

Efficient Single-Server Online-Offline PIR without Periodic Preprocessing

Hoang-Dung Nguyen
Virginia Tech
nhd@vt.edu

Jorge Guajardo
Robert Bosch LLC — RTC
jorge.guajardomerchan@us.bosch.com

Thang Hoang
Virginia Tech
thanghoang@vt.edu

Abstract—Private Information Retrieval (PIR) allows a client to retrieve an entry from a public database without revealing the entry of interest. Standard PIR, however, requires the server to perform expensive computation that is linear in the database size per client query. To reduce this online cost, Online-Offline PIR (OO-PIR) was proposed, allowing the client to precompute a query-independent hint table that enables sublinear online query complexity. Unfortunately, existing OO-PIR protocols require either a non-colluding two-server setting or a single-server setting with expensive periodic preprocessing, where the entire hint table must be rebuilt after a limited number of online queries. This results in extremely high bandwidth or computation overhead.

We present **ESCAPE**, a novel OO-PIR protocol for the single-server setting that completely eliminates the expensive periodic preprocessing, supporting unlimited online queries in sublinear time with low constant response bandwidth. The core innovation in **ESCAPE** lies in reconciling a new hint sampling strategy with Linearly Homomorphic Encryption (LHE) to conceal the correlation between any hint and any online query, while allowing the consumed hint to be refreshed on the fly in sublinear time. We design a random sampling structure that aligns with deterministic, precomputable linear functions, enabling the protocol to exploit the streamlined preprocessing of efficient LHE instantiations. We fully implement **ESCAPE**, evaluate it on large-scale databases, and release our implementation as open source. Experimental results show that **ESCAPE** radically reduces end-to-end latency to under a second for 1-8 TiB database sizes and 8-16 KiB entries, achieving up to two orders of magnitude lower bandwidth and up to three orders of magnitude lower computation than state-of-the-art PIR.

1. Introduction

Private Information Retrieval (PIR), introduced by Chor, Goldreich, Kushilevitz, and Sudan [18], permits any client to retrieve an entry at index $i \in \{1, \dots, N\}$ from a public database of N entries, without revealing the requested entry to the database server. PIR is a well-known primitive that has been extensively studied in theory and applied to many applications: password breach alerts [7, 19, 20], private advertising [21, 22, 23], private database search [24, 25], private communications [26, 27, 28], private media consumption [29]. An efficient PIR scheme tailored for such applications (i.e., moderate entry size from hundreds of bytes to a few KiB) would bring the primitive closer to practice.

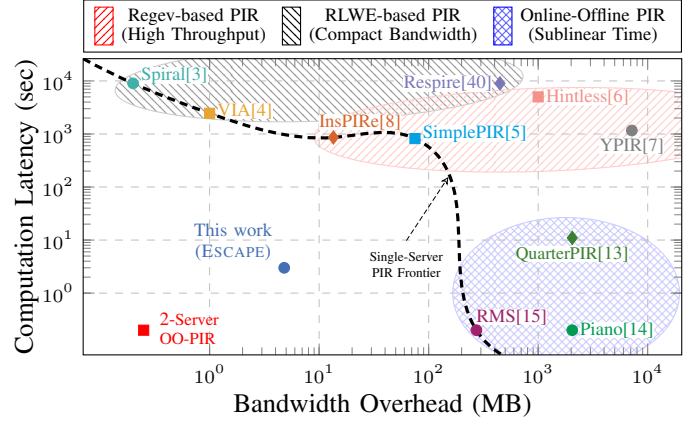


Figure 1: Advancing Single-Server PIR Frontier.
(Bandwidth vs. Computation – $2^{30} \times 64$ -KiB entry)

Originally, PIR was studied in the multi-server setting and achieved information-theoretic privacy [30, 31, 32, 33, 34]. This model assumes the servers are non-colluding yet maintain the same database state, which may be difficult to justify for practical deployment. Subsequent works [35, 36, 37, 38] move PIR to the single-server setting by relaxing PIR privacy to computational hardness. Despite its strong privacy, PIR overhead is substantial in both computation and bandwidth, where the server must process the entire database per query as the lower bound [32, 39], resulting in a linear-size query. A long line of research has pursued a balanced trade-off between server computation and bandwidth for the single-server PIR, shown in Figure 1 and detailed below.

Lattice-based PIR. When the client cannot keep any private state, linear database processing is unavoidable. Compressing linear-size queries to polylogarithmic levels needs advanced lattice-based primitive [1, 2, 3, 4, 19, 40, 41, 42]. To evaluate a compressed query while keeping the response correct and small, the server must perform substantially heavier work in the encrypted domain. Recent works [5, 6, 7, 8, 9, 43] instead allow both query and response to grow sublinearly with the database, trading slightly larger bandwidth for higher server throughput via cheaper Regev-based constructions. Even so, these schemes cannot surpass the linear-processing barrier.

Online-Offline PIR. When the client can store private state, OO-PIR [44] lets the client preprocess a query-independent hint table for sublinear-time online query. Since two-server OO-PIR [15, 45, 46, 47, 48, 49] achieves sublinear time and low bandwidth, follow-up works [10, 11, 12, 14, 15, 16, 17, 45] attempt to realize OO-PIR in the single-server setting.

TABLE 1: Asymptotic Cost Comparison with Prior Works in Single-Server PIR.

Scheme [‡]	Amortized Bandwidth		Amortized Computation	Periodic Rebuild	Offline Preprocessing		Client Storage
	Query	Response			Bandwidth	Computation	
XPIR[1], FastPIR[2]	$\mathcal{O}_\lambda(N)$	$\mathcal{O}_\lambda(\beta)$	$\mathcal{O}_\lambda(\beta N)_\mathbb{R}$	No	—	—	$\mathcal{O}_\lambda(1)$
Spiral[3], VIA[4]	$\mathcal{O}_\lambda(1)$	$\mathcal{O}_\lambda(\beta)$	$\mathcal{O}_\lambda(\beta N)_\mathbb{R}$	No	$\mathcal{O}_\lambda(\log N)$	$\mathcal{O}_\lambda(\beta N)_{\text{NTT}}$	$\mathcal{O}_\lambda(1)$
SimplePIR[5]	$\mathcal{O}_\lambda(\sqrt{\beta N})$	$\mathcal{O}_\lambda(\sqrt{\beta N})$	$\mathcal{O}(\beta N)_\odot$	No	$\mathcal{O}_\lambda(\beta\sqrt{N})$	$\mathcal{O}_\lambda(\beta N)_\odot$	$\mathcal{O}_\lambda(\sqrt{\beta N})$
HintlessPIR[6]	$\mathcal{O}_\lambda(\sqrt{\beta N})$	$\mathcal{O}_\lambda(\sqrt{\beta N})$	$\mathcal{O}(\beta N)_\odot$	No	—	$\mathcal{O}_\lambda(\beta N)_\odot$	$\mathcal{O}_\lambda(1)$
YPIR[7], InsPIRe[8]	$\mathcal{O}_\lambda(\sqrt{N})$	$\mathcal{O}_\lambda(\beta)$	$\mathcal{O}(\beta N)_\odot$	No	—	$\mathcal{O}_\lambda(\beta N)_\odot$	$\mathcal{O}_\lambda(1)$
ZipPIR[9]	$\mathcal{O}_\lambda(\sqrt{\beta N})$	$\mathcal{O}_\lambda(\sqrt{\beta N})$	$\mathcal{O}(\beta N)_\odot$	No	$\mathcal{O}_\lambda(1)$	$\mathcal{O}_\lambda(\beta N)_\odot$	$\mathcal{O}_\lambda(1)$
LP[10], ZLTS[11]	$\mathcal{O}_\lambda(\log N)$	$\mathcal{O}_\lambda(\beta)$	$\mathcal{O}_\lambda(\beta\sqrt{N} \log N)_{\text{FHE}}$	Yes	$\mathcal{O}_\lambda(\beta\sqrt{N})$	$\mathcal{O}_\lambda(\beta N \log N)_{\text{FHE}}$	$\mathcal{O}_\lambda(\beta\sqrt{N})$
CHK[12, Thm 4.1]	$\mathcal{O}_\lambda(\sqrt{N} \log N)$	$\mathcal{O}_\lambda(\beta\sqrt{N} \log N)$	$\mathcal{O}_\lambda(\beta N^{3/4} \log N)_{\text{LHE}}$	Yes	$\mathcal{O}_\lambda(\beta N^{3/4} \log N)$	$\mathcal{O}_\lambda(\beta N \log N)_{\text{LHE}}$	$\mathcal{O}_\lambda(\beta N^{3/4} \log N)$
QuarterPIR[13]	$\mathcal{O}_\lambda(N^{1/4})$	$\mathcal{O}(\beta\sqrt{N})$	$\mathcal{O}_\lambda(\beta\sqrt{N})_\oplus$	Yes	$\mathcal{O}(\beta N)$	$\mathcal{O}_\lambda(\beta N)_\oplus$	$\mathcal{O}_\lambda(\beta\sqrt{N})$
Piano[14], RMS[15]	$\mathcal{O}(\sqrt{N} \log N)$	$\mathcal{O}(\beta\sqrt{N}/\lambda)$	$\mathcal{O}(\beta\sqrt{N})_\oplus$	Yes	$\mathcal{O}(\beta N)$	$\mathcal{O}_\lambda(\beta N)_\oplus$	$\mathcal{O}_\lambda(\beta\sqrt{N})$
WR[16]	$\mathcal{O}(\sqrt{N} \log N)$	$\mathcal{O}(\beta\sqrt{N})$	$\mathcal{O}(\beta\sqrt{N})_\oplus$	Yes	$\mathcal{O}(\beta N)$	$\mathcal{O}(\beta N)_\oplus$	$\mathcal{O}(\beta\sqrt{N})$
ThorPIR[17]	$\mathcal{O}(N^{1/3} \log N)$	$\mathcal{O}_\lambda(\beta N^{1/3})$	$\mathcal{O}_\lambda(\beta N^{2/3})_{\text{FHE}}$	Yes	$\mathcal{O}_\lambda(\beta N^{2/3})$	$\mathcal{O}_\lambda(\beta N)_{\text{FHE}}$	$\mathcal{O}(\beta N^{2/3})$
This work	$\mathcal{O}_\lambda(N^{2/3})$	$\mathcal{O}_\lambda(\beta)$	$\mathcal{O}(\beta N^{2/3})_\odot$	No	$\mathcal{O}(\beta N)^*$	$\mathcal{O}_\lambda(\beta N)_\odot$	$\mathcal{O}_\lambda(\beta N^{2/3})$

[‡] We denote β for entry size, N for number of database entries, $\odot$ for addition/multiplication over 32-bit finite fields, $\oplus$ for XOR operation, $\mathbb{R}$ for ring-polynomial multiplication via NTT (Number Theoretic Transform), FHE/LHE for generic fully/linearly homomorphic operation. $\mathcal{O}_\lambda(\cdot)$ indicates the cost hides polylogarithmic factor of security parameter λ in some schemes (e.g., LP, ZLTS). In practice, the computational cost order is: $\text{FHE} > \mathbb{R} > \text{LHE} > \odot > \oplus$.

* If $\beta = \Omega(N^{2/3})$, this can be reduced to $\mathcal{O}_\lambda(\beta N^{2/3})$ with an increased multiplicative factor of $\mathcal{O}(N^{1/3})$ into the *one-time* server preprocessing computation.

However, without having the second server to separate the offline and online phases, the offline hint preprocessing must be entirely hidden from the server processing the online queries to preserve PIR privacy. One line of works [10, 11, 12, 17] leverage Fully Homomorphic Encryption (FHE) to delegate preprocessing to the server obliviously. Other works [13, 14, 15, 16] rely only on one-way functions and avoid FHE completely, but requires the client to perform preprocessing itself by streaming the entire database. Regardless, preprocessing only delivers a limited number of hints, each consumed per query. If all hints are used up, the client cannot query until another preprocessing completes in full. Thus, periodic preprocessing is inevitable in these prior works.

The Tension. Single-server PIR with low bandwidth incurs linear server work with expensive primitive. Sublinear server work demands an expensive periodic preprocessing cost in either server time or client bandwidth to sustain unlimited queries. We thus target the following research question:

Can we design a lightweight single-server PIR scheme supporting unlimited queries at sublinear online time and low bandwidth, without expensive periodic preprocessing?

1.1. Our Contributions

We answer this question affirmatively by constructing a novel single-server PIR called ESCAPE – Efficient Scalable Priate Information Retrieval. Table 1 compares ESCAPE performance with state-of-the-art single-server PIR. To our knowledge, ESCAPE is the first to address this tension with the following desirable properties:

- **Unlimited Queries without Periodic Hint Table Rebuild:** The most unique property of ESCAPE is that it permits the computation of individual hint instantly on the fly without expensive cryptographic primitives. Our preprocessing is truly one time, such that in the online phase, the client can perform unlimited arbitrary queries: each query consumes

one hint, which gets refreshed in sublinear server time and constant response bandwidth.

- **Sublinear Server Computation:** ESCAPE maintains the sublinear efficiency of the two-server OO-PIR. For each query, the server and client compute $\mathcal{O}(N^{2/3})$ modular arithmetic operations over a small finite field, which is much cheaper than $\tilde{\mathcal{O}}_\lambda(\sqrt{N})$ amortized FHE operations from prior studies [10, 11, 17].
- **Minimal Response Bandwidth:** The response bandwidth in ESCAPE is independent of the number of database entries. To retrieve an entry with a hint and then refresh that hint privately, the client only downloads two aggregated entries, which is much lower than state-of-the-art [14, 15, 16] with $\mathcal{O}(\sqrt{N})$ entries for amortized preprocessing bandwidth.
- **Hardware-Friendly Query Evaluation with Vectorized Native Integer Arithmetic:** ESCAPE is simple by design, with all computations realized via highly vectorizable modular arithmetic using SIMD intrinsics that are supported by commodity hardware. We implemented ESCAPE and evaluated its performance on large databases. The codebase is available at <https://github.com/vt-asaplab/Escape>.
- **Extremely Low Latency on TiB-Scale Database:** With low bandwidth cost, sublinear computation, and hardware-friendly simplicity, ESCAPE achieves extremely low latency. At TiB scales, ESCAPE is two to three orders of magnitude faster than lattice-based PIR at a similarly low bandwidth, taking a second to query a database with billions of 16 KiB entries, cutting OO-PIR bandwidth by at least 95%.
- **Technique: Pseudorandom Set on Structured Matrix** At the core, we formulate a novel strategy to sample a random hint set that remains hidden from the server across both offline preprocessing and online queries. We further design this sampling on a preprocessed matrix structure so that existing high-throughput LHE evaluates each query in sublinear time. We rigorously analyze and formally prove that ESCAPE satisfies PIR security.

1.2. Technical Overview

We recap how two-server OO-PIR works, then explain the challenges in realizing OO-PIR with a single server, and finally present our main idea to address the challenges.

Two-Server OO-PIR. The high-level idea is to let the client preprocess $M = \mathcal{O}(\sqrt{N} \log N)$ hints $\mathcal{H} = (h_1, \dots, h_M)$ that can accelerate online query time. The client first executes an offline phase with one server (Left) to prepare the hints. Each hint $h_i = (\mathcal{S}_i, \rho_i)$ depicts a set $\mathcal{S}_i = (s_1, \dots, s_{\sqrt{N}})$ of $\sqrt{N}$ random DB indices where $\rho_i = \sum_{s \in \mathcal{S}_i} \text{DB}[s]$ is the offline parity sum computed by server Left given $\mathcal{S}_i$. M hints ensure that per online query, the client can recover any entry $\text{DB}[x]$ via any hint $h = (\mathcal{S}, \rho)$ where $x \in \mathcal{S}$, by combining the offline parity sum ρ with a punctured parity $\hat{\rho} = \sum_{s \in \mathcal{S}} \text{DB}[s]$ obtained on revealing an online set $\mathcal{S}' \leftarrow \mathcal{S} \setminus \{x\}$ to server Right. For security, the same hint $h = (\mathcal{S}, \rho)$ cannot be reused to retrieve another entry index $x' \in \mathcal{S}$ as server Right can take set difference of $\mathcal{S} \setminus \{x\}$ and $\mathcal{S} \setminus \{x'\}$ to learn the private indices (x, x') . To maintain the hint distribution for next queries, the client needs to refresh the consumed hint h (used to recover $\text{DB}[x]$) by preprocessing a new random hint that also covers entry $\text{DB}[x]$ using the server Left.

1-Server Hint Preprocessing Challenge. To coalesce the Left and Right views under one server, prior works conceal the Left preprocessing view via FHE at the server [10, 11, 12, 17] or by building the entire hint table at the client side [13, 14, 15, 16]. This is because if the server observes both the online query set $\mathcal{S} \setminus \{x\}$ and the offline query set $\mathcal{S}$ previously preprocessed for hint $h = (\mathcal{S}, \rho)$, it can learn the private index x being retrieved. With $M = \mathcal{O}(\sqrt{N} \log N)$ hints, there is only one hint h that has the offline query $\mathcal{S}$ overlapping at least $\sqrt{N} - 1$ (out of $\sqrt{N}$) indices with the online query $\mathcal{S} \setminus \{x\}$. We call such a unique intersection between an online set and its offline set as the *parity set*.

To conceal the linkability caused by the parity set, the server-side methods [10, 11, 12, 17] must preprocess entire hint table remotely with an expensive per-client super-linear FHE circuit evaluation of $\mathcal{O}(N \log N)$ over the full database. The client-side methods [13, 14, 15, 16] can avoid FHE but require the client to stream the whole database to preprocess the hint table locally, which incurs bandwidth $\mathcal{O}(N)$ database entries. In both cases, the preprocessing cannot be efficiently done per hint as the same expensive cost remains when hiding any $\sqrt{N}$ indices on domain N . In addition, one preprocessing round only support $\mathcal{O}(\sqrt{N} \log N)$ online queries since each hint is single-use. To support unlimited queries, all existing single-server OO-PIR schemes must periodically rebuild the entire hint table via the linear-cost preprocessing after only a sublinear number of queries, leading to an unfavorable trade-off between server computation and client bandwidth.

Idea 1: Private Parity Set from Random Set Intersection. To conceal the linkability caused by the unique parity set, we take a different route by ensuring that the distribution of offline query $\mathcal{S}$ and online query $\mathcal{S}'$ are indistinguishable from random. Our core idea is to reconcile two independent query sets $\mathcal{S}$ and $\mathcal{S}'$ to represent a single parity set $\mathcal{A}$ such

that revealing the offline query $\mathcal{S}$ allows us to preprocess a private hint parity ρ over set $\mathcal{A}$, while revealing the online query $\mathcal{S}'$ yields a private punctured parity $\hat{\rho}$ over set $\mathcal{A} \setminus \{x\}$ for the online recovery of $\text{DB}[x] = \rho - \hat{\rho}$.

This relies on a simple overlap property: two random sets of m indices $\{1, \dots, N\}$ almost surely overlap once m is large relative to N . This is because the probability two sets stay disjoint can be approximated by $(1 - m/N)^m \approx e^{-m^2/N}$, super-polynomially small once $m^2 = \Theta(N^{1+\alpha}) \forall \alpha \in (0, 1)$. We exploit this overlap property to define a parity set as the intersection of two random sets $\mathcal{S}$ and $\mathcal{S}'$ that will be used as offline and online queries, respectively. As the joint view of online and offline sets are identically distributed, when an online set $\mathcal{S}'$ is revealed, it cannot be linked to any offline set $\mathcal{S}$ to reveal the parity set $\mathcal{A}$. Also, the parity set size $|\mathcal{A}|$ and the query set size $m = |\mathcal{S}| = |\mathcal{S}'|$ can be tunable (sublinearly to N) so the hint table covers all database entries.

Given a private parity set $\mathcal{A}$, the next step is to privately compute the offline parity ρ and the online punctured parity $\hat{\rho}$. This can be realized with Linearly Homomorphic Encryption (LHE) [50, 51, 52, 53, 54], especially Regev-based preprocessing LHE [5, 50], which supports efficient private evaluation of linear sums. However, directly applying Regev-based LHE to the OO-PIR model faces a fundamental mismatch: its preprocessing requires the data vector on which client queries are evaluated to be fixed across queries (as in standard PIR model where every query is evaluated over the same full database). In OO-PIR model, by contrast, each client query is evaluated over a fresh random sub-database determined by the query set, and this sub-database changes dynamically from one query to another, which makes Regev-based preprocessing incompatible with random set sampling.

Idea 2: Random Set Intersection on Preprocessed Matrix. Our key idea is to confine the randomness of sampling to a fixed, structured domain on which Regev preprocessing remains valid. If the database DB is organized as a matrix, we can sample a random row as the offline set $\mathcal{S}$ and a random column as the online set $\mathcal{S}'$ (or vice versa). Unlike an arbitrary subset of entries, a row or a column is fixed, and the randomness per query relies on which row and column a hint selects, so each row and column is preprocessed once and reused across all queries. Since a row and a column of a matrix share exactly one entry, $\mathcal{S}$ and $\mathcal{S}'$ intersect in a single, uniformly random entry, defining the parity set $\mathcal{A}$.

