Towards Making Doubly-Efficient PIR Practical

Pan Xiao[1,2], Heng Zhang[1,2], Rending Ouyang[1,2], Cong Zhang[1,2], Jian Liu[1,2✉], Kui Ren[1,2], and Chun Chen[1,2]

[1] The State Key Laboratory of Blockchain and Data Security, Zhejiang University, {pan.xiao,heng.zhang,oyyy,0022090,liujian2411,kuiren,chenc}@zju.edu.cn
[2] Hangzhou High-Tech Zone (Binjiang) Blockchain and Data Security Research Institute, Hangzhou, China

Abstract. Doubly-efficient private information retrieval (DEPIR) enables sublinear per-query work (in the database size N) for both client and server, while requiring no client state. Despite its theoretical promise, single-server DEPIR exhibits a prohibitive concrete efficiency gap: for $N = 2^{23}$, the state-of-the-art construction (Eurocrypt '25) requires a 733TB server state and over 2^{37} online RAM/Disk reads, rending it infeasible to execute.

This paper advances single-server DEPIR towards practicality through a series of algorithmic innovations. It outperforms the state-of-the-art by achieving a reduction of up to 4 orders of magnitude in server state and 6 orders of magnitude in query time. In particular, for the same level database ($N = 2^{23}$), querying 5 461 elements in a single batch requires only 171GB server state and 2^{24} online RAM/disk reads, yielding a 112s total query time and a 21ms amortized query time.

Keywords: Private information retrieval · Doubly-efficient · Polynomial preprocessing.

1 Introduction

(Single-server) *private information retrieval* (PIR) is a cryptographic protocol that allows a client to retrieve an element $\mathsf{DB}[i]$ from a database $\mathsf{DB} \in \{0,1\}^N$ held by a server, without revealing the index $i \in [N]$. A trivial solution is for the client to download the entire DB; PIR achieves the same functionality with only sublinear communication in the database size N. However, the server must still access the entire DB to answer a query, making its computational cost at least linear in N. If the server skipped any location, it is clear that the client's query is different from that location, thereby violating privacy. This inherent overhead severely limits the practicality of PIR.

Doubly-efficient PIR (DEPIR) overcomes this limitation by preprocessing DB into a server-side data structure, enabling sublinear communication and computation. The concept was first introduced by Beimel, Ishai and Malkin [5], and two initial constructions were concurrently proposed by Canetti et al. [11] and

✉ Jian Liu is the corresponding author.

Boyle et al. [7], both requiring client-specific preprocessing. A significant breakthrough came from Lin, Mook, and Wichs (LMW) [33], who proposed the first DEPIR scheme supporting global preprocessing. However, as noted by Zhou et al. [49], this construction is "far from practical or even implementable". Okada et al. (OPPW) [40,41] refuted the second part of this claim, demonstrating that implementation barriers can be overcome through appropriate optimizations.

The LMW blueprint for DEPIR begins with the server interpolating DB as an m-variate polynomial $f(X_1, \ldots, X_m)$ of total degree bounded by D, transforming database indexing into multivariate polynomial evaluation. The client then encrypts the query variables $[x_1, \ldots, x_m]$ using *algebraic somewhat homomorphic encryption* (ASHE), enabling the server to directly evaluate f on the ciphertexts (over a ring R) as if operating on plaintexts. To achieve sublinear cost, the server preprocesses $f(X_1, \ldots, X_m)$ over R into a data structure that supports sublinear-time evaluation in the number of monomials, i.e., N. Specifically, let M denote the maximum value that a "lifted"[3] version of $f(X_1, \ldots, X_m)$ can take, and let $[p_1, \ldots, p_t]$ be a vector of primes s.t. $\prod_{i=1}^{t} p_i \geq M$; this allows $f(X_1, \ldots, X_m)$ over R to be evaluated modulo each p_i separately, with the result reconstructed via the *Chinese Remainder Theorem* (CRT). The server then constructs t tables, where the i-th table stores all p_i^m possible evaluations of $f(X_1, \ldots, X_m)$ modulo p_i, effectively transforming homomorphic polynomial evaluation into simple table lookups. If the tables are too large to store, the evaluation of f modulo each p_i can be recursively reduced to evaluations modulo a set of even smaller primes.

Despite the theoretical promise of DEPIR, it exhibits a prohibitive concrete efficiency gap, e.g., OPPW becomes infeasible to execute when $N = 2^{23}$ (cf. Table 3 in [40]). The practical challenges are summarized as follows:

Challenge 1: Interpolating DB as a multivariate polynomial of total degree bounded by D takes $O(N^2)$ time,[4] which is infeasible for large N.

