK-RAM, but does not meet our goal of breaking the $O(T)$ communication and verifier computation barrier.

Lookup Arguments from Bilinear Maps. Lookup arguments [21, 24, 59, 68, 69] allow proving that elements of a committed vector are in a large committed table, doubly efficiently. It can be used to obtain ZK-RAM for read-only memory with $O(T)$ prover time and constant communication and verifier overhead. However, these constructions make use of bilinear maps, thus not meeting our goal of using symmetric cryptography. Moreover, they crucially rely on a trusted setup with a structured reference string (SRS) that grows linearly with the database size and cannot be delegated to the prover (even if one gives up on practical efficiency). Indeed, soundness crucially depends on hiding the secret embedded in the SRS from the prover. Updatable lookup arguments [10, 20, 57] further support updates, but still inherit the limitations discussed above.

2 Technical Overview

Our starting point is the doubly efficient black-box **Arg-RAM** (Definition 1 without updatability) construction proposed in [42]. In this section, we begin by offering a high-level description of their idea. We will then delve into the challenges and potential pitfalls of extending their idea to construct **ZK-RAM**.

2.1 Projection Codes and **Arg-RAM** [42]

In the technical overview, we simplify the setting of **Arg-RAM**. In this setting, the prover $\mathcal{P}$ commits to the database $\mathbf{x}$ in the offline phase, as is done in **Arg-RAM**. The difference lies in the online phase. In each online phase, $\mathcal{P}$ proves an NP-statement on the projected message $\mathbf{x}|_{\mathbf{s}}$. Specifically, for an NP relation $\mathcal{R}$, $\mathcal{P}$ proves that $\exists w$ such that $\mathcal{R}(\mathbf{x}|_{\mathbf{s}}, w) = \text{accept}$ (Contrast this with the online phase of **Arg-RAM**, as described in Definition 1 without updatability). A doubly-efficient solution to this problem can be easily extended to obtain a doubly-efficient **Arg-RAM**. The idea is to view the bits accessed by the RAM program as the subset of bits, i.e., $\mathbf{s}$, on which we would like to prove an input statement and use standard reduction of RAM programs to TMs [35].

The construction template from [42] using *projection codes* (see Sect. 1.3 for an informal description) is as follows: In the offline phase, $\mathcal{P}$ computes an encoding of $\mathbf{x}$, denoted as $\mathbf{X}$, using projection codes. $\mathcal{P}$ then publicly commits to $\mathbf{X}$ using a vector commitment scheme [12, 49, 50]. During the online phase, $\mathcal{P}$ commits to an encoding of the set $\mathbf{s}$, denoted as $\mathbf{T}$, to $\mathcal{V}$ using vector commitments. Additionally, $\mathcal{P}$ obtains a witness w . $\mathcal{P}$ generates a Probabilistically Checkable Proof of Proximity (PCPP) [6]¹ π with respect to the input encodings: $\mathbf{T}$ and $\mathbf{Y}$, where $\mathbf{Y}$ is an encoding of $\mathbf{x}|_{\mathbf{s}}$. This is where [42] makes use of the projection property. For encoding of $\mathbf{x}|_{\mathbf{s}}$, $\mathcal{P}$ computes the projected sequence of codeword coordinates $\mathbf{S} \in [N]^K$ for the input sequence $\mathbf{s}$. $\mathcal{P}$ uses the projected codeword $\mathbf{X}|_{\mathbf{s}}$ as the encoding of the input $\mathbf{x}|_{\mathbf{s}}$ in the generation of the PCPP proof π . Since the projected codeword $\mathbf{X}|_{\mathbf{s}}$ is part of the committed global codeword $\mathbf{X}$, this naturally binds the proof to the committed input. $\mathcal{P}$ commits to the proof π using vector commitment. $\mathcal{V}$ then runs the PCPP verifier, denoted as $\mathcal{V}'$, in its head. For a query for the bit at index i in π or $\mathbf{T}$ (the encoding of input $\mathbf{s}$), $\mathcal{V}$ simply requests the commitment opening of π or $\mathbf{T}$ at index i . For a query for bit at index i of the projected codeword $\mathbf{X}|_{\mathbf{s}}$, $\mathcal{V}$ determines the index j in the global codeword $\mathbf{X}$ that corresponds to index i of the projected codeword and requests the commitment opening of $\mathbf{X}$ at index j . $\mathcal{V}$ verifies the openings and forwards the responses to $\mathcal{V}'$. In this high-level overview, we have omitted some details; namely, $\mathcal{P}$ needs to prove that the projection is performed correctly, and in the context of **Arg-RAM**, how the set $\mathbf{s}$ is proven to be consistent with the set of bits accessed by RAM program in extending this solution for simpler problem to obtain **Arg-RAM**, which are addressed in [42].

¹ PCPPs are proof systems that allow probabilistic verification of statements by querying a few bits of the encoded input and a purported proof.

Challenges in Extending to ZK-RAM: The approach described above does not ensure zero-knowledge, as $\mathcal{V}$ queries the input encoding $\mathbf{X}$ and the PCPP proof π . Queries to the projected codeword $\mathbf{X}|_{\mathbf{s}}$ can leak bits of $\mathbf{x}$, violating *input privacy*. Additionally, the projection reveals the codeword sequence $\mathbf{S}$, compromising *access pattern privacy* by exposing the coordinates of $\mathbf{s}$. Moreover, the input and access pattern can also be inferred from the PCPP proof.

2.2 Addressing Input Privacy

Let us first explore potential approaches to obtain *input privacy*. In this context, we're in a setting typically referred to as the Unprotected Access Pattern (UAP) setting, where revealing the access pattern is acceptable.

Zero-Knowledge PCPPs (ZKPCPPs). We consider replacing PCPPs with ZKPCPPs [43], which are PCPPs with a zero-knowledge guarantee: the view of any (potentially malicious) verifier, who makes at most q^* queries to the input and the proof oracle, can be efficiently simulated by making q^* queries to the input oracle alone. In the offline phase, $\mathcal{P}$ generates $t > q^*$ additive secret-shares of each bit of $\mathbf{x}$, encoding them using projection code to obtain $\mathbf{X}$. In the online phase, $\mathcal{P}$ generates a ZKPCPP proof π for the statement $\exists w$ s.t. $\mathcal{R}(\mathbf{x}|_{\mathbf{s}}, w) = \text{accept}$ using² the projected codeword of the secret-shares of $\mathbf{x}|_{\mathbf{s}}$. Since the verifier makes at most q^* queries, generating $t > q^*$ secret-shares ensures input privacy. This approach fails to guarantee input privacy across multiple online phases, as $\mathcal{V}$ could eventually query enough bits of $\mathbf{X}$ to surpass the threshold t , revealing secret-shares of the database. We mention ZKPCPP only to illustrate the challenge of obtaining input privacy; it is not used in our construction.

MPC-in-the-Head and Proactive Secret Sharing. In the offline phase, $\mathcal{P}$ computes the projection encoding of $\mathbf{x}$ to obtain the codeword $\mathbf{X} \in \{0, 1\}^N$. For each $i \in [N]$, $\mathcal{P}$ computes $(2\tau, \eta)$ -Shamir secret sharing [61] of bit $\mathbf{X}[i]$, where $\eta = O(\lambda)$ denotes the number of shares and $2\tau = c\eta$ denotes the corruption threshold, $0 < c < 1$, $\mathbf{X}[i]$ denotes bit at index i in $\mathbf{X}$. Let the generated shares be denoted as $(\langle \mathbf{X}[i] \rangle_j)_{j \in [\eta]}$ and let $\langle \mathbf{X} \rangle_j = (\langle \mathbf{X}[1] \rangle_j, \dots, \langle \mathbf{X}[N] \rangle_j)$. For each $j \in [\eta]$, $\mathcal{P}$ sends vector commitment $\text{com}_j^{(x)}$ on $\langle \mathbf{X} \rangle_j$ to $\mathcal{V}$.

In the online phase, $\mathcal{P}$ generates a PCPP proof π of length ℓ_p for the NP statement $\exists w$ such that $\mathcal{R}(\mathbf{x}|_{\mathbf{s}}, w) = \text{accept}$ with respect to the projected codeword $\mathbf{X}|_{\mathbf{s}}$. Note that, the PCPP proof π is generated using the projected unshared global codeword. For each $i \in [\ell_p]$, $\mathcal{P}$ computes (τ, η) -Shamir secret sharing of bit $\pi[i]$. Let the generated shares be denoted as $(\langle \pi[i] \rangle_j)_{j \in [\eta]}$ and let $\langle \pi \rangle_j = (\langle \pi[1] \rangle_j, \dots, \langle \pi[\ell_p] \rangle_j)$. For each $j \in [\eta]$, $\mathcal{P}$ sends vector commitment $\text{com}_j^{(\pi)}$ on $\langle \pi \rangle_j$ to $\mathcal{V}$.

The PCPP verifier $\mathcal{V}^{(\pi)}$, on randomness r , generates Q -many queries $\mathcal{Q} = q_1, \dots, q_Q$ non-adaptively in the first phase. (independent of the proof π and

² Proof π also uses the encoding of $\mathbf{s}$, i.e., $\mathbf{T}$. Since it is not crucial for conveying the high-level concern, we omit this detail for brevity in the rest of the discussion.

encoding $\mathbf{X}|_s$. The queries are either to π or $\mathbf{X}|_s$. Let the response to these queries be denoted by $\Psi = b_1, \dots, b_Q$. The second phase of $\mathcal{V}^{(\pi)}$ on input (r, Q, Ψ) outputs **accept** if the statement is true; otherwise it outputs **reject** with overwhelming probability. Let Φ denote the second phase of execution of $\mathcal{V}^{(\pi)}$. Define function $f(r, Q, (\langle b_i \rangle_j)_{i \in [Q], j \in [\eta]}) = \Phi(r, Q, (\text{Reconstruct}(\langle b_i \rangle_1, \dots, \langle b_i \rangle_\eta))_{i \in [Q]}) = \Phi(r, Q, b_1, \dots, b_Q)$. Let Π_f be an MPC protocol that computes f with τ -perfect privacy, τ -perfect robustness [41, Section 2.3].

$\mathcal{P}$ and $\mathcal{V}$ jointly sample the randomness r . $\mathcal{P}$ emulates η parties $Z_1, \dots, Z_\eta$ and runs protocol Π_f with public inputs (r, Q) and private inputs $(\langle b_i \rangle_j)_{i \in [Q]}$ of Z_j . In the execution, let the view of Z_i be denoted by $\mathbf{V}(Z_i)_{\Pi}$. $\mathcal{P}$ commits to the views of the MPC emulation. $\mathcal{V}$ selects a set $\mathcal{J} = \{j_1, \dots, j_\tau\}$ and requests that $\mathcal{P}$ open the commitments of the views of the parties in $\mathcal{J}$. $\mathcal{P}$ provides the openings of these commitments, revealing the views of the parties in the MPC emulation $(\mathbf{V}(Z_j)_{\Pi_f})_{j \in \mathcal{J}}$. For every index i accessed in the PCPP proof/ $\mathbf{X}$, $\mathcal{P}$ opens the vector commitment $\text{com}_j^{(\pi)} / \text{com}_j^{(x)}$ to the value $\langle \pi[i] \rangle_j / \langle \mathbf{X}[i] \rangle_j$ for each $j \in \mathcal{J}$. $\mathcal{V}$ verifies the consistency of the obtained views $(\mathbf{V}(Z_j)_{\Pi_f})_{j \in \mathcal{J}}$. Additionally, it also verifies that each view $\mathbf{V}(Z_j)_{\Pi_f}$ is consistent with vector openings of $\langle \pi \rangle_j / \langle \mathbf{X} \rangle_j$. If all the checks pass $\mathcal{V}$ outputs **accept**.

For a single execution, the construction is input private, as $\mathcal{V}$ learns only τ -shares of any bit in $\mathbf{X}/\pi$. However, in multiple executions, $\mathcal{V}$ will recover bits of $\mathbf{X}$ as in each execution it may obtain shares for different set of index set $\mathcal{J}$. To prevent this leakage, we replace the use of standard secret-sharing with *Proactive secret sharing (PSS)* [56]. PSS supports a *refresh* procedure, where parties use their old shares as input to generate new shares of the same secret. Honest parties then discard the old shares. A phase denotes the duration between two executions of refresh. $(2\tau, \eta)$ -PSS guarantees that in a phase if at most 2τ are corrupted, no information about the underlying secret is revealed to adversary.

$\mathcal{P}$ after emulating $\mathcal{V}^{(\pi)}$ in MPC-in-the-head, emulates the refresh algorithm. $\mathcal{P}$ commits to these concatenated views for every party Z_i . $\mathcal{P}$ and $\mathcal{V}$ then proceed in a similar manner as before. In a phase of PSS share, $\mathcal{V}$ may learn at most 2τ in our approach, thus ensuring input privacy. We rely on the robustness of MPC protocol Π_f , soundness of PCPPs, cut-and-choose argument and robustness of PSS scheme to ensure that the underlying bits of database $\mathbf{x}$ is extractable.

2.3 Access Pattern Privacy via Locally Updatable Projection Codes

As the projection algorithm is deterministic, $\mathbf{S}$ leaks the access pattern. Thus, it is not clear how to obtain ZK-RAM by making a direct black-box use of projection codes even in the read-only setting. To address this, one approach is to transform the RAM computation into an Oblivious RAM computation (ORAM)³, as done in the construction of Zero-Knowledge Arguments for RAM Programs by [40, 53]. However, this approach isn't efficiently composable with projection codes. The problem arises because an oblivious RAM evaluation necessarily involves both

³ ORAM transforms any program into a functionally equivalent one in which memory access pattern is independent of the program's secret inputs.

physical reads and writes, even when the logical RAM operations are read-only. This is precisely where our novel primitive, LUPC, proves to be advantageous.

At a high level, the idea of our U-ZK-RAM construction is to first compute an ORAM encoding $\hat{\mathbf{x}}$ of database $\mathbf{x}$ and then compute $\mathbf{X} \leftarrow \text{LUPC.Enc}(\hat{\mathbf{x}})$. Having obtained the global codeword, parties proceed similar to procedure described under MPC-in-the-head + PSS.

Relaxed Notion of Robustness. The robustness notion mentioned in Sect. 1.3 is called the Prefix Hamming Metric (PHM) [13]. In addition to PHM, we consider another robustness notion, Buffered Hamming Distance (BHD), also introduced in [13]. BHD envisions a codeword $\mathbf{X}$ comprises buckets of varying sizes. The robustness notion defined by BHD permits a constant fraction of bits to be corrupted within each bucket.

Challenges in Defining LUPC. We introduce the notion of LUPC defined in Definition 3 (see Definition 10 for formal details). One of our challenge was to devise a meaningful definition of LUPC - one that is not excessively stringent, preventing any instantiations, and yet not overly lenient, failing to offer adequate soundness guarantees in potential applications.