A single matrix, however, fixes the intersection to one entry, whereas the parity set size must be tunable for the hint table to cover the entire database entries. Thus, we partition the database into n independent matrices, and for each partition, the offline set $\mathcal{S}$ draws one row and the online set $\mathcal{S}'$ draws one column. Every partition then contributes one shared entry, and the n entries form the parity set $\mathcal{A}$. The partition count n thus controls its size, while every sampled row and column stays preprocessable. As the partitions are independent Regev-based LHE instances, linearity sums their per-partition sub-parities into a single parity over $\mathcal{A}$.

Putting All Together. With Idea 1, the private parity set can help refresh consumed hints on the fly, supporting unlimited

queries without periodic rebuilding. Idea 2 places those sets on a preprocessed matrix, retrieving each parity in sublinear time at Regev-based LHE throughput. Together, we obtain a single-server OO-PIR approaching two-server efficiency without non-colluding assumption or periodic preprocessing.

2. Preliminaries

Notation. We denote $[N]$ as the set $\{1, \dots, N\}$, λ as a security parameter and $\text{negl}(\cdot)$ as a negligible function. PPT refers to Probabilistic Polynomial Time. We denote $x \xleftarrow{\$} [N]$ as sampling x uniformly at random from the set $[N]$. For a distribution $\mathcal{D}$, we denote $x \xleftarrow{\$} \mathcal{D}$ as a random sample of $\mathcal{D}$. We denote $\mathbb{Z}_q$ as the set of integers modulo q . We denote the inner product between two matrices $\mathbf{A}, \mathbf{B}$ as $\mathbf{A} \cdot \mathbf{B}$.

2.1. Linearly Homomorphic Encryption (LHE)

LHE permits a plaintext $\mathbf{v}$ to be encrypted such that a linear function f_θ can be homomorphically evaluated on the ciphertext and decryption yields $f_\theta(\mathbf{v})$. We define a linear function $f_\theta(\mathbf{v}) := \langle \theta, \mathbf{v} \rangle = \sum_{i=1}^m \theta_i \nu_i$ as the inner product of an input $\mathbf{v} = (\nu_1, \dots, \nu_m)$ and a vector $\theta = (\theta_1, \dots, \theta_m)$.

Definition 1 (LHE). An IND-CPA secure LHE scheme is a tuple of PPT algorithms $\text{LHE} = (\text{Setup}, \text{Enc}, \text{Eval}, \text{Dec})$:

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$: Given a security parameter λ , it outputs public parameters pp as implicit input to other algorithms.
- $\mathbf{c} \leftarrow \text{Enc}(\mathbf{k}, \mathbf{v})$: Given a secret $\mathbf{k} \in \mathcal{K}$ and a plaintext vector $\mathbf{v}$, it outputs a ciphertext vector $\mathbf{c}$.
- $\text{ct} \leftarrow \text{Eval}(\theta, \mathbf{c})$: Given a vector θ specifying function f_θ and a ciphertext $\mathbf{c}$, it outputs a ciphertext ct encrypting $f_\theta(\mathbf{v})$ by homomorphically evaluating $f_\theta(\mathbf{c})$.
- $v \leftarrow \text{Dec}(\mathbf{k}, \text{ct})$: Given a secret $\mathbf{k}$ and a ciphertext ct , it outputs a decrypted value $v = f_\theta(\mathbf{v})$.

Regev-based Preprocessing LHE. Proposed by Henzinger et. al. [5], Regev-based PLHE can precompute a function hint $\mathbf{d}_\theta$ for a linear function f_θ to reduce the cost of multiple evaluations. PLHE is based on LWE assumption, involving a public matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times \eta}$ of m LWE samples $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^\eta$, a secret space $\mathcal{K} \leftarrow \mathbb{Z}_q^\eta$ with secret dimension η , and an error space $\chi \sim D_{\mathbb{Z}, \sigma}$ with standard deviation σ .

Definition 2 (Regev-based PLHE). A Regev-based PLHE constructed from LWE parameters (η, m, q, χ) is a tuple of PPT algorithms $\text{PLHE} = (\text{Setup}, \text{Prep}, \text{Enc}, \text{Eval}, \text{Dec})$:

- $\mathbf{A} \leftarrow \text{Setup}(1^\lambda)$: Given a security parameter λ , it outputs public matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times \eta}$.
- $\mathbf{d}_\theta \leftarrow \text{Prep}(\theta)$: Given a function $f_\theta := \langle \theta, \cdot \rangle$, it outputs a function hint $\mathbf{d}_\theta \leftarrow \theta \cdot \mathbf{A}$.
- $\mathbf{c} \leftarrow \text{Enc}(\mathbf{A}, \mathbf{k}, \mathbf{v})$: Given a matrix $\mathbf{A}$, a secret $\mathbf{k} \xleftarrow{\$} \mathcal{K}$, a plaintext $\mathbf{v} \in \mathbb{Z}_p^m$, it outputs a ciphertext $\mathbf{c} \leftarrow \mathbf{A} \cdot \mathbf{k} + \mathbf{e} + \Delta \cdot \mathbf{v}$ using an error $\mathbf{e} \xleftarrow{\$} \chi^m$ and a scalar $\Delta = \lfloor q/p \rfloor$.
- $\text{ct} \leftarrow \text{Eval}(\theta, \mathbf{c})$: Given a vector $\theta \in \mathbb{Z}_p^m$ specifying function f_θ and a ciphertext $\mathbf{c}$, it outputs a ciphertext $\text{ct} \leftarrow f_\theta(\mathbf{c})$.
- $v \leftarrow \text{Dec}(\mathbf{d}_\theta, \mathbf{k}, \text{ct})$: Given a function hint $\mathbf{d}_\theta$, a secret $\mathbf{k}$, and a ciphertext ct , it outputs $v \leftarrow \lfloor \text{ct} - \mathbf{d}_\theta \cdot \mathbf{k} \rfloor_\Delta / \Delta$.

Definition 3 (Regev-based PLHE Correctness). A Regev-based PLHE with LWE parameters (η, m, q, χ) is correct if for all $\mathbf{v} \in \mathbb{Z}_p^m$, there is a negligible δ :

$$\Pr \left[v = f_\theta(\mathbf{v}) \mid \begin{array}{l} \mathbf{A} \leftarrow \text{Setup}(1^\lambda) \\ \mathbf{d}_\theta \leftarrow \text{Prep}(\theta) \\ \mathbf{c} \leftarrow \text{Enc}(\mathbf{A}, \mathbf{k}, \mathbf{v}) \\ \text{ct} \leftarrow \text{Eval}(\theta, \mathbf{c}) \\ v \leftarrow \text{Dec}(\mathbf{d}_\theta, \mathbf{k}, \text{ct}) \end{array} \right] \geq 1 - \delta$$

Definition 4 (Regev-based PLHE Security). A Regev-based PLHE with LWE parameters (η, m, q, χ) is (T, ϵ) -secure if every adversary with time T has advantage at most ϵ in distinguishing LWE output from a uniform sample $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^m$:

$$\{(\mathbf{A}, \mathbf{A}\mathbf{k} + \mathbf{e})\} \approx \{(\mathbf{A}, \mathbf{r})\}$$

Remark. By Definition 3 and Definition 4, there exists a secure Regev-based PLHE [5] with LWE parameters $(\eta, m, q, \chi \sim D_{\mathbb{Z}, \sigma})$ and plaintext modulus p when:

$$\lfloor q/p \rfloor \geq \sqrt{2}\sigma p \cdot \sqrt{m} \cdot \sqrt{\ln(2/\delta)}$$

3. Models

System Model. We consider a server with a public database DB of N entries of size $|\beta|$, queryable by any client. Each client maintains a hint table. We model this as an Online-Offline PIR (OO-PIR) scheme, as follows.

Definition 5 (OO-PIR). An OO-PIR scheme is a tuple of PPT algorithms $\text{OO-PIR} = (\text{Setup}, \text{Query}, \text{Refresh})$:

- $(\mathcal{H}, \text{pp}) \leftarrow \text{Setup}(1^\lambda, \text{DB}, N)$: Given a security parameter λ , an N -entry database DB, it outputs a hint table $\mathcal{H}$ and public parameters pp as implicit input to other algorithms.
- $\beta_x \leftarrow \text{Query}(x, \mathcal{H})$: Given an index x and a hint table $\mathcal{H}$, it outputs an entry $\beta_x = \text{DB}[x]$.
- $\mathcal{H}' \leftarrow \text{Refresh}(\beta_x, \mathcal{H})$: Given a hint table $\mathcal{H}$ that was used to query β_x , it outputs an updated hint table $\mathcal{H}'$.

Definition 6 (OO-PIR Correctness). An OO-PIR scheme is correct if given a database DB of N entries, a security parameter λ , $(\mathcal{H}, \text{pp}) \leftarrow \text{Setup}(1^\lambda, \text{DB}, N)$, and any number of prior executions, there exists $\text{negl}(\lambda)$ for all $x \in [N]$:

$$\Pr \left[\beta_x = \text{DB}[x] \mid \begin{array}{l} \beta_x \leftarrow \text{Query}(x, \mathcal{H}) \\ \mathcal{H}' \leftarrow \text{Refresh}(\beta_x, \mathcal{H}) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

Threat Model. We adopt the semi-honest model, where the client is trusted and the server is adversarial that follows the protocol faithfully but attempts to infer the client's query.

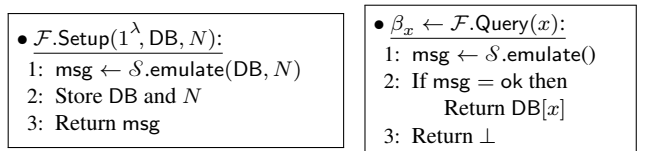


Figure 2: OO-PIR Ideal Functionality.

Security Model. We define OO-PIR security via Ideal/Real paradigm, such that an adversary $\mathcal{A}$ corrupting the server cannot learn the queried entry. Let $\mathcal{F}$ be the OO-PIR ideal functionality for database lookup (Figure 2). Let $\mathcal{S}$ be an ideal simulator that emulates real-world adversary views. Let $\mathcal{Z}$ be the environment that adaptively provides inputs for all entities based on adversary views and receives outputs.

- **Ideal:** In the offline phase, given the input (DB, N) from $\mathcal{Z}$, $\mathcal{F}$ notifies $\mathcal{S}$ about DB content. $\mathcal{S}$ emulates $\mathcal{A}$'s view for offline execution and replies to $\mathcal{F}$ with ok or $\perp$. For each online step t , given the index $x_t \in [N]$ from $\mathcal{Z}$, $\mathcal{F}$ notifies $\mathcal{S}$ about the event (but not x_t). $\mathcal{S}$ emulates $\mathcal{A}$'s view for online execution and replies to $\mathcal{F}$ with ok or $\perp$. If $\mathcal{S}$ says ok, $\mathcal{F}$ returns $\text{DB}[x_t]$ to $\mathcal{Z}$.
- **Real:** In the offline phase, $\mathcal{Z}$ provides (DB, N) to the server. The client honestly executes $(\mathcal{H}, \text{pp}) \leftarrow \text{Setup}(1^\lambda, \text{DB}, N)$ with the server to obtain hint table $\mathcal{H}$. For each online step t , given the index $x_t \in [N]$ from $\mathcal{Z}$, the client honestly executes $\beta_{x_t} \leftarrow \text{Query}(x_t, \mathcal{H})$ and $\mathcal{H}' \leftarrow \text{Refresh}(\beta_{x_t}, \mathcal{H})$ with the server. The client returns β_{x_t} to $\mathcal{Z}$.

Definition 7 (OO-PIR Security). A protocol $\Pi_{\mathcal{F}}$ securely realizes $\mathcal{F}$ if for every PPT real-world adversary $\mathcal{A}$, there is a PPT simulator $\mathcal{S}$, such that for all non-uniform PPT environment $\mathcal{Z}$:

$$|\Pr[\text{REAL}_{\Pi_{\mathcal{F}}, \mathcal{A}, \mathcal{Z}}(\lambda) = 1] - \Pr[\text{IDEAL}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}(\lambda) = 1]| \leq \text{negl}(\lambda)$$

4. ESCAPE with Generic LHE

In this section, we first describe the hint structure in our scheme. Then we present the offline and online phase of our scheme using generic LHE.

Database. The database DB is an array of N entries of size β . For simplicity, each entry labeled x_i is stored at $\text{DB}[x_i]$.

Pseudorandom Set (PRS). A PRS [45], constructed from a pseudorandom function (PRF) $F : \{0, 1\}^\lambda \times [m] \rightarrow [N]$ and a PRF key $\text{sk} \xleftarrow{\$} \{0, 1\}^\lambda$, is an algorithm $\mathcal{S} \leftarrow \text{Set}(\text{sk})$ that returns a set $\mathcal{S} = (s_1, \dots, s_m)$ uniformly from $[N]$:

$$\mathcal{S} \leftarrow \text{Set}(\text{sk}) := \{s_i \mid s_i = F_{\text{sk}}(i) \ \forall i \in [m]\}$$

Hint Structure. The client maintains a private hint table $\mathcal{H} = (h_1, \dots, h_M)$ consisting of M hints $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$, with two PRF keys $(\text{sk}_i, \text{sk}'_i)$ representing two PRS sets $(\mathcal{S}_i, \mathcal{S}'_i)$, respectively, and a parity sum $\rho_i = \sum_{a \in \mathcal{A}_i} \text{DB}[a]$, where $\mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}'_i$ is the parity set with $\mathbb{E}[|\mathcal{A}_i|] = n$.

4.1. Offline Phase

The construction consists of an offline and online phases such that, the hint table $\mathcal{H} = (h_1, \dots, h_M)$ is created in the offline phase and the client later uses it to privately retrieve some DB entries in the online phase. Under a single-server view, our strategy is to create $\mathcal{H}$ such that any pair of offline and online query sets that correspond to the same parity hint h_i must be independent and uniformly distributed.

Figure 3 shows how our offline phase works. In short, the client creates each hint h_i by revealing m random

```

•  $(\mathcal{H}, \text{pp}) \leftarrow \text{Setup}(1^\lambda, \text{DB}, N)$ :
0: Let  $\text{pp} \leftarrow \text{LHE.Setup}(1^\lambda)$ 
1: For each  $i \in [M]$  do
2:   Let  $\text{sk}_i, \text{sk}'_i \xleftarrow{\$} \{0, 1\}^\lambda$ . Let  $\mathcal{A}_i \leftarrow \text{Set}(\text{sk}_i) \cap \text{Set}(\text{sk}'_i)$ 
3:   Let  $(s_1, \dots, s_m) \leftarrow \text{Set}(\text{sk}_i)$  be the set of random DB indices
4:   Let  $\mu_i \leftarrow (\mu_1, \dots, \mu_m)$  with  $\mu_j = \mathbb{1}_{[s_j \in \mathcal{A}_i]} \ \forall j \in [m]$ 
5:   Let  $\mathbf{k} \xleftarrow{\$} \mathcal{K}$  and  $\mathbf{c}_i \leftarrow \text{LHE.Enc}(\mathbf{k}, \mu_i)$ 
6:    $\gamma_i \leftarrow \text{LHE.Eval}(\theta = (\text{DB}[s_1], \dots, \text{DB}[s_m]), \mathbf{c}_i) \triangleright \text{Server}$ 
7:   Let  $h_i \leftarrow (\text{sk}_i, \text{sk}'_i, \rho_i)$  with parity  $\rho_i \leftarrow \text{LHE.Dec}(\mathbf{k}, \gamma_i)$ 
8: Return  $\mathcal{H} \leftarrow (h_1, \dots, h_M)$  and  $\text{pp}$ 

```

Figure 3: ESCAPE with Generic LHE: Offline Phase.

DB indices, and ask the server to compute a sum ρ_i of $n < m$ random entries from the m indicated DB entries. Let $(s_1, \dots, s_m) \leftarrow \mathcal{S}_i$ be the set of random DB indices sampled by a PRF key sk_i (line 3). The client creates a bit vector $\mu_i = (\mu_1, \dots, \mu_m)$ with each bit $\mu_j = 1$ if $\text{DB}[s_j]$ is chosen as part of the sum ρ_i for all $j \in [m]$ (line 4). Thus, vector μ_i has n active bits, and the remaining bits are zero. To hide which n indices are selected from the set $\mathcal{S}_i$, the client encrypts the bit vector μ_i with LHE to produce a ciphertext vector $\mathbf{c}_i = \text{LHE.Enc}(\mathbf{k}, \mu_i)$ (line 5). The server homomorphically evaluates the linear function f_θ , specified by $\theta = (\text{DB}[s_1], \dots, \text{DB}[s_m])$, on the ciphertext $\mathbf{c}_i$ (line 6) to produce $\gamma_i = f_\theta(\mathbf{c}_i)$, which can be decrypted to the plaintext parity ρ_i by LHE correctness.

To identify the indices $(a_1, \dots, a_n)$ of the offline set $\mathcal{S}_i = (s_1, \dots, s_m)$ for creating the bit vector μ_i , the client samples another uniformly random online set $\mathcal{S}'_i = (s'_1, \dots, s'_m)$ to pair with $\mathcal{S}_i$ and computes the set intersection $\mathcal{A}_i = (a_1, \dots, a_n)$ (line 2). Since $\mathcal{A}_i$ exists in both sets $\mathcal{S}_i$ and $\mathcal{S}'_i$, the client uses $\mathcal{A}_i$ as the hint parity set and retrieves the corresponding parity ρ_i by sending an encrypted bit vector μ_i that privately selects the indices $(a_1, \dots, a_n)$ from $\mathcal{S}_i$. Thus, the set $\mathcal{A}_i$ is described by two sets $\mathcal{S}_i$ and $\mathcal{S}'_i$, sampled by random keys sk_i and sk'_i , respectively. We analyze set size $|\mathcal{A}_i|$ in §A.

4.2. Online Phase

In the online phase, the client can privately recover an entry $\text{DB}[x]$ using a hint $h_i \in \mathcal{H}$ whose parity ρ_i contains $\text{DB}[x]$. Figure 4 shows how the client generates a retrieval query for recovery. Figure 5 shows how to refresh hint h_i to maintain security for future queries.

Hint Search. The client finds hint $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$ that covers $\text{DB}[x]$ by checking if index x exists in the intersection $\mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}'_i$, with $\mathcal{S}_i \leftarrow \text{Set}(\text{sk}_i)$ and $\mathcal{S}'_i \leftarrow \text{Set}(\text{sk}'_i)$. We know $x \in \mathcal{A}_i$ if $x \in \mathcal{S}_i$ and $x \in \mathcal{S}'_i$. To recover $\text{DB}[x]$ via the parity sum $\rho_i = \sum_{a \in \mathcal{A}_i} \text{DB}[a]$, the client needs a punctured parity $\hat{\rho}'$ such that $\text{DB}[x] = \rho_i - \hat{\rho}'$. The correctness holds as long as the client can find a hint where $x \in \mathcal{A}_i$.

Theorem 1 (Correctness). With the client hint table $\mathcal{H} = (h_1, \dots, h_M)$ where h_i describes $\mathcal{A}_i$ of size $\mathbb{E}[|\mathcal{A}_i|] = n$, our ESCAPE with generic LHE satisfies Definition 6 by setting the hint table size $M = \mathcal{O}(N/n(\lambda + \ln N))$.

• $\beta_x \leftarrow \text{Query}(x, \mathcal{H})$:

- 1: Parse $\mathcal{H} = (h_1, \dots, h_M)$
- 2: Select hint $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$ where $x \in \text{Set}(\text{sk}_i) \cap \text{Set}(\text{sk}'_i)$
- 3: Let $\mathcal{A}_i \leftarrow \text{Set}(\text{sk}_i) \cap \text{Set}(\text{sk}'_i)$. Let $\mathcal{S}'_i \leftarrow \text{Set}(\text{sk}'_i)$.
- 4: Let $b \xleftarrow{\$} \text{Bernoulli}(m/N)$. If $b = 1$ then $z \leftarrow x$ else $z \xleftarrow{\$} [N] \setminus \mathcal{S}'_i$
- 5: Let $(s'_1, \dots, s'_m) \leftarrow \mathcal{S}'_i \setminus \{x\} \cup \{z\}$ be the set of indices
- 6: Let $\mu' \leftarrow (\mu'_1, \dots, \mu'_m)$ with $\mu'_j = \mathbb{1}[s'_j \in \mathcal{A}_i \setminus \{z\}] \ \forall j \in [m]$
- 7: Compute $\mathbf{c}' \leftarrow \text{LHE.Enc}(\mathbf{k}, \mu')$ with $\mathbf{k} \xleftarrow{\$} \mathcal{K}$
- 8: Compute $\gamma' \leftarrow \text{LHE.Eval}(\theta = (\text{DB}[s'_1], \dots, \text{DB}[s'_m]), \mathbf{c}') \triangleright \text{Server}$
- 9: Return $\beta_x \leftarrow \rho_i - \hat{\rho}'$ with $\hat{\rho}' \leftarrow \text{LHE.Dec}(\mathbf{k}, \gamma')$

Figure 4: ESCAPE with Generic LHE: Online Query.