Challenge 2: An ASHE ciphertext consists of two elements $(a, b) \in R^2$, doubling the number of variables from m to $2m$.[5]

Challenge 3: Choosing a small m (e.g., 4–6) is preferable, but requires a larger D to satisfy $\binom{D+m}{m} \geq N$, which increases M and thus the number of required primes, consequently raising both storage and online time.

Challenge 4: Despite recursive reductions of the primes, we prove that the largest prime ultimately stabilizes at approximately $O(D \cdot \log D)$ (cf. Theorem 4), establishing a storage barrier that CRT cannot overcome.

[3] The coefficients of f are lifted from R to $\mathbb{Z}$.

[4] LMW chooses f with *individual* degree bounded by d, achieving $O(Nm \log d)$ interpolation time, but this supports only d^m points. In contrast, OPPW [40,41] chooses f with total degree bounded by D, enabling interpolation over $\binom{D+m}{m}$ points—an improvement by a factor exponential in m, at the cost of $O(N^2)$ interpolation time.

[5] OPPW [40] mitigates this issue by employing a relaxed variant of RLWE, termed {0,1}-CRT RLWE.

Our Contributions. In this paper, we present a novel DEPIR scheme that overcomes the above challenges through the following algorithmic innovations:

Fast interpolation with total degree. To overcome "Challenge 1", we propose a *novel* interpolation algorithm that achieves $O(N \cdot \log N)$ time complexity to interpolate DB as a multivariate polynomial of total degree bounded by D. In particular, we reduce the problem of polynomial interpolation with total degree to the *global postage-stamp problem* (cf. Section 3). This leads to upto $2\,760\,300\times$ reduction in preprocessing time (cf. Table 2).

ASHE with a single ring element. To overcome "Challenge 2", we adopt an approach inspired by SimplePIR [24] that samples a fresh encryption key for each query, allowing us to fix one ring element across all queries (cf. Section 4). This reduces the number of variables from $2m$ to m, achieving a 2^m-fold storage reduction compared to $\{0,1\}$-CRT RLWE in OPPW.

Polynomial decomposition. To overcome "Challenge 3", we propose a *novel* algorithm that interpolates DB as $\sum_{i=0}^{\ell-1} f_i(X_1, \ldots, X_m) \cdot X_{m+1}^i$, where each f_i has a smaller total degree D' satisfying $\binom{D'+m}{m} \geq N/\ell$. Crucially, the multiplications involving the encrypted powers X_{m+1}^i can be carried out efficiently in the online phase (cf. Section 5). This reduces the input of interpolation by ℓ-times, while introducing $\ell + L + 2H - 5$ (where $L \cdot H = \ell$) homomorphic multiplications online. This tradeoff is valuable when DB is too large to preprocess.

Polynomial preprocessing with p-adic numbers. To overcome "Challenge 4", we propose a *novel* algorithm that decomposes M into a set of p-adic numbers p^k (instead of primes)[6], circumventing the $O(D \cdot \log D)$ barrier for the largest prime (cf. Section 6). This leads to $3\times$ reduction in the largest prime, yielding $12.5\times$ storage reduction and $2.8\times$ fewer table reads (see Table 3).

Experimental results show that our DEPIR achieves a reduction of up to 4 orders of magnitude in server state and 6 orders of magnitude in query time. In particular, for $N = 2^{23}$, querying 5 461 elements in a single batch requires 171GB of server state and 2^{24} online RAM/disk reads, yielding a 112s total query time and a 21ms amortized query time.

We summarize our contributions as follows:

- We establish a lower bound on the prime size when CRT is applied recursively (cf. Theorem 4), which we believe offers an important insight for polynomial preprocessing.
- We prove new asymptotic results for polynomial preprocessing (cf. Lemma 3, Theorem 5).
- We propose a novel algorithm for multivariate polynomial interpolation with total degree (cf. Theorem 1).
- We provide a systematic benchmark, and the results show that our protocol achieves a $7\,505\times$ reduction in server state and a $3\,481\,944\times$ reduction in query time (cf. Table 4).

[6] OPPW [40] also considers p-adic, but only for power-of-two cases, and treats it separately from CRT.

2 Preliminaries

This section outlines required notations, definitions and assumptions.

2.1 Notations

We use λ to denote the computational security parameter. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and we use $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$. We use bold letters to denote vectors (e.g., $\mathbf{x}$). We use both x_i and $\mathbf{x}[i]$ to denote the i-th coordinate of $\mathbf{x}$. To enhance readability, we provide a summary of the notations in Appendix A.