An observant reader may notice that while the Proj algorithm of projection codes does not require access to the codeword $\mathbf{X}$, the Proj/Update algorithm in the definition of LUPC does. This modification reflects that Proj/Update require adaptive access to $\mathbf{X}$, while the projection codes' Proj algorithm requires non-adaptive access. Furthermore, we treat these adaptive accesses to $\mathbf{X}$ as random accesses, enabling sublinear-time access. The adaptive nature is necessary because if the Update algorithm runs in sublinear time, it is unclear how to determine the location of updated bits in a non-adaptive manner, given that the codeword may be modified with arbitrary updates. Indeed, this requirement will be evident in our LUPC construction.

Our LUPC Construction. Our LUPC construction has a hierarchical structure similar to hierarchical ORAM schemes [55] and the LULDC construction of [13]. The construction comprises approximately $\approx \log n$ buckets $\mathcal{B}_0, \dots, \mathcal{B}_{\theta+1}$ of increasing size. $\mathcal{B}_j$ comprises of 2^j many entries of the form (index, value) pair, for $j \in [\theta]$. The last bucket, i.e., $\mathcal{B}_{\theta+1}$, stores the entire message. Each bucket is encoded with standard projection codes to provision for efficient projection of the encoded message. Updates are written to the top bucket. Buckets may eventually fill in; hence, the buckets are periodically combined and moved to an empty bucket at a lower level. During updates, the modified buckets are re-encoded using projection codes. As each bucket is separately encoded, it is not hard to see that the global codeword satisfies BHD robustness.

The process of projection is quite involved. At a high level, given a sequence of message coordinates $\mathbf{s}$, determine the sequence of codeword coordinates $\mathbf{S}_i$ for $i \in [\theta + 1]$ in each bucket corresponding to $\mathbf{s}$. Let $\mathbf{S} = \mathbf{S}_0 \parallel \mathbf{S}_1 \parallel \dots \parallel \mathbf{S}_{\theta+1}$ and $\mathbf{X} = \mathcal{B}_0 \parallel \mathcal{B}_1 \parallel \dots \parallel \mathcal{B}_{\theta+1}$. Then $\mathbf{X}_{|\mathbf{S}}$ is almost the projected codeword. Note that it is not a codeword, as the codeword is not robust due to being projected from

buckets of exponentially decreasing length. To address this, we use repetition codes for projections from small-sized buckets to ensure that projections from all $\log n$ buckets are of the same length. This results in a projection with $O(1/\log n)$ -robustness, which suffices for the application of PCPPs [4].

We will show that our construction is an LUPC with BHD-robustness. LUPC with BHD-robustness guarantee suffices for our U-Arg-RAM and U-ZK-RAM constructions. While the stronger PHM-robustness is not required in our applications, we also prove that our construction satisfies PHM-robustness as we believe that LUPC with PHM-robustness can have potentially new real-world applications, as it better captures a more real-world scenario.

LU-HPCPs-RAM. Locally Updatable Holographic PCPs for RAM programs (LU-HPCPs-RAM) can be seen in our context as an information-theoretic analogue of U-Arg-RAM, where the commitment is replaced by a robust encoding. This holographic PCP serves as a key building block for constructing U-Arg-RAM and more advanced U-ZK-RAM schemes. Beyond extending the Holographic PCPs for RAM programs [42] to support write operations on the database $\mathbf{x}$, the LU-HPCPs-RAM construction also abstracts the information-theoretic soundness aspects of U-Arg-RAM and U-ZK-RAM, making their soundness proofs modular.

Performance. Projection codes in two-parameter regime were proposed in [42]. In the first regime, the runtime of **Enc** and **Proj** is $\tilde{O}(n^{c_\alpha})$ and $\tilde{O}(k^{1+\alpha})$ respectively, where k is the size of projected message, $\alpha > 0$, $c_\alpha = 1 + 1/\alpha$. Whereas, in the second regime, it is $\tilde{O}(n^{1+\alpha})$ and $\tilde{O}(k^2)$ respectively. The **Update** algorithm in our construction invokes **Enc** algorithm of projection codes.

Moreover, the online phase of LU-HPCPs-RAM (in turn, online algorithm of U-Arg-RAM, U-ZK-RAM) requires executions of both **Proj** and **Update**. If we were to use projection codes with parameters in the second regime, it would result in an LU-HPCPs-RAM scheme with $\tilde{O}(\max(T^c, T^{1+\alpha}))$ online time, where $c = 1 + 1/\alpha$. Thus, the online prover time in our constructions is quadratic in T . It is likely that future improved constructions of projection codes will bypass this barrier and is left as an interesting open question.

Organization of Paper. In Sect. 4, we provide formal definitions of U-Arg-RAM and U-ZK-RAM. We introduce the notion of LUPC and present a construction in Sect. 5. In Sect. 6, we propose the concept of LU-HPCPs-RAM, and offer a construction using LUPC. In Sect. 7, we present our doubly efficient U-ZK-RAM construction. Finally, in the full version, we provide a simpler U-Arg-RAM construction using a Kilian-style [44] approach on LU-HPCPs-RAM.

3 Preliminaries

Notations. For a sequence/vector $\mathbf{x} = (x_1, x_2, \dots, x_\ell)$, $|\mathbf{x}|$ denotes its length; $x[i]$ denotes value at $i \in [\ell]$; $x[i_1, \dots, i_2]$ denotes $(x_{i_1}, x_{i_1+1}, \dots, x_{i_2})$, where $i_1 \leq i_2$; for $a \leq \ell$ -sized sequence $A = (j_1, j_2, \dots, j_a) \in [\ell]^a$ (possibly non-distinct), $\mathbf{x}|_A = (x_{j_1}, x_{j_2}, \dots, x_{j_a})$ and $\mathbf{x}||_A = ((j_1, x_{j_1}), (j_2, x_{j_2}), \dots, (j_a, x_{j_a}))$. For integer $\ell > 0$, $[\ell] = \{1, \dots, \ell\}$ and $[\![\ell]\!] = \{0, \dots, \ell\}$. $\parallel$ denotes concatenation. If not mentioned,

assume that the alphabet size for all the codewords used is q , where $\mathbb{F}_q$ represents the code alphabet space. For $\mathbf{u}$ and $\mathbf{v} \in \mathbb{F}_q^\ell$, $w_H(\mathbf{u}, \mathbf{v})$ represents the Hamming distance between $\mathbf{u}$ and $\mathbf{v}$. $\log(\cdot)$ is used to denote the base-2 logarithm.

3.1 Model of Computation

We use $\text{DTIME}(T(n))$ to denote deterministic TMs that run in time $T(n)$ (where $T: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$) on input of length n . Our definition of RAM model [1], as adopted from [29], models it as an interactive TM communicating with a RAM. Our focus is on the execution of RAM programs M on a RAM memory $\mathbf{x}$ of size n . The RAM memory $\mathbf{x}$ consists of the database as well as any additional working space required by M . As in [32], we consider RAM programs such as “ M ” to be defined by a *next-instruction TM* Π that takes as input 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. M consists of a series of executions of Π , initialized with start state, until an instance of Π outputs the operation **Stop**. Formally, we initialize $\mathbf{st} = \text{start}$ and $b = 0$, then iteratively compute $(\text{op}, \ell, b', \mathbf{st}') \leftarrow \Pi(\mathbf{st}, b)$ until termination, updating $\mathbf{st}$ to $\mathbf{st}'$. The possible operations op are as follows: (1) If $\text{op} = \text{Stop}$, terminate with output b' . (2) If $\text{op} = \text{Read}$, set $b = x_\ell$. (3) If $\text{op} = \text{Write}$, set $x_\ell = b'$ and $b = b'$. We are interested in next-instruction TM Π whose computation time is $\text{polylog}(n)$.

3.2 Encoding Scheme

An encoding scheme over $\mathbb{F}_q$ with message length n , codeword length N , relative distance δ , and relative decoding radius ρ , comprises of **Enc** and **Dec** algorithms. **Enc** maps a message $\mathbf{x} \in \mathbb{F}_q^n$ to codeword $\mathbf{X} \in \mathbb{F}_q^N$ such that for any $\mathbf{x} \neq \mathbf{x}'$, $w_H(\text{Enc}(\mathbf{x}), \text{Enc}(\mathbf{x}')) \geq \delta N$. **Dec** takes as input (possibly corrupted) codeword $\mathbf{X}' \in \mathbb{F}_q^N$ and outputs $\mathbf{x}$ if $\exists \mathbf{x} \in \mathbb{F}_q^n$ such that $w_H(\text{Enc}(\mathbf{x}), \mathbf{X}') \leq \rho N$, where $\rho < \delta/2$; else **Dec** outputs $\perp$.

An encoding scheme is (Q, δ', ϵ) -locally testable, if $\exists$ PPT algorithm $\mathcal{A}$ that makes Q -queries to codeword $\mathbf{X}' \in \mathbb{F}_q^N$ such that if the codeword is more than δ' -far away from a valid codeword, it accepts with probability at most ϵ . An encoding scheme is (Q, δ', ϵ) -locally decodable, if $\exists$ PPT algorithm $\mathcal{A}$ that on input $i \in [n]$ makes Q -queries to codeword $\mathbf{X}' \in \mathbb{F}_q^N$ such that if the codeword is at most δ' -far away from a valid codeword $\mathbf{X} = \text{Enc}(\mathbf{x})$, it outputs x_i with probability at least $1 - \epsilon$. An encoding scheme is (Q, δ', ϵ) -locally correctable, if $\exists$ PPT algorithm $\mathcal{A}$ that on input $i \in [N]$ makes Q -queries to codeword $\mathbf{X}' \in \mathbb{F}_q^N$ such that if the codeword is at most δ' -far away from a valid codeword $\mathbf{X} = \text{Enc}(\mathbf{x})$, it outputs X_i with probability at least $1 - \epsilon$.

There exists Reed-Muller codes with parameters $q = \text{polylog}(n)$, $N = O(n^{1+\alpha})$ for $\alpha > 0$ (cf. [37, 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 [23, Theorem 7], [60, Theorem 9], and (Q, δ, ϵ) -locally decodable/ (Q, δ, ϵ) -locally correctable code [67, Proposition 2.6].

3.3 Projection Codes [42]

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 $\text{Dec}(\tilde{\mathbf{X}}|_{\mathbf{s}}) = \mathbf{x}|_{\mathbf{s}}$. For formal definition refer [42, Definition 7].

Theorem 4 (Projection Codes [42, Theorem 3]). *For any $\alpha > 0$, there exists an efficient $(n, N = O(n^{1+\alpha}), k, K = \tilde{O}(k), \delta_P = \Omega(1))$ -projection code. The running time of Enc and Proj algorithms is $\tilde{O}(n^{1+\alpha})$ and $\tilde{O}(k^2)$ respectively.*

3.4 Probabilistically Checkable Proofs of Proximity (PCPPs)

As in [6], we deal with “pair languages” represented as $(\mathbf{v}, \mathbf{x})$ inputs. $\mathcal{V}$ fully reads explicit input $\mathbf{v} \in \{0, 1\}^\ell$ and is given oracle access to implicit input $\mathbf{x} \in \{0, 1\}^n$ (typically with $\ell \ll n$). A probabilistic proof system $(\mathcal{P}, \mathcal{V})$ for an NP relation $\mathcal{R}((\mathbf{v}, \mathbf{x}), w) \in \text{DTIME}(T(n))$ is $(Q, t_p, t_v, \ell_p, \epsilon, \delta)$ -PCPP system [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)] with respect to encoding function $\mathbf{E}$ if the following holds $\forall(\mathbf{v}, \mathbf{x})$: $\mathcal{P}$ on input $\mathbf{v}$, $\mathbf{x}$ and witness w outputs encoding $\mathbf{X} = \mathbf{E}(\mathbf{x})$, and proof π of length $\ell_p(n)$ in time $t_p(n)$. $\mathcal{V}$ on input $\mathbf{v}$ runs in time $t_v(n)$ and makes $Q(n)$ queries to $\mathbf{X}$ and π combined. If $(\mathbf{v}, \mathbf{X}^*)$ are such that $\forall \mathbf{x}'$ for which $\exists w$ s.t. $\mathcal{R}(\mathbf{v}, \mathbf{x}', w) = \text{accept}$ and $\mathbf{X}^*$ is δ -far from $\mathbf{E}(\mathbf{x}')$, then $\mathcal{V}$ accepts with probability $\leq \epsilon = \epsilon(n)$.

Below, we state the result from [4] regarding the existence of a “quasi-optimal” PCPP proof system for any NP relation $\mathcal{R}$.⁴

Theorem 5 (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 $T : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$. There exists a $(Q = \text{polylog}(T(n)), t_p = \tilde{O}((T(n))), t_v = \text{polylog}(T(n)), \ell_p = \tilde{O}((T(n))), \epsilon = \text{negl}(n), \delta = 1/\text{polylog}(T(n)))$ -PCPP system (P, V) for $\mathcal{R}$ with respect to an encoding function $\mathbf{E}$.*

3.5 Secure Multiparty Computation (MPC)

In MPC [11, 27, 41], we have η parties denoted by $Z_1, \dots, Z_\eta$. We adopt formalization of [41]. Public input is denoted by $v^{(\text{pub})}$. Private input and private randomness of Z_i is denoted by $v_i^{(\text{priv})}$ and r_i respectively. We consider MPC protocols Π that securely realize an η -party functionality f , where f maps inputs $v^{(\text{pub})}, v_1^{(\text{priv})}, \dots, v_\eta^{(\text{priv})}$ to an output in $\{0, 1\}$. Let $V(Z_i)$ denote the view of Z_i . We consider the standard notion of perfect correctness, perfect τ -privacy and perfect τ -robustness.

⁴ Theorem 1 in [4] applies to PCPs, but it can be extended to the PCPP case [3]. A related PCPP with slightly higher prover complexity is discussed in [5].

Theorem 6. [16, Section 5] Let $\tau < \eta/3$. There exists a perfectly correct, τ -private and τ -robust MPC protocol that has a communication cost of $\tilde{O}(\eta|C|)$, where $|C|$ denotes the size of circuit C .

3.6 Commitments and Vector Commitments

A commitment scheme C comprise of PPT algorithms ($\text{Gen}, \text{Commit}, \text{Verify}$). $\text{Gen}(1^\lambda)$ outputs commitment key ck . On input ck and message $\mathbf{v}$, Commit outputs (commitment, hint) pair $(\text{com}, \text{decom})$. On input $\text{ck}, \text{com}, \text{decom}, \mathbf{v}$, Verify outputs **accept** if com is a valid commitment of $\mathbf{v}$; otherwise it outputs **reject**. We use statistically hiding and computationally binding commitment scheme [27] in this work.