• $\mathcal{H}' \leftarrow \text{Refresh}(\beta_x, \mathcal{H})$:

- 1: Parse $\mathcal{H} = (h_1, \dots, h_M)$. Let $h_i \in \mathcal{H}$ be the used hint.
- 2: Let $\text{sk}, \text{sk}' \xleftarrow{\$} \{0, 1\}^\lambda$ where $x \in \text{Set}(\text{sk}) \cap \text{Set}(\text{sk}')$
- 3: Let $\mathcal{A} \leftarrow \text{Set}(\text{sk}) \cap \text{Set}(\text{sk}')$. Let $\mathcal{S} \leftarrow \text{Set}(\text{sk})$.
- 4: Let $b \xleftarrow{\$} \text{Bernoulli}(m/N)$. If $b = 1$ then $z \leftarrow x$ else $z \xleftarrow{\$} [N] \setminus \mathcal{S}$
- 5: Let $(s_1, \dots, s_m) \leftarrow \mathcal{S} \setminus \{x\} \cup \{z\}$ be the set of indices
- 6: Let $\mu \leftarrow (\mu_1, \dots, \mu_m)$ with $\mu_j = \mathbb{1}[s_j \in \mathcal{A} \setminus \{z\}] \ \forall j \in [m]$
- 7: Compute $\mathbf{c} \leftarrow \text{LHE.Enc}(\mathbf{k}, \mu)$ with $\mathbf{k} \xleftarrow{\$} \mathcal{K}$
- 8: Compute $\gamma \leftarrow \text{LHE.Eval}(\theta = (\text{DB}[s_1], \dots, \text{DB}[s_m]), \mathbf{c}) \triangleright \text{Server}$
- 9: Let $h \leftarrow (\text{sk}, \text{sk}', \hat{\rho} + \beta_x)$ with $\hat{\rho} \leftarrow \text{LHE.Dec}(\mathbf{k}, \gamma)$
- 10: Return $\mathcal{H}' \leftarrow \mathcal{H}_{h_i \leftarrow h}$

Figure 5: ESCAPE with Generic LHE: Hint Refresh.

Proof. Corrigan-Gibbs and Kogan [45] proved that for an index $x \in [N]$ to be included in a set with probability $\mathcal{R}$, we can define the number of hint sets as $M = \mathcal{O}(1/\mathcal{R}(\lambda + \ln N))$ and achieve negligible error probability (no set covers x). With $\mathbb{E}[|\mathcal{A}_i|] = n$, we have $\Pr[x \in \mathcal{A}_i] = n/N$. Thus, setting hint size $M = \mathcal{O}(N/n(\lambda + \ln N))$ satisfies Definition 6. $\square$

Online Query. To retrieve the punctured parity $\hat{\rho}'$ of $\mathcal{A}_i \setminus \{x\}$ that aligns with the parity ρ_i of $\mathcal{A}_i$ (for $\text{DB}[x]$ recovery), the client creates a retrieval query via the remaining set $\mathcal{S}'_i \supset \mathcal{A}_i$. Since $\mathcal{S}'_i$ is independent of $\mathcal{S}_i$ (and others $\mathcal{S}_j \neq \mathcal{S}_i$) from the offline phase, we can create a query out of $\mathcal{S}'_i$ that privately indicates $\mathcal{A}_i \setminus \{x\}$ via an encrypted bit vector to obtain $\hat{\rho}'$.

The query only needs to ensure that the appearance of $x \in \mathcal{S}'_i$ is uniformly random, such that on receiving $\mathcal{S}'_i$, the server cannot distinguish if $x \in \mathcal{S}'_i$ or $x \in [N] \setminus \mathcal{S}'_i$. For any random set of size m , any index appears with probability m/N . Thus, with $1 - m/N$ probability, the client replaces x with another random index z from $[N] \setminus \mathcal{S}'_i$ (See §A).

In Figure 4, the client performs a Bernoulli trial (line 4) to see if index $x \in \mathcal{S}'_i \leftarrow \text{Set}(\text{sk}'_i)$ should be replaced or not. With the modified set $(s'_1, \dots, s'_m) \leftarrow \mathcal{S}'_i \setminus \{x\} \cup \{z\}$ (line 5), the client only wants the punctured parity $\hat{\rho}'$ corresponding to $\mathcal{A}_i \setminus \{z\}$. The client creates a bit vector $\mu' = (\mu'_1, \dots, \mu'_m)$ with each bit $\mu'_j = 1$ if the client chooses $\text{DB}[s'_j]$ as part of $\hat{\rho}'$ for all $j \in [m]$ (line 6). The remaining steps are to use LHE to retrieve $\hat{\rho}'$ for recovering $\text{DB}[x] = \rho_i - \hat{\rho}'$.

Hint Refresh. After consuming the hint $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$ for the online query set $\mathcal{S}'_i \setminus \{x\} \cup \{z\}$, we must refresh h_i . Reusing h_i to recover x would indicate same access

pattern as the server observes an almost identical query $\mathcal{S}'_i \setminus \{x\} \cup \{z'\}$. Reusing h_i for another x' would leak both x and x' due to set difference between $\mathcal{S}'_i \setminus \{x\} \cup \{z\}$ and $\mathcal{S}'_i \setminus \{x'\} \cup \{z'\}$. Note that these problems still exist if we choose another $\mathcal{S}'_i$ that contains the same intersection set $\mathcal{A}_i$ (i.e., trying to reuse the parity ρ_i). Thus, the client must discard h_i entirely. Since h_i was consumed and is subject to containing $\text{DB}[x]$, the client needs a new hint that also includes $\text{DB}[x]$, to maintain the hint table distribution.

In Figure 5, the client creates a new hint that covers $\text{DB}[x]$ by sampling two new random sets $\mathcal{S} \leftarrow \text{Set}(\text{sk})$ and $\mathcal{S}' \leftarrow \text{Set}(\text{sk}')$ from two new keys (sk, sk') with $x \in \mathcal{S} \cap \mathcal{S}'$ (line 2). With $\mathcal{A} \leftarrow \mathcal{S} \cap \mathcal{S}'$ as the new set intersection (line 3), the client needs a punctured parity $\hat{\rho}$ corresponding to $\mathcal{A} \setminus \{x\}$, for computing the parity $\hat{\rho} + \text{DB}[x]$ of the set $\mathcal{A}$. To make use of $\mathcal{S}$ for retrieving $\hat{\rho}$, the client performs a Bernoulli trial to replace x (lines 4 to 5), and then creates a corresponding bit vector to retrieve $\hat{\rho}$ (lines 6 to 9), similar to the online query steps. The client replaces h_i with a new hint $h = (\text{sk}, \text{sk}', \hat{\rho} + \text{DB}[x])$.

Theorem 2 (Security). ESCAPE with generic LHE is secure by Definition 7.

Proof. Given a simulator $\mathcal{S}_{\text{LHE}}$ for LHE, we construct a simulator $\mathcal{S}$ (Figure 6) in IDEAL with distribution $\tilde{\mathcal{D}} = \{\mathcal{S} \subseteq [N] : |\mathcal{S}| = m\}$ such that $\mathcal{S}, \mathcal{S}', (\mathcal{S}_1, \dots, \mathcal{S}_M)$ are randomly independent sets of indices, and $\mathbf{r}, \mathbf{r}', (\mathbf{r}_1, \dots, \mathbf{r}_n)$ are random LHE ciphertexts.

• $\mathcal{S}.\text{Offline}(\text{DB})$: 1: $(\mathcal{S}_1, \dots, \mathcal{S}_M) \xleftarrow{\\$} \tilde{\mathcal{D}}^M$ 2: $Q_i = (\mathcal{S}_i, \mathbf{r}_i), \mathbf{r}_i \xleftarrow{\\$} \mathcal{S}_{\text{LHE}}$ 3: Output $(Q_1, \dots, Q_M)$	• $\mathcal{S}.\text{Online}(\text{DB})$: 1: $\mathcal{S}', \mathcal{S} \xleftarrow{\\$} \tilde{\mathcal{D}}$ and $\mathbf{r}', \mathbf{r} \xleftarrow{\\$} \mathcal{S}_{\text{LHE}}$ 2: Output $Q' = (\mathcal{S}', \mathbf{r}')$ 3: Output $Q = (\mathcal{S}, \mathbf{r})$
--	--

Figure 6: The Simulator $\mathcal{S}$ for ESCAPE under Generic LHE.

We define a sequence of hybrid games Hyb_i , where each Hyb_i replaces some protocol transcripts with the simulator outputs, and show that Real and Ideal are indistinguishable:

$$|\Pr[\text{REAL}_{\Pi_{\mathcal{F}}, \mathcal{A}, \mathcal{Z}}(\lambda) = 1] - \Pr[\text{IDEAL}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}(\lambda) = 1]| \leq \text{negl}(\lambda)$$

See Appendix §C for the detailed hybrid arguments. $\square$

4.3. Asymptotic Analysis

Assuming an LHE scheme, which encrypts each selection bit with $\mathcal{O}_\lambda(1)$ -size ciphertext, we analyze the efficiency cost with N as the number of DB entries (of size β), m as the size of query sets, $n = m^2/N$ as the hint set intersection size, and $M = \mathcal{O}(N/n(\lambda + \ln N))$ as the number of hints.

- **Offline Bandwidth:** The client sends $\mathcal{O}(M)$ PRF keys and $\mathcal{O}(mM)$ ciphertexts, and receives $\mathcal{O}(\beta M)$ ciphertexts.
- **Offline Computation:** The server executes $\mathcal{O}(mM)$ PRF and $\mathcal{O}_\lambda(\beta mM)$ LHE evaluation. The client executes $\mathcal{O}(M)$ PRF key generation, $\mathcal{O}_\lambda(mM)$ LHE encryption, and $\mathcal{O}_\lambda(\beta M)$ LHE decryption.

- **Online Bandwidth:** The client sends $\mathcal{O}(m)$ indices and $\mathcal{O}(m)$ ciphertexts, receives $\mathcal{O}(\beta)$ ciphertexts.
- **Online Computation:** The client executes $\mathcal{O}(M)$ PRF, $\mathcal{O}_\lambda(m)$ LHE encryption, $\mathcal{O}_\lambda(\beta)$ LHE decryption. The server executes $\mathcal{O}_\lambda(\beta m)$ LHE evaluation.

5. Efficient ESCAPE with Regev-based PLHE

Most existing LHE schemes [51, 52, 53, 54] either have (1) large ciphertext expansion for security, or (2) heavy client decryption due to discrete-log solver, or (3) expensive homomorphic evaluation due to ciphertext rotations. To gain efficiency, Regev-based PLHE [5] was proposed, where the ciphertext $\mathbf{c} = (\mathbf{A}, \mathbf{b})$ has a public matrix $\mathbf{A}$ independent with the encrypted bit vector $\mathbf{b}$. As the homomorphic evaluation between $\mathbf{c}$ and $\mathbf{DB}$ is $\text{LHE.Eval}(\mathbf{DB}, \mathbf{c}) = (\mathbf{DB} \cdot \mathbf{A}, \mathbf{DB} \cdot \mathbf{b})$, we can preprocess $(\mathbf{DB} \cdot \mathbf{A})$ since $\mathbf{DB}$ is fixed. In our case, the evaluation (Figure 3 line 6, Figure 4 line 8) is performed on a random sub-database $\theta^* = (\mathbf{DB}[s_1], \dots, \mathbf{DB}[s_m])$ which is only known when creating query, so preprocessing of $\theta^* \cdot \mathbf{A}$ is not possible. We need to design a pseudorandom set sampling structure that is efficiently aligned with Regev-based PLHE.

5.1. Technical Challenges

In standard PIR with Regev-based PLHE, the query is evaluated with every $\sqrt{N}$ rows of the $\mathbf{DB}$ matrix $[\sqrt{N}] \times [\sqrt{N}]$ to satisfy PIR security. However, we observe that correctness holds even if the evaluation is on a *single* row containing x . With a simple extension, we can reuse PLHE to retrieve the sum of entries $(s_1, \dots, s_m)$ by revealing a subset of rows that includes them. In contrast to PIR, OO-PIR allows us to securely operate on a subset of entries per query.

The problem with this approach is that m random independent entries can scatter across $m/2$ rows with high probability. With $m = \sqrt{N}$, we operate on roughly half of the database and fall back to linear processing, which is not desirable in the OO-PIR model. If we choose all entries from a single row and sample multiple rows to construct the query sets, so that the processing complexity is sublinear, their set intersection is either all indices from that row or not. Indices correlated from the same row creates linkability with their offline set that, if not removed, can point out the row containing query index x and break security. Removing them all, however, breaks correctness as we cannot get the punctured parity we need.

5.2. New Hint Sampling Structure

To support random set intersection on a fixed, structured domain, we present a new database structure for set sampling as follows. Given a database $\mathbf{DB}$ of N entries, we divide $\mathbf{DB}$ into n partitions, each partition P_k is an $\eta \times \eta$ matrix covering η^2 entries with labels in range $[(k-1)\eta^2 + 1, \dots, k\eta^2]$ for $k \in [n]$. Thus, $\mathbf{DB}$ is a $t \times \eta$ matrix with $t = n \cdot \eta$ rows, η columns. We define $[m]_k = [(k-1)\eta + 1, \dots, k\eta]$ as the

row indices for partition P_k . We define a public transposed $\mathbf{DB}^\top$ where each $P_k \in \mathbf{DB}$ is transposed. Example 1 shows $(\mathbf{DB}, \mathbf{DB}^\top)$ as wide matrices, with vertical highlight indicate row access. Each cell is labeled by the entry label it stores. We denote $\mathbf{DB}[i, *]$ as the entries located at row (i) . For convenience, we also denote $\mathbf{DB}(i)$ for the set of entry labels at row (i) . We use the same notations for $\mathbf{DB}^\top$.

1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
1	4	7	10	13	16	19	22	25	1	2	3	10	11	12	19	20	21
2	5	8	11	14	17	20	23	26	4	5	6	13	14	15	22	23	24
3	6	9	12	15	18	21	24	27	7	8	9	16	17	18	25	26	27
P_1			P_2			P_3			$P_1^\top$			$P_2^\top$			$P_3^\top$		
DB with $\text{DB}(2) = (4, 5, 6)$									DB $^\top$ with $\text{DB}^\top(5) = (11, 14, 17)$								

Example 1: $(\mathbf{DB}, \mathbf{DB}^\top)$ with $n = 3$ partitions, $\eta = 3$ columns, $t = 9$ rows. We highlight the labels from $\mathbf{DB}(2)$, $\mathbf{DB}^\top(5)$.

Given an entry labeled $x \in [N]$, let $\phi : [N] \rightarrow [t] \times [m]$ be the mapping for coordinates (u, v) of the entry in $(\mathbf{DB}, \mathbf{DB}^\top)$. The below formula help us formally convert between an entry label and its coordinate via base- η expansion of x :

$$x = a \cdot \eta^2 + b \cdot \eta + c + 1$$

$$\phi(x) = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \cdot \eta + b + 1 \\ c + 1 \end{bmatrix} \quad \phi^\top(x) = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \cdot \eta + c + 1 \\ b + 1 \end{bmatrix}$$

By designing the matrix pairs $(\mathbf{DB}, \mathbf{DB}^\top)$, each row $r_j \in P_k^\top$ is a distinct column from P_k due to transposition. In a matrix, a row and a column share only one entry:

Row Pairing Property. For each pair of rows (r_i, r_j) from a partition pair $(P_k, P_k^\top) \forall k \in [n]$, it follows that:

$$|\mathbf{DB}(r_i) \cap \mathbf{DB}^\top(r_j)| = 1 \quad \forall (r_i, r_j) \in P_k \times P_k^\top$$

PRS on Matrix. With the above property, we can uniformly sample a label $x \in P_k$ by computing the intersection of two random rows from P_k and $P_k^\top$. Thus, a set of random labels can be defined by two sets of random rows from $(\mathbf{DB}, \mathbf{DB}^\top)$. To sample a set of entry labels on n random rows $\mathcal{R} = (r_1, \dots, r_n)$ with respect to $\mathbf{DB}$ or $\mathbf{DB}^\top$, we adjust the PRS notation $\mathcal{S} \leftarrow \text{Set}(\text{sk})$ for our proposed matrix database structure as follows:

$$\begin{aligned} \mathcal{F} &: \{0, 1\}^\lambda \times [n] \rightarrow [m] \\ \mathcal{R} &\leftarrow \text{Row}(\text{sk}) := \{(k-1)\eta + \mathcal{F}_{\text{sk}}(k) \mid \forall k \in [n]\} \\ \mathcal{S} &\leftarrow \text{Set}(\text{sk}) := \bigcup_{r \in \text{Row}(\text{sk})} \mathbf{DB}(r) \\ \mathcal{S}' &\leftarrow \text{TSet}(\text{sk}') := \bigcup_{r \in \text{Row}(\text{sk}')} \mathbf{DB}^\top(r) \end{aligned}$$

Set Intersection. Given a PRF key pair (sk, sk') describing $(\mathcal{S}, \mathcal{S}')$, we show how to compute $\mathcal{A} \leftarrow \text{Set}(\text{sk}) \cap \text{TSet}(\text{sk}')$:

$$\begin{aligned} (r_1, \dots, r_n) &\leftarrow \text{Row}(\text{sk}) \quad \wedge \quad (r'_1, \dots, r'_n) \leftarrow \text{Row}(\text{sk}') \\ \mathcal{A} &\leftarrow \{a_k \mid a_k = \mathbf{DB}(r_k) \cap \mathbf{DB}^\top(r'_k) \mid \forall k \in [n]\} \end{aligned}$$

Hint Structure. The client maintains Regev-based PLHE public parameters and function hints $\text{pp} = (\bar{\mathbf{A}}, \mathbf{D}, \bar{\mathbf{A}}^\top, \mathbf{D}^\top)$ for $(\mathbf{DB}, \mathbf{DB}^\top)$, with a hint table $\mathcal{H} = (h_1, \dots, h_M)$. Each hint $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$ describes a parity set $\mathcal{A}_i = (a_1, \dots, a_n)$ of n random labels and a parity sum $\rho_i = \sum_{a \in \mathcal{A}_i} \mathbf{DB}[\phi(a)]$.

• $(\mathcal{H}, \text{pp}) \leftarrow \text{Setup}(1^\lambda, \text{DB}, N)$:

- 1: For each $r \in [t]$ do
- 2: $\mathbf{A}_r \leftarrow \text{PLHE.Setup}(1^\lambda)$, $\mathbf{d}_r \leftarrow \text{PLHE.Prepare}(\text{DB}[r, *]) \triangleright \text{Server}$
- 3: $\mathbf{A}'_r \leftarrow \text{PLHE.Setup}(1^\lambda)$, $\mathbf{d}'_r \leftarrow \text{PLHE.Prepare}(\text{DB}^\top[r, *]) \triangleright \text{Server}$
- 4: Let $\mathbf{D} \leftarrow (\mathbf{d}_1, \dots, \mathbf{d}_t)$, $\bar{\mathbf{A}} \leftarrow (\mathbf{A}_1, \dots, \mathbf{A}_t)$
- 5: Let $\mathbf{D}^\top \leftarrow (\mathbf{d}'_1, \dots, \mathbf{d}'_t)$, $\bar{\mathbf{A}}^\top \leftarrow (\mathbf{A}'_1, \dots, \mathbf{A}'_t)$
- 6: For each $i \in [M]$ do
- 7: Let $\text{sk}_i, \text{sk}'_i \xleftarrow{\$} \{0, 1\}^\lambda$. Let $\mathcal{A}_i \leftarrow \text{Set}(\text{sk}_i) \cap \text{TSet}(\text{sk}'_i)$
- 8: Let query $Q_i \leftarrow \emptyset$, secret key $\mathbf{k}_i \xleftarrow{\$} \mathcal{K}$.
- 9: For each row $r \in \text{Row}(\text{sk}_i)$ do
- 10: Let $(s_1, \dots, s_m) \leftarrow \text{DB}(r)$ be the set of labels
- 11: Let $\boldsymbol{\mu} \leftarrow (\mu_1, \dots, \mu_m)$ with $\mu_j = \mathbb{1}[s_j \in \mathcal{A}_i] \forall j \in [m]$
- 12: Let $Q_i \leftarrow Q_i \cup \{(r, \mathbf{c})\}$ with $\mathbf{c} \leftarrow \text{PLHE.Enc}(\mathbf{A}_r, \mathbf{k}_i, \boldsymbol{\mu})$
- 13: Compute $\gamma_i \leftarrow \sum_{(r, \mathbf{c}) \in Q_i} \text{PLHE.Eval}(\text{DB}[r, *], \mathbf{c}) \triangleright \text{Server}$
- 14: Compute $\rho_i \leftarrow \text{PLHE.Dec}(\mathbf{d}^*, \mathbf{k}_i, \gamma_i)$ with $\mathbf{d}^* \leftarrow \sum_{r \in \mathcal{R}} \mathbf{d}_r$
- 15: Compute $h_i \leftarrow (\text{sk}_i, \text{sk}'_i, \rho_i)$
- 16: Return $\mathcal{H} \leftarrow (h_1, \dots, h_M)$ and $\text{pp} \leftarrow (\bar{\mathbf{A}}, \mathbf{D}, \bar{\mathbf{A}}^\top, \mathbf{D}^\top)$

Figure 7: Efficient ESCAPE with Regev PLHE: Offline Phase.