2.2 Ring Learning with Errors

The *learning with errors* (LWE) assumption [42] is parameterized by (n, Q, χ), where n is the lattice dimension (typically a power of 2), $Q \gg p$ is the modulus, and χ over $\mathbb{Z}_Q$ is a discrete Gaussian distribution. The decision version of the assumption asserts that the following distributions are computationally indistinguishable:

$$(\mathbf{a}, \mathbf{s} \cdot \mathbf{a} + \epsilon) \approx_c (\mathbf{a}, r),$$

where $\mathbf{s} \in_\$ \mathbb{Z}_Q^{1\times n}$, $\mathbf{a} \in_\$ \mathbb{Z}_Q^{n\times 1}$, $\epsilon \in \chi$, and $r \in_\$ \mathbb{Z}_Q$.

Ring learning with errors (RLWE) is a variant of LWE, defined over the cyclotomic ring $R_Q = \mathbb{Z}_Q[Z]/(Z^n + 1)$. To facilitate our proof, we present the RLWE definition following the style of OPPW (cf. Definition 2.6 in [40]).

Definition 1 (RLWE). *Let R be a number ring, q, m be positive integers, $\mathcal{B}$,$\mathcal{E}$ distributions over R_q, and $\mathcal{S}$ a distribution over the unit group R_q^*. We say that an algorithm solves $\mathsf{RLWE}(R, q, \mathcal{B}, \mathcal{E}, \mathcal{S}, m)$ if it has non-negligible advantage in distinguishing the following two scenarios:*

- *In* Scenario 1*, on input $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_m] \in R_p^m$, the oracle returns $(\mathbf{a}, \mathbf{b}) \in R_q^m \times R_q^m$, where $s \leftarrow \mathcal{S}$, $\{b_i \leftarrow \mathcal{B},\ \epsilon_i \leftarrow \mathcal{E}\}_{i\in[m]}$, and*

$$a_i := s^{-1}(b_i - \mu_i - \epsilon_i) \qquad \forall i \in [m].$$

- *In* Scenario 2*, on input $\boldsymbol{\mu} = [\mu_1, \ldots, \mu_m] \in R_p^m$, the oracle returns $(\mathbf{a}, \mathbf{b}) \in R_q^m \times R_q^m$, where $\{b_i \leftarrow \mathcal{B},\ a_i \xleftarrow{\$} R_q\}_{i\in[m]}$.*

We use an RLWE variant in which the public vector $\mathbf{b}$ is sampled once and reused across oracle calls. We establish its hardness under the RLWE assumption in Definition 1 using a hybrid argument similar to that of SimplePIR [24]; see Appendix D.

2.3 Polynomial Evaluation with Preprocessing

Kedlaya and Umans (KU) [28] show how to preprocess a multivariate polynomial $f \in R[X_1, \ldots, X_m]$ into a data structure that allows evaluation on any input $\mathbf{x} \in R^m$ in time sublinear in the number of monomials of f. The basic idea is to build a lookup table containing $\{\mathbf{x}, f(\mathbf{x})\}_{\mathbf{x} \in R^m}$, which already suffices when $|R|$ is small. For larger rings, KU can produce a compact data structure of size $\mathsf{poly}(\log |R|)^m$ or even $\mathsf{poly}(\log \log |R|)^m$.

Consider $R = \mathbb{Z}_q$. If q is highly smooth, the idea above combines naturally with the Chinese Remainder Theorem (CRT). Specifically, if $q = \prod_{i=1}^{t} p_i$, then $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_t}$, and it suffices to store t tables, each containing $\{\mathbf{x}, f(\mathbf{x})\}_{\mathbf{x} \in \mathbb{Z}_{p_i}^m}$. For a generic q, one can choose primes $[p_1, \ldots p_t]$ such that $P = \prod_{i=1}^{t} p_i \geq M$, where M is the maximum absolute value attained by a "lifted" version of f (i.e., the coefficients of f are lifted from $\mathbb{Z}_q$ to $\mathbb{Z}$). This ensures that the final CRT-reconstructed value modulo P is exactly the integer result (i.e., no wrap-around occurs). By choosing the t smallest primes for a sufficiently large t, the resulting data structure achieves a size of $\mathsf{poly}(\log q)^m$. If some p_i is still too large, KU applies the procedure recursively: each evaluation of f over $\mathbb{Z}_{p_i}$ can itself be reduced to t_i' evaluations over smaller primes $\mathbb{Z}_{p_{ij}'}$. This further reduces the storage complexity to depend only on $\log \log q$, at the cost of a higher evaluation time, bounded by $t \cdot \max_i t_i'$. The process can be iterated further, continually trading storage size for evaluation time. However, we prove that the largest prime ultimately stabilizes (cf. Theorem 4).