Theorem 7 (Statistically Hiding Commitment Schemes [17, 18, 36]). Assuming collision-resistant hash functions, there exists a commitment scheme that is statistically hiding and computationally binding. The size of the commitment key ck is λ , the execution time of Commit and Verify is $O(\ell\lambda)$.

Vector Commitment [12, 50] (VC) enables commitment of a vector of messages in a manner that later permits opening the commitment with respect to a specific index. A VC scheme comprises of the following PPT algorithms:

1. $\text{pp} \xleftarrow{\$} \text{Gen}(1^\lambda, n)$: Given vector size n , Gen outputs public parameters pp .
2. $(\text{com}, \text{st}) \xleftarrow{\$} \text{Commit}(\text{pp}, \mathbf{m} = (m_1, \dots, m_n))$: Given pp , a message vector $\mathbf{m}$, Commit outputs commitment com and committer state st .
3. $\sigma_i \leftarrow \text{Open}(\text{pp}, m, i, \text{st})$: On input pp , message m , index $i \in [n]$ and st , Open outputs proof σ_i for the statement that the message at the i^{th} index of the committed message vector is m .
4. $b \leftarrow \text{Verify}(\text{pp}, \text{com}, i, m, \sigma)$: Given pp , com , $i \in [n]$, message m and proof σ , Verify validates proof σ for the statement that com is a commitment on message vector $\mathbf{m}$ such that $m_i = m$ and outputs **accept/reject**.
5. $(\text{com}', \text{aux}, \zeta) \xleftarrow{\$} \text{Update}(\text{pp}, \text{com}, (i, m_i)_{i \in \mathcal{I}}, (i, m'_i)_{i \in \mathcal{I}}, \text{st})$: The algorithm is executed by a committer when messages at indices $\mathcal{I} \subseteq [n]$ are updated from $(m_i)_{i \in \mathcal{I}}$ to $(m'_i)_{i \in \mathcal{I}}$. In addition to the list of updates, Update algorithm takes as input pp , com , and st . It outputs the updated commitment com' , information aux to obtain updated state, along with update information ζ .
6. $\text{st}' \leftarrow \text{UpdState}(\text{st}, \text{aux})$: It outputs updated committer state st' .
7. $\text{com}', (i, \sigma'_i)_{i \in \mathcal{I}} \leftarrow \text{ProofUpdate}(\text{pp}, \text{com}, (i, \sigma_i, m_i^*)_{i \in \mathcal{I}}, \zeta)$: This algorithm can be executed by any user. Given pp , com , ζ , and a sequence of $(i, \sigma_i, m_i)_{i \in \mathcal{I}}$, where σ_i is proof w.r.t. index i prior to the update. It allows the user to compute updated proofs $(i, \sigma'_i)_{i \in \mathcal{I}}$ and updated commitment com' such that σ'_i is proof for the statement that com' is a commitment on message vector $\mathbf{m}'$ (updated message) such that $m'_i = m_i^*$ (possibly updated).

If VC is statistically hiding, we have that commitments on $\mathbf{m}$ and $\mathbf{m}'$ are statistically indistinguishable even after seeing openings at indices i on which $\mathbf{m}$

and $\mathbf{m}'$ agree. VC is index binding if, for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ can generate a commitment c , two distinct values $m \neq m'$ with respective proofs σ and σ' , such that both verify successfully for the same index i is negligible. We make use Merkle Trees [50]. While Merkle Trees are index binding, we can make them hiding by committing each bit of the message using statistically hiding scheme of Theorem 7, and then committing using Merkle Trees.

3.7 Secret-Sharings

A (τ, η) secret-sharing scheme (SS) comprises of **Share** and **Reconstruct** protocols. Given a secret m , the dealer executes **Share** protocol with parties $Z_1, \dots, Z_n$ such that party Z_i obtains share $\langle m \rangle_i$. SS is τ -private if the distribution of any subset of fewer than τ shares output by **Share** protocol is independent of the secret m . Given any $\rho > \tau$ shares of parties, **Reconstruct** outputs a message m or $\perp$. A secret-sharing scheme is τ -robust if more than $\eta - \tau$ are uncorrupted, then **Reconstruct** on input the shares from all parties always outputs the underlying secret. We use Shamir secret-sharing [61].

A (τ, η) -proactive secret sharing (PSS) [39, 56] scheme comprises of an additional algorithm called **Refresh** algorithm. For $i \in [\eta]$, Z_i participates in **Refresh** algorithm with share s_i . **Refresh** outputs updated share s'_i to P_i . Phase is the duration of time between two consecutive executions of **Refresh** protocol. The duration between **Share** and first **Refresh** is a phase. Similarly, the duration between the last **Refresh** and **Reconstruct** forms a phase. PSS is τ -private if $\leq \tau$ shares obtained in a phase reveals no information about the secret. If atmost τ parties are corrupt in each phase, then PSS scheme is τ -robust, if **Reconstruct** outputs the secret correctly. We use Proactive Secret Sharing based on Shamir secret-sharing [39, 56].

3.8 Oblivious RAM (ORAM)

ORAM [29], is a method for transforming any program into an equivalent one in terms of functionality, while ensuring that the memory access patterns remain oblivious to the program's secret inputs. We follow the formalization of [53]. Consider a RAM program Π executed on memory $\mathbf{x}$ with additional auxiliary input v , where $\mathbf{x}'$ and z represent the updated memory and output resulting from the execution of Π .

An ORAM comprises of two algorithms. $\text{RAMInit}(1^\lambda, \mathbf{x})$ takes memory $\mathbf{x}$ as input, and outputs encoded memory $\hat{\mathbf{x}}$ and state st . $\text{RAMEval}(\Pi, v, \text{st}, \hat{\mathbf{x}})$ takes as input description of Π , v , ORAM state st , and is given random access to $\hat{\mathbf{x}}$. It outputs the updated ORAM state st' , and the output of execution, i.e., z . In the process, it updates the encoded memory which we denote by $\hat{\mathbf{x}}'$. Occasionally, we explicitly specify the randomness employed in the two protocols, RAMInit and RAMEval , represented as ω . In cases where it is not explicitly mentioned, assume that the randomness is uniformly sampled. Below, we informally describe the properties of an ORAM scheme (See [53, Definition 1,2,3] for formal definitions).

An ORAM scheme is *complete* if the output of the real execution agrees with the output of the oblivious execution with overwhelming probability. If the outputs disagree, then the output in the oblivious execution is $\perp$.

An ORAM scheme is *extractable* if $\exists \text{RAMExtract}$, that, given the encoded RAM memory $\hat{x}$ and ORAM state st , outputs the plaintext memory $\mathbf{x}$.

An ORAM scheme is secure if the access pattern AP of RAMEval doesn't leak any information about memory $\mathbf{x}$ or auxiliary input v .

We use the a modified Path-ORAM scheme [62] mentioned in [53]. In this scheme, the execution time of RAMInit and RAMEval is $\tilde{O}(n)$ and $\tilde{O}(T)$, where n is the size of memory $\mathbf{x}$, and T denotes the runtime of the RAM program Π .

4 Definitions

We extend the notion of argument for RAM programs on committed *read-only* database [42, Definition 11], referred to as Arg-RAM to define U-Arg-RAM , denoting argument for RAM programs on committed *updatable* database. We build on the definition of U-Arg-RAM to introduce the concept of argument for RAM programs on committed database with *zero-knowledge* property. Let $\mathcal{R}$ and $\mathcal{U}$ denote an NP relation and an update algorithm respectively. Let P be a RAM program with random access to the database $\mathbf{x} \in \{0, 1\}^n$ and takes $w \in \{0, 1\}^\mu$ as input. Let ℓ denote the length of the description of RAM program P ($\ell \ll n$, typically $\ell \leq \text{polylog}(n)$). P computes $\mathcal{R}(\mathbf{x}, w)$ and then modifies the configuration of the database to $\mathbf{x}' \leftarrow \mathcal{U}(\mathbf{x}, w)$ in time $T \ll n$. We denote $P = (\mathcal{R}, \mathcal{U})$. Subsequently, the next program $P' = (\mathcal{R}', \mathcal{U}')$, corresponding to NP relation $\mathcal{R}'$ and update algorithm $\mathcal{U}'$, operates on this updated database $\mathbf{x}'$ instead of the original configuration $\mathbf{x}$. We formalize this syntax in Definition 4.

Definition 4. (Syntax of Reactive Argument System) A reactive argument system 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 $\mathbf{x}_0 \in \{0, 1\}^n$.
 - **Offline Commitment:** On receiving input, $\mathcal{P}_{\text{off}}$ outputs a state st_0 and $\mathbf{x}_0$; and publishes commitment com_0 on $\mathbf{x}_0$ using commitment key ck . We require $\text{com}_0 \leq c_{\text{off}}(n, \lambda)$ and the runtime of $\mathcal{P}_{\text{off}}$ within $t_{\text{off}}(n, \lambda)$.
- For $i = 1, 2, 3, \dots$:
 - i^{th} -**Invocation of Online Phase:**
 - **Input:** $\mathcal{P}_{\text{on}}$ is given random access to st_{i-1} and $\mathbf{x}_{i-1}$, and receives RAM program $P_i = (\mathcal{R}_i, \mathcal{U}_i) \in \{0, 1\}^\ell$ (defining NP-relation $\mathcal{R}_i$ and update algorithm $\mathcal{U}_i$), and a witness $w_i \in \{0, 1\}^\mu$ as input. $\mathcal{V}$ receives $n, \lambda \in \mathbb{N}$, P_i , ck , com_{i-1} as input.

- *Interaction:* $\mathcal{P}_{\text{on}}$ and $\mathcal{V}$ interact and $\mathcal{V}$ outputs $\text{ver}_i = \text{accept/reject}$ based on the interaction and commitment com_{i-1} . $\mathcal{P}_{\text{on}}$ obtains updated state st_i and updated database $\mathbf{x}_i \in \{0, 1\}^n$; and $\mathcal{V}$ obtains updated commitment com_i . We require the runtime of $\mathcal{P}_{\text{on}}$ and $\mathcal{V}$ to be respectively bounded by $t_{\text{on}}(n, \lambda, T_i)$ and $t_v(n, \lambda)$, and the communication cost to be within $c_{\text{on}}(n, \lambda)$, where T_i denotes the bound on the runtime of P_i .

Definition 5. (Completeness) A reactive argument system $(\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}}), \mathcal{V})$ (Definition 4) is considered complete if the following conditions hold. For every $\mathbf{x}_0 \in \{0, 1\}^n$, $P_i = (\mathcal{R}_i, \mathcal{U}_i)$, and w_i , we have:

- $\text{ck} \xleftarrow{\$} \text{Setup}(1^\lambda)$. $(\text{st}_0, \mathbf{x}_0, \text{com}_0) \leftarrow \mathcal{P}_{\text{off}}(\lambda, \text{ck}, \mathbf{x}_0)$.
- For $i = 1, 2, 3, \dots$:
 - $\mathcal{P}_{\text{on}}(\text{st}_{i-1}, \mathbf{x}_{i-1}, P_i, w_i)$ and $\mathcal{V}(n, \lambda, P_i, \text{ck}, \text{com}_{i-1})$ interact, and $\mathcal{V}$ outputs verdict ver_i . $\mathcal{P}_{\text{on}}$ obtains st_i and $\mathbf{x}_i$, and $\mathcal{V}$ obtains com_i .

Let $\mathbf{x}'_0 \leftarrow \mathbf{x}_0$. $\forall i$, let $\mathbf{x}'_i \leftarrow \mathcal{U}_i(\mathbf{x}'_{i-1}, w_i)$. $\forall i$, if $\mathcal{R}_i(\mathbf{x}'_{i-1}, w_i) = \text{accept}$, then after the honest $\mathcal{P}$ and $\mathcal{V}$ execute the offline phase followed by invocations of online phase, $\forall i$, $\text{ver}_i = \text{accept}$, $\mathbf{x}_i = \mathbf{x}'_i$ and com_i is a commitment on $\mathbf{x}_i$.

Our definition of *soundness* of an argument system is stronger than the *standard* concept of soundness, which ensures the existence of a database $\mathbf{x}$ consistent with the offline commitment and proven statements in the online phase. In contrast, we focus on *database extractability*, akin to *knowledge soundness* [4, Definition 12.35], which implies the existence of a PPT *extractor* EXT . This extractor can, with black-box oracle access to a potentially corrupted prover $\mathcal{P}^*$, extract a database consistent with the commitments and proven statements.

Definition 6. (Soundness) A reactive argument system $(\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}}), \mathcal{V})$ (Definition 4) is $\gamma(n, \lambda)$ -sound, if there exists PPT oracle machine EXT , called extractor, such that for every non-uniform polynomial time prover $\mathcal{P}^*$, security parameter λ , and $n = n(\lambda)$ (for some polynomial $n(\cdot)$), equation (1) holds for the two experiments $\text{Real-Exp}_{(\mathcal{P}^*, \mathcal{V})}^{\text{Sound}}(\lambda, n)$ and $\text{Ideal}_{(\mathcal{P}^*, \text{EXT})}^{\text{Sound}}(\lambda, n)$ defined below. Here, $\mathcal{P}^*$ is restricted to databases (initial or updated) of size n .

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

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

1. $\text{ck} \xleftarrow{\$} \text{Setup}(1^\lambda)$. $(\text{st}_0^*, \mathbf{x}_0^*, \text{com}_0^*, P_1^* = (\mathcal{R}_1^*, \mathcal{U}_1^*)) \leftarrow \mathcal{P}^*(\text{ck})$.
2. Set $\text{continue} \leftarrow \text{true}$ and $i \leftarrow 1$.
3. while continue : //Iterates at most polynomial times $U(\lambda)$
 - (a) $\mathcal{P}^*(\text{st}_{i-1}^*, \mathbf{x}_{i-1}^*, P_i^*, w_i/\perp)$ and $\mathcal{V}(n, \lambda, P_i^*, \text{ck}, \text{com}_{i-1}^*)$ interact in the online phase and $\mathcal{V}$ outputs verdict ver_i . $\mathcal{P}^*$ outputs st_i^* , $\mathbf{x}_i^*$, com_i^* and continue . If $\text{continue} = \text{true}$, $\mathcal{P}^*$ additionally outputs $P_{i+1}^* = (\mathcal{R}_{i+1}^*, \mathcal{U}_{i+1}^*)$.
 - (b) Set $i \leftarrow i + 1$.