1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
1	4	7	10	13	16	19	22	25	1	2	3	10	11	12	19	20	21
2	5	8	11	14	17	20	23	26	4	5	6	13	14	15	22	23	24
3	6	9	12	15	18	21	24	27	7	8	9	16	17	18	25	26	27
P_1			P_2			P_3			$P_1^\top$			$P_2^\top$			$P_3^\top$		

DB rows (1, 4, 8) sampled by sk_i DB^T rows (2, 6, 7) sampled by sk'_i

Example 2: A hint h_i describes a set $\mathcal{A}_i = (2, 12, 22)$. Client reveals left highlighted rows (1, 4, 8) in the offline phase.

5.3. Offline Phase

Figure 7 illustrates how our offline phase works. We first precompute PLHE public parameters consisting of random matrices and function hints, so that a subset of rows from DB and DB^T can be used as linear functions for query evaluations (lines 1 to 5). Then, we compute offline hints using the idea of set intersection as described in our generic construction (lines 6 to 15). The main difference is how we sample each hint using the two (logical) matrix views of the database.

In short, we create each hint h_i by sampling an offline set $\mathcal{S}_i$ and another online set $\mathcal{S}'_i$, respectively from the matrices DB and DB^T. We want to compute the set intersection $\mathcal{A}_i$ of the two sets $\mathcal{S}_i \leftarrow \text{Set}(\text{sk}_i)$ and $\mathcal{S}'_i \leftarrow \text{TSet}(\text{sk}'_i)$ for hint h_i (line 7). Example 2 shows a set intersection using random independent rows from the two matrices.

For the remaining steps (Figure 7, line 8 to line 15), the client retrieves a parity ρ_i corresponding to the parity set $\mathcal{A}_i$. For each row r of the set $\mathcal{S}_i$ being revealed, the client creates a bit vector $\boldsymbol{\mu} = (\mu_1, \dots, \mu_m)$ with bit $\mu_j = 1$ if the entry $\text{DB}[\phi(s_j)]$ from the set of labels $(s_1, \dots, s_m) \leftarrow \text{DB}(r)$ is chosen by $\mathcal{A}_i$ as part of parity ρ_i for all $j \in [m]$ (line 11). Given a row vector of entries $\boldsymbol{\theta} = \text{DB}[r, *]$ specifying a linear function $f_{\boldsymbol{\theta}} := \langle \boldsymbol{\theta}, \cdot \rangle$, the client retrieves $f_{\boldsymbol{\theta}}(\boldsymbol{\mu})$ by sending ciphertext $\mathbf{c} \leftarrow \text{PLHE.Enc}(\mathbf{A}_r, \mathbf{k}_i, \boldsymbol{\mu})$ to the server (line 12) with $\mathbf{A}_r$ as the PLHE public matrix for $\text{DB}[r, *]$. The query

• $\beta_x \leftarrow \text{Query}(x, \mathcal{H})$:

- 1: Parse $\mathcal{H} = (h_1, \dots, h_M)$, $\text{pp} = (\bar{\mathbf{A}}, \mathbf{D}, \bar{\mathbf{A}}^\top, \mathbf{D}^\top)$
- 2: Parse $\bar{\mathbf{A}}^\top = (\mathbf{A}'_1, \dots, \mathbf{A}'_t)$, $\mathbf{D}^\top = (\mathbf{d}'_1, \dots, \mathbf{d}'_t)$. Let $\mathbf{k} \xleftarrow{\$} \mathcal{K}$.
- 3: Select $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$ where $x \in \text{Set}(\text{sk}_i) \cap \text{TSet}(\text{sk}'_i)$
- 4: Let $\mathcal{A}_i \leftarrow \text{Set}(\text{sk}_i) \cap \text{TSet}(\text{sk}'_i)$. Let query $Q' \leftarrow \emptyset$, $k = \lceil x/m^2 \rceil$
- 5: Let $\mathcal{R} \leftarrow \text{Row}(\text{sk}'_i)$. Select row $\tau \in \mathcal{R}$ where $x \in \text{DB}^\top(\tau)$
- 6: Let $b \xleftarrow{\$} \text{Bernoulli}(1/m)$. If $b = 1$ then $z \leftarrow \tau$ else $z \xleftarrow{\$} [m]_k \setminus \{\tau\}$
- 7: For each row $r \in \mathcal{R} \setminus \{\tau\} \cup \{z\}$ do
- 8: Let $(s_1, \dots, s_m) \leftarrow \text{DB}^\top(r)$ be the set of labels
- 9: $\boldsymbol{\mu} \leftarrow (\mu_1, \dots, \mu_m)$ with $\mu_j = \mathbb{1}[s_j \in \mathcal{A}_i \setminus \text{DB}^\top(z)] \forall j \in [m]$
- 10: Let $Q' \leftarrow Q' \cup \{(r, \mathbf{c})\}$ with $\mathbf{c} \leftarrow \text{PLHE.Enc}(\mathbf{A}'_r, \mathbf{k}, \boldsymbol{\mu})$
- 11: Compute $\gamma' \leftarrow \sum_{(r, \mathbf{c}) \in Q'} \text{PLHE.Eval}(\text{DB}^\top[r, *], \mathbf{c}) \triangleright \text{Server}$
- 12: Compute $\rho' \leftarrow \text{PLHE.Dec}(\mathbf{d}^*, \mathbf{k}, \gamma')$ with $\mathbf{d}^* \leftarrow \sum_{r \in \mathcal{R}} \mathbf{d}'_r$
- 13: Return $\beta_x \leftarrow \rho_i - \rho'$

Figure 8: Efficient ESCAPE with Regev PLHE: Online Query.

(τ)									(z)								
1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
1	2	3	10	11	12	19	20	21	1	2	3	10	11	12	19	20	21
4	5	6	13	14	15	22	23	24	4	5	6	13	14	15	22	23	24
7	8	9	16	17	18	25	26	27	7	8	9	16	17	18	25	26	27
$P_1^\top$			$P_2^\top$			$P_3^\top$			$P_1^\top$			$P_2^\top$			$P_3^\top$		

DB^T rows (2, 6, 7) sampled by sk'_i Replace DB^T row (τ) with (z)

Example 3: Client modifies TSet(sk'_i) to recover $\text{DB}[\phi(12)]$.

Q_i has n pairs of row and its corresponding bit vector, so the homomorphic evaluation from the server (line 13) is $f_{\boldsymbol{\theta}_1}(\mathbf{c}_1) + \dots + f_{\boldsymbol{\theta}_n}(\mathbf{c}_n)$ with $\boldsymbol{\theta}_k = \text{DB}[r_k, *]$. By Regev-based PLHE correctness, the client can decrypt the parity $\rho_i = f_{\boldsymbol{\theta}_1}(\boldsymbol{\mu}_1) + \dots + f_{\boldsymbol{\theta}_n}(\boldsymbol{\mu}_n)$. For each query Q_i , let $(\mathbf{A}_1, \dots, \mathbf{A}_n)$ be the public matrices corresponding to $(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n)$:

$$\begin{aligned}
\gamma_i &= \boldsymbol{\theta}_1(\mathbf{A}_1 \mathbf{k}_i + \mathbf{e}_1 + \Delta \boldsymbol{\mu}_1) + \dots + \boldsymbol{\theta}_n(\mathbf{A}_n \mathbf{k}_i + \mathbf{e}_n + \Delta \boldsymbol{\mu}_n) \\
&= (\boldsymbol{\theta}_1 \mathbf{A}_1 + \dots + \boldsymbol{\theta}_n \mathbf{A}_n) \mathbf{k}_i \\
&\quad + (\boldsymbol{\theta}_1 \mathbf{e}_1 + \dots + \boldsymbol{\theta}_n \mathbf{e}_n) + (\boldsymbol{\theta}_1 \Delta \boldsymbol{\mu}_1 + \dots + \boldsymbol{\theta}_n \Delta \boldsymbol{\mu}_n) \\
&= (\mathbf{d}_1 + \dots + \mathbf{d}_n) \mathbf{k}_i \\
&\quad + (\boldsymbol{\theta}_1 \mathbf{e}_1 + \dots + \boldsymbol{\theta}_n \mathbf{e}_n) + (\boldsymbol{\theta}_1 \Delta \boldsymbol{\mu}_1 + \dots + \boldsymbol{\theta}_n \Delta \boldsymbol{\mu}_n)
\end{aligned}$$

With $\mathbf{d}^* = (\mathbf{d}_1 + \dots + \mathbf{d}_n)$ (Figure 7, line 14), we can decrypt the parity ρ_i as long as the accumulated error $(\boldsymbol{\theta}_1 \mathbf{e}_1 + \dots + \boldsymbol{\theta}_n \mathbf{e}_n)$ does not overflow into the plaintext.

5.4. Online Phase

In the online phase, the client wants to recover an entry $\text{DB}[\phi(x)]$ using a hint $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$. As presented in Section 4, the key idea is to use the set $\mathcal{S}'_i \leftarrow \text{TSet}(\text{sk}'_i)$ paired with $\mathcal{S}_i \leftarrow \text{Set}(\text{sk}_i)$ from the offline phase. We need to retrieve the punctured parity ρ' corresponding to $\mathcal{S}'_i \cap \mathcal{S}_i \setminus \{x\}$ without revealing the row containing the entry label x .

Recall that in the generic construction, the label x must be removed from $\mathcal{S}'_i$ with some probability so that $\mathcal{S}'_i$ looks uniformly random. As we now operate on the matrix views of the database, the row τ covering label x must be replaced by a random row z from its partition with $1 - 1/m$ probability.

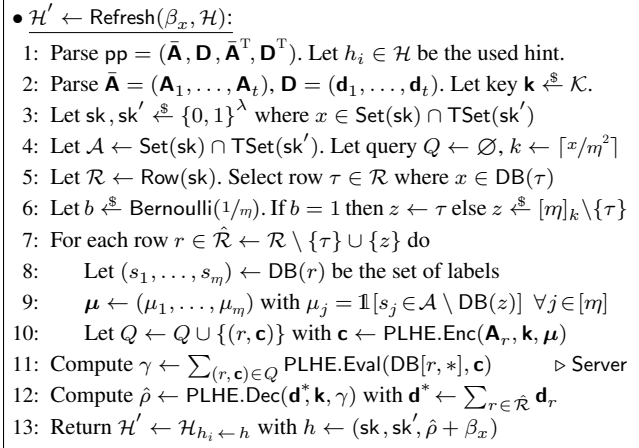


Figure 9: Efficient ESCAPE with Regev PLHE: Hint Refresh.

Figure 8 shows how our online query works. Let $\mathcal{R}$ be the set of rows covering $\mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}'_i$ (line 5). The client performs a Bernoulli trial to replace the row $\tau \in \mathcal{R}$ with a random row z (line 6). For each row $r \in \mathcal{R} \setminus \{\tau\} \cup \{z\}$, the client creates a bit vector μ to select all entry labels chosen by $\mathcal{A}_i$, excluding the replacement row z (line 9) such that μ correctly indicate the set of labels $\mathcal{S}'_i \cap \mathcal{S}_i \setminus \{x\}$ for the punctured parity $\hat{\rho}'$. Example 3 shows the server view after replacing row τ correlated to label x , with a random row z . For the remaining steps (Figure 8, line 10 to line 12), the client uses PLHE to encrypt the bit vector per chosen row and retrieve $\hat{\rho}'$ to recover $\text{DB}[x]$.

Figure 9 shows how to refresh the consumed hint h_i with a new hint to maintain the joint distribution of the hint table. Similar to the generic construction, the client samples two new sets $\mathcal{S} \leftarrow \text{Set}(\text{sk})$ and $\mathcal{S}' \leftarrow \text{TSet}(\text{sk}')$ from two new random keys (sk, sk') such that $x \in \mathcal{S} \cap \mathcal{S}'$ (line 3). Again, each key defines a set of random rows per matrix. The difference is that we constrain $x \in \mathcal{S} \cap \mathcal{S}'$ to preserve hint distribution, but we do not need rejection sampling to satisfy the constraint, as we can use a tuple $(\text{sk}, \text{sk}', x)$ to denote that x will always be included in the set evaluation of $\mathcal{S}$ and $\mathcal{S}'$. This does not impact the entry label distribution covered by the hint table, as each partition always contributes one label in the parity set $\mathcal{A} \leftarrow \mathcal{S} \cap \mathcal{S}'$ and partitions are independent from each others. Regardless, the set of rows revealed by $\mathcal{S}$ to the server only contains row τ according to the random distribution, independent of client interest, due to the Bernoulli trial for row replacement.

Theorem 3 (Correctness). *Our efficient ESCAPE with Regev-based PLHE is correct by Definition 6 if both OO-PIR and PLHE correctness hold, except with an error $\delta = 2^{-\lambda}$:*

(1) *Given a DB of N entries, with n matrix partitions of size $m \times m$, the client maintains the hint table $\mathcal{H} = (h_1, \dots, h_M)$ of size $M = \mathcal{O}(m^2(\lambda + \ln N))$ such that each entry $\text{DB}[\phi(x)]$ is covered by at least one hint.*

(2) *Given PLHE with LWE params $(\eta, m, q, \chi \sim D_{\mathbb{Z}, \sigma})$, the client can decrypt a parity sum from n linear evaluations on n DB row vectors $(\theta_1, \dots, \theta_n) \in \mathbb{Z}_p^{n \times m}$, with modulus p :*

1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
1	4	7	10	13	16	19	22	25	1	2	3	10	11	12	19	20	21
2	5	8	11	14	17	20	23	26	4	5	6	13	14	15	22	23	24
3	6	9	12	15	18	21	24	27	7	8	9	16	17	18	25	26	27
P_1			P_2			P_3			$P_1^\top$			$P_2^\top$			$P_3^\top$		

DB rows (3, 4, 7) sampled by sk DB^T rows (3, 6, 8) sampled by sk'

Example 4: A new hint that contains $\text{DB}[\phi(12)]$.

$$\lfloor q/p \rfloor \geq \sqrt{2}\sigma p \cdot \sqrt{nm} \cdot \sqrt{\ln(2/\delta)}$$

Proof. See Appendix §B □

Theorem 4 (Security). *Our efficient ESCAPE with Regev-based PLHE is secure by Definition 7.*

Proof. See Appendix §D □

5.5. Asymptotic Analysis and Optimization

Given the Regev-based PLHE with LWE parameters η as secret dimension, m as the number of LWE samples, q as ciphertext modulus, p as plaintext modulus. We denote $\mathbb{F}_q$ as the finite field. We analyze the cost of our efficient ESCAPE with N as the number of DB entries (of size β), nm as the size of query sets (n rows, each size m), n as the hint set intersection size, and M as the number of hints.

- **Online Bandwidth:** The client sends $\mathcal{O}(nmq)$ ciphertext bits, $\mathcal{O}(n)$ row indices, and receives $\mathcal{O}(\beta q/p)$ ciphertext bits, $\mathcal{O}(\beta \eta q/p)$ bits of PLHE function hints. The response can be compressed to $\mathcal{O}(\beta q/p)$ bits (i.e., removing the dimension η) by delegating the inner product between the function hints and the PLHE secret key to the server, and packing those $\mathcal{O}(\beta q/p)$ bits of result into one Paillier ciphertext, leveraging ZipPIR analysis [9].
- **Online Computation:** The client executes $\mathcal{O}(M)$ PRF evaluations and $\mathcal{O}(nm)$ multiplications over $\mathbb{F}_q$ for PLHE encryption. The server executes $\mathcal{O}(\beta nmq/p)$ multiplications over $\mathbb{F}_q$ for PLHE evaluation, $\mathcal{O}(\beta \eta nq/p)$ additions of PLHE function hints over $\mathbb{F}_q$. When compressing the response, the server executes $\mathcal{O}(\beta \eta q/p)$ extra multiplication under Paillier, which can be significantly accelerated via low-level optimization.
- **Offline Bandwidth:** The client sends $\mathcal{O}(nmMq)$ ciphertext bits, $\mathcal{O}(M)$ PRF keys, and receives $\mathcal{O}(\beta Mq/p)$ ciphertext bits, $\mathcal{O}(\beta \eta Mq/p)$ bits of PLHE function hints. To remove the η factor, the client can apply response compression as above. When $\beta = o(nm)$, the client can instead stream $\mathcal{O}(N)$ entries to compute $\mathcal{O}(M)$ hints locally *once*.
- **Offline Computation:** The server executes $\mathcal{O}(mM)$ PRF evaluations, $\mathcal{O}(\beta \eta Nq/p)$ multiplications over $\mathbb{F}_q$ for the PLHE preprocessing phase, $\mathcal{O}(\beta nmMq/p)$ $\mathbb{F}_q$ operations to compute the client hint table *once*. The client executes $\mathcal{O}(M)$ PRF key generation, $\mathcal{O}(nmM)$ multiplications over $\mathbb{F}_q$ for encryption and $\mathcal{O}(\beta \eta Mq/p)$ $\mathbb{F}_q$ operations for decryption. When $\beta = o(nm)$, we can alternatively let the client compute $\mathcal{O}(M)$ hints locally *once* by executing $\mathcal{O}(\beta nMq/p)$ additions over $\mathbb{F}_q$ at the client.

6. Experimental Evaluation

6.1. Implementation

We implemented ESCAPE from scratch in C/C++ and incorporated the optimization described in Section 5.5. We used hardware intrinsics (AVX-512, VAES, IFMA52) and out-of-order scheduler to dispatch multiple micro-ops to independent execution units (ALU, load/store, vector) per cycle. For fast PLHE encryption that must reproduce many random PLHE public parameters $\mathbf{A}$, we utilized VAES across eight 512-bit registers to produce 128 random 32-bit unsigned per 100 CPU cycles. We vectorized the inner products as 16×32 -bit multiply-accumulate via AVX-512. To compress the response via Paillier, we implemented fast multiplication with Karatsuba algorithm [55] and Montgomery reduction [56] over operands held as 80 limbs of 52 bits unsigned. We leveraged IFMA52 `vpadd52luq/vpmadd52huq` to increase CPU throughput by processing eight independent limbs per 512-bit register. Our implementation is publicly available at: <https://github.com/vt-asaplab/Escape>

6.2. Configuration

Hardware Setting. We used a dual-socket server Intel Xeon Platinum 8360Y (Ice Lake-SP) with 48 cores, supporting AVX-512, VAES, and IFMA52 instructions. The platform is equipped with 1 TiB DDR4-3200 DRAM, distributed as 512 GiB per socket with an average of 80 GiB/s read and write.

Database Setting. To benchmark the database processing cost of all PIR schemes, we created databases of sizes ranging from 1 TiB to 64 TiB, with three different entry sizes 8 KiB, 16 KiB, 64 KiB, and the number of entries N varies from 2^{24} to 2^{32} . Since not all PIR schemes support such large-scale experiments, we measured their performance on small database size up to 1 TiB, then extrapolated the computational delay up to 64 TiB, neglecting the memory overhead. For ESCAPE, our experiment measured its actual performance by generating the database subset that it needs to process.

Network Setting. We simulated the network for concurrent usage of daily tasks (e.g. file streaming, cloud backup, agentic workflows) and configured client-server bandwidth to be around 70 Mbps with 15 ms round-trip latency.

Parameter Selection. For our efficient ESCAPE on databases with the number of entries $N = n \times \eta \times \eta$, we choose $n = \eta = N^{1/3}$. To securely aggregate $n\eta$ entries for N up to 2^{32} entries, we choose PLHE parameters $q = 2^{32}$, $p = 2^7$, error standard deviation $\sigma = 3.2$, and $\eta = 1024$ for binary secret dimension to achieve ~ 112 bits of security according to the LWE estimator [57]. We set $M = N^{2/3}(\lambda + \ln N)$ with $\lambda = 40$ for $2^{-\lambda}$ probability of failure.

Counterpart Comparison. We compared ESCAPE with state-of-the-art single-server PIR in the standard arbitrary query sequence assumption, including OO-PIR model: Piano [14], RMS [15], and lattice-based PIR model: SimplePIR [5], InsPIRe [8], VIA [4]. For Piano and RMS, we amortize their offline costs over the number of queries they can support.