Notice that KU requires no special properties of the underlying ring. LMW [33] leverages this to construct a "sufficiently homomorphic" map between $\mathbb{Z}_q$ and $\mathbb{Z}_Q[Y, Z]/(f(Y), g(Z))$ for a sufficient large q and monic polynomials f and g. However, this comes at the cost of a tremendously large q. OPPW [41] circumvents this issue using *double-CRT*, which we introduce in the next subsection.

2.4 Double-CRT

For a ring $R_Q = \mathbb{Z}_Q[Z]/(Z^n + 1)$, if $Q = \prod_{i=1}^{r} q_i$ with $\{q_i \equiv 1 \pmod{2n}\}_{i \in [r]}$, the ideal $(Z^n + 1) \subseteq \mathbb{Z}_{q_i}[Z]/(Z^n + 1)$ decomposes into n different degree-1 prime ideals $\{(Z - \rho_i^{2j-1})\}_{j \in [n]}$ with $\rho_i^{2n} \equiv 1 \pmod{q_i}$. This yields the isomorphism

$$R_Q = \mathbb{Z}_Q[Z]/(Z^n + 1) \cong \bigoplus_{i=1}^{r} \bigoplus_{j=1}^{n} \mathbb{Z}_{q_i}.$$

To evaluate a polynomial $G \in R_Q[X_1, \ldots, X_m]$, we first leverage the isomorphism

$$R_Q[X_1, \ldots, X_m] \cong \bigoplus_{i=1}^{r} R_{q_i}[X_1, \ldots, X_m]$$

to decompose G into r polynomials $\{G'_i \in R_{q_i}[X_1, \ldots, X_m]\}_{i\in[r]}$. Then, for each $i \in [r]$ we employ the isomorphism

$$R_{q_i}[X_1, \ldots, X_m] \cong \bigoplus_{j=1}^{n} R_{q_i}[X_1, \ldots, X_m]/(Z - \rho_i^{2j-1}) \cong \bigoplus_{j=1}^{n} \mathbb{Z}_{q_i}[X_1, \ldots, X_m]$$

to further decompose G'_i into n polynomials $\{g_{i,j} \in \mathbb{Z}_{q_i}[X_1, \ldots, X_m]\}_{j\in[n]}$. We could use KU to build a data structure for each $g_{i,j}$ (cf. Section 2.3).

Given an input $[\mu_1, \ldots, \mu_m] \in R_Q^m$, we evaluate $G(\mu_1, \ldots, \mu_m)$ as follows:

1. For each $k \in [m]$ and $i \in [r]$, compute $\mu'_{k,i} := \mu_k \pmod{q_i} \in R_{q_i}$.
2. For each $k \in [m]$, $i \in [r]$ and $j \in [n]$, compute $x_{k,i,j} := \mu'_{k,i} \pmod{Z - \rho_i^{2j-1}} \in \mathbb{Z}_{q_i}$.
3. For each $i \in [r]$ and $j \in [n]$, compute $y_{i,j} := g_{i,j}(x_{1,i,j}, \ldots, x_{m,i,j}) \in \mathbb{Z}_{q_i}$.
4. For each $i \in [r]$, use $[y_{i,1}, \ldots, y_{i,n}]$ to recover $G'_i \in R_{q_i}[X_1, \ldots, X_m]$ via CRT.
5. Use $[G'_1, \ldots, G'_r]$ to recover $G \in R_Q[X_1, \ldots, X_m]$ via CRT.

3 Fast Interpolation with Total Degree

In this section, we show how we interpolate DB as an m-variate polynomial of total degree bounded D in $O(N \cdot \log N)$ time. To the best of our knowledge, no prior work has addressed this specific problem: while polynomial interpolation is classical, interpolation with total-degree constraints appears unexplored; OPPW's adoption of $O(N^2)$ interpolation algorithm strongly indicates that no faster method existed.

We leverage the *global postage-stamp problem* (GPSP) to reduce multivariate polynomial interpolation to a univariate problem. We begin by reviewing univariate polynomial interpolation and GPSP.