4. Set $U \leftarrow i - 1$. Return 1 if $\forall i \in [U]$, $\text{ver}_i = \text{accept}$, else return 0.

$\text{Ideal}_{(\mathcal{P}^*, \text{EXT})}^{\text{Sound}}(\lambda, n) :$

1. $\text{ck} \xleftarrow{\$} \text{Setup}(1^\lambda)$. $(\text{st}_0^*, \mathbf{x}_0^*, \text{com}_0^*, P_1^* = (\mathcal{R}_1^*, \mathcal{U}_1^*)) \leftarrow \mathcal{P}^*(\text{ck})$.
2. Set $\text{continue} \leftarrow \text{true}$ and $i \leftarrow 1$.
3. while continue :
 - (a) $\mathcal{P}^*(\text{st}_{i-1}^*, \mathbf{x}_{i-1}^*, P_i^*, w_i/\perp)$ and $\mathcal{V}(n, \lambda, P_i^*, \text{ck}, \text{com}_{i-1}^*)$ interact in the online phase. $\mathcal{P}^*$ outputs st_i^* , $\mathbf{x}_i^*$, com_i^* and continue . If $\text{continue} = \text{true}$, $\mathcal{P}^*$ additionally outputs $P_{i+1}^* = (\mathcal{R}_{i+1}^*, \mathcal{U}_{i+1}^*)$.
 - (b) Set $i \leftarrow i + 1$.
4. Set $U \leftarrow i - 1$.
5. Initially, EXT is given ck , com_0^* , and P_1^* as input. For $i \in [U]$,
 - EXT additionally gets oracle access to $\mathcal{P}^*(\text{st}_{i-1}^*, \mathbf{x}_{i-1}^*, P_i^*, w_i/\perp)$, and is given input com_i^* .
 - if $i = 1$, EXT outputs $(\mathbf{x}'_0, \mathbf{x}'_1)$. Else EXT outputs $\mathbf{x}'_i$.
 - EXT is given $P_{i+1}^* = (\mathcal{R}_{i+1}^*, \mathcal{U}_{i+1}^*)$ as input.
6. Return 1 if $\forall i \in [U]$, $\exists w_i$ s.t. $\mathcal{R}_i^*(\mathbf{x}'_{i-1}, w_i) = \text{accept}$, $\mathbf{x}'_i = \mathcal{U}_i^*(\mathbf{x}'_{i-1}, w_i)$, com_{i-1}^* is a commitment of $\mathbf{x}'_{i-1}$, and com_U^* is a commitment of $\mathbf{x}'_U$.
Else return 0.

Definition 7. (U-Arg-RAM) A reactive argument system $(\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}}), \mathcal{V})$ (Definition 4) is U-Arg-RAM if it satisfies Definitions 5, 6 with $\gamma = \text{negl}(\lambda)$.

We formalize the notion of *zero knowledge* which captures that in the online phase, when proving NP statements related to NP relation $\mathcal{R}$, and executing update algorithm $\mathcal{U}$, the only information disclosed to $\mathcal{V}$ is the execution time of the RAM program $P = (\mathcal{R}, \mathcal{U})$, denoted as T .

Definition 8. (Zero-Knowledge) A reactive argument system $(\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}}), \mathcal{V})$ (Definition 4) is zero knowledge if for every PPT verifier $\mathcal{V}^*$, there exists an expected PPT simulator $\mathcal{S}_{\mathcal{V}^*}$ such that for every database $\mathbf{x}_0 = \{0, 1\}^n$ and efficient non-uniform distinguisher $\mathcal{D}$, equation (2) holds for the two experiments $\text{Real-Exp}_{(\mathcal{P}, \mathcal{V}^*)}^{\text{ZK}}(\lambda, n)$ and $\text{Ideal}_{(\mathcal{V}^*, \mathcal{S}_{\mathcal{V}^*})}^{\text{ZK}}(\lambda, n)$ defined below.

$$|\Pr[D(\text{Real-Exp}_{(\mathcal{P}, \mathcal{V}^*)}^{\text{ZK}}(\lambda, n)) = 1] - \Pr[D(\text{Ideal}_{(\mathcal{V}^*, \mathcal{S}_{\mathcal{V}^*})}^{\text{ZK}}(\lambda, n)) = 1]| \leq \text{negl}(\lambda). \quad (2)$$

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

1. $\text{ck} \xleftarrow{\$} \text{Setup}(1^\lambda)$. $(\text{st}_0, \mathbf{x}_0, \text{com}_0) \xleftarrow{\$} P_{\text{off}}(\lambda, \text{ck}, \mathbf{x}_0)$.
2. $(\text{st}_0^*, P_1^* = (\mathcal{R}_1^*, \mathcal{U}_1^*)) \leftarrow \mathcal{V}^*(n, \lambda, \text{ck}, \text{com}_0)$.
3. Set $\text{continue} \leftarrow \text{true}$ and $i \leftarrow 1$.
4. while continue : //Iterates at most polynomial times $U(\lambda)$
 - $\mathcal{P}_{\text{on}}(\text{st}_{i-1}, \mathbf{x}_{i-1}, P_i^*, w_i)$ and $\mathcal{V}^*(\text{st}_{i-1}^*, n, \lambda, P_i^*, \text{ck}, \text{com}_{i-1})$ interact in the online phase. $\mathcal{P}_{\text{on}}$ outputs st_i , $\mathbf{x}_i$, com_i . $\mathcal{V}^*$ outputs st_i^* and continue . If $\text{continue} = \text{true}$, $\mathcal{V}^*$ additionally outputs $P_{i+1}^* = (\mathcal{R}_{i+1}^*, \mathcal{U}_{i+1}^*)$. Let $\mathbf{V}_i$ denote the transcript of $\mathcal{V}^*$. Set $i \leftarrow i + 1$.

5. Set $U \leftarrow i - 1$. return $(V_1, \dots, V_U)$.

$\text{Ideal}_{(\mathcal{V}^*, \mathcal{S}_{\mathcal{V}^*})}^{\text{ZK}}(\lambda, n) :$

1. $\text{ck} \xleftarrow{\$} \text{Setup}(1^\lambda)$.
2. $(\text{st}_0^S, \text{com}_0) \leftarrow \mathcal{S}_{\mathcal{V}^*}(\lambda, n, \text{ck})$.
3. $(\text{st}_1^*, P_1^* = (\mathcal{R}_1^*, \mathcal{U}_1^*)) \leftarrow \mathcal{V}^*(n, \lambda, \text{ck}, \text{com}_0)$.
4. Set $\text{continue} \leftarrow \text{true}$ and $i \leftarrow 1$.
5. while continue : //Iterates at most polynomial times $U(\lambda)$
 - Let T_i denote the runtime of $P_i^*(\mathbf{x}_{i-1}, w_i)$. $\mathcal{S}_{\mathcal{V}^*}(\text{st}_{i-1}^S, T_i, \mathcal{R}_i^*(\mathbf{x}_{i-1}, w_i))$ is given oracle access to $\mathcal{V}^*(\text{st}_{i-1}^*, n, \lambda, P_i^*, \text{ck}, \text{com}_{i-1})$. $\mathcal{S}_{\mathcal{V}^*}$ outputs simulated transcript V'_i , st_i^S and com_i . $\mathcal{S}_{\mathcal{V}^*}$ rolls $\mathcal{V}^*$ to end of execution of i^{th} -invocation of online phase. $\mathcal{V}^*$ from this state outputs st_i^* and continue . If $\text{continue} = \text{true}$, $\mathcal{V}^*$ additionally outputs $P_{i+1}^* = (\mathcal{R}_{i+1}^*, \mathcal{U}_{i+1}^*)$. Set $i \leftarrow i + 1$.
6. Set $U \leftarrow i - 1$. return $(V'_1, \dots, V'_U)$.

Definition 9. ($\mathcal{U}$ -ZK-RAM) An argument system $(\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}}), \mathcal{V})$ (Definition 4) is $\mathcal{U}$ -ZK-RAM if it satisfies Definitions 5, 6, and 8 with $\gamma = \text{negl}(\lambda)$.

5 Locally Updatable Projection Codes (LUPC)

In this section, we propose the notion of LUPC that extends the notion of projection codes [42]. Let $\mathbf{x}$ and $\hat{\mathbf{x}}$ be two messages, where $\hat{\mathbf{x}}$ is obtained by updating the bit at index i of $\mathbf{x}$ to b . We denote this update as $\mathbf{x}^{(i,b)}$. Extending this notion, $\hat{\mathbf{x}} = \mathbf{x}^{(i_1, b_1), \dots, (i_t, b_t)}$ means that the message $\hat{\mathbf{x}}$ is obtained by updating the bits at indices $i_1, \dots, i_t$ of m to $b_1, \dots, b_t$, where the indices $i_1, \dots, i_t$ need not be distinct. The updates in the sequence $(i_1, b_1), \dots, (i_t, b_t)$ are ordered from earliest to latest, i.e., the update corresponding to index i_1 is the earliest, followed by that corresponding to i_2 , and so on. For an update sequence $\mathbf{u} = (i_1, b_1), \dots, (i_t, b_t)$, we use the notation $\mathbf{x}^{\mathbf{u}}$ to denote $\mathbf{x}$ is updated with update sequence $\mathbf{u}$.

5.1 Definition

While the definition of projection codes includes the parameters k and K , the length of sequence of message and codeword coordinates, respectively, [42] omits them in the application due to the variability of k , which in turn affects K . In line with the application of LUPC, we omit these varying parameters from the definition below.

Definition 10. (**LUPC: Syntax**) Let dist be a distance metric and let $0 \leq \delta_E, \delta_P < 1$. A (binary) $(n, N, \delta_E, \delta_P, \text{dist})$ locally updatable projection code (LUPC) comprises of algorithms $\Pi = (\text{Enc}, \text{Update}, \text{Proj}, \text{Dec}_E, \text{Dec}_P)$.

- $\mathbf{X} \leftarrow \text{Enc}(\mathbf{x})$: Enc maps a message $\mathbf{x} \in \{0, 1\}^n$ to a codeword $\mathbf{X} \in \{0, 1\}^N$.

- $\mathbf{X}' \leftarrow \text{Update}(\mathbf{X}, \mathbf{u})$: Given a sequence of updates $\mathbf{u} = ((i_1, b_1), \dots, (i_t, b_t))$ and a random access to a (possibly updated) codeword $\mathbf{X}$, Update updates the current codeword to $\mathbf{X}'$.
- $\mathbf{S} \stackrel{\$}{\leftarrow} \text{Proj}(\mathbf{X}, \mathbf{s})$: Let $\mathbf{s} \in [n]^k$ denote a sequence of distinct k message coordinates. Given $\mathbf{s}$ and random access to codeword $\mathbf{X}$, (possibly probabilistic) algorithm Proj outputs a sequence of (possibly non-distinct) K codeword coordinates $\mathbf{S} \in [N]^K$.
- $\mathbf{x} \leftarrow \text{Dec}_E(\mathbf{X})$: Given codeword $\mathbf{X}$, Dec_E outputs message $\mathbf{x}$.
- $\mathbf{y} \leftarrow \text{Dec}_P(\mathbf{s}, \mathbf{S}, \mathbf{Y})$: Given a sequence of distinct k message coordinates $\mathbf{s}$, a sequence of K codeword coordinates $\mathbf{S}$, and a projected codeword $\mathbf{Y} \in \{0, 1\}^K$, Dec_P outputs $\mathbf{y}$, a sequence of k (coordinate, bit) pairs.

Codewords corresponding to $\mathbf{x}$ can be obtained in two different ways: 1) It can be obtained by computing $\text{Enc}(\mathbf{x})$. 2) It can be obtained by computing an update on codeword corresponding to $\hat{\mathbf{x}}$ with update sequence $\mathbf{u}$ s.t. $\mathbf{x} = \hat{\mathbf{x}}^{\mathbf{u}}$. The Update algorithm can potentially result in multiple codewords corresponding to the same message $\mathbf{x}$. We use $\mathcal{C}_{\mathbf{x}}$ to denote this set of codewords for $\mathbf{x}$.

Definition 11. (Set $\mathcal{C}_{\mathbf{x}}$). $\mathbf{X}$ is in $\mathcal{C}_{\mathbf{x}}$, if: (i) $\mathbf{X} = \text{Enc}(\mathbf{x})$. (ii) There exists $\hat{\mathbf{x}}$ and a sequence of non-empty sequence of updates $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_t$, s.t. $\mathbf{x} = \hat{\mathbf{x}}^{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_t}$, then $\text{Update}(\dots \text{Update}(\text{Update}(\text{Enc}(\hat{\mathbf{x}}), \mathbf{u}_1), \mathbf{u}_2), \dots, \mathbf{u}_t) \in \mathcal{C}_{\mathbf{x}}$.

Definition 12. (LUPC: Completeness.) An $(n, N, \delta_E, \delta_P)$ -LUPC $\Pi = (\text{Enc}, \text{Update}, \text{Proj}, \text{Dec}_E, \text{Dec}_P)$ is complete if it satisfies the following:

1. $\forall \mathbf{x} \in \{0, 1\}^n, \forall \mathbf{X} \in \mathcal{C}_{\mathbf{x}}$, we have $\mathbf{x} = \text{Dec}_E(\mathbf{X})$.
2. $\forall \mathbf{x} \in \{0, 1\}^n, \forall \mathbf{X} \in \mathcal{C}_{\mathbf{x}}$ and sequence of distinct message coordinates $\mathbf{s}$, there exists $\mathbf{S}^*$ s.t. for all random sequence r , $\text{Proj}(\mathbf{X}, \mathbf{s}; r) = \mathbf{S}^*$. Moreover, $\mathbf{x}|_{\mathbf{s}} = \text{Dec}_P(\mathbf{s}, \mathbf{S}, \mathbf{X}|_{\mathbf{s}})$.