We used the publicly open-source implementations of **Piano**, **RMS**, **SimplePIR**, **VIA**, **InsPIRe** to assess their performance. Note that more recent OO-PIR schemes exist (e.g., [16, 49, 58]), but they primarily optimize orthogonal dimensions (e.g., storage, preprocessing). Thus, they retain similar amortized bandwidth scaling relevant to Piano and RMS that we used as representative baselines for comparison.

6.3. Results

Online Latency. ESCAPE takes only a few seconds to answer a query, which is one to three orders of magnitude lower end-to-end latency (up to $1575\times$) than all prior single-server PIR schemes.

For a database with $N = 2^{30}$ entries of 64 KiB, Figure 10 shows that ESCAPE answers a query in 4.37s, against the tens of minutes to hours required by other lattice-based standard PIR. ESCAPE is faster than SimplePIR (830s) by $190\times$, InsPIRe (843s) by $193\times$, and VIA (6881s) by $1575\times$. Even against other OO-PIR, whose online server is nearly free by construction, ESCAPE still wins outright and is $56\times$ faster than Piano (245s) and $7.5\times$ faster than RMS (33s).

In deployment terms, this efficiency allows ESCAPE to serve 824 queries per hour on $N = 2^{30}$ entries of 64 KiB each, against only 4.3 queries for SimplePIR, 4.3 for InsPIRe, 110 for RMS, 14.7 for Piano, and 0.52 for VIA. In other words, serving 1,000 queries per day costs ESCAPE under 1.2 hours of computation, against 9.6, 9.8, 80, and 2.8 server-days for SimplePIR, InsPIRe, VIA, and Piano respectively, which heavily affect bandwidth or computation scalability.

ESCAPE worst case at 64 KiB (4.37s at $N = 2^{30}$) still falls below every counterparts best case for same entry size (reached at $N = 2^{24}$): 5.11s for RMS, 14.2s for SimplePIR, 15.86s for InsPIRe, 30.68s for Piano, and 107.6s for VIA. ESCAPE end-to-end latency barely moves across the sweep (only a $1.27\times$ factor) while other lattice-based schemes grow by $64\times$ and Piano or RMS by $8\times$. Put plainly, ESCAPE at its largest N already exceeds every counterpart at its smallest.

Notably, with the same 64 TiB database scale, ESCAPE advantage carries to smaller blocks with larger ratios. Moving from $N = 2^{30}$ entries of 64 KiB to $N = 2^{32}$ entries of 16 KiB, ESCAPE end-to-end latency halves (4.37 to 2.16s) while others stay pinned, so the speedups are doubled: from $190\times$ to $385\times$ over SimplePIR, from $193\times$ to $393\times$ over InsPIRe, and from $1575\times$ to $3186\times$ over VIA, while the Piano and RMS hold near $56\times$ and $7\times$. This proves ESCAPE potential across varying entry size β . For $N = 2^{32}$ entries of 8 KiB each, ESCAPE 1.66s is $263\times$ faster than InsPIRe (436s) and $2073\times$ faster than VIA (3441s).

Unlike its counterparts, ESCAPE balances latency across compute and network. For $N = 2^{30}$ entries of 64 KiB each, its latency divides into 3.58s of server time and 0.58s of network. Lattice-based standard PIR are instead bounded by their linear server time, with the server dwarfing the network by two to four orders of magnitude. VIA, in particular, squanders its hard-won bandwidth lead on a 6881-second server. Conversely, the offline-online schemes are purely

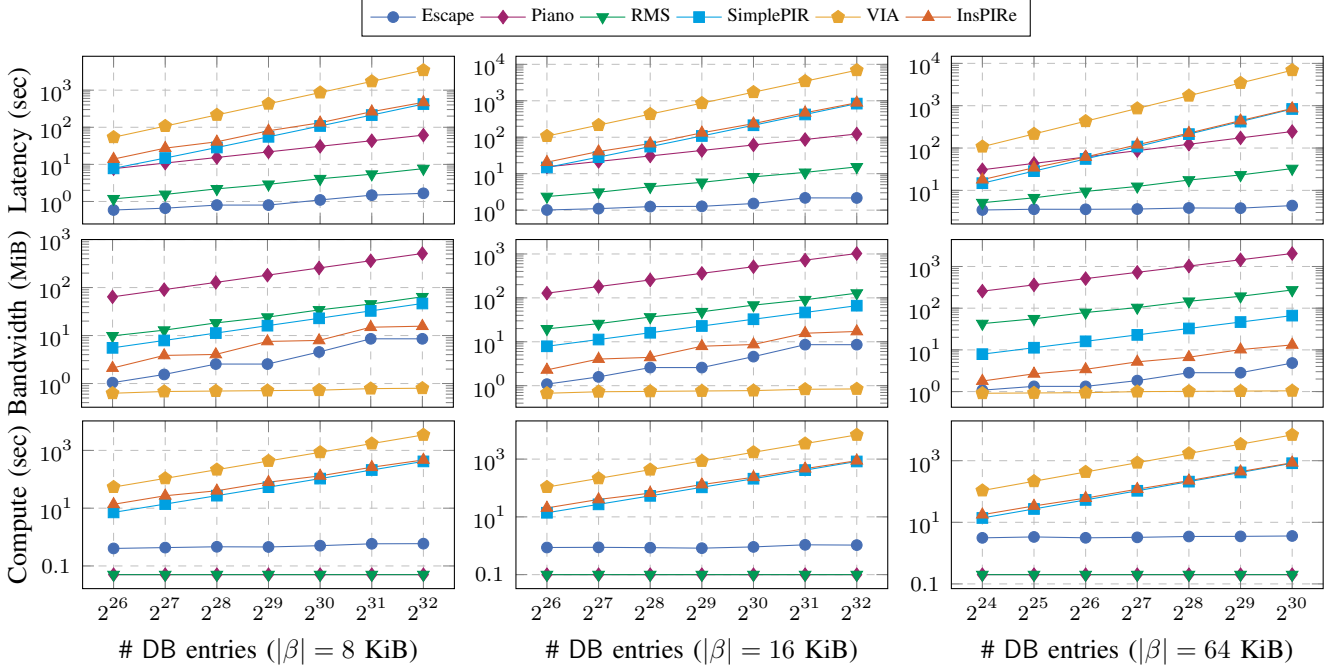


Figure 10: Total Online E2E Latency (w/ Computation and Bandwidth Cost) for Varied Database and Entry Sizes.

TABLE 2: Concrete Cost Comparison in Large-Scale Database Settings.

Scheme	$ \beta = 64 \text{ KiB}, N = 2^{30}$ (64 TiB DB)					$ \beta = 16 \text{ KiB}, N = 2^{32}$ (64 TiB DB)				
	Query	Response	Server	Client	Total	Query	Response	Server	Client	Total
SimplePIR	33.03 MiB	33.33 MiB	819.20 s	3.61 s	830.16 s	33.17 MiB	33.20 MiB	819.20 s	3.64 s	830.20 s
InsPIRe	7.16 MiB	6.00 MiB	830.95 s	10.8 s	843.52 s	14.16 MiB	3.00 MiB	825.08 s	21.6 s	849.08 s
VIA	696 KiB	368 KiB	6881.28 s	9.9 ms	6881.41 s	768 KiB	92 KiB	6881.28 s	10.8 ms	6881.39 s
Piano	60 KiB	2048.00 MiB	0 ms	28.8 ms	245.43 s	128 KiB	1024.00 MiB	0 ms	14.3 ms	122.73 s
RMS	64 KiB	273.00 MiB	0 ms	0.96 ms	32.72 s	136 KiB	128.00 MiB	0 ms	0.45 ms	15.35 s
ESCAPE	4.50 MiB	338 KiB	3.58 s	213 ms	4.37 s	8.50 MiB	84 KiB	932 ms	195 ms	2.16 s

Scheme	$ \beta = 64 \text{ KiB}, N = 2^{29}$ (32 TiB DB)					$ \beta = 16 \text{ KiB}, N = 2^{30}$ (16 TiB DB)				
	Query	Response	Server	Client	Total	Query	Response	Server	Client	Total
SimplePIR	23.01 MiB	23.51 MiB	409.60 s	2.47 s	417.45 s	16.25 MiB	16.37 MiB	204.80 s	1.76 s	210.48 s
InsPIRe	7.16 MiB	3.00 MiB	415.48 s	10.47 s	427.17 s	7.16 MiB	1.50 MiB	207.74 s	11.2 s	220.25 s
VIA	686 KiB	368 KiB	3440.64 s	9.5 ms	3440.77 s	696 KiB	92 KiB	1720.32 s	9.9 ms	1720.42 s
Piano	42 KiB	1448.12 MiB	0 ms	20.4 ms	173.54 s	60 KiB	512.00 MiB	0 ms	7.20 ms	61.36 s
RMS	45 KiB	193.00 MiB	0 ms	0.70 ms	23.13 s	64 KiB	68.25 MiB	0 ms	0.24 ms	8.19 s
ESCAPE	2.50 MiB	338 KiB	3.28 s	176 ms	3.80 s	4.50 MiB	84 KiB	850 ms	123 ms	1.52 s

network-bound, with Piano carrying 2 GiB at the same setting. This split exposes two pillars underpinning ESCAPE latency win: constant response bandwidth and sublinear server time.

Online Bandwidth. ESCAPE bandwidth is up to $424\times$ smaller than prior OO-PIR schemes, more than two orders of magnitude, while retaining sublinear online time.

Figure 10 shows that ESCAPE lives in the polylogarithmic bandwidth tier, shared by only VIA and InsPIRe. At the largest setting we measure, $N = 2^{30}$ entries of 64 KiB each, ESCAPE consumes only 4.83 MiB per query, which is smaller than Piano (2048 MiB) by $424\times$, RMS (273 MiB) by $56\times$, SimplePIR (66 MiB) by $13.7\times$, and is comparable to InsPIRe

(13.16 MiB). In absolute terms, ESCAPE per-query traffic is only 0.24% of Piano and 7.3% of SimplePIR.

The lead widens monotonically as parameter N grows. Across 64 KiB sweep from $N = 2^{24}$ to $N = 2^{30}$, ESCAPE bandwidth rises by $4.5\times$ only in the MiB range (from 1.08 to 4.83 MiB) while Piano grows $8\times$ into the GiB range (256 MiB to 2 GiB) and SimplePIR grows $8\times$ in step (7.9 to 66 MiB). ESCAPE advantage climbs from $39\times$ to $56\times$ over RMS, and from $1.65\times$ to $2.7\times$ over InsPIRe. Piano smallest point (256 MiB at $N = 2^{24}$) is still $53\times$ above ESCAPE worst case (4.83 MiB at $N = 2^{30}$).

The savings compound under a critical load. At 70 Mbps

endpoint link, Piano needs 234s per query for its 2048 MiB download, RMS needs 31s for 273 MiB, SimplePIR needs 7.6s for 66 MiB, while ESCAPE clears its 4.83 MiB in 552 ms. Across one million queries, the egress totals diverge sharply: 4.6 TiB for ESCAPE compared with 2 PiB for Piano (420 $\times$), 267 TiB for RMS (56 $\times$), and 65 TiB for SimplePIR (14 $\times$).

The same picture holds at 16 KiB entries, where ESCAPE stays orders of magnitude below the $\mathcal{O}(\sqrt{N})$ tier and remains comparable to InsPIRe. VIA is the sole scheme that beats ESCAPE on bandwidth, staying near 1 MiB for a roughly 5 $\times$ advantage. However, this gap narrows in the large-entry regime where most PIR applications live (encrypted mail, document retrieval, ...), shrinking from 10.2 $\times$ at 16 KiB (8.59 against 0.84 MiB at $N = 2^{32}$) to 4.6 $\times$ at 64 KiB (4.83 against 1.04 MiB at $N = 2^{30}$). As the next section shows, this 5 $\times$ deficit is dominated by the server-time gap, where ESCAPE outpaces VIA by two to three orders of magnitude.

Online Computation. ESCAPE server time is two to three orders of magnitude (up to 1912 $\times$) faster than prior single-server PIR, which is sublinear to the parameter N , while other lattice-based standard PIR scales linearly.

Figure 10 reports server time across database sizes. At 64 KiB, ESCAPE server time rises just 1.15 $\times$ (from 3.1 to 3.6s) across a 64 $\times$ growth in N from 2^{24} to 2^{30} . By contrast, SimplePIR, InsPIRe, and VIA each sweep the full $N \times \beta$ database on every query, so their cost scales linearly with N : by the midpoint $N = 2^{27}$ they already take 104, 111, and 860s respectively (while ESCAPE is still at 3.2s), reaching hundreds to thousands of seconds by 2^{30} .

For a database with $N = 2^{30}$ entries of 64 KiB each, ESCAPE 3.6s outpaces SimplePIR (819s) by 227 $\times$, InsPIRe (830s) by 230 $\times$, and VIA (6881s) by 1912 $\times$. In wall-clock terms, that is 13.6, 13.8, and 115 minutes per query, during which ESCAPE has already processed 227, 230, and 1912 queries. The advantage also compounds with N : ESCAPE speedup over SimplePIR grows through 4 $\times$, 17 $\times$, 60 $\times$, and 230 $\times$ at $N = 2^{24}, 2^{26}, 2^{28}, 2^{30}$, with InsPIRe (5.0 $\times$ to 235 $\times$) and VIA (34 $\times$ to 1912 $\times$) following the same trajectory. Each doubling of the database roughly doubles ESCAPE lead and a 64 $\times$ growth in N is equivalent to a 57 $\times$ growth in ESCAPE lead. VIA 8 $\times$ markup over SimplePIR comes from its heavier ring-polynomial processing, whose multi-core scaling is limited by NTT memory pressure.

At 16 KiB, the story repeats with larger speedup ratios. ESCAPE server time rises from 0.87s at $N = 2^{26}$ to 0.93s at $N = 2^{32}$, while the other lattice-based PIR schemes stay the same, with 819, 825, and 6881s for SimplePIR, InsPIRe, and VIA. As ESCAPE per-query cost shrinks by 3.87 $\times$ from 64 KiB to 16 KiB entries (3.6s to 0.93s), the speedups over SimplePIR, InsPIRe, and VIA climb respectively to 878 $\times$, 886 $\times$, and 7398 $\times$ on $N = 2^{32}$ entries of 16 KiB each.

Piano and RMS achieve sub-second server computation via cheap XORs, but only by shifting an $\mathcal{O}(\beta\sqrt{N})$ preprocessing burden onto every client, which then resurfaces as the network-bound latency we already quantified.

Cost Breakdown Analysis. Table 2 shows the detailed cost of all schemes. The total column includes server and client

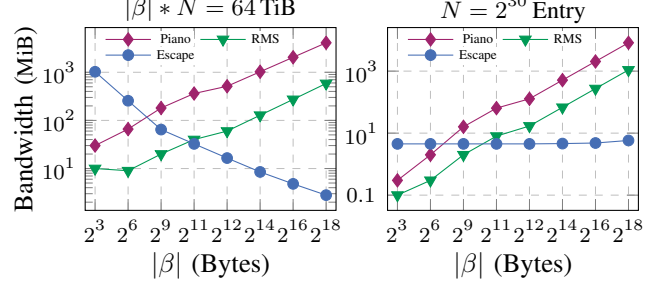


Figure 11: Bandwidth Cost with Varied Entry Size.

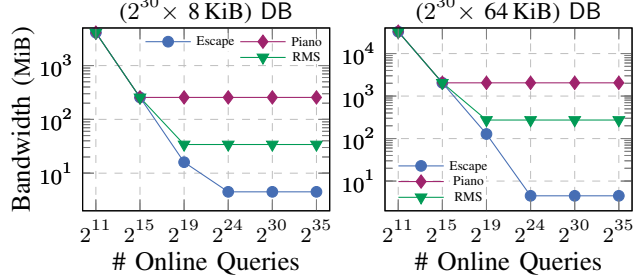


Figure 12: Hint-Update Average Bandwidth Per Query.

computation delay, plus query and response bandwidth delay. We observe while lattice-based PIR can balance query and response by trading larger number of entries for smaller entry size, the query and response size in OO-PIR is imbalanced. Moreover, the balancing technique from lattice-based PIR cannot be used to improve bandwidth overhead for OO-PIR schemes Piano and RMS. This is because building OO-PIR hint on a smaller entry size results in a smaller entry chunk being recovered per query and hint usage. To retrieve the same amount of information, Piano and RMS with rebalanced bandwidth need more queries and hint usage, which reduces their number of supported retrievals and incurs the amortized bandwidth cost for database streaming. For example, with 64 TiB = $2^{32} \times 16$ KiB, we can rebalance $N = 2^{44}$ and $|\beta| = 4$ bytes. The query size is $2^{22} \times 22$ bits and the amortized response is $2^{22} \times 4$ bytes to retrieve a 4-byte chunk. To retrieve a 16-KiB entry, we need 2^{12} queries out of the 2^{22} query (hint) budget, so the number of supported retrieval reduced to 2^{10} . The amortized bandwidth cost is $2^{34} \times 4$ bytes, which is worse than fetching $2^{16} \times 16$ KiB. Thus, Piano and RMS with $\mathcal{O}(\beta\sqrt{N})$ response size are not efficiently scalable for large entry size settings compared to ESCAPE.

Small Entry Size Settings. As ESCAPE $\mathcal{O}(\beta)$ response size is independent of the number of entries N at the cost of $\mathcal{O}(N^{2/3})$ query size, we can improve ESCAPE bandwidth cost in small entry size settings by grouping entries into larger blocks, reducing the number of entry indices for smaller query sizes. Figure 11 reports the bandwidth of ESCAPE and Piano/RMS for varied entry sizes from 8 bytes to 128 KiB on databases with (a) size 64 TiB or (b) 2^{30} entries.

Overall Performance. The cross-entry-size picture is clear: At the largest N for each entry size, ESCAPE latency stays under 4.5s (4.37, 2.16, and 1.66s for 64, 16, and 8 KiB), while the linear-server counterparts stay between 418s and

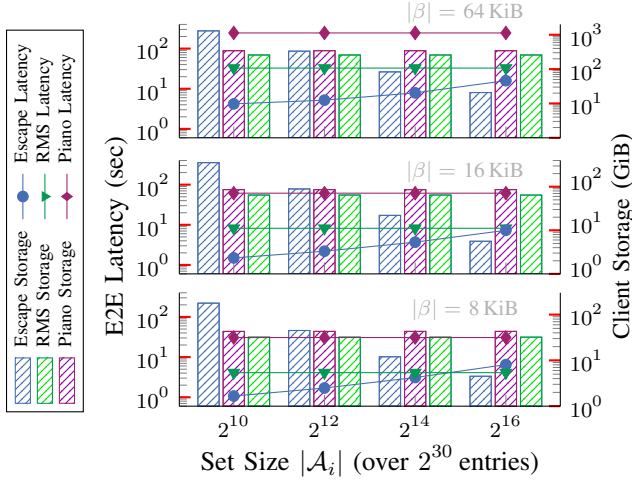


Figure 13: Client Storage vs. Latency with Varied Set Size

6881s. Against RMS, ESCAPE is strictly smaller for every pair of entry size β and entry count N we measure. In short, ESCAPE dominates the PIR frontier and is the first PIR in single-server setting to scale bandwidth and computation efficiently on databases with billions of entries in KiB regime, pushing online latency into the low-seconds regime.

Amortized Offline Cost Per Query. Figure 12 reports the offline bandwidth cost amortized over a varied number of online queries for databases with 2^{30} entries. The offline cost includes the one-time cost to build the hint table before the first query and maintaining the table per query afterward. ESCAPE, Piano, RMS all stream the entire database at first to preprocess hints. However, after about 2^{19} queries, this amortized cost no longer decreases for Piano or RMS as they have to periodically stream the database. ESCAPE amortized bandwidth keeps decreasing as more queries are executed. After the one-time database streaming, ESCAPE can refresh hint remotely on the server, and the response time per hint refresh is exactly as one online query (Figure 10).

Client Storage. Depending on the database and parameter settings, the client in ESCAPE stores at most 640 GiB, 1280 GiB, and 1280 GiB of hints to retrieve 8 KiB, 16 KiB, or 64 KiB entry, respectively, which supports an unlimited number of queries for databases up to 64 TiB. Asymptotically, ESCAPE client storage is $\mathcal{O}_\lambda(\beta N^{2/3})$ compared to $\mathcal{O}_\lambda(\beta \sqrt{N})$ of Piano and RMS. However, we note that concretely, Piano and RMS also need to store replacement entries, backup hints, and cache for rebuilding hints.

As shown in Figure 13, RMS client storage for 64 TiB database ($2^{30} \times 64$ KiB) is 257.7 GiB. Piano client storage is 343.6 GiB. For ESCAPE, we can reduce the number of hints by choosing set size 2^{12} per hint, which is slightly above $N^{1/3}$, for $N = 2^{30}$ and $|\beta| = 64$ KiB, such that the required client storage reduces to 332.8 GiB, with 5.2s response time.