Fact 1 (NTT-based univariate polynomial interpolation [1]). *Let $\mathbb{Z}_p$ be a prime field of order p satisfying $p \equiv 1 \pmod{N}$. Then there exists a primitive N-th root of unity $\omega \in \mathbb{Z}_p$, and we denote $\{\omega_i = \omega^i\}_{i\in[N]}$. For any vector $[c_1, \ldots, c_N] \in \mathbb{Z}_p^N$, the Number Theoretic Transform (NTT) is defined as*

$$\left\{ y_i = \sum_{j=1}^{N} c_j \, \omega_i^{j-1} \right\}_{i\in[N]}.$$

Both the transform and its inverse can be computed in $O(N \cdot \log N)$ time via a divide-and-conquer algorithm, analogous to the Fast Fourier Transform.

Consequently, there exists an interpolation algorithm which, given $[\omega_1, \ldots, \omega_N] \in \mathbb{Z}_p^N$ and any $[y_1, \ldots, y_N] \in \mathbb{Z}_p^N$ with $N \le p$, outputs a univariate polynomial $f \in \mathbb{Z}_p[X]$ of degree less than N such that $\{f(\omega_i) = y_i\}_{i\in[N]}$. The algorithm runs in time $O(N \cdot \log N)$.

Table 1. Representative solutions for GPSP.

m	D	$N_{m,D,\varrho}$	ϱ	m	D	$N_{m,D,\varrho}$	ϱ
	7	165	1 5 24 37		7	345	1 8 11 64 102
	8	234	1 6 25 65		8	512	1 9 15 78 115
	9	326	1 5 34 60		9	797	1 9 23 108 181
4	22	4368	1 12 121 561	5	10	1055	1 8 27 119 194
	40	33981	1 30 343 2634		64	1782370	1 52 705 13100 99644
	70	254900	1 52 959 12172		65	1888725	1 48 698 12988 111755
	71	269519	1 55 1014 12870		66	2036874	1 48 746 13252 113747

Definition 2 (Global postage-stamp problem (GPSP)). *Given $m, D \in \mathbb{N}$, define*

- *Extremal (postage-stamp) basis:* $\varrho = [e_1 = 1 < e_2 < \cdots < e_m] \in \mathbb{N}^m$;
- *Coefficient vectors:* $\mathcal{C}_{m,D} = \left\{ [c_1, \ldots, c_m] \in \mathbb{N}^m \;\middle|\; \sum_{i=1}^{m} c_i \leq D \right\}$;
- *Extremal set:* $\Phi_{m,D,\varrho} = \left\{ \sum_{i=1}^{m} c_i e_i \;\middle|\; [c_1, \ldots, c_m] \in \mathcal{C} \right\}$,
- *Extremal value:* $N_{m,D,\varrho} = \min\{k \in \mathbb{N} \mid k \notin \Phi_{m,D,\varrho}\}$, *i.e., the smallest positive integer not in* $\Phi_{m,D,\varrho}$. *Notice that* $\forall k' \in [N_{m,D,\varrho} - 1]$, $k' \in \Phi_{m,D,\varrho}$.

Given m and D, GPSP(m, D) *asks to determine an extremal basis ϱ that maximizes the extremal value $N_{m,D,\varrho}$.*

Although GPSP(m, D) is NP-hard in general, we restrict our consideration to small values of m where solutions can be computed via exhaustive search [12]. Representative solutions are listed in Table 1. Given m and D, we could retrieve the extremal basis ϱ and the extremal value $N_{m,D,\varrho}$ from this table.

Theorem 1 (Multi-variate polynomial interpolation). *Let $\mathbb{Z}_p$ be a prime field of order p satisfying $p \equiv 1 \pmod{N}$, and let $m \in \mathbb{N}$. Given any set $\{y_i \in \mathbb{Z}_p\}_{i \in [N]}$, there exists an algorithm that constructs an m-variate polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_m]$ with the following properties:*

- *For every $i \in [N]$, there exists $[x_1, \ldots, x_m] \in \mathbb{Z}_p^m$ s.t. $f(x_1, \ldots, x_m) = y_i$.*
- *The total degree of f satisfies* $\deg(f) \leq D = (1 + o(1)) \cdot \alpha \cdot m \cdot N^{\frac{1}{m}}$, *where $\alpha < 1$ is a constant.*
- *The algorithm runs in time $O(N \cdot \log N)$.*

Proof. For given values of N and m, the minimal integer D (and its corresponding extremal basis $\varrho = [e_1 = 1 < e_2 < \cdots < e_m]$) satisfying $N_{m,D,\varrho} > N$ can be found in Table 1. This implies that all $k \in [N]$ belong to the extremal set $\Phi_{m,D,\varrho}$. For each $k \in [N]$, we compute the corresponding coefficient vector $[c_1^{(k)}, \ldots, c_m^{(k)}]$, which satisfies:

$$\sum_{j=1}^{m} c_j^{(k)} \leq D \text{ and } \sum_{j=1}^{m} c_j^{(k)} e_j = k.$$

Using dynamic programming, we could compute all N coefficient vectors in $O(mN)$ time, which can be reused across all runs.