Definition 13. (LUPC: Soundness.) An $(n, N, \delta_E, \delta_P)$ -LUPC $\Pi = (\text{Enc}, \text{Update}, \text{Proj}, \text{Dec}_E, \text{Dec}_P)$ is sound if it satisfies the following:

1. $\forall \mathbf{x} \neq \mathbf{x}' \in \{0, 1\}^n, \forall (\mathbf{X}, \mathbf{X}') \in \mathcal{C}_{\mathbf{x}} \times \mathcal{C}_{\mathbf{x}'}$, $\text{dist}(\mathbf{X}, \mathbf{X}') \geq \delta_E N$.
2. For a (possibly corrupted) codeword $\tilde{\mathbf{X}} \in \{0, 1\}^N$, if for $\mathbf{x} \in \{0, 1\}^n$ there exists $\mathbf{X} \in \mathcal{C}_{\mathbf{x}}$ s.t. $\text{dist}(\tilde{\mathbf{X}}, \mathbf{X}) < (\delta_E/2)N$, then $\mathbf{x} = \text{Dec}_E(\tilde{\mathbf{X}})$.
3. For a (possibly corrupted) codeword $\tilde{\mathbf{X}}_0 \in \{0, 1\}^N$, if for $\mathbf{x} \in \{0, 1\}^n$ there exists $\mathbf{X}_0 \in \mathcal{C}_{\mathbf{x}}$ s.t. $\text{dist}(\tilde{\mathbf{X}}_0, \mathbf{X}_0) < (\delta_E/2)N$, then if $\tilde{\mathbf{X}}_1 \leftarrow \text{Update}(\tilde{\mathbf{X}}_0, \mathbf{u})$, then there exists codeword $\mathbf{X}_1 \in \mathcal{C}_{\mathbf{x}^{\mathbf{u}}}$ s.t. $\text{dist}(\tilde{\mathbf{X}}_1, \mathbf{X}_1) < (\delta_E/2)N$.
4. For a (possibly corrupted) codeword $\tilde{\mathbf{X}} \in \{0, 1\}^N$, if for $\mathbf{x} \in \{0, 1\}^n$ there exists $\mathbf{X} \in \mathcal{C}_{\mathbf{x}}$ s.t. $\text{dist}(\tilde{\mathbf{X}}, \mathbf{X}) < (\delta_E/2)N$, then

$$\Pr[\tilde{\mathbf{S}} \neq \mathbf{S} | \tilde{\mathbf{S}} = \text{Proj}(\tilde{\mathbf{X}}, \mathbf{s}) \text{ and } \mathbf{S} = \text{Proj}(\mathbf{X}, \mathbf{s})] \leq \text{negl}(\lambda),$$

where the probability is taken over the randomness used in Proj algorithm.

Moreover, given $\mathbf{S} \stackrel{\$}{\leftarrow} \text{Proj}(\mathbf{X}, \mathbf{s})$ and $w_H(\tilde{\mathbf{X}}|_{\mathbf{s}}, \mathbf{X}|_{\mathbf{s}}) < (\delta_P/2)K$, then $\mathbf{x}|_{\mathbf{s}} = \text{Dec}_P(\mathbf{s}, \mathbf{S}, \tilde{\mathbf{X}}|_{\mathbf{s}})$.

5.2 Construction

Our LUPC construction is inspired by the LULDC construction of [13] and uses hierarchical data structure from the first ORAM construction [29]. We adopt a similar template to [13] for the **Enc** and **Update** algorithms, with key differences: 1) We use projection codes instead of LDCs as the base encoding; 2) Our **Update** algorithm supports multiple updates while preserving the previous codeword state, unlike the single update in [13]. This enables generating robust PCPP proofs to verify the correct execution of the **Update** algorithm.

Let $\theta = \log n - \log(\log n + 1) - 1$. A codeword consists of $\theta + 2$ buckets $\mathcal{B}_0, \dots, \mathcal{B}_\theta, \mathcal{B}_{\theta+1}$, where each $\mathcal{B}_i$ for $i \in [\theta]$ encodes up to 2^i updates. $\mathcal{B}_{\theta+1}$ is a special bucket $\mathcal{B}^*$ that stores the encoding of the entire message. The construction ensures each bucket is either full or empty at all times. Our construction is presented in Figs. 1, 2, with instructions provided below.

Enc. Given $\mathbf{x}$, **Enc** algorithm (Fig. 1) initializes empty buckets $\mathcal{B}_0, \dots, \mathcal{B}_\theta$ and stores the global encoding of $\mathbf{x}$ in bucket $\mathcal{B}_{\theta+1}$.

Update. In our LUPC implementation for constructing U-Arg-RAM and U-ZK-RAM, preserving the initial state of codeword $\mathbf{X}$ after executing **Update** is essential. Thus, **Update** (Fig. 1) avoids in-place updates to $\mathbf{X}$, instead generating auxiliary buckets $\mathcal{B}'_j$ for the updated sections. The variable m , initialized in step 1, tracks the number of auxiliary buckets created. For example, if $m = 5$, **Update** has created six auxiliary buckets: $\mathcal{B}'_0, \dots, \mathcal{B}'_5$.

The for loop starting at step 2 iterates through the updates, sequentially integrating each into the updated codeword in order of arrival. In step 3, $\text{ptr}(\mathcal{B})$ refers to the pointer (physical address) of bucket $\mathcal{B}$. These steps track the updated state of each bucket at level i . $\mathcal{B}^*_i$ points to $\mathcal{B}_i$ if no auxiliary bucket $\mathcal{B}'_i$ exists; otherwise, it points to $\mathcal{B}'_i$, which holds the updated state. In the algorithm, $\text{bucket}(\mathcal{B}^*)$ refers to the content of the bucket addressed by $\mathcal{B}^*$.

If bucket at level 0 is empty, then an encoding of the update is stored in bucket at level 0 (step 4). Else, we determine β such that bucket at level β is empty (step 8), and update states of buckets as described in steps 8-12. If no such empty bucket is encountered (step 13), the global codeword at level $\theta + 1$ is updated. As can be observed in step 20, the initial codeword $\mathbf{X}$ and updated codeword $\mathbf{X}'$ are maintained during the execution of **Update** algorithm.

Proj. The **Proj** algorithm, shown in Fig. 2, uses the local decoder $\mathcal{V}^{(LDC)}$ for RM codes (step 7) and relies on its negligible failure probability. For each level j and index i in $\mathbf{s}$, **Proj** checks if an update for i is in $\mathcal{B}_j$. If present, it projects the corresponding message coordinates (step 7); otherwise, it projects coordinates proving the absence of the update (step 8). Repetition codes ensure robustness and maintain nearly equal lengths for projected codewords at each level.

Dec_E. Parse codeword $\mathbf{X}$ as $(\mathcal{B}_0, \dots, \mathcal{B}_{\theta+1})$. For each $j \in [\theta + 1]$, if $\mathcal{B}_j$ is not empty, compute $\mathbf{v}_j \leftarrow \Pi_j.\text{Dec}_E(\mathcal{B}_j)$. From $\mathbf{v}_{\theta+1}, \mathbf{v}_\theta, \dots, \mathbf{v}_0$, determine the latest bit b_i corresponding to each index $i \in [n]$. Return $\mathbf{x} \leftarrow b_1 \parallel \dots \parallel b_n$.

Dec_P. On input $(\mathbf{s} \in [n]^k, \mathbf{S} \in [N]^K, \mathbf{Y} \in \{0, 1\}^K)$,

Parameters.

1. Number of Buckets $(\theta') = \theta + 2$, where $\theta = \lceil \log n - \log(\log n + 1) - 1 \rceil$.
2. For $i \in [\theta]$, $n_i = 2^i(\lceil \log n \rceil + 1)$ and $n_{\theta+1} = n$.
3. Family of projection codes $\Pi_i = (\text{Enc}, \text{Proj}, \text{Dec})$ with parameters (n_i, N_i, δ) and associated global decoder Dec_E (for Reed-Muller codes) and local decoder $\mathcal{V}^{(LDC)}$, for $i \in [\theta + 1]$.

$\mathbf{X} \leftarrow \text{Enc}(\mathbf{x} \in \{0, 1\}^n)$:

1. Associate bucket $\mathcal{B}_i$ with projection code Π_i , for $i \in [\theta + 1]$. For $i \in [\theta]$, $\mathcal{B}_i$ has the capacity to hold projection encoding of a set of updates of size 2^i , where each update is of size $\lceil \log n \rceil + 1$. Bucket $\mathcal{B}_{\theta+1}$ stores encoding of a message of length n .
2. Set $\mathcal{B}_i \leftarrow \perp$ (empty buckets), $\forall i \in [\theta]$. Set $\mathcal{B}_{\theta+1} \leftarrow \Pi_{\theta+1}.\text{Enc}(\mathbf{x})$. Return $(\mathcal{B}_i)_{i \in [\theta+1]}$.

$\mathbf{X}' \leftarrow \text{Update}(\mathbf{X} \in \{0, 1\}^N, \mathbf{u} = ((i_1, b_1), \dots, (i_t, b_t)))$:

1. Parse codeword $\mathbf{X} = (\mathcal{B}_0, \dots, \mathcal{B}_{\theta+1})$. Set $m \leftarrow -1$.
// We ensure that a bucket $\mathcal{B}_i$ is either empty or full
2. **for** $a \in [t]$ **do**
 - // Assign bucket pointers
 3. **If** $m = -1$, set $\mathcal{B}_i^* \leftarrow \text{ptr}(\mathcal{B}_i), \forall i \in [\theta + 1]$. **Else**, set $\mathcal{B}_i^* \leftarrow \text{ptr}(\mathcal{B}_i'), \forall i \in [m]$ and $\mathcal{B}_i^* \leftarrow \text{ptr}(\mathcal{B}_i), \forall i \in [m + 1, \dots, \theta + 1]$.
// First bucket is empty
 4. **if** $\text{bucket}(\mathcal{B}_0^*) = \perp$ **then**
 5. | Compute $\Pi_0.\text{Enc}((i_a, b_a))$ and store it in $\mathcal{B}_0'$. **If** $m < 0$, set $m \leftarrow 0$.
 6. | **else**
 7. | Find the smallest index $\beta \in [\theta]$ such that $\text{bucket}(\mathcal{B}_\beta^*)$ is empty. **If** such an index doesn't exist, set $\beta \leftarrow \perp$.
// β^{th} bucket is empty
 8. | **if** $\beta \neq \perp$ **then**
 9. | | Compute list $\mathbf{v}_j \leftarrow \Pi_j.\text{Dec}_E(\text{bucket}(\mathcal{B}_j^*))$, $\forall j \in [\beta - 1]$.
 10. | | Merge all the entries of the form (i, b) from $\mathbf{v}_0, \dots, \mathbf{v}_{\beta-1}$ as well as the update pair (i_a, b_a) in ascending order, sorted based on index i in the entry. While merging, in case there are multiple entries corresponding to the same index i , we keep the repetitions but ensure that each entry with index i stores the "latest bit". **If** $i_a = i$, then b_a is the "latest bit". **Else**, the 'latest bit' refers to the bit associated with index i , found in the list $\mathbf{v}_j$ with the smallest value of j . This merged and sorted list is denoted as $\mathbf{v}_\beta$.
 11. | | Compute $\mathcal{B}_\beta' \leftarrow \Pi_\beta.\text{Enc}(\mathbf{v}_\beta)$. Set $\mathcal{B}_{\beta-1}', \dots, \mathcal{B}_0'$ to $\perp$ in that order.
 12. | | **If** $m < \beta$, set $m \leftarrow \beta$.
 13. | **else**
 14. | | // All buckets are full
 $\forall j \in [\theta + 1]$, compute $\mathbf{v}_j \leftarrow \Pi_j.\text{Dec}_E(\text{ptr}(\mathcal{B}_j^*))$. From $\mathbf{v}_{\theta+1}, \mathbf{v}_\theta, \dots, \mathbf{v}_0$ and the update pair (i_a, b_a) , determine the latest bit b_i corresponding to each index $i \in [n]$. Set $\mathbf{v} \leftarrow b_1 \parallel \dots \parallel b_n$.
 15. | | Compute $\mathcal{B}_{\theta+1}' \leftarrow \Pi_{\theta+1}.\text{Enc}(\mathbf{v})$. Set $\mathcal{B}_\theta', \dots, \mathcal{B}_0'$ to $\perp$ in that order.
 16. | | Set $m \leftarrow \theta + 1$.
 17. | **end**
 18. | **end**
 19. **end**
 20. Buckets $(\mathcal{B}_0, \dots, \mathcal{B}_{\theta+1})$ form initial codeword $\mathbf{X}$, and buckets $(\mathcal{B}_0', \dots, \mathcal{B}_m', \mathcal{B}_{m+1}, \dots, \mathcal{B}_{\theta+1})$ form updated codeword $\mathbf{X}'$.
 21. Replace $\mathcal{B}_j$ with $\mathcal{B}_j'$ in $\mathbf{X}$, $\forall j \in [m]$, to update state to updated codeword.

Fig. 1. Our LUPC construction Π_{LUPC}

```

 $\mathbf{S} \xleftarrow{\$} \text{Proj}(\mathbf{X} \in \{0, 1\}^N, \mathbf{s} \in [n]^k):$ 
1. Parse codeword  $\mathbf{X} = (\mathcal{B}_0, \dots, \mathcal{B}_{\theta+1})$ .
   //  $\mathcal{B}_0$  has atmost one update. Either it corresponds to an index in
   //  $\mathbf{s}$ , or serves as proof of the non-existence of updates for
   // indices in  $\mathbf{s}$ . Thus, include the encoding in projection.
2. If  $\mathcal{B}_0 \neq \perp$ , then set  $\mathbf{S}_0 \leftarrow (1, \dots, N_0)$ . Else, set  $\mathbf{S}_0 \leftarrow \perp$ .
3. for  $j \in [\theta]$  do
4.   if  $\mathcal{B}_j \neq \perp$  then
5.     Set  $\mathbf{r}_j \leftarrow ()$ 
6.     for  $i \in \mathbf{s}$  do
7.       Perform a binary search for index  $i$  in  $\mathcal{B}_j$  using  $\mathcal{V}^{(LDC)}$  of
       Reed-Muller codes. If update for index  $i$  is found, let  $\mathbf{a} = (a_0, \dots,$ 
8.        $a_{\lceil \log n \rceil})$  be the sequence of indices corresponding to this update.
       Else, determine (i) two consecutive updates in  $\mathcal{B}_j$  corresponding to
       indices  $g$  and  $h$  such that  $g < i < h$ ; Or (ii) update with the
       smallest index  $g$  in  $\mathcal{B}_j$  s.t.  $i < g$ ; Or (iii) update with the largest
       index  $h$  in  $\mathcal{B}_j$  s.t.  $h < i$ ; through the trace of binary search, as a
       proof of non-existence of update corresponding to index  $i$  in  $\mathcal{B}_j$ . Let
        $\mathbf{a}$  be the sequence of indices corresponding to this update(s). Merge
       sequence  $\mathbf{a}$  to sequence list  $\mathbf{r}_j$  without repetition.
9.     end
10.    Compute  $\mathbf{S}_j \leftarrow \Pi_j.\text{Proj}(\mathbf{r}_j)$ .
11.   else
12.      $\mathbf{S}_j \leftarrow \perp$ .
13.   end
14. end
15. Compute  $\mathbf{S}_{\theta+1} \leftarrow \Pi_{\theta+1}.\text{Proj}(\mathbf{s})$ .
   // As  $\mathbf{S}_j$  are likely of different length, we need to repeat
   // smaller sequences to ensure robustness of projected codeword.
16. Determine  $k'$ , the length of the longest sequence among  $\mathbf{S}_0, \dots, \mathbf{S}_{\theta+1}$ .
17.  $\forall j \in [\theta + 1]$ , if  $\mathbf{S}_j \neq \perp$ , repeat sequence " $\mathbf{S}_j$ "  $\ell_j = \lceil k' / |\mathbf{S}_j| \rceil$  times. Denote
   this repeated sequence by  $\mathbf{S}'_j$ . Else, set  $\mathbf{S}'_j \leftarrow \perp$ . Return  $\mathbf{S} \leftarrow \mathbf{S}'_0 \parallel \dots \parallel \mathbf{S}'_{\theta+1}$ .