ESCAPE can reach asymptotic storage cost $\mathcal{O}_\lambda(\beta \sqrt{N})$ to match prior works, with parameter setting $n \approx N^{1/2}$, $\eta \approx N^{1/4}$ given the number of entries $N = n \times \eta \times \eta$. In such cases, ESCAPE concrete storage cost can be lower, since ESCAPE is not bounded by the required number of backup

hints for supporting $\sqrt{N}$ queries as in Piano and RMS, whose storage cost cannot be reduced even if the set size per hint is larger. For 8 TiB database ($2^{30} \times 8$ KiB), ESCAPE client storage is merely 10 GiB, while incurring 7.1s response time, compared to RMS 30 GiB storage with 4.1s time.

7. Related Work

Doubly Efficient PIR (DEPIR). DEPIR reduces per-query server cost by encoding the database once, independently of any client. Beimel et al. [32] first showed a construction that imposes $\mathcal{O}(N^{3.2})$ size of database encoding for $\mathcal{O}(N^{0.6})$ server time. Canetti et al. [59] and Boyle et al. [60] introduced single-server schemes from secretly permuted Reed-Muller codes with superlinear, per-client-key encoding. Boyle et al. [60] made their scheme public-key via a non-standard assumption. Lin et al. [61] later achieved a scheme from the Ring-LWE assumption at polylogarithmic cost. Lazzaretti et al. [62] gave the first information-theoretic multi-server DEPIR with unbounded queries. Lin et al. [63] prove that black-box use of generic primitives cannot build DEPIR, so constructions must exploit algebraic structure. Okada et al. [64] lower-bounded the ciphertext ring size such schemes need. An efficient DEPIR remains an open problem.

Batch PIR. Instead of speeding up one query, batched PIR [65] was proposed to amortize the server work across Q queries via database combinatoric encoding. Angel et al. [26] first made batch PIR practical via randomized batch codes, where Q queries can be processed with a concretely cheaper cost than running Q PIR instances. The encoded databases in [26, 65], however, increased the response size. Angel et al. [41] reduced the response size with a construction that achieves $\mathcal{O}(N)$ server time for size- Q batch while only returning $\mathcal{O}(Q)$ ciphertexts. Mughees and Ren [66] reduced the response through a vectorized batch PIR that can pack many entries in a single ciphertext. Recent schemes further reduced these costs. PIRANA [67] accelerated multi-query retrieval with constant-weight codes. Bienstock et al. [68] and Respire [40] shrunk the batch response via oblivious ciphertext compression and ring-switching response packing.

Multi-Server OO-PIR. To achieve sublinear server online time, Corrigan-Gibbs et al. [45] proposed the first OO-PIR scheme in the two-server setting, where client maintains a state of size sublinear, requiring λ protocol executions for query correctness. [46, 47] showed how to eliminate such λ executions, requiring $\mathcal{O}(\sqrt{N})$ entries to be downloaded or $\mathcal{O}(N)$ client computation/storage. Shi et al. [69] leveraged λ executions to achieve a scheme with polylogarithmic bandwidth and sublinear complexity for the client and server overall. [15, 48] proposed patching technique to enable the OO-PIR query security that formerly relied on the λ executions, thus removing the need for λ executions while maintaining other metrics. Lazzaretti et al. [49] leveraged $\mathcal{O}(\sqrt{N})$ response bandwidth to remove both the λ executions and the λ factor in client storage cost. The techniques from [15, 49, 69] have been studied in single-server setting but have yet to efficiently balance computation and bandwidth.

Acknowledgment

This work was supported by the National Science Foundation (NSF) under grant CNS-2350215, an unrestricted gift from Robert Bosch, and the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

References

- [1] C. A. Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, “XPIR: Private Information Retrieval for Everyone,” *PETS (2016)*.
- [2] I. Ahmad, Y. Yang, D. Agrawal, A. El Abbadi, and T. Gupta, “Addra: Metadata-Private Voice Communication Over Fully Untrusted Infrastructure,” in *15th USENIX Symposium (OSDI 21)*, 2021.
- [3] S. J. Menon and D. J. Wu, “Spiral: Fast, High-Rate Single-Server PIR via FHE Composition,” in *IEEE S&P*, 2022.
- [4] C. Liu, X. Wang, and Z. Zhang, “VIA: Communication-Efficient Single-Server Private Information Retrieval,” in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026.
- [5] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, “One Server For The Price of Two: Simple and Fast Single-Server Private Information Retrieval,” in *USENIX*, 2023.
- [6] B. Li, D. Micciancio, M. Raykova, and M. Schultz-Wu, “Hintless Single-Server Private Information Retrieval,” in *Annual International Cryptology Conference*, 2024.
- [7] S. J. Menon and D. J. Wu, “YPIR: High-Throughput Single-Server PIR with Silent Preprocessing,” in *USENIX Security*, 2024.
- [8] R. A. Mahdavi, S. Patel, J. Y. Seo, and K. Yeo, “InsPIR: Communication-Efficient PIR with Server-side Preprocessing,” in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026.
- [9] R. A. Mahdavi, A. Diaa, and F. Kerschbaum, “ZipPIR: High-throughput Single-server PIR without Client-side Storage,” in *35th USENIX Security Symposium (USENIX Security 26)*, 2026.
- [10] A. Lazzaretti and C. Papamanthou, “Near-Optimal Private Information Retrieval with Preprocessing,” in *Theory of Cryptography (TCC 23)*.
- [11] M. Zhou, W. Lin, Y. Tselekounis, and E. Shi, “Optimal Single-Server Private Information Retrieval,” in *Annual International Conference on Theory and Applications of Cryptographic Techniques*, 2023.
- [12] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan, “Single-Server Private Information Retrieval with Sublinear Amortized Time,” in *Advances in Cryptology – EUROCRYPT 2022*. Springer, 2022.
- [13] A. Ghoshal, M. Zhou, and E. Shi, “Efficient Pre-Processing PIR Without Public-Key Cryptography,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2024.
- [14] M. Zhou, A. Park, W. Zheng, and E. Shi, “PIANO: Extremely Simple, Single-Server PIR with Sublinear Server Computation,” in *2024 IEEE Symposium on Security and Privacy*, May 2024.
- [15] L. Ren, M. H. Mughees, and I. Sun, “Simple and Practical Amortized Sublinear Private Information Retrieval using Dummy Subsets,” in *ACM SIGSAC Conference (CCS)*, 2024.
- [16] Z. Wang and L. Ren, “Single-Server Client Preprocessing PIR with Tight Space-Time Trade-Off,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2025.
- [17] B. Fisch, A. Lazzaretti, Z. Liu, and C. Papamanthou, “ThorPIR: Single Server PIR via Homomorphic Thorp Shuffles,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2024.
- [18] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, “Private Information Retrieval,” *Journal of the ACM (JACM)*, 1998.
- [19] A. Ali, T. Lepoint, S. Patel, M. Raykova, P. Schoppmann, K. Seth, and K. Yeo, “Communication-Computation Trade-offs in PIR,” in *30th USENIX Security Symposium*, 2021.
- [20] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh *et al.*, “Protecting Accounts from Credential Stuffing with Password Breach Alerting,” in *28th USENIX Security Symposium*, 2019.
- [21] M. Green, W. Ladd, and I. Miers, “A Protocol For Privately Reporting Ad Impressions At Scale,” in *ACM SIGSAC Conference (CCS)*, 2016.
- [22] S. Servan-Schreiber, K. Hogan, and S. Devadas, “AdVeil: A Private Targeted Advertising Ecosystem,” *Cryptology ePrint Archive*, 2021.
- [23] M. Backes, A. Kate, M. Maffei, and K. Pecina, “ObliviAd: Provably Secure and Practical Online Behavioral Advertising,” in *2012 IEEE Symposium on Security and Privacy*.
- [24] S. Servan-Schreiber, S. Langowski, and S. Devadas, “Private Approximate Nearest Neighbor Search with Sublinear Communication,” in *2022 IEEE Symposium on Security and Privacy*.
- [25] M. Zhou, E. Shi, and G. Fanti, “Pacmann: Efficient Private Approximate Nearest Neighbor Search,” in *International Conference on Learning Representations*, 2025.
- [26] S. Angel and S. Setty, “Unobservable Communication Over Fully Untrusted Infrastructure,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.
- [27] R. Cheng, W. Scott, E. Masserova, I. Zhang, V. Goyal, T. Anderson, A. Krishnamurthy, and B. Parno, “Talek: Private Group Messaging with Hidden Access Patterns,” in *Proceedings of the 36th Annual Computer Security Applications Conference*, 2020.
- [28] P. Mittal, F. Olumofin, C. Troncoso, N. Borisov, and I. Goldberg, “PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval,” in *20th USENIX Security Symposium*, 2011.
- [29] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish, “Scalable and Private Media Consumption with Popcorn,” in *13th USENIX Symposium (NSDI 16)*, 2016.
- [30] A. Ambainis, “Upper Bound on The Communication Complexity of Private Information Retrieval,” in *International Colloquium on Automata, Languages, and Programming*, 1997.
- [31] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, “Breaking The $O(n^{1/(2^k-1)})$ Barrier for Information-theoretic Private Information Retrieval,” in *43rd Annual IEEE Symposium (FOCS)*, 2002.
- [32] A. Beimel, Y. Ishai, and T. Malkin, “Reducing The Servers Computation In Private Information Retrieval: PIR with Preprocessing,” in *CRYPTO 2000: Annual International Cryptology Conference*, 2000.
- [33] Z. Dvir and S. Gopi, “2-Server PIR With Subpolynomial Communication,” *Journal of the ACM (JACM)*, 2016.
- [34] N. Gilboa and Y. Ishai, “Distributed Point Functions and Their Applications,” in *Advances in Cryptology – EUROCRYPT 2014*.
- [35] E. Kushilevitz and R. Ostrovsky, “Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval,” in *38th Annual Symposium on Foundations of Computer Science*, 1997.
- [36] C. Cachin, S. Micali, and M. Stadler, “Computationally Private Information Retrieval with Polylogarithmic Communication,” in *Advances in Cryptology – EUROCRYPT*, 1999.
- [37] C. Gentry and Z. Ramzan, “Single-Database Private Information Retrieval with Constant Communication Rate,” in *International Colloquium on Automata, Languages, and Programming*, 2005.
- [38] H. Lipmaa, “An Oblivious Transfer Protocol with Log-Squared Communication,” in *Information Security: 8th International Conference*.
- [39] R. Sion and B. Carbanar, “On The Computational Practicality of Private Information Retrieval,” in *NDSS*, 2007.
- [40] A. Burton, S. J. Menon, and D. J. Wu, “Respire: High-rate PIR for Databases with Small Records,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2024.

- [41] S. Angel, H. Chen, K. Laine, and S. Setty, “PIR with Compressed Queries and Amortized Query Processing,” in *IEEE S&P*, 2018.
- [42] M. H. Mughees, H. Chen, and L. Ren, “OnionPIR: Response Efficient Single-Server PIR,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2021.
- [43] A. Davidson, G. Pestana, and S. Celi, “FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval,” *PETS*, 2023.
- [44] S. Patel, G. Persiano, and K. Yeo, “Private Stateful Information Retrieval,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.
- [45] H. Corrigan-Gibbs and D. Kogan, “Private Information Retrieval with Sublinear Online Time,” in *EUROCRYPT 2020*.
- [46] A. Lazaretti and C. Papamanthou, “TreePIR: Sublinear-Time and Polylog-Bandwidth Private Information Retrieval from DDH,” in *Advances in Cryptology – CRYPTO 2023*.
- [47] D. Kogan and H. Corrigan-Gibbs, “Private Blocklist Lookups With Checklist,” in *30th USENIX Security Symposium*, 2021.
- [48] H. Nguyen, J. Guajardo, and T. Hoang, “Client-Efficient Online-Offline Private Information Retrieval,” *PETS (2025)*.
- [49] A. Lazaretti and C. Papamanthou, “Single-Pass Client Preprocessing Private Information Retrieval,” in *USENIX Security Symposium*, 2024.
- [50] O. Regev, “On Lattices, Learning With Errors, Random Linear Codes, and Cryptography,” *Journal of the ACM (JACM)*, 2009.
- [51] P. Paillier, “Public-Key Cryptosystems Based On Composite Degree Residuosity Classes,” in *International Conference on The Theory and Applications of Cryptographic Techniques*, 1999.
- [52] I. Damgård and M. Jurik, “A Generalisation, A Simplification and Some Applications of Paillier’s Probabilistic Public-Key System,” in *International Workshop on Public Key Cryptography*, 2001.
- [53] R. Cramer, R. Gennaro, and B. Schoenmakers, “A Secure and Optimally Efficient Multi-Authority Election Scheme,” *European Transactions on Telecommunications*, 1997.
- [54] V. Lyubashevsky, C. Peikert, and O. Regev, “On Ideal Lattices and Learning With Errors Over Rings,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010.
- [55] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley.
- [56] P. L. Montgomery, “Modular Multiplication without Trial Division,” *Mathematics of Computation*, 1985.
- [57] M. R. Albrecht, R. Player, and S. Scott, “On the Concrete Hardness of Learning with Errors,” *Journal of Mathematical Cryptology*, 2015.
- [58] P. S. Arunachalam and L. Ren, “Single-Server Stateful PIR with Verifiability and Balanced Efficiency,” in *IEEE S&P*, 2026.
- [59] R. Canetti, J. Holmgren, and S. Richelson, “Towards Doubly Efficient Private Information Retrieval,” in *Theory of Cryptography: 15th International Conference, TCC 2017*. Springer, 2017.
- [60] E. Boyle, Y. Ishai, R. Pass, and M. Wootters, “Can We Access a Database both Locally and Privately?” in *Theory of Cryptography: 15th International Conference*, 2017.
- [61] W.-K. Lin, E. Mook, and D. Wichs, “Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation From Ring LWE,” in *55th Annual ACM Symposium (STOC)*, 2023.
- [62] A. Lazaretti, Z. Liu, B. Fisch, P. Miao, and C. Papamanthou, “Multi-server Doubly Efficient PIR in the Classical Model and Beyond,” in *Theory of Cryptography Conference*, 2025.
- [63] W.-K. Lin, E. Mook, and D. Wichs, “Black Box Crypto is Useless for Doubly Efficient PIR,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2025.
- [64] H. Okada, R. Player, S. Pohmann, and C. Weinert, “On Algebraic Homomorphic Encryption and Its Applications to Doubly Efficient PIR,” in *Advances in Cryptology – EUROCRYPT 2025*.
- [65] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, “Batch Codes and Their Applications,” in *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, 2004.
- [66] M. H. Mughees and L. Ren, “Vectorized Batch Private Information Retrieval,” in *IEEE Symposium on Security and Privacy*, 2023.
- [67] J. Liu, J. Li, D. Wu, and K. Ren, “Pirana: Faster Multi-Query PIR via Constant-Weight Codes,” in *IEEE S&P 2024*, 2024.
- [68] A. Bienstock, S. Patel, J. Y. Seo, and K. Yeo, “Batch PIR and Labeled PSI with Oblivious Ciphertext Compression,” in *33rd USENIX Security Symposium*, 2024.
- [69] E. Shi, W. Aqeel, B. Chandrasekaran, and B. Maggs, “Puncturable Pseudorandom Sets and Private Information Retrieval with Near-Optimal Online Bandwidth and Time,” in *Proceedings of the 41st Annual International Cryptology Conference*, 2021.

Appendix A. Deferred Lemma and Analysis

Lemma 1. *Let $\mathcal{A}_i = \mathcal{S}_i \cap \mathcal{S}'_i$ be the intersection of two random sets $(\mathcal{S}_i, \mathcal{S}'_i)$ with $|\mathcal{S}_i| = |\mathcal{S}'_i| = m$. Then $\mathbb{E}[|\mathcal{A}_i|] = m^2/N$. Furthermore, $||\mathcal{A}_i| - m^2/N| < \mathcal{O}(\sqrt{m^2/N})$.*

Proof. For every index $x \in [N]$, we define indicator variable $Y_x = \mathbb{1}[x \in \mathcal{S}_i \cap \mathcal{S}'_i]$. Since $\mathcal{S}_i$ and $\mathcal{S}'_i$ are independent:

$$\mathbb{E}[|\mathcal{A}_i|] = \sum_{x \in [N]} \mathbb{E}[Y_x] = N \cdot \Pr[x \in \mathcal{S}_i \cap \mathcal{S}'_i] = \sqrt{m^2/N}$$

So $|\mathcal{A}_i|$ is hypergeometric with mean $\mu = m^2/N$. Via Chernoff bounds, we have: $\Pr[||\mathcal{A}_i| - \mu| \geq t] \leq 2e^{-t^2/3\mu}$

When $\text{RHS} \leq \delta$, we have $t \geq \sqrt{3 \ln(2/\delta)} \cdot \sqrt{\mu}$. Thus:

$$||\mathcal{A}_i| - m^2/N| < \sqrt{3 \ln(2/\delta)} \cdot \sqrt{m^2/N} \quad \square$$

Lemma 2. *If x is replaced by $z \in [N] \setminus \mathcal{S}'_i$ with probability $1 - m/N$, then $\mathcal{S}'_i \setminus \{x\} \cup \{z\}$ is uniformly distributed.*

Proof. Let R_x be the event a chosen index x got replaced. Let x^* be the guess based on the revealed set $\mathcal{S}^* \leftarrow \mathcal{S}'_i \setminus \{x\} \cup \{z\}$.

$$\begin{aligned} \Pr[x = x^*] &= \Pr[x^* \in \mathcal{S}^*] \cdot \Pr[\neg R_x] = \frac{1}{m} \cdot \frac{m}{N} = \frac{1}{N} \\ &= \Pr[x^* \in [N] \setminus \mathcal{S}^*] \cdot \Pr[R_x] = \frac{1}{N-m} \cdot \left(1 - \frac{m}{N}\right) = \frac{1}{N} \end{aligned}$$

In both cases, the probability of guessing x is $1/N$, so x is uniformly distributed in the revealed set $\mathcal{S}^*$. $\square$

Appendix B. Proof of Correctness (Efficient ESCAPE)

We prove two correctness points (1) and (2) below:

(1) For hint coverage, the client can recover $\text{DB}[\phi(x)]$ if there is $h_i = (\text{sk}_i, \text{sk}'_i, \rho_i)$ and $x \in \mathcal{A}_i \leftarrow \text{Set}(\text{sk}_i) \cap \text{TSet}(\text{sk}'_i)$.

By intersection definition, $\mathcal{A}_i = (a_1, \dots, a_n)$ samples a label $a_k \xleftarrow{\$} P_k \forall k \in [n]$, with $|P_k| = m^2$. Let $A_{x,k}$ denote $x \in P_k$ uncovered in M hints. As partitions are independent, probability of at least one uncovered label $x \in \text{DB}$:

$$\Pr \left[\bigcup_{k=1}^n \bigcup_{x \in P_k} A_{x,k} \right] \leq \sum_{k=1}^n \sum_{x \in P_k} \Pr[A_{x,k}] \leq nm^2 \left(1 - \frac{1}{m^2}\right)^M \leq \delta$$

With $\delta = 2^{-\lambda}$, we have $M \geq m^2 \ln N + \lambda m^2 \ln(2)$, and the client can find a hint for recovery if the hint size is:

$$M = \mathcal{O}(m^2(\lambda + \ln N))$$

Let $\mathcal{A}_1, \dots, \mathcal{A}_M$ be uniform random sets of size n from $[N]$. Given $x \in [N]$ and a hint index $I = \min\{j : x \in \mathcal{A}_j\}$, we replace $\mathcal{A}_I$ with a uniform $\mathcal{A}^* \supset x$. We want:

$$(\mathcal{A}_1, \dots, \mathcal{A}_{I-1}, \mathcal{A}^*, \mathcal{A}_{I+1}, \dots, \mathcal{A}_M) =_s (\mathcal{A}_1, \dots, \mathcal{A}_M)$$

Let $T_1, \dots, T_M$ be a fresh sample for uniform sets size n , and $J = \min\{j : x \in T_j\}$. In both processes, each set contains x with probability $1/m^2$. For any $k \in \{1, \dots, M\}$:

$$\Pr[I = k] = \Pr[J = k] = (1 - 1/m^2)^{k-1} \cdot 1/m^2$$

By independence of $T_1, \dots, T_m$ (target) and conditional on $I = k$ (hint), both distributions decompose slot-wise as:

- For $(1, \dots, k-1)$: uniform over sets without x (T_i vs $\mathcal{A}_i$)
- For k : uniform over sets cover x (T_k vs $\mathcal{A}^*$)
- For $(k+1, \dots, m)$: uniform over all sets size n (T_i vs $\mathcal{A}_i$)

With the same marginal for the index and the same conditional given it, the joint distributions agree:

$$(\mathcal{A}_1, \dots, \mathcal{A}_{I-1}, \mathcal{A}^*, \mathcal{A}_{I+1}, \dots, \mathcal{A}_M) \equiv (T_1, \dots, T_M).$$

The post-refresh hint distribution is identical to a fresh sample of M uniform hint sets from the offline.