Let $\omega \in \mathbb{Z}_p$ be a primitive N-th root of unity, and define $\{\omega_i = \omega^i\}_{i\in[N]}$. For each $i \in [N]$, let $\Omega_i = [\omega_i^{e_1}, \ldots, \omega_i^{e_m}] \in \mathbb{Z}_p^m$. Define the polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_m]$ as:

$$f(X_1, \ldots, X_m) = \sum_{k=1}^{N} C_k g_k(X_1, \ldots, X_m), \text{ where } g_k(X_1, \ldots, X_m) = \prod_{j=1}^{m} X_j^{c_j^{(k)}}.$$

For each Ω_i, we have

$$g_k(\Omega_i) = \prod_{j=1}^{m} (\omega_i^{e_j})^{c_j^{(k)}} = \omega_i^{\sum_{j=1}^{m} c_j^{(k)} e_j} = \omega_i^k.$$

Thus, evaluating f at Ω_i yields:

$$f(\Omega_i) = \sum_{k=1}^{N} C_k g_k(\Omega_i) = \sum_{k=1}^{N} C_k \omega_i^k = h(\omega_i),$$

where $h(X) = \sum_{k=1}^{N} C_k X^k$ is a univariate polynomial. Let $h'(X) = X^{-1} \cdot h(X) = \sum_{k=1}^{N} C_k X^{k-1}$. By Fact 1, the coefficients $[C_1, \ldots, C_N]$ of h' (and hence f) can be computed from the pairs $\{(\omega_i, y_i \cdot \omega_i^{-1})\}_{i\in[N]}$ in time $O(N \cdot \log N)$.

The total degree of f is bounded by D, since each monomial g_k has degree $\sum_{j=1}^{m} c_j^{(k)} \leq D$. As shown in [29], D admits the asymptotic expression:

$$D = (1 + o(1)) \cdot \alpha \cdot m \cdot N^{\frac{1}{m}},$$

where the constant α depends on m. □

Concretely, $\alpha \in (0.8077, 0.8401)$ when $m = 4$; $\alpha \in (0.6201, 0.7244)$ when $m = 5$ and $N \leq 2^{21}$ [12]. We remark that any new progress on the GPSP can be directly incorporated into our interpolation algorithm.

In the context of DEPIR, the server constructs an m-variate polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_m]$ with total degree bounded by D by applying Theorem 1 to its database $\{y_i \in \mathbb{Z}_p\}_{i\in[N]}$, and publishes ω and the extremal basis $\varrho = \{e_j\}_{j\in[m]}$. To query y_i, the client computes

$$\left\{x_j := (\omega^i)^{e_j}\right\}_{j\in[m]},$$

whereupon the server and client privately evaluates f at $[x_1, \ldots, x_m]$.

Table 2 compares the interpolation time of three algorithms. LMW [33] interpolates polynomials with individual degree bounded d, achieving reasonable interpolation time, at the cost of doubling the total degree. OPPW [40,41] adopts an $O(N^2)$ interpolation algorithm, the cost of which quickly becomes prohibitive,

Table 2. Comparison of different interpolation methods.

N	2^{12}	2^{15}	2^{18}	2^{21}
p	65 537	786 433	5 767 169	104 857 601
m	4	4	4	5
LMW: individual degree interpolation in $O(N \cdot m \cdot \mathsf{poly}\log d)$ [33]				
d	8	14	23	19
D	32	64	92	95
Interpolation time	16.2ms	182.4ms	1.5s	14.7s
OPPW: total degree interpolation in $O(N^2)$ [40, 41]				
D	16	28	48	45
Interpolation time	622.3ms	35.4s	2 207s	128 630s
Our work: total degree interpolation in $O(N \cdot \log N)$				
D	23	40	71	67
Interpolation time	0.1ms	0.6ms	5.1ms	46.6ms

e.g., interpolating a database of size $N = 2^{21}$ takes nearly 36 hours. In contrast, our method completes the same task in only 46.6ms, with a slightly larger D. Notice that our method is also significantly faster than that of LMW. The reason is that LMW employs a general interpolation algorithm supporting arbitrary $\mathbf{x}_i \in \mathbb{Z}_p^m$ in the input set $\{\mathbf{x}_i, y_i\}_{i\in[N]}$, whereas our method restricts each $\mathbf{x}_i$ to $[(\omega^i)^{e_1}, \ldots, (\omega^i)^{e_m}]$. This restriction does not compromise the applicability of our approach in the context of DEPIR and significantly enhances efficiency.