```

Fig. 2. Our LUPC construction Π_{LUPC} (Continued)

1. Parse $\mathbf{S}$ as $\mathbf{S}'_0 \parallel \dots \parallel \mathbf{S}'_{\theta+1}$.
2. Partition codeword $\mathbf{Y}$ to $\mathbf{Y}'_0 \parallel \dots \parallel \mathbf{Y}'_{\theta+1}$, such that $\mathbf{Y}'_i$ is the projected codeword at coordinates $\mathbf{S}'_i$ of bucket i of the global codeword.
3. For each $j \in [\theta + 1]$: (a) If $\mathbf{Y}'_j \neq \perp$, let $\mathbf{S}_j$ be the sequence obtained from $\mathbf{S}'_j$ after removing repetitions. Remove repetitions from $\mathbf{Y}'_j$ to obtain $\mathbf{Y}_j$, i.e., set $\mathbf{Y}_j$ to be the first $|\mathbf{S}_j|$ bits of $\mathbf{Y}'_j$. Compute $\mathbf{v}_j \leftarrow \Pi_j.\text{Dec}(\mathbf{Y}_j)$. (b) If $\mathbf{Y}'_j = \perp$, set $\mathbf{v}_j \leftarrow \perp$.
4. From $\mathbf{v}_{\theta+1}, \mathbf{v}_\theta, \dots, \mathbf{v}_0$, determine the latest bit b_i corresponding to each index $i \in \mathbf{s}$. return $\mathbf{y} = (i, b_i)_{i \in \mathbf{s}}$.

Definition 14 (Buffered Hamming Distance (BHD) [13]). Let the codeword $\mathbf{X} \in \{0, 1\}^N$ comprise buckets $\mathcal{B}_0, \dots, \mathcal{B}_{\theta+1}$ of lengths $N_0, \dots, N_{\theta+1}$ as defined in Fig. 1. Additionally, consider another codeword $\mathbf{X}' \in \{0, 1\}^N$, consisting of

buckets $\mathcal{B}'_0, \dots, \mathcal{B}'_{\theta+1}$. We define the buffered hamming distance $w_{BH}(\mathbf{X}, \mathbf{X}') \leq \delta N$, if $\forall i \in [\theta + 1]$, $w_H(\mathcal{B}_i, \mathcal{B}'_i) \leq \delta N_i$.

Due to space constraints, we provide a formal proof in the full version. In the full version, we recall the notion of “prefix hamming metric” from [13] and show that our construction satisfies it in the context of LUPC.

Theorem 8 (LUPC for BHD metric). *For any $\alpha > 0$, Π_{LUPC} (described in Figs. 1 and 2) is an $(n, N = O(n^{1+\alpha}), \delta_E = \Omega(1), \delta_P = 1/\log(n))$ for the “BHD metric” with execution time of Enc , $t_{\text{Enc}} = \tilde{O}(n^{1+\alpha})$. Moreover, for any $t = t(n)$ -length sequence of updates, the runtime of Update , $t_{\text{Update}} = \tilde{O}(t \cdot n^\alpha)$, and for any sequence of $k = k(n)$ message coordinates, $\mathbf{S}$ is of length $\tilde{O}(k)$ and runtime of Proj , $t_{\text{Proj}} = \tilde{O}(k^2)$.*

6 Locally Updatable Holographic PCPs for RAM

In this section, we extend the notion of Holographic PCPs for RAM programs, as proposed in [42], to provide support for write operations on the database $\mathbf{x}$. We term this notion ‘Locally Updatable Holographic PCPs for RAM Programs’ (abbreviated as LU-HPCPs-RAM).

Definition 15. (LU-HPCPs-RAM) *An LU-HPCPs-RAM with respect to $(n, N, \delta_E, \delta_P)$ -LUPC scheme Π_L for distance metric dist , 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 $\mathbf{x}_0 \in \{0, 1\}^n$.
 - **Database Encoding:** $\mathcal{P}_{\text{off}}$ computes $\mathbf{X}_0 = \Pi_L.\text{Enc}(\mathbf{x}_0)$ and outputs state st_0 and $\mathbf{x}_0$. We require that $|\mathbf{X}| \leq p_{\text{off}}(n)$ and is computed within time $t_{\text{off}}(n)$ by prover’s RAM program.
- **i^{th} -Invocation of Online Phase:**
 - **Inputs:** $\mathcal{P}_{\text{on}}$ receives state st_{i-1} , $\mathbf{x}_{i-1}$, RAM program $P_i = (\mathcal{R}_i, \mathcal{U}_i) \in \{0, 1\}^\ell$ (defining NP-relation $\mathcal{R}_i$ and update algorithm $\mathcal{U}_i$) and a witness $w_i \in \{0, 1\}^\mu$ as input. $\mathcal{V}$ receives $n \in \mathbb{N}$, and $P_i \in \{0, 1\}^\ell$ as input, and is given oracle access to $\mathbf{X}_{i-1}$.
 - **Proof Generation:** $\mathcal{P}_{\text{on}}$ computes a proof π_i , updated database $\mathbf{x}_i \in \{0, 1\}^n$, an updated codeword $\mathbf{X}_i$, and an updated state st_i . $\mathcal{V}$ gets oracle access to π_i and $\mathbf{X}_i$. We require that the length of π_i , denoted as $|\pi|$, and the number of bits updated to change the state of codeword $\mathbf{X}_{i-1}$ to $\mathbf{X}_i$, are bounded by $p_{\text{on}}(n, T_i)$. Additionally, the execution time of $\mathcal{P}_{\text{on}}$ should be bounded by $t_{\text{on}}(n, T_i)$. Here T_i represents the bound on the runtime of P_i .

- **Verification:** $\mathcal{V}$ makes at most $Q(n)$ queries to $(\mathbf{X}_{i-1}, \mathbf{X}_i, \pi_i)$, 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:** For every $\mathbf{x}_0 \in \{0,1\}^n$, $P_i = (\mathcal{R}_i, \mathcal{U}_i)$ and w_i , we have:
 1. $(\text{st}_0, \mathbf{x}_0, \mathbf{X}_0) \leftarrow \mathcal{P}_{\text{off}}(\mathbf{x}_0)$.
 2. For $i = 1, 2, 3, \dots$: (a) $(\pi_i, \mathbf{x}_i, \mathbf{X}_i, \text{st}_i) \leftarrow \mathcal{P}_{\text{on}}(\text{st}_{i-1}, \mathbf{x}_{i-1}, P_i, w_i)$. (b) $\text{ver}_i \xleftarrow{\$} \mathcal{V}^{(\mathbf{X}_{i-1}, \mathbf{X}_i, \pi_i)}(n, P_i)$.
 Let $\mathbf{x}'_0 \leftarrow \mathbf{x}_0$. $\forall i$, let $\mathbf{x}'_i \leftarrow \mathcal{U}_i(\mathbf{x}'_{i-1}, w_i)$. $\forall i$, if $\mathcal{R}_i(\mathbf{x}'_{i-1}, w_i) = \text{accept}$, then after the honest $\mathcal{P}$ and $\mathcal{V}$ execute the offline phase followed by invocations of online phase, $\forall i$, $\text{ver}_i = \text{accept}$, and $\mathbf{X}_i$ is an encoding of $\mathbf{x}_i = \mathbf{x}'_i$.
- **ϵ -Soundness:** Consider the following:
 1. $(\text{st}_0^*, \mathbf{x}_0^*, \mathbf{X}_0^*, P_i^* = (\mathcal{R}_i^*, \mathcal{U}_i^*)) \leftarrow \mathcal{P}^*$.
 2. Set **continue** $\leftarrow \text{true}$ and $i \leftarrow 1$.
 3. **while** **continue**: //Iterates at most polynomial times $U(\lambda)$
 - $(\pi_i^*, \mathbf{x}_i^*, \mathbf{X}_i^*, \text{st}_i^*, \text{continue}, P_{i+1}^* = (\mathcal{R}_{i+1}^*, \mathcal{U}_{i+1}^*)) \leftarrow \mathcal{P}^*(\text{st}_{i-1}^*, \mathbf{x}_{i-1}^*, P_i^*, w_i/\perp)$. Here, $P_{i+1}^* = \perp$ if **continue** = **false**. Set $i \leftarrow i + 1$.
 4. Set $U \leftarrow i - 1$.
 Now, $\forall i \in [U]$: If $\forall i \in [U]$, $\mathcal{V}^{(\mathbf{X}_{i-1}^*, \mathbf{X}_i^*, \pi_i^*)}(n, P_i^*)$ accepts, then except with probability at most $\epsilon = \epsilon(n)$: $\forall i \in [U]$, $\exists \mathbf{X}_{i-1} \in \mathcal{C}_{\mathbf{x}'_{i-1}}$ s.t. $w_{BH}(\mathbf{X}_{i-1}, \mathbf{X}_{i-1}^*) < (\delta_E/2)N$ and $\exists w_i$ s.t. $\mathcal{R}_i^*(\mathbf{x}'_{i-1}, w_i) = \text{accept}$ and $\exists \mathbf{X}_i \in \mathcal{C}_{\mathbf{x}'_i}$ s.t. $\mathbf{x}_i^* = \mathcal{U}_i^*(\mathbf{x}'_{i-1}, w_i)$ and $w_{BH}(\mathbf{X}_i, \mathbf{X}_i^*) < (\delta_E/2)N$.
 In this definition, δ denotes the proximity parameter of our LU-HPCPs-RAM.

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

Due to space constraints, we present our LU-HPCPs-RAM construction in the full version that proves the below theorem.

Theorem 9 (LU-HPCPs-RAM). For any $\alpha > 0$, there exists an $(n, Q = \text{polylog}(n), t_{\text{off}} = \tilde{O}(n^{1+\alpha}), p_{\text{off}} = O(n^{1+\alpha}), t_{\text{on}} = \tilde{O}(T^2), p_{\text{on}} = \tilde{O}(T), t_v = \text{polylog}(n), \delta = 1/\text{polylog}(n), \epsilon = \text{negl}(n))$ -LU-HPCPs-RAM w.r.t. BHD metric.

7 U-ZK-RAM

In the construction, we use (1) a commitment scheme $\mathcal{C}$; (2) a vector commitment scheme $\mathcal{VC}$; (3) an ORAM scheme ($\text{RAM}_{\text{init}}, \text{RAM}_{\text{eval}}$); (4) a $(2\tau, \eta)$ -secret sharing scheme $\mathcal{SS} = (\text{Share}, \text{Reconstruct})$; (5) a $(2\tau, \eta)$ proactive secret sharing scheme $\mathcal{PSS}$ that is τ_r -robust; (6) a (τ, η) -robust MPC protocol $\Pi^{(\text{MPC})}$; (7) our LU-HPCPs-RAM scheme $(\mathcal{P}^{(H)} = (\mathcal{P}_{\text{off}}^{(H)}, \mathcal{P}_{\text{on}}^{(H)}), \mathcal{V}^{(H)})$ with respect to LUPC scheme Π_{LUPC} ; (8) ℓ represents the length of randomness for $\mathcal{V}^{(H)}$.

Computation of Prover (Fig. 3 and 4). In the offline phase, $\mathcal{P}_{\text{off}}$ on input $\mathbf{x}$, computes ORAM encoded memory and ORAM state $(\hat{\mathbf{x}}, \text{st}^{(o)})$ in Step 1. $\mathcal{P}_{\text{off}}$

computes LUPC encoding, $\mathbf{X}$, of $(\text{st}^{(o)} \parallel \hat{\mathbf{x}})$ in Step 2. Encoding $\text{st}^{(o)}$ may reveal accesses to $\text{st}^{(o)}$ ⁵. Thus, we tweak RAMEval so that any access to $\text{st}^{(o)}$ touches all bits in $\text{st}^{(o)}$. Since $\text{st}^{(o)}$ is succinct, this incurs negligible overhead. Then, $\mathcal{P}$ generates a PSS sharing of each bit of $\mathbf{X}$ and commits to these shares using VC.

In the online phase, $\mathcal{P}_{\text{on}}$ first receives a commitment to the challenge set from $\mathcal{V}$. In step 2, $\mathcal{P}_{\text{on}}$ invokes the LU-HPCPs-RAM prover and generates a proof for the ORAM evaluation⁶ RAMEval of the RAM program $P_{\mathcal{R}}$. At the end of this step, $\mathcal{P}_{\text{on}}$ obtains proof $\pi^{(H)}$ and updated buckets $\mathbf{Y}$ (auxiliary buckets in LUPC construction). In step 3, $\mathcal{P}_{\text{on}}$ generates and commits to secret-shares of proof $\pi^{(H)}$. In steps 4–6, $\mathcal{P}_{\text{on}}$ generates secret-shares of updated codeword, updates the commitment and sends the updated commitments to $\mathcal{V}$. Here, $\mathcal{P}_{\text{on}}$ doesn't perform in place commitment update. Rather, it preserves the auxiliary state so that it can efficiently respond to queries to initial codeword (older commitment) and final codeword (newer commitment). In steps 7–10, $\mathcal{P}_{\text{on}}$ jointly samples the randomness to emulate computation of LU-HPCPs-RAM verifier $\mathcal{V}^{(H)}$. In step 11, $\mathcal{P}_{\text{on}}$ emulates the MPC-in-the-head computation of $\mathcal{V}^{(H)}$. It commits the views to $\mathcal{V}$. In steps 12–16, $\mathcal{P}_{\text{on}}$ prepares list of openings to commitment of $\pi^{(H)}$ at indices accessed by $\mathcal{V}^{(H)}$. Similarly for $\mathbf{X}$ and $\mathbf{X}'$. $\mathcal{P}_{\text{on}}$ then renews sharings of bits accessed in $\mathbf{X}'$. For this, $\mathcal{P}_{\text{on}}$ emulates the Refresh procedure of PSS to renew shares of $\mathbf{X}'$ at indices accessed by $\mathcal{V}^{(H)}$. $\mathcal{P}_{\text{on}}$ updates the secret sharings of $\mathbf{X}$ at the accessed locations in step 27. $\mathcal{P}_{\text{on}}$ sends commitments to views generated in emulation of Refresh procedure and updated commitment of database. Upon receiving the decommitment to challenge party set $\mathcal{I}$ from $\mathcal{V}$, on verification of decommitments, $\mathcal{P}_{\text{on}}$ opens the views in MPC-in-the-head emulation of $\mathcal{V}^{(H)}$ and Refresh procedure for parties in $\mathcal{I}$. It also sends opening of shares of proof and database encodings.