(2) To correctly decrypt parity ρ_i per hint h_i , observing the transcript for query $Q_i = \{(r_1, \mathbf{c}_1), \dots, (r_n, \mathbf{c}_n)\}$:

$$\gamma_i \leftarrow \sum_{k=1}^n \text{PLHE.Eval}(\theta_k = \text{DB}[r_k, *], \mathbf{c}_k) = \sum_{k=1}^n f_{\theta_k}(\mathbf{c}_k)$$

Recall the correctness proof [5] for one evaluation $f_{\theta}(\mathbf{c})$ with $\mathbf{c} \leftarrow \text{Enc}(\mathbf{A}, \mathbf{k}, \mathbf{v}) := (\mathbf{A} \cdot \mathbf{k} + \mathbf{e} + \Delta \cdot \mathbf{v})$, $\Delta = \lfloor q/p \rfloor$.

By expanding the decryption transcript, we have:

$$v^* = \text{ct} - \mathbf{d}_{\theta} \cdot \mathbf{k} = \theta(\mathbf{A}\mathbf{k} + \mathbf{e} + \Delta\mathbf{v}) - \theta\mathbf{A}\mathbf{k} = \theta\mathbf{e} + \Delta\theta\mathbf{v}$$

With bit vector $\mathbf{v} = \boldsymbol{\mu}$ for row θ , we obtain $f_{\theta}(\boldsymbol{\mu}) = \theta\boldsymbol{\mu} = \lfloor v^* \rfloor_{\Delta} / \Delta$ if noise $\theta\mathbf{e}$ is below rounding bound $\Delta/2$.

Since $\mathbf{e} \leftarrow^{\$} \chi^m$, $\chi \sim D_{\mathbb{Z}, \sigma}$ and $\sigma^2 = s^2/2\pi$ for some $s > 0$, applying Lindner-Peikert and Banaszczyk tail bound:

$$\Pr[|\theta\mathbf{e}| \geq T \cdot s \cdot \|\theta\|] \leq 2 \exp(-\pi T^2)$$

With $T = \Delta/2s\|\theta\|$ and error probability δ , we have:

$$\Pr[|\theta\mathbf{e}| \geq \Delta/2] \leq 2 \exp\left(-\pi \cdot \frac{\Delta^2}{2s^2\|\theta\|^2}\right) \leq \delta$$

With norm $\|\theta\| \leq \sqrt{m\lfloor p/2 \rfloor^2}$ for θ containing $m \times \mathbb{Z}_p$ entries in range $[-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]$, and $s = \sigma\sqrt{2\pi}$, we have:

$$\lfloor q/p \rfloor \geq \sqrt{2}\sigma p \cdot \sqrt{m} \cdot \sqrt{\ln(2/\delta)}$$

For correctness of the sum $f_{\theta_1}(\mathbf{c}_1) + \dots + f_{\theta_n}(\mathbf{c}_n)$ from n evaluations with n vectors $(\theta_1, \dots, \theta_n) \in \mathbb{Z}_p^{n \times m}$, notice the concatenation of $(\theta_1, \dots, \theta_n)$ into one $\theta^\circ = (\theta_1 || \dots || \theta_n)$ with $\mathbf{c}^\circ = (\mathbf{c}_1 || \dots || \mathbf{c}_n)$ and corresponding noise $\mathbf{e}^\circ = (\mathbf{e}_1 || \dots || \mathbf{e}_n)$:

$$f_{\theta_1}(\mathbf{c}_1) + \dots + f_{\theta_n}(\mathbf{c}_n) = \langle \theta^\circ, \mathbf{c}^\circ \rangle$$

Using the bound from Lindner-Peikert and Banaszczyk with noise $(\theta^\circ \cdot \mathbf{e}^\circ)$, and the norm $\|\theta^\circ\|$ for concatenation with $\|\theta^\circ\|^2 = \sum_{k=1}^n \|\theta_k\|^2 \leq n \cdot m \cdot \lfloor p/2 \rfloor^2$, the proof carries and the decryption correctness of the sum $f_{\theta_1}(\mathbf{c}_1) + \dots + f_{\theta_n}(\mathbf{c}_n)$ on n rows $(\theta_1, \dots, \theta_n) \in \mathbb{Z}_p^{n \times m}$ holds when modulus p :

$$\lfloor q/p \rfloor \geq \sqrt{2}\sigma p \cdot \sqrt{nm} \cdot \sqrt{\ln(2/\delta)}$$

Appendix C.

Proof of Security (ESCAPE with Generic LHE)

We construct hybrid arguments as follows:

Hybrid 0. We define Hyb_0 as $\text{REAL}_{\Pi_{\mathcal{F}}, \mathcal{A}, \mathcal{Z}}$ with adversarial $\mathcal{A}$, environment $\mathcal{Z}$ and our generic construction.

Hybrid 1. We define Hyb_1 identical to Hyb_0 , except that the client replaces PRS evaluation $\text{Set}(\text{sk}_i)$ and $\text{Set}(\text{sk}'_i)$ by uniformly sampling $\mathcal{S}_i$ and $\mathcal{S}'_i$ from $\tilde{\mathcal{D}}$. The client discards all keys $(\text{sk}, \text{sk}'_i)$. Each hint becomes $h_i = (\mathcal{S}_i, \mathcal{S}'_i, \rho_i)$.

We argue that Hyb_0 and Hyb_1 are indistinguishable to $\mathcal{Z}$. In PRS, indices are sampled via $F : \{0, 1\}^\lambda \times [N] \rightarrow [N]$

$$(s_1, \dots, s_m) \leftarrow \{s_i \mid s_i = F_{\text{sk}}(i) \ \forall i \in [m]\}$$

By the standard PRF security assumption, we have:

$$|\Pr[\mathcal{A}^{F_{\text{sk}}(\cdot)} = 1 : \text{sk} \leftarrow^{\$} \{0, 1\}^\lambda] - \Pr[\mathcal{A}^{R_N(\cdot)}]| \leq \text{negl}(\lambda)$$

where $R_N(\cdot)$ is a truly random function on output domain $[N]$ and $\mathcal{A}$ has oracle access to $F_{\text{sk}}(\cdot)$ or $R_N(\cdot)$ but cannot see the key sk . Since $\text{Set}(\cdot)$ with m $F_{\text{sk}}(\cdot)$ invocations, are indistinguishable from $\tilde{\mathcal{D}}$ with m $R_N(\cdot)$ invocations (without replacement), $\mathcal{Z}$ cannot distinguish Hyb_0 from Hyb_1 .

Hybrid 2. We define Hyb_2 identical to Hyb_1 , except that for each $h_i = (\mathcal{S}_i, \mathcal{S}'_i, \rho_i)$, the client no longer obtains the parity $\rho_i = \sum_{a \in \mathcal{A}_i} \text{DB}[a]$ with $\mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}'_i$ via LHE. In Hyb_2 , the client invokes $\mathcal{S}_{\text{LHE}}$ to generate random LHE ciphertexts and sends them to the server, by which no ρ_i for $\mathcal{A}_i$ can be obtained. For correctness, the client invokes $\mathcal{F}$ to obtain $\text{DB}[x]$, discards all ρ_i , so each hint $h_i = (\mathcal{S}_i, \mathcal{S}'_i)$. By LHE security, true LHE ciphertexts are indistinguishable from $\mathcal{S}_{\text{LHE}}$ outputs, so Hyb_1 and Hyb_2 are indistinguishable to $\mathcal{Z}$.

Hybrid 3. We define Hyb_3 identical to Hyb_2 , except that in the online, the client finds $h_i = (\mathcal{S}_i, \mathcal{S}'_i)$, $x \in \mathcal{S}_i \cap \mathcal{S}'_i$ but discards the set $\mathcal{S}'_i$ paired with $\mathcal{S}_i$ from the offline. The client uniformly samples a new set $\mathcal{S}^* \in \tilde{\mathcal{D}}$ such that $x \in \mathcal{S}_i \cap \mathcal{S}^*$, then proceeds with $h_i = (\mathcal{S}_i, \mathcal{S}^*)$ and $\mathcal{A}^* \leftarrow \mathcal{S}_i \cap \mathcal{S}^*$ to create online query Q' . The hint refresh remains unchanged.

We argue that Hyb_2 and Hyb_3 are indistinguishable to $\mathcal{Z}$ as $(\mathcal{S}_i, \mathcal{S}'_i)$ and $(\mathcal{S}_i, \mathcal{S}^*)$ are identically distributed. Let $\mathcal{X} \subset \tilde{\mathcal{D}}$ be the subset of samples cover x , with $|\mathcal{X}| = \binom{N-1}{m-1}$ since $m-1$ remaining indices are from $[N-1]$.

Lemma 3. Given label x and $\mathcal{H} = (h_1, \dots, h_M)$, the chosen hint $h_i = (\mathcal{S}_i, \mathcal{S}'_i)$ preserves the uniformity of $\mathcal{S}'_i$ over $\mathcal{X}$, even for bounded hint size M :

$$\mathcal{D}(\mathcal{S}_i, \mathcal{S}'_i \mid x, M) \equiv \mathcal{D}(\mathcal{S}_i, \mathcal{S}^* \mid x, M), \quad \mathcal{S}^* \leftarrow^{\$} \mathcal{X}$$

Proof. We first analyze the joint probability for $(\mathcal{S}_i, \mathcal{S}'_i)$.

Let a_i be the probability that h_i is the first hint satisfying $x \in \mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}'_i$. Let $E_x^{(i)}$ be the sub-event that h_i matches and $E_x = \bigcup_i E_x^{(i)}$ be the event for at least one match. The matches are mutually exclusive with $\Pr[E_x] = \sum_{i=1}^M a_i = a$. Conditioned on $E_x^{(i)}$, for any specific pair $(U, V) \in \mathcal{X} \times \mathcal{X}$:

$$\Pr[\mathcal{S}_i = U, \mathcal{S}'_i = V \mid E_x^{(i)}] = 1/|\mathcal{X}|^2$$

Marginalizing over sub-event $E_x^{(i)}$ (for bounded size M):

$$\Pr \left[\begin{matrix} \mathcal{S}_i = U \\ \mathcal{S}'_i = V \end{matrix} \right] = \sum_{i=1}^M \Pr[E_x^{(i)}] \cdot \Pr \left[\begin{matrix} \mathcal{S}_i = U \\ \mathcal{S}'_i = V \end{matrix} \mid E_x^{(i)} \right] = \frac{a}{|\mathcal{X}|^2}$$

Finally, condition on the success of matching event E_x :

$$\Pr \left[\begin{matrix} \mathcal{S}_i = U \\ \mathcal{S}'_i = V \end{matrix} \mid E_x \right] = \frac{\Pr[\mathcal{S}_i = U, \mathcal{S}'_i = V]}{\Pr[E_x]} = \frac{a/|\mathcal{X}|^2}{a} = \frac{1}{|\mathcal{X}|^2}$$

For distribution $(\mathcal{S}_i, \mathcal{S}^*)$, since $\mathcal{S}_i$ is extracted from $(\mathcal{S}_i, \mathcal{S}'_i)$ and independent from the fresh $\mathcal{S}^*$, their joint probability is the product of their marginals:

$$\Pr[\mathcal{S}_i = U \mid E_x] \cdot \Pr[\mathcal{S}^* = V \mid E_x] = 1/|\mathcal{X}|^2 \quad \square$$

The lemma is proved. It follows that $\mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}'_i$ and $\mathcal{A}^* \leftarrow \mathcal{S}_i \cap \mathcal{S}^*$ are identically distributed. $\mathcal{Z}$ cannot distinguish if query Q' is created from Hyb₂ or Hyb₃.

Hybrid 4. We define Hyb₄ identical to Hyb₃, except that the client picks a random hint $h_j = (\mathcal{S}_j, \mathcal{S}'_j)$ independent to x in the online, and samples a new $\mathcal{S}^* \xleftarrow{\$} \tilde{\mathcal{D}}$ with $x \in \mathcal{S}^*$. The client uses $h_j = (\mathcal{S}_j, \mathcal{S}^*)$ to create query Q' . Since h_j is random, the client refreshes h_j with a new random hint.

Lemma 4. *Marginalized over the posterior of x on event E_{x^*} , the joint $(\mathcal{S}_i, \mathcal{S}^*)$ in Hyb₃ is statistically indistinguishable from $(\mathcal{S}_j, \mathcal{S}^*)$ in Hyb₄, except with probability δ .*

$$\sum_{x^* \in [N]} \Pr[x = x^* \mid \mathcal{H}] \cdot \mathcal{D}(\mathcal{S}_i, \mathcal{S}^* \mid \mathcal{H}, E_{x^*}) \approx_s \mathcal{D}(\mathcal{S}_j, \mathcal{S}^* \mid \mathcal{H})$$

Proof. Let event E_{x^*} denotes at least one hint $h_i = (\mathcal{S}_i, \mathcal{S}'_i)$ matches $x^* \in \mathcal{S}_i \cap \mathcal{S}'_i$. Notice that we only interact with the set of hints matching x^* , and $\mathcal{S}_i$ is statistically independent of the remaining hints in $\mathcal{H}$ that were generated but ignored.

To measure the uniformity of $\mathcal{D}(\mathcal{S}_i, \mathcal{S}^* \mid \mathcal{H}, E_{x^*})$ where $\mathcal{S}_i$ depends on E_{x^*} , we take the expectation over $\mathcal{H}$ and focus on the marginal probability of $\mathcal{S}_i$ for each $U \in \mathcal{X}$ that depends exclusively on the uniform randomness of $\mathcal{H}$:

$$\mathbb{E}_{\mathcal{H}} [\Pr[\mathcal{S}_i = U \mid \mathcal{H}, E_{x^*}]] = \Pr[\mathcal{S}_i = U \mid E_{x^*}]$$

When x is unknown, we want to sum over every possible index x^* to analyze the distribution. However, $\mathcal{H}$ can fail to cover some indices. Let $E^\circ = \bigcap_{x^* \in [N]} E_{x^*}$ the global event that $\mathcal{H}$ covers every label. We have:

$$\sum_{x^* \in [N]} \Pr[x = x^* \mid \mathcal{H}, E^\circ] \cdot \Pr[\mathcal{S}_i = U \mid E_{x^*}, E^\circ]$$

An index x^* can only be chosen if $\mathcal{H} = (h_1, \dots, h_M)$ covers x^* . Given event E° , every x^* is guaranteed to be

covered by $\mathcal{H}$. There is also no prior bias on the true target x . Thus, the posterior of x remains uniform:

$$\Pr[x = x^* \mid \mathcal{H}, E^\circ] = 1/N$$

For $\Pr[\mathcal{S}_i = U \mid E_{x^*}, E^\circ]$, $\mathcal{S}_i$ depends only on target U containing x^* . Conditioned on E_{x^*} , the global event E° (for other indices) is invariant to U , so conditional independence holds. We can drop the condition E° to simplify:

$$\Pr[\mathcal{S}_i = U \mid E_{x^*}, E^\circ] = \Pr[\mathcal{S}_i = U \mid E_{x^*}] = 1/|\mathcal{X}|$$

Next, we look at the term $\Pr[\mathcal{S}_i = U \mid E_{x^*}, E^\circ]$ to filter out the sum. Within E° , the local event E_{x^*} satisfied. However, if $x^* \notin U$, the event E_{x^*} fails since $\mathcal{S}_i$ must contain x^* . In that case, $\mathcal{S}_i \neq U$ and the conditional probability = 0. With $U = (u_1, \dots, u_m)$ containing m indices, exactly m out of $[N]$ indices in the sum survive:

$$\begin{aligned} & \sum_{x^* \in (u_1, \dots, u_m)} \Pr[x = x^* \mid \mathcal{H}, E^\circ] \cdot \Pr[\mathcal{S}_i = U \mid E_{x^*}, E^\circ] \\ &= m \cdot \frac{1}{N} \cdot \frac{1}{|\mathcal{X}|} = m \cdot \left(\frac{1}{N} \cdot \frac{1}{\binom{N-1}{m-1}} \right) = \frac{1}{\binom{N}{m}} \end{aligned}$$

This equals $\Pr[\mathcal{S}_i = U \mid E^\circ]$ since $\binom{N}{m}$ is the total number of all valid sets by choosing m indices from $[N]$. So $(\mathcal{S}_i \mid E^\circ)$ is uniform. Since $\mathcal{S}^*$ is independent of $\mathcal{H}$, the joint $\mathcal{D}(\mathcal{S}_i, \mathcal{S}^* \mid E^\circ)$ is determined by independent marginals.

In Hyb₄, $(\mathcal{S}_j, \mathcal{S}^*)$ are independent by construction, with $\mathcal{S}_j$ uniform over m^{2n} valid sets. It follows that:

$$\mathcal{D}(\mathcal{S}_i, \mathcal{S}^* \mid E^\circ) \equiv \mathcal{D}(\mathcal{S}_j, \mathcal{S}^* \mid E^\circ)$$

As a result, Hyb₃ and Hyb₄ can diverge only on $\neg E^\circ$, where $\mathcal{H}$ fails to cover some entries. We have:

$$\Delta(\mathcal{D}(\mathcal{S}_i, \mathcal{S}^* \mid \mathcal{H}), \mathcal{D}(\mathcal{S}_j, \mathcal{S}^* \mid \mathcal{H})) \leq \Pr[\neg E^\circ] \leq \delta \quad \square$$

Hence, $\mathcal{Z}$ cannot distinguish Hyb₃ from Hyb₄ except with negligible probability δ .

Hybrid 5. We define Hyb₅ identical to Hyb₄, except that in the online, the client samples a complete random $\mathcal{S}^* \xleftarrow{\$} \mathcal{D}^T$ independent of x . We argue that $\mathcal{Z}$ cannot distinguish if $\mathcal{S}^*$ priorly contains the index x , as the client performs Bernoulli $_{(m/N)}$ to replace the index x with z from $[N] \setminus \mathcal{S}^*$.

Note that Hyb₅ is identical to the simulator $\mathcal{S}$ in IDEAL. Thus, IDEAL and REAL are indistinguishable to $\mathcal{Z}$.

Appendix D.

Proof of Security (Efficient ESCAPE)

We construct a simulator $\mathcal{S}^+$ for our ESCAPE with Regev-based PLHE as in Figure 14. Given any matrix with $t = n\eta$ rows, we define the distribution $\mathcal{D}^* = [m]_1 \times \dots \times [m]_n$ with $[m]_k = [(k-1)\eta + 1, \dots, k\eta] \forall k \in [n]$ is the row indices for partition P_k . Thus, a uniform sample from $\mathcal{D}^*$ draws one row per $[m]_k$. Given DB with $t = n\eta$ rows, we define distribution $\mathcal{D} = P_1 \times \dots \times P_n$ to sample a set of labels using random row $r_k \in P_k$ with $r_k \xleftarrow{\$} [m]_k$:

$$(\mathcal{S} \xleftarrow{\$} \mathcal{D}) \stackrel{\text{def}}{=} \left[\bigcup_i \text{DB}(r_i), (r_1, \dots, r_n) \xleftarrow{\$} \mathcal{D}^* \right]$$

In other words, $\mathcal{D}$ outputs the union set of labels from random DB rows r_k . For $\text{DB}^\top$ with $t = nm\eta$ rows, we also have $\mathcal{D}^\top = P_1^\top \times \dots \times P_n^\top$. We define $\mathcal{S}' \stackrel{\$}{\leftarrow} \mathcal{D}^\top$ analogously:

$$(\mathcal{S}' \stackrel{\$}{\leftarrow} \mathcal{D}^\top) \stackrel{\text{def}}{=} \left[\bigcup_i \text{DB}^\top(r_i), (r_1, \dots, r_n) \stackrel{\$}{\leftarrow} \mathcal{D}^* \right]$$

$\bullet \mathcal{S}^+.\text{Offline}(\text{DB}):$ 1: $(\mathbf{A}_1, \dots, \mathbf{A}_{2t}) \stackrel{\$}{\leftarrow} (\mathbb{Z}_q^{m \times \eta})^{2t}$ 2: $(\mathcal{S}_1, \dots, \mathcal{S}_M) \stackrel{\$}{\leftarrow} \mathcal{D}^M$ 3: $Q_i = (\mathcal{S}_i, \mathbf{r}_i), \mathbf{r}_i \stackrel{\$}{\leftarrow} \mathbb{Z}_q^{nm\eta}$ 4: Output $(Q_1, \dots, Q_M)$	$\bullet \mathcal{S}^+.\text{Online}(\text{DB}):$ 1: $\mathcal{S}' \stackrel{\$}{\leftarrow} \mathcal{D}^\top$ and $\mathbf{r}' \stackrel{\$}{\leftarrow} \mathbb{Z}_q^{nm\eta}$ 2: $\mathcal{S} \stackrel{\$}{\leftarrow} \mathcal{D}$ and $\mathbf{r} \stackrel{\$}{\leftarrow} \mathbb{Z}_q^{nm\eta}$ 3: Output $Q' = (\mathcal{S}', \mathbf{r}')$ 4: Output $Q = (\mathcal{S}, \mathbf{r})$
--	---

Figure 14: The Simulator $\mathcal{S}^+$ for ESCAPE with Regev PLHE.