4 ASHE with a Single Ring Element

Recall that the server needs to evaluate an m-variate polynomial f on $[x_1, \ldots, x_m]$, where each x_i is encrypted as $(a_i, b_i) \in R_Q^2$ with $R_Q = \mathbb{Z}_Q[Z]/(Z^n + 1)$. This doubles the number of variables from m to $2m$. To address this, we adopt an approach inspired by SimplePIR [24], fixing one ring element across all queries to maintain the number of variables at m. The key insight from SimplePIR is that (R)LWE-based encryption remains secure when a is reused for polynomially many messages, provided that each ciphertext employs an independent secret and error.

Fig. 1 details our ASHE scheme. To align with the context of DEPIR, the encryption algorithm encrypts m plaintexts using a single secret (cf. Steps 1–2 of $\mathsf{ASHE.Enc}$). Security is preserved, as each ciphertext uses a unique b (cf. Step 5 of $\mathsf{ASHE.Setup}$). Inspired by OPPW [40], we fix b instead of a, enabling the elimination of one variable during the evaluation of f (cf. Steps 2–4 of $\mathsf{ASHE.Eval}$).

Since our DEPIR protocol (cf. Section 7.1) leverages SIMD parallelism, we formulate our ASHE scheme to support SIMD operations. Specifically, each input plaintext μ provided to $\mathsf{ASHE.Enc}$ is an SIMD-style polynomial encoded by $\mathsf{ASHE.Encode}$. Let $\{\omega_i\}_{i\in[n]}$ represent the n-th roots of

$\mathsf{ASHE.Setup}(1^\lambda, D, p, N) \to n, Q, \chi, R_p, R_Q, R_R, [b_1, \ldots, b_m]$:

1. Set the gap parameter $\Delta := D \log p + \log N + \log p + 1$.
2. Choose $n = \mathsf{poly}(\lambda, \Delta)$ and $Q = \lambda^{\mathsf{poly}(\Delta)}$.
3. Choose a discrete Gaussian distribution over R: $\chi = \chi(\lambda)$.
4. Define the rings $R_p = \mathbb{Z}_p[Z]/(Z^n + 1)$, $R_Q = \mathbb{Z}_Q[Z]/(Z^n + 1)$ and $R_R = R_Q[Y]/(Y^{D+1} + 1) \cong \mathbb{Z}_Q[Y, Z]/(Z^n + 1, Y^{D+1} + 1)$.
5. Sample $\{b_i \xleftarrow{\$} R_Q\}_{i \in [m]}$.

$\mathsf{ASHE.Encode}([x_1, \ldots, x_n]) \to \mu \in R_p$:
On input $[x_1, \ldots, x_n] \in \mathbb{Z}_p^n$, output $\mu \leftarrow \mathsf{NTT}^{-1}([x_1, \ldots, x_n]) \in R_p$.

$\mathsf{ASHE.Enc}([b_1, \ldots, b_m], [\mu_1, \ldots, \mu_m]) \to [\tilde{\mu}_1, \ldots, \tilde{\mu}_m] \in R_R^m, s \in R_Q$:
On input $[b_1, \ldots, b_m] \in R_Q^m$ and $[\mu_1, \ldots, \mu_m] \in R_p^m$,

1. Sample $s \leftarrow \chi$ and $\{\epsilon_i \leftarrow \chi\}_{i \in [m]}$.
2. Compute $\{a_i := s^{-1}(b_i - \mu_i - p \cdot \epsilon_i)\}_{i \in [m]} \in R_Q^m$.
3. Output the ciphertexts $\{\tilde{\mu}_i := b_i - a_i \cdot Y\}_{i \in [m]} \in R_R^m$ and the encryption key $s \in R_Q$.

$\mathsf{ASHE.Dec}(\tilde{\mu}, s) \to \mu \in R_p$:
On input a ciphertext $\tilde{\mu} \in R_R$ and an encryption key $s \in R_Q$,

1. Interpret $\tilde{\mu}$ as $\tilde{\mu}(Y) \in R_Q[Y]/(Y^{D+1} + 1)$.
2. Output $\mu := \tilde{\mu}(s) \pmod p \in R_p$.