Computation of Verifier (Fig. 5). $\mathcal{V}$ generates and sends commitment to challenge set $\mathcal{I}$ in step 1. In step 2, $\mathcal{V}$ receives commitments to sharings of LU-HPCPs-RAM proof $\pi^{(H)}$ and to sharings of updated codeword. In steps 3–6, $\mathcal{V}$ jointly samples randomness with $\mathcal{P}_{\text{on}}$ used to execute the LU-HPCPs-RAM verifier in the MPC-in-the-head approach. $\mathcal{V}$ obtains commitments to the views generated in the MPC-in-the-head emulations of $\mathcal{V}^{(H)}$ and PSS.Refresh procedure along with the commitment to updated sharings of input encoding. In step 8, $\mathcal{V}$ sends decommitments corresponding to the challenge set $\mathcal{I}$. $\mathcal{V}$ receives the openings to views of parties in execution of $\mathcal{V}^{(H)}$, PSS.Refresh procedure for the parties in $\mathcal{I}$. For parties in $\mathcal{I}$, $\mathcal{V}$ also receives openings to commitment of the corresponding shares of the LU-HPCPs-RAM proof and input encoding (including update information for the renewed shares). $\mathcal{V}$ checks if the views are consistent given query responses, the list of broadcast messages, etc. It also verifies that the

⁵ Since the client accesses $\text{st}^{(o)}$ locally, the ORAM server never observes them, and they are not included in ORAM security.

⁶ $\mathcal{P}$ can choose randomness in RAMEval since soundness of ORAM [53, Definition 1] ensures that for any choice of randomness, no malicious prover can prove a RAM program outputs 1 when it actually outputs 0.

Offline Phase.

Input: $\mathcal{P}_{\text{off}}$ receives λ , (ck, pp) , and $\mathbf{x} \in \{0, 1\}^n$.

Offline Commitment: $\mathcal{P}_{\text{off}}$ performs the following computation:

1. $(\widehat{\mathbf{x}}, \text{st}^{(o)}) \xleftarrow{\$} \text{RAMInit}(1^\lambda, \mathbf{x})$.
2. Invoke $\mathcal{P}_{\text{off}}^{(H)}$ on input $(\text{st}^{(o)} \parallel \widehat{\mathbf{x}})$ and obtain $(\text{st}^{(H)}, \mathbf{X} = (\mathcal{B}_0, \dots, \mathcal{B}_{\theta+1}))$.
Generate shares $\langle \mathbf{X} \rangle_1, \dots, \langle \mathbf{X} \rangle_\eta \leftarrow \text{PSS.Share}(\mathbf{X})$. Compute $(\text{st}_i^{(x)}, \text{com}_i^{(x)}) \leftarrow \text{VC.Commit}(\text{pp}, \langle \mathbf{X} \rangle_i)$.
3. Set $\text{st}^{(P)} \leftarrow \{\text{ck}, \text{pp}, \mathbf{X}, \text{st}^{(H)}, (\langle \mathbf{X} \rangle_i, \text{st}_i^{(x)})_{i \in [\eta]}\}$ and publish
 $\text{com}^{(x)} = (\text{com}_i^{(x)})_{i \in [\eta]}$.

Online Phase.

Input: $\mathcal{P}_{\text{on}}$ receives state $\text{st}^{(P)} \leftarrow \{\text{ck}, \text{pp}, \mathbf{x}, \mathbf{X}, \text{st}^{(H)}, (\langle \mathbf{X} \rangle_i, \text{st}_i^{(x)})_{i \in [\eta]}\}$, RAM program $P = (\mathcal{R}, \mathcal{U}) \in \{0, 1\}^\ell$, and a witness $w \in \{0, 1\}^\mu$ as input.

Interaction:

1. Receive $(\text{com}_i^{(T)})_{i \in [\tau]}$.
2. Let $P^{(O)}$ denote the description of ORAM Evaluation of RAM program P using RAMEval . Let ω denote the randomness used in the execution of RAMEval .
Invoke $\mathcal{P}_{\text{on}}^{(H)}$ with state $\text{st}^{(H)}$ and inputs $P^{(O)}$ and (w, ω) . $\mathcal{P}_{\text{on}}^{(H)}$ returns proof $\pi^{(H)}$ and updated buckets $\mathbf{Y} = (\mathcal{B}'_0, \dots, \mathcal{B}'_m)$, where $m \in [\theta + 1]$.
3. Generate η shares $\langle \pi^{(H)} \rangle_1, \dots, \langle \pi^{(H)} \rangle_\eta \leftarrow \text{SS.Share}(\pi^{(H)})$. Compute
 $(\text{st}_i^{(\pi)}, \text{com}_i^{(\pi)}) \leftarrow \text{VC.Commit}(\text{pp}, \langle \pi^{(H)} \rangle_i)$. Send $\{\text{com}_i^{(\pi)}\}_{i \in [\eta]}$ to $\mathcal{V}$.
4. Generate η shares $\langle \mathbf{Y} \rangle_1, \dots, \langle \mathbf{Y} \rangle_\eta \leftarrow \text{SS.Share}(\mathbf{Y})$.
5. Let IL denote the sequence of indices corresponding to updated buckets $(\mathcal{B}'_0, \dots, \mathcal{B}'_m)$. The oracle to the shares of the updated codeword, i.e., $\langle \mathbf{X}' \rangle_i$, can be mimicked by using $\langle \mathbf{Y} \rangle_i$ for queries at indices in IL . For all indices not in IL , the response can be obtained from $\langle \mathbf{X} \rangle_i$.
6. $\forall i \in [\eta]$, compute $\mathbf{u}'_i$ the sequence of pairs of "index" in IL and the corresponding bit in list $\langle \mathbf{Y} \rangle_i$. Similarly, compute $\mathbf{u}_i$ the sequence of pairs of "index" in IL and the corresponding bit in list $\langle \mathbf{X} \rangle_i$. Compute
 $(\widetilde{\text{com}}_i^{(x)}, \widetilde{\text{aux}}_i, \widetilde{\zeta}_i) \leftarrow \text{VC.Update}(\text{pp}, \text{com}_i^{(x)}, \mathbf{u}_i, \mathbf{u}'_i, \text{st}_i^{(x)})$. Send $\{\widetilde{\text{com}}_i^{(x)}\}_{i \in [\eta]}$ to $\mathcal{V}$.
// $\widetilde{\text{com}}_i^{(x)}$ can be treated as commitment to $\langle \mathbf{X}' \rangle_i$
7. Initialize $r \leftarrow 0^\ell$.
8. **for** $i \in [\ell]$ **do**
9. Generate random bit $\psi_i^{(P)} \xleftarrow{\$} \{0, 1\}$. $(\text{com}_i^{(r)}, \text{decom}_i^{(r)}) \leftarrow$
 $\text{C.Commit}(\text{ck}, \psi_i^{(P)})$. Send $\text{com}_i^{(r)}$ to $\mathcal{V}$. Receive bit $\psi_i^{(V)}$ from $\mathcal{V}$. Send
 $(\psi_i^{(P)}, \text{decom}_i^{(r)})$ to $\mathcal{V}$. Set $r[i] \leftarrow \psi_i^{(P)} \oplus \psi_i^{(V)}$.
10. **end**
11. Let f denote the n -party function corresponding to V_H . Emulate $\Pi_f^{(MPC)}$ with Z_i given $P_{\mathcal{R}}^{(O)}$ and r as input and oracle access to $\langle \mathbf{X} \rangle_i$, $\langle \mathbf{X}' \rangle_i$ and $\langle \pi^{(H)} \rangle_i$, for $i \in [\eta]$. Based on the execution, let the view of party Z_i be denoted by $\mathbf{V}_i^{(H)}$.
Let $\text{BM-List}^{(H)}$ denote the list of broadcasted messages, which includes $\text{QL}^{(X)}$, $\text{QL}^{(X')}$, and $\text{QL}^{(\pi)}$, representing the bits accessed in the input encodings and the proof. Compute $(\text{com}_i^{(H)}, \text{decom}_i^{(H)}) \leftarrow \text{C.Commit}(\text{ck}, \mathbf{V}_i^{(H)})$. Send
 $\{\text{BM-List}^{(H)}, \{\text{com}_i^{(H)}\}_{i \in [\eta]}\}$ to $\mathcal{V}$.

Fig. 3. Prover $\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}})$ of our U-ZK-RAM construction

```

12.  $\forall i \in [\eta]$ , create  $|\text{QL}^{(\pi)}|$ -sized lists  $v_i, \rho_i$ .
13. for  $j \in |\text{QL}^{(\pi)}|$  do
14.    $\forall i \in [\eta]$ , set  $v_i[j] = \langle \pi \rangle_i[\text{QL}^{(\pi)}[j]]$ . Compute
       $\rho_i[j] \leftarrow \text{VC.Open}(\text{pp}, v_i[j], \text{QL}^{(\pi)}[j])$ .
15. end
16.  $\forall i \in [\eta]$ , create list  $\text{QA}_i^{(\pi)} \leftarrow ((\text{QL}^{(\pi)}[j], v_i[j], \rho_i[j]))_{j \in |\text{QL}^{(\pi)}|}$ .
17.  $\forall i \in [\eta]$ , create  $|\text{QL}^{(X)}|$ -sized lists  $s_i, \sigma_i$ .
18. for  $j \in |\text{QL}^{(X)}|$  do
19.    $\forall i \in [\eta]$ , set  $s_i[j] = \langle \mathbf{X} \rangle_i[\text{QL}^{(X)}[j]]$ . Compute
       $\sigma_i[j] \leftarrow \text{VC.Open}(\text{pp}, s_i[j], \text{QL}^{(X)}[j])$ .
20. end
21.  $\forall i \in [\eta]$ , create list  $\text{QA}_i^{(X)} \leftarrow ((\text{QL}^{(X)}[j], s_i[j], \sigma_i[j]))_{j \in |\text{QL}^{(X)}|}$ .
22. Similarly,  $\forall i \in [\eta]$ , create list  $\text{QA}_i^{(X')}$ .
23. Let  $\text{QL}_S^{(X)} \subseteq \text{QL}^{(X)}$  denote the queries made to indices not in  $\text{IL}$ , i.e., the
    shares that are shared between the two state of codewords  $\mathbf{X}$  and  $\mathbf{X}'$ . Set
     $\text{QL} \leftarrow \text{QL}^{(X')} \cup \text{QL}_S^{(X)}$ . For  $i \in [\eta]$ , create  $|\text{QL}|$ -sized lists  $\sigma_i, s_i, \mathbf{V}_i^{(S)}, s'_i$ .
    // State  $\widetilde{\text{st}}_i^{(x)}$  can be emulated using  $\text{st}_i^{(x)}$  and  $\widetilde{\text{aux}}_i$ 
24. for  $j \in |\text{QL}|$  do
25.    $\forall i \in [\eta]$ , set  $s_i[j] = \langle \mathbf{X}' \rangle_i[\text{QL}[j]]$  and  $\sigma_i[j] \leftarrow \text{VC.Open}(\text{pp}, \langle \mathbf{X}' \rangle_i[\text{QL}[j]],$ 
       $\text{QL}[j], \widetilde{\text{st}}_i^{(x)})$ .
26.   Emulate MPC computation  $\text{PSS.Refresh}$  with  $Z_i$  given current share  $s_i[j]$ 
      as input, for  $i \in [\eta]$ . Set  $\mathbf{V}_i^{(S)}[j]$  to be view of party  $Z_i$  in this computation,
      and set  $s'_i[j]$  to new share received as output by party  $Z_i$ .
27.    $\forall i \in [\eta]$ , set  $\langle \mathbf{X}' \rangle_i[\text{QL}[j]] \leftarrow s'_i[j]$ .
28. end
29.  $\forall i \in [\eta]$ , create list  $\text{QA}_i \leftarrow ((\text{QL}[j], s_i[j], \sigma_i[j], s'_i[j]))_{j \in |\text{QL}^{(\pi)}|}$ .
30.  $\forall i \in [\eta]$ , Set  $\widetilde{\text{st}}_i^{(x)} \leftarrow \text{UpdState}(\text{st}_i^{(x)}, \widetilde{\text{aux}}_i)$ . Then, compute
       $(\text{com}_i^{(x)'}, \text{aux}_i^{(x)'}, \zeta_i^{(x)'}) \leftarrow$ 
       $\text{VC.Update}(\text{pp}, \widetilde{\text{com}}_i^{(x)}, (\text{QL}[j], s_i[j])_{j \in |\text{QL}|}, (\text{QL}[j], s'_i[j])_{j \in |\text{QL}|}, \widetilde{\text{st}}_i^{(x)})$ .
31. Set  $\text{com}^{(x)'} \leftarrow (\text{com}_i^{(x)'})_{i \in [\eta]}$ ,
       $\text{st}^{(x)'} \leftarrow (\text{st}_i^{(x)'} = \text{VC.UpdState}(\widetilde{\text{st}}_i^{(x)}, \text{aux}_i^{(x)'})_{i \in [\eta]}$ .
32.  $\forall i \in [\eta]$ , compute  $(\text{com}_i^{(S)}, \text{decom}_i^{(S)}) \leftarrow \text{C.Commit}(\mathbf{V}_i^{(S)})$ .
33. Send  $(\text{com}_i^{(S)})_{i \in [\eta]}$  and  $\text{com}^{(x)'}$  to  $\mathcal{V}$ .
34. Receive  $(\text{decom}_1^{\mathcal{T}}, i_1), \dots, (\text{decom}_\tau^{\mathcal{T}}, i_\tau)$  from  $\mathcal{V}$ .  $\forall j \in [\tau]$ , if
       $\text{Verify}(\text{ck}, \text{com}_j^{\mathcal{T}}, \text{decom}_j^{\mathcal{T}}, i_j) = \text{reject}$ , abort. Send
       $((\mathbf{V}_i^{(H)}, \text{decom}_i^{(H)}), (\mathbf{V}_i^{(S)}, \text{decom}_i^{(S)}))_{i \in \mathcal{I}}$  and  $(\text{QA}_i^{(\pi)}, \text{QA}_i^{(X)}, \text{QA}_i^{(X')}, \text{QA}_i,$ 
 $\zeta_i^{(x)'})_{i \in \mathcal{I}}$  to  $\mathcal{V}$ .
35. Set  $\text{st}^{(\mathcal{P})} \leftarrow \{\text{ck}, \text{pp}, \mathbf{x}, \mathbf{X}', \text{st}^{(H)'}, (\langle \mathbf{X}' \rangle_i, \text{st}_i^{(x)'})_{i \in [\eta]}\}$ .