In Ideal, $\mathcal{S}$, $\mathcal{S}'$, and $(\mathcal{S}_1, \dots, \mathcal{S}_M)$ are random independent sets of labels. The vectors $\mathbf{r}, \mathbf{r}'$, and $(\mathbf{r}_1, \dots, \mathbf{r}_n)$ are random independent vectors from $\mathbb{Z}_q^{nm\eta}$. We define the hybrids:

Hybrid 0. We define Hyb_0 as $\text{REAL}_{\Pi, \mathcal{F}, \mathcal{A}, \mathcal{Z}}$ with adversarial $\mathcal{A}$, environment $\mathcal{Z}$ and our efficient instantiation.

Hybrid 1. We define Hyb_1 identical to Hyb_0 , except that the client replaces the PRS evaluation $\text{Set}(\text{sk}_i)$ and $\text{TSet}(\text{sk}_i')$ by uniformly sampling $\mathcal{S}_i$ and $\mathcal{S}_i'$ from $\mathcal{D}$ and $\mathcal{D}^\top$. The client discards $(\text{sk}, \text{sk}_i')$. Each hint becomes $h_i = (\mathcal{S}_i, \mathcal{S}_i', \rho_i)$.

We argue that Hyb_0 and Hyb_1 are indistinguishable to $\mathcal{Z}$. Recall that during set evaluation, the client samples random rows via $F: \{0, 1\}^\lambda \times [n] \rightarrow [m]$

$$(r_1, \dots, r_n) \leftarrow \{(k-1)n + F_{\text{sk}}(k) \mid k \in [n]\}$$

By the standard PRF security assumption, it follows that:

$$|\Pr[\mathcal{A}^{F_{\text{sk}}(\cdot)} = 1 : \text{sk} \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda] - \Pr[\mathcal{A}^{R_m(\cdot)}]| \leq \text{negl}(\lambda)$$

where $R_m(\cdot)$ is a truly random function on output domain $[m]$ and $\mathcal{A}$ has oracle access to $F_{\text{sk}}(\cdot)$ or $R_m(\cdot)$ but cannot see the key sk . Since n independent $R_m(\cdot)$ invocations is identical to $\mathcal{D}^*$, $\mathcal{D}$ and $\mathcal{D}^\top$ with $\mathcal{D}^*$ underlying is identical to $\mathcal{D}$ and $\mathcal{D}^\top$ with n independent $R_m(\cdot)$ invocations, and are indistinguishable from $\text{Set}(\cdot)$ and $\text{TSet}(\cdot)$ with n independent $F_{\text{sk}}(\cdot)$ invocations. $\mathcal{Z}$ cannot distinguish Hyb_0 with Hyb_1 .

Hybrid 2. We define Hyb_2 identical to Hyb_1 , except that to obtain parity ρ_i for hint $h_i = (\mathcal{S}_i, \mathcal{S}_i', \rho_i)$ with $\mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}_i'$, the client samples $\mu_k \stackrel{\$}{\leftarrow} \{0, 1\}^m \forall k \in [n]$, and receives from server a random parity ρ_i independent of $\mathcal{A}_i$ and cannot recover any entry. For correctness, the client invokes $\mathcal{F}$ to obtain $\text{DB}[x]$, and the client discards the random ρ_i .

We argue that Hyb_1 and Hyb_2 are indistinguishable to $\mathcal{Z}$ for any bit vectors $(\mu_1, \dots, \mu_n) \in \{0, 1\}^{m \times n}$. Recall in Hyb_1 , the parity $\rho_i = \sum_{a \in \mathcal{A}_i} \text{DB}[\phi(a)]$ is obtained by revealing the ciphertexts of n bit vectors $(\mu_1, \dots, \mu_n)$ indicate $\mathcal{A}_i \subset \mathcal{S}_i$.

Lemma 5. *If the PLHE scheme with LWE parameters (η, m, q, χ) is (T, ϵ) -secure, then any algorithm with running time $T = \mathcal{O}(nm)$, public input $(\mathbf{A}_1, \dots, \mathbf{A}_n) \leftarrow (\mathbb{Z}_q^{m \times \eta})^n$ and oracle access to $\text{PLHE.Enc}(\cdot, \mathbf{k}, \cdot)$ with $\mathbf{k} \leftarrow \mathbb{Z}_q^\eta$, has advantage at most ϵ in distinguishing the two distributions:*

$$\begin{aligned} \mathcal{D}_Q &= \{(\mathbf{A}_k, \text{PLHE.Enc}(\mathbf{A}_k, \mathbf{k}, \mu_k))\}_{k=1}^n \\ \tilde{\mathcal{D}} &= \{(\mathbf{A}_k, \mathbf{r}_k) : \mathbf{r}_k \leftarrow \mathbb{Z}_q^{nm\eta}\}_{k=1}^n \end{aligned}$$

Proof. We define the baseline distributions derived from the underlying LWE assumption that provides the (T, ϵ) -security:

$$\begin{aligned} \mathcal{D}_1 &= \{(\mathbf{A}_k, \mathbf{A}_k \cdot \mathbf{k} + \mathbf{e}_k) : \mathbf{e}_k \leftarrow \chi^m\}_{k=1}^n \\ \mathcal{D}_2 &= \{(\mathbf{A}_k, \mathbf{r}_k) : \mathbf{r}_k \leftarrow \mathbb{Z}_q^m\}_{k=1}^n \end{aligned}$$

Observe that distribution $\mathcal{D}_1$ defines n distinct LWE samples of dimension m , all sharing the same secret vector $\mathbf{k} \in \mathbb{Z}_q^\eta$. By vertically concatenating matrices $\mathbf{A}_1, \dots, \mathbf{A}_n$ into a single matrix $\mathbf{A}^* \in \mathbb{Z}_q^{nm \times \eta}$, and similarly concatenating the error vectors $\mathbf{e}_1, \dots, \mathbf{e}_n$ into $\mathbf{e}^* \leftarrow \chi^{nm}$, the sequence of pairs in $\mathcal{D}_1$ is perfectly isomorphic to a single standard LWE instance $(\mathbf{A}^*, \mathbf{A}^* \mathbf{k} + \mathbf{e}^*)$. Thus, the indistinguishability between $\mathcal{D}_1$ and $\mathcal{D}_2$ holds, provided the underlying LWE problem remains (T, ϵ) -secure on nm samples.

Now, consider a reduction algorithm $\mathcal{B}$ such that, given an input sequence $\{(\mathbf{A}_k, \mathbf{v}_k)\}_{k=1}^n \in (\mathbb{Z}_q^{m \times \eta} \times \mathbb{Z}_q^m)^n$, computes and outputs $\{(\mathbf{A}_k, \mathbf{v}_k + \lfloor q/p \rfloor \cdot \mu_k)\}_{k=1}^n$. We observe that:

- (1) If $\{(\mathbf{A}_k, \mathbf{v}_k)\} \stackrel{\$}{\leftarrow} \mathcal{D}_1$, the output of $\mathcal{B}$ is distributed identically to valid PLHE encryptions (exactly $\mathcal{D}_Q$).
- (2) If $\{(\mathbf{A}_k, \mathbf{v}_k)\} \stackrel{\$}{\leftarrow} \mathcal{D}_2$, the output of $\mathcal{B}$ is distributed identically to $\tilde{\mathcal{D}}$, since adding a deterministic encoding to vector $\mathbf{r}_k$ still results in a uniformly random vector.

As PLHE is (T, ϵ) -secure, any algorithm with runtime T has an advantage $< \epsilon$ in distinguishing $\mathcal{D}_1$ and $\mathcal{D}_2$. The reduction algorithm $\mathcal{B}$ performs n vector additions of size m with runtime $\mathcal{O}(nm)$. Thus, any algorithm distinguishing $\mathcal{D}_Q$ and $\tilde{\mathcal{D}}$ in $T - \mathcal{O}(nm)$ success with probability $< \epsilon$. $\square$

Hybrid 3. We define Hyb_3 identical to Hyb_2 , except that in the online, the client finds $h_i = (\mathcal{S}_i, \mathcal{S}_i')$, $x \in \mathcal{S}_i \cap \mathcal{S}_i'$ but discards the set $\mathcal{S}_i'$ paired with $\mathcal{S}_i$ from the offline. The client uniformly samples a new set $\mathcal{S}^* \in \mathcal{D}^\top$ such that $x \in \mathcal{S}_i \cap \mathcal{S}^*$, then proceeds with $h_i = (\mathcal{S}_i, \mathcal{S}^*)$ and $\mathcal{A}^* \leftarrow \mathcal{S}_i \cap \mathcal{S}^*$ to create online query Q' . The hint refresh remains unchanged.

Let $\mathcal{X}^\top \subset \mathcal{D}^\top$ and $\mathcal{X} \subset \mathcal{D}$ be the subset of samples cover x . We apply Lemma 3 with $|\mathcal{X}| = |\mathcal{X}^\top| = (m^2)^{n-1}$ to have that $(\mathcal{S}_i, \mathcal{S}_i')$ and $(\mathcal{S}_i, \mathcal{S}^*)$ are identically distributed. $\mathcal{A}_i \leftarrow \mathcal{S}_i \cap \mathcal{S}_i'$ and $\mathcal{A}^* \leftarrow \mathcal{S}_i \cap \mathcal{S}^*$ are identically distributed. $\mathcal{Z}$ cannot distinguish query Q' from Hyb_2 or Hyb_3 .

Hybrid 4. We define Hyb_4 identical to Hyb_3 , except that the client picks a random hint $h_j = (\mathcal{S}_j, \mathcal{S}_j')$ independent to x in the online, and samples a new $\mathcal{S}^* \stackrel{\$}{\leftarrow} \mathcal{D}^\top$ with $x \in \mathcal{S}^*$. The client uses $h_j = (\mathcal{S}_j, \mathcal{S}^*)$ to create query Q' . Since h_j is random, the client refreshes h_j with a new random hint.

With the size n for both sets $\mathcal{S}_i$ in Hyb_3 $\mathcal{S}_j$ in Hyb_4 and $|\mathcal{X}| = (m^2)^{n-1}$, we apply Lemma 4 to have that:

$$\Delta(\mathcal{D}(\mathcal{S}_i, \mathcal{S}^* \mid \mathcal{H}), \mathcal{D}(\mathcal{S}_j, \mathcal{S}^* \mid \mathcal{H})) \leq \delta$$

Hence, $\mathcal{Z}$ cannot distinguish Hyb_3 from Hyb_4 except with negligible probability δ .

Hybrid 5. We define Hyb_5 identical to Hyb_4 , except that in the online phase, the client samples $\mathcal{S}^* \stackrel{\$}{\leftarrow} \mathcal{D}^\top$ uniformly random, independent of x . $\mathcal{Z}$ cannot distinguish if $\mathcal{S}^*$ priorly contained x as the client performs Bernoulli($1/m$) to replace row τ containing x with row z from the same partition P_k .

Note that Hyb_5 is identical to the simulator $\mathcal{S}^+$ in IDEAL. Thus, IDEAL and REAL are indistinguishable to $\mathcal{Z}$.

Appendix E. Algorithmic Optimization

After processing (Figure 8-9), the server holds PLHE ciphertext $(\mathbf{d}^*, \gamma)$. ZipPIR [9] showed how to compress the response by computing $v^* = \gamma + \sum_i^\eta (q - \mathbf{d}^*[i]) \cdot \mathbf{k}[i]$ at the server side with a Paillier-encrypted secret $\mathbf{k}$ to remove the dimension η . The method is as follows:

$$\text{Enc}\left(\sum_i^\eta (q - \mathbf{d}^*[i]) \cdot \mathbf{k}[i]\right) = \prod_i^\eta \text{Enc}(\mathbf{k}[i])^{(q - \mathbf{d}^*[i])} \pmod{N_{\text{Pai}}^2}$$

Without PLHE modulus q for reduction, we can bound $v^* \leq q + \eta q$ when secret $\mathbf{k}$ is binary and $(\mathbf{d}^*, \gamma) \in \mathbb{Z}_q^\eta \times \mathbb{Z}_q$. We have many chunks v^* and for correctness, each chunk needs $\log q + \log(\eta + 1)$ bits of the Paillier plaintext, where the $\log(\eta + 1)$ upper bits denote overflow $\sum_i^\eta q \cdot \mathbf{k}[i]$ in mod q . We can reduce the bit allocation to $\log q$ by letting the overflow grows into next chunk lower-bit noise and further to $\log q'$ by switching $(\mathbf{d}^*, \gamma)$ to smaller modulus $q' < q$, only when γ has error $|e| < \Delta/4$ ([9, Sect 3.3]).

For fast computation, we can decompose $r = q - \mathbf{d}^*[i]$ as a sum of T digits $(b_0, \dots, b_{T-1})$, each having z bits. Thus:

$$\text{Enc}(\mathbf{k}[i])^r = \prod_{t=0}^{T-1} \text{Enc}(\mathbf{k}[i])^{b_t} \cdot 2^{tz} \pmod{N_{\text{Pai}}^2}$$

The same $\text{Enc}(\mathbf{k}[i])$ is used to compress $|\beta|/p$ chunks, so we precompute a lookup table per $\text{Enc}(\mathbf{k}[i])$ for all possible $(b_0, \dots, b_{T-1})$. With $z = 11$, 32-bit q and $\eta = 1024$, the lookup table is $4096 \cdot 2^{11} \cdot 3 \cdot 1024 = 3$ GiB in exchange for 3η 4096-bit modmul on N_{Pai}^2 , and we can batch-process multiple modmul via one intrinsic invocation.

Appendix F. Low-Level Optimization

We optimized ESCAPE, consisting of three main execution paths: (1) encryption of unit vector via Regev-based PLHE, (2) the inner product of a DB row and a PLHE ciphertext $\mathbf{c}$, and (3) the PLHE hint $(\mathbf{D} = \text{DB} \cdot \mathbf{A})$ row aggregation and compression. We leverage Paillier-based compression method from ZipPIR [9] to compress (remove dimension η) the sum $\mathbf{d}^*$ of subset rows from $\mathbf{D}$.

A. Hardware Configuration

We used Intel Ice Lake-SP Xeon Platinum 8360Y with 48 cores across two sockets. The platform supports VAES, AVX-512, and IFMA52, with 32 vector registers (zmm0 .. zmm31) of 512 bits and 8 opmask registers (k0 .. k7). VAES performs one AES round on four 128-bit blocks packed into one zmm register. IFMA52 performs unsigned $52 \times 52 \rightarrow 104$ bit multiply-accumulate operations, producing separate 52-bit low and high halves. Both VAES and IFMA52 incur 4-cycle latency, but each core can sustain a dispatch throughput of 1 instruction/cycle per port. For each core, port 5 handles

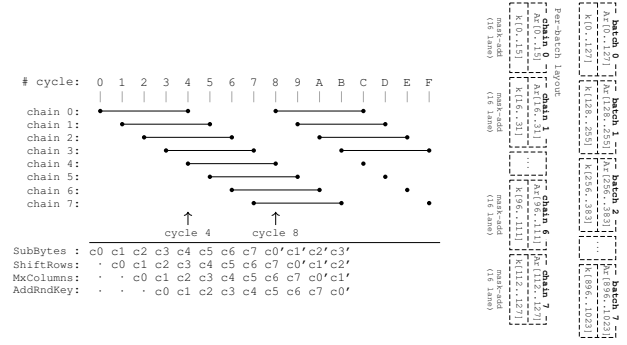


Figure 15: VAES Chain Scheduling and Data Layout.

IFMA52, port 0 handles both IFMA52 and VAES, which are the primary bottlenecks in our workload.

Our implementation utilized AVX-512, IFMA52, VAES intrinsics and out-of-order scheduler (per core) to dispatch micro-ops (independent of program order) to a small set of execution ports, which connect the scheduler to execution units (ALUs, vector, load/store) and accept at most one micro-op per cycle. Because the ports are physically separated, they can handle independent micro-ops in the same cycle. An execution port is saturated when the processor can supply it with independent ops every cycle to maximize the throughput. A port is free if no op was dispatched to it in a cycle.

B. Hardware-Vectorized PLHE Encryption

In our scheme, the client chose one DB row per partition and encrypts its column choice (unit vector) as an LWE ciphertext. The public matrix $\mathbf{A}$ for that DB row needs to be reproduced. Each row of $\mathbf{A}$ is $\eta = 1024$ random u32int numbers. We use AVX-512 VAES to pseudorandomly generate 8 batches of 128 random u32int numbers.

At a high level, AVX-512 VAES can produce four 128-bit random numbers after applying VAES instruction on a seed for 10 rounds. Each round takes 4 CPU cycles with distinct execution units correspond to SubBytes, ShiftRows, MixColumns, or AddRoundKey.

To increase throughput, we dispatch eight independent chains (illustrated by above figures) of VAES workloads on eight registers. After 10 cycles, we have applied one VAES round per register. After 100 cycles, all registers finished with 10 VAES rounds and we have 32 random numbers of 128 bits, viewed as 128 random u32int.

Notice each batch of 128 numbers (denoted as $\text{Ar}[0..127]$) will be used as part of the inner product with the secret key for PLHE encryption. Our inner product is simple since the secret $\mathbf{k}$ is binary: each term $\text{Ar}[i]$ contributes to the sum or nothing. We use AVX-512 masked-add instruction (vpadd with `__mmask16` opmask) to implement this conditional add.

The secret is preprocessed once into one 16-lane mask per 16 secret bits (a single AVX-512 mask register) so the mask is ready to be applied for the VAES 512-bit output after 10 rounds (no data leaves the registers). We cast each VAES

output into 16 numbers of u32int and do 16 conditional adds in one cycle. The inner product between the secret ($\eta = 1024$) and one A row consumes 8 batches, which are roughly 800 cycles and cost $0.3 \mu s$ per core at 3 GHz.

C. Karatsuba-Montgomery Multiplication

To optimize the multiplication of 4096-bit Paillier entries from the precomputed table, we implemented Montgomery Multiplication (IEEE Micro, 1996) using IFMA52 intrinsic, which provides $52 \times 52 \rightarrow 104$ -bit fused multiply-adds. Briefly, Montgomery helps us avoid long-division reduction while IFMA52 let us decompose 4096-bit multiplication onto 80 limbs of 52 bits. Thus, multiplying two 4096-bit Paillier entries costs 6400 IFMA52 multiply-adds on one 80×52 -bit limb structure. We need 3η such multiplications per chunk compression.

To reduce the 6400 multiply-adds per multiplication, we applied Karatsuba algorithm to decompose each 80-limb multiplication into three nested levels:

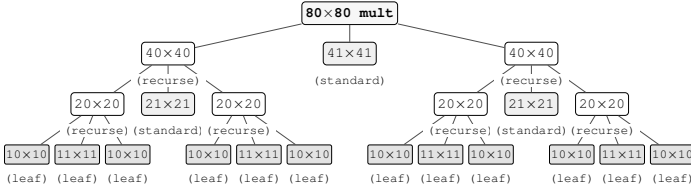


Figure 16: Triple Nested Karatsuba

Let u and v be two $2k$ -limb numbers, each limb is b bits. With $u = u_L + u_H \cdot b^k$ and $v = v_L + v_H \cdot b^k$, we compute the $4k$ -limb product $u \cdot v = z_0 + z_1 \cdot b^k + z_2 \cdot b^{2k}$ with:

$$\begin{aligned} z_0 &= (u_L \cdot v_L) & z_2 &= (u_H \cdot v_H) \\ z_1 &= (u_L + u_H)(v_L + v_H) - z_0 - z_2 \end{aligned}$$

With three nested Karatsuba levels, we can reduce the IFMA52 multiply-add count on 80 limbs by 40%, from 6400 down to 3847:

$$\text{Level 2 (one branch): } 10^2 + 11^2 + 10^2 = 321$$

$$\text{Level 1 (one branch): } 2 \cdot 321 + 21^2 = 1083$$

$$\text{Level 0 (one branch): } 2 \cdot 1083 + 41^2 = 3847$$

Note that per response chunk, 3η 4096-bit Paillier entries (on 80×52 -bit limbs) are waiting to be multiply-added. We utilize IFMA52 `vpadd52luq/vpadd52huq` to increase throughput by processing 8 independent 52-bit limbs from distinct chunks, packed into a 512-bit register as 8×64 -bit lanes. With two IFMA52 execution ports (0 and 5), we can dispatch two such 8-wide chains of 52-bit multiply-adds to process for 16 distinct chunks per cycle.