```

Fig. 4. Prover $\mathcal{P} = (\mathcal{P}_{\text{off}}, \mathcal{P}_{\text{on}})$ of our U-ZK-RAM construction (Continued)

query responses are consistent with the commitment. Finally, it verifies that the shares are renewed consistently and commitments are appropriately updated.

Input: $\mathcal{V}$ receives $n, \lambda \in \mathbb{N}$, $P_{\mathcal{R}}$, pp , and $\text{com}^{(x)} = (\text{com}_i^{(x)})_{i \in [\eta]}$ as input.
Interaction:

1. Sample distinct $i_1, \dots, i_\tau \xleftarrow{\$} [\eta]$. Compute $(\text{com}_j^{(X)}, \text{decom}_j^{(X)}) \leftarrow \text{C.Commit}(i_j), \forall j \in [\tau]$. Send $(\text{com}_i^{(X)})_{i \in [\tau]}$.
2. Receive $(\text{com}_i^{(\pi)}, \widetilde{\text{com}}_i^{(x)})_{i \in [\eta]}$ from $\mathcal{P}_{\text{on}}$.
3. Initialize $r' \leftarrow 0^\ell$.
4. **for** $i \in [\ell]$ **do**
5. Receive $\text{com}_i^{(r)}$ from $\mathcal{P}_{\text{on}}$. Generate random bit $\psi_i^{(V)} \xleftarrow{\$} \{0, 1\}$. Send $\psi_i^{(V)}$ to $\mathcal{P}_{\text{on}}$. Receive $(\psi_i^{(P)}, \text{decom}_i^{(r)})$ from $\mathcal{P}_{\text{on}}$. If $\text{C.Verify}(\text{ck}, \text{com}_i^{(r)}, \text{decom}_i^{(r)}, \psi_i^{(P)}) = \text{reject}$, abort else continue. Set $r'[i] \leftarrow \psi_i^{(P)} \oplus \psi_i^{(V)}$.
6. **end**
7. Receive $\{\text{BM-List}^{(H)}, \{\text{com}_i^{(H)}\}_{i \in [\eta]}\}, (\text{com}_i^{(S)})_{i \in [\eta]}$ and $\text{com}^{(x)'} from $\mathcal{P}_{\text{on}}$.$
8. Send $(\text{decom}_1^{(X)}, i_1), \dots, (\text{decom}_\tau^{(X)}, i_\tau)$ to $\mathcal{V}$. Receive $(V_i^{(H)}, \text{decom}_i^{(H)}), (V_i^{(S)}, \text{decom}_i^{(S)})_{i \in \mathcal{I}}$ and $(\text{QA}_i^{(\pi)}, \text{QA}_i^{(X)}, \text{QA}_i^{(X')}, \text{QA}_i, \zeta_i^{(x)'})_{i \in \mathcal{I}}$ from $\mathcal{P}_{\text{on}}$.
// Check consistency of query responses w.r.t. commitment
9. **for** $i \in \mathcal{I}$ **do**
10. $\forall \text{entry} = (\text{ind}, v, \rho) \in \text{QA}_i^{(\pi)}$, compute $\text{ver} \leftarrow \text{Verify}(\text{pp}, \text{com}_i^{(\pi)}, \text{ind}, v, \rho)$. If $\text{ver} = \text{reject}$, abort.
11. **end**
12. Similarly, check consistency of query response in lists $\text{QA}^{(X)}$ and $\text{QA}^{(X)'}$
13. Compute $C_H \leftarrow \Psi(\mathcal{V}^{(H)})$.
14. $\forall i \in \mathcal{I}$, if $\text{C.Verify}(\text{ck}, \text{com}_i^{(H)}, \text{decom}_i^{(H)}, V_i^{(H)}) = \text{reject}$, abort.
15. Let $P_{\mathcal{R}}^{(O)}$ denote the description of ORAM Evaluation of RAM program $P_{\mathcal{R}}$ using RAMEval algorithm. Check consistency of all the views $(V_i^{(H)})_{i \in \mathcal{I}}$ for MPC computation of f , given $P_{\mathcal{R}}^{(O)}$, list of broadcasted messages $\text{BM-List}^{(H)}$, randomness r , query responses in $(\text{QA}_i^{(X)})_{i \in \mathcal{I}}$, $(\text{QA}_i^{(X')})_{i \in \mathcal{I}}$ and $(\text{QA}_i^{(\pi)})_{i \in \mathcal{I}}$, and the output is accept in all these views. If not consistent, abort.
// Verify if shares are refreshed and committed correctly
16. From $\text{BM-List}^{(H)}$, determine query lists $\text{QL}^{(X')}$ and $\text{QL}^{(X)}$. Set $\text{QL} \leftarrow \text{QL}^{(X')} \cup \text{QL}^{(X)}$.
17. $\forall i \in \mathcal{I}$, if $\text{C.Verify}(\text{ck}, \text{com}_i^{(S)}, \text{decom}_i^{(S)}, V_i^{(S)}) = \text{reject}$, abort.
18. $\forall j \in |\text{QL}|$, check consistency of views $(V_i^{(S)}[j])_{i \in \mathcal{I}}$ for MPC computation of $\text{PSS.Refresh}()$ circuit, given $(\text{QA}_i[j])_{i \in \mathcal{I}}$. If not consistent, abort.
19. $\forall i \in \mathcal{I}$, entry $\text{QA}_i[j]$ is of the form $(\text{QL}[j], s_i[j], \sigma_i[j], s'_i[j])$. Now, $\forall i \in \mathcal{I}$, compute $\widetilde{\text{com}}_i^{(x)}, (\text{QL}[j], \sigma_i^{(j)'})_{j \in |\text{QL}|} \leftarrow \text{ProofUpdate}(\text{pp}, \text{com}_i^{(x)}, (\text{QL}[j], \sigma_i[j], s'_i[j]), s'_i[j])_{j \in |\text{QL}|}, \zeta_i^{(x)'})$. If $\widetilde{\text{com}}_i^{(x)} \neq \text{com}_i^{(x)'}$, abort.

Fig. 5. Verifier $\mathcal{V}$ of our U-ZK-RAM construction

We set $\tau = \Omega(\lambda)$, $\tau_r = 2\tau$ and $\eta = c\lambda$ in our construction. We prove the below Theorem in the full version.

Theorem 10. (U-ZK-RAM) *For every $\alpha > 0$, there exists an $(n, t_{\text{off}} = \tilde{O}(n^{1+\alpha}), c_{\text{off}} = \text{poly}(\lambda, \log(n)), t_{\text{on}} = \tilde{O}(T^2), c_{\text{on}} = \text{poly}(\lambda, \log(n)), t_v = \text{poly}(\lambda, \log(n)), \gamma = \text{negl}(\lambda))$ -U-ZK-RAM that makes a black-box use of CRHF.*

Acknowledgements. We would like to thank anonymous Eurocrypt reviewers for their helpful comments. Y. Ishai was supported by ISF grants 2774/20 and 3527/24, BSF grant 2022370, and ISF-NSFC grant 3127/23. R. Ostrovsky was supported in part by NSF grants CNS-2246355, CCF-2220450, US-Israel BSF grant 2022370, and by Sunday Group.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)
2. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac’n’Cheese: zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 92–122. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_4
3. Ben-Sasson, E., Chiesa, A.: Personal Communication (2023)
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: On the concrete efficiency of probabilistically-checkable proofs. In: STOC. ACM (2013)
5. 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)
6. 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)
7. Bitansky, N., et al.: The hunting of the SNARK. J. Cryptol. (2017)
8. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. **37**(2), 156–189 (1988)
9. Büinz, B., Chen, B.: Protostar: generic efficient accumulation/folding for special-sound protocols. In: ASIACRYPT. Springer, Cham (2023)
10. Campanelli, M., Fiore, D., Han, S., Kim, J., Kolonelos, D., Hyunok, O.: Succinct zero-knowledge batch proofs for set accumulators. In: ACM CCS. ACM (2022)
11. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
12. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_5
13. Chandran, N., Kanukurthi, B., Ostrovsky, R.: Locally updatable and locally decodable codes. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 489–514. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_21
14. 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

15. Choudhuri, A.R., Garg, S., Goel, A., Sekar, S., Sinha, R.: Sublonk: sublinear prover plonk. In: PoPETs (2024)
16. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_32
17. Damgård, I.B., Pedersen, T.P., Pfitzmann, B.: On the existence of statistically hiding bit commitment schemes and fail-stop signatures. *J. Cryptol.* **10**(3), 163–194 (1997). <https://doi.org/10.1007/s001459900026>
18. Damgård, I., Pedersen, T.P., Pfitzmann, B.: Statistical secrecy and multibit commitments. *IEEE Trans. Inf. Theory* (1998)
19. Di, Z., Xia, L., Nguyen, W.D., Tyagi, N.: Muxproofs: succinct arguments for machine computation from tuple lookups. *IACR Cryptol. ePrint Arch.* 974 (2023)
20. Dutta, M., Ganesh, C., Patranabis, S., Prakash, S., Singh, N.: Batching-efficient RAM using updatable lookup arguments. In: ACM CCS. ACM (2024)
21. Eagen, L., Fiore, D., Gabizon, A.: CQ: cached quotients for fast lookups. *IACR Cryptol. ePrint Arch.* 1763 (2022)
22. 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)
23. Friedl, K., Sudan, M.: Some improvements to total degree tests. In: ISTCS. IEEE Computer Society (1995)
24. Gabizon, A., Khovratovich, D.: flookup: fractional decomposition-based lookups in quasi-linear time independent of table size. *IACR Cryptol. ePrint Arch.* 1447 (2022)
25. 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
26. Goel, A., Hall-Andersen, M., Kaptchuk, G.: Dora: processor expressiveness is (nearly) free in zero-knowledge for RAM programs. In: ACM CCS. ACM (2024)
27. Goldreich, O.: The Foundations of Cryptography - Volume 1: Basic Techniques. Cambridge University Press, Cambridge (2001)
28. Goldreich, O.: On doubly-efficient interactive proof systems. *Found. Trends Theor. Comput. Sci.* (2018)
29. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. *J. ACM* (1996)
30. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. *J. ACM* (2015)
31. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. *SIAM J. Comput.* **18**(1), 186–208 (1989)
32. Gordon, S.D., et al.: Secure two-party computation in sublinear (amortized) time. In: CCS, pp. 513–524. ACM (2012)
33. 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
34. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11

35. Gurevich, Y., Shelah, S.: Nearly linear time. In: Meyer, A.R., Taitslin, 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
36. Halevi, S., Micali, S.: Practical and provably-secure commitment schemes from collision-free hashing. In: Kobitz, N. (ed.) *CRYPTO 1996*. LNCS, vol. 1109, pp. 201–215. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_16
37. 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
38. Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: Canteaut, A., Ishai, Y. (eds.) *EUROCRYPT 2020*. LNCS, vol. 12107, pp. 569–598. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_19
39. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: how to cope with perpetual leakage. In: Coppersmith, D. (ed.) *CRYPTO 1995*. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_27
40. 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
41. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: *STOC*. ACM (2007)
42. Ishai, Y., Ostrovsky, R., Shah, A.: Succinct arguments for RAM programs via projection codes. In: Handschuh, H., Lysyanskaya, A. (eds.) *CRYPTO 2023*. LNCS, vol. 14082, pp. 159–192. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38545-2_6
43. Ishai, Y., Weiss, M.: Probabilistically checkable proofs of proximity with zero-knowledge. In: Lindell, Y. (ed.) *TCC 2014*. LNCS, vol. 8349, pp. 121–145. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_6
44. 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)
45. Kosba, A.E., Papadopoulos, D., Papamanthou, C., Song, D.: MIRAGE: succinct arguments for randomized algorithms with applications to universal ZK-snarks. In: *USENIX Security*. USENIX Association (2020)
46. Kothapalli, A., Setty, S.T.V.: Supernova: proving universal machine executions without universal circuits. *IACR Cryptol. ePrint Arch.* 1758 (2022)
47. Kothapalli, A., Setty, S.: Hypernova: recursive arguments for customizable constraint systems. In: Reyzin, L., Stebila, D. (eds.) *CRYPTO 2024*. LNCS, vol. 14929, pp. 345–379. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-68403-6_11
48. Kothapalli, A., Setty, S., Tzialla, I.: Nova: recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) *CRYPTO 2022*. LNCS, vol. 13510, pp. 359–388. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15985-5_13
49. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Micciancio, D. (ed.) *TCC 2010*. LNCS, vol. 5978, pp. 499–517. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11799-2_30

50. 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
51. Micali, S.: CS proofs (extended abstracts). In: FOCS. IEEE Computer Society (1994)
52. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: FOCS. IEEE Computer Society (2003)
53. 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
54. 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
55. Ostrovsky, R., Shoup, V.: Private information storage (extended abstract). In: ACM Symposium on the Theory of Computing, pp. 294–303. ACM (1997)
56. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: PODC. ACM (1991)
57. Ozdemir, A., Wahby, R.S., Whitehat, B., Boneh, D.: Scaling verifiable computation using efficient set accumulators. In: USENIX Security. USENIX Association (2020)
58. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: IEEE SP. IEEE Computer Society (2013)
59. Posen, J., Kattis, A.A.: Caulk+: table-independent lookup arguments. IACR Cryptol. ePrint Arch. 957 (2022)
60. Rubinfeld, R., Sudan, M.: Robust characterizations of polynomials with applications to program testing. SIAM J. Comput. **25**(2), 252–271 (1996)
61. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
62. Stefanov, E., et al.: Path ORAM: an extremely simple oblivious RAM protocol. In: ACMCCS. ACM (2013)
63. 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
64. Wahby, R.S., Setty, S., Howald, M., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. In: NDSS. The Internet Society (2015)
65. Yang, Y., Heath, D., Hazay, C., Kolesnikov, V., Venkitasubramaniam, M.: Batchman and robin: batched and non-batched branching for interactive ZK. In: ACM CCS. ACM (2023)
66. Yang, Y., Heath, D., Hazay, C., Kolesnikov, V., Venkitasubramaniam, M.: Tight ZK CPU: batched ZK branching with cost proportional to evaluated instruction. In: ACM CCS. ACM (2024)
67. Yekhanin, S.: Locally decodable codes. Found. Trends® Theor. Comput. Sci. **6**(3), 139–255 (2012)
68. Zapico, A., Buterin, V., Khovratovich, D., Maller, M., Nitulescu, A., Simkin, M.: Caulk: lookup arguments in sublinear time. In: ACM CCS. ACM (2022)
69. Zapico, A., Gabizon, A., Khovratovich, D., Maller, M., Ràfols, C.: Baloo: nearly optimal lookup arguments. IACR Cryptol. ePrint Arch. 1565 (2022)
70. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vRAM: faster verifiable RAM with program-independent preprocessing. In: IEEE SP. IEEE Computer Society (2018)