$\mathsf{ASHE.Decode}(\mu) \to [x_1, \ldots, x_n] \in \mathbb{Z}_p^n$:
On input a polynomial $\mu \in R_p$, output $[x_1, \ldots, x_n] \leftarrow \mathsf{NTT}(\mu) \in \mathbb{Z}_p^n$.

$\mathsf{ASHE.Eval}(f, [b_1, \ldots, b_m], [\tilde{\mu}_1, \ldots, \tilde{\mu}_m]) \to \tilde{\mu}' \in R_R$:
On input $f \in R_R[X_1, \ldots, X_m]$ of total degree bounded by D, and m freshly encrypted ciphertexts $[\tilde{\mu}_1, \ldots, \tilde{\mu}_m] \in R_R^m$,

1. Compute $\{a_i := b_i - \tilde{\mu}_i(1)\}_{i \in [m]} \in R_Q^m$.
2. Derive $f(\tilde{\mu}_1, \ldots, \tilde{\mu}_m) = f(b_1 - a_1 Y, \ldots, b_m - a_m Y)$
   $= \sum_{i=1}^{D} Y^i f_i(a_1, \ldots, a_m) + f_0(b_1, \ldots, b_m)$
   $= \sum_{i=1}^{D} Y^i a_m^i f_i(\frac{a_1}{a_m}, \ldots, \frac{a_{m-1}}{a_m}, 1) + f_0(b_1, \ldots, b_m)$.
3. For each $i \in [D]$, define
   $g_i(X_1, \ldots, X_{m-1}) = f_i(X_1, \ldots, X_{m-1}, 1) \in R_Q[X_1, \ldots, X_{m-1}]$.
4. Output $\tilde{\mu}' := \sum_{i=1}^{D} Y^i a_m^i g_i(\frac{a_1}{a_m}, \ldots, \frac{a_{m-1}}{a_m}) + f_0(b_1, \ldots, b_m)$.

Notice that $f_i(a_1, \ldots, a_m) = a_m^i f_i(\frac{a_1}{a_m}, \ldots, \frac{a_{m-1}}{a_m}, 1)$ holds because each f_i is homogeneous.

Fig. 1. ASHE with a single ring element.

$Z^n + 1$ over $\mathbb{Z}_p$; $\mathsf{ASHE.Encode}$ transforms $[x_1, \ldots, x_n] \in \mathbb{Z}_p^n$ into a polynomial $\mu \in R_p = \mathbb{Z}_p[Z]/(Z^n + 1)$ using the inverse Number Theoretic Transform (NTT): $\mu \leftarrow \mathsf{NTT}^{-1}([x_1, \ldots, x_n])$, ensuring that $\{\mu(\omega_i) = x_i\}_{i\in[n]}$. This encoding establishes the isomorphism:

$$\mathbb{Z}_p[Z]/(Z^n + 1) \cong \bigoplus_{i=1}^{n} \mathbb{Z}_p,$$

enabling simultaneous evaluation of n values in a single ciphertext.

Theorem 2 (ASHE Security). *Under the* $\mathsf{RLWE}(R, q, \mathcal{B}, \mathcal{E}, \mathcal{S}, m)$ *assumption, the construction in Fig. 1 is IND-CPA secure.*

Proof. Suppose an adversary $\mathcal{A}$ wins the IND-CPA game with non-negligible advantage. Given an RLWE oracle as described in Definition 1, we construct an algorithm $\mathcal{A}'$ that solves $\mathsf{RLWE}(R, q, \mathcal{B}, \mathcal{E}, \mathcal{S}, m)$ by running $\mathcal{A}$ as a subroutine. Specifically, $\mathcal{A}'$ simulates the encryption oracle for $\mathcal{A}$: on each encryption call with $[\mu_1, \ldots, \mu_m] \in R_p^m$, it queries the RLWE oracle with $[p^{-1}\mu_1, \ldots, p^{-1}\mu_m]$, receives $(\mathbf{a}, \mathbf{b}) \in (R_Q^m, R_Q^m)$, and returns $\{\tilde{\mu}_i := p \cdot b_i - a_i \cdot Y\}_{i\in[m]}$.

- If the RLWE oracle outputs valid samples, then for each $i \in [m]$,

$$\begin{aligned}\tilde{\mu}_i &= p \cdot b_i - a_i \cdot Y = p \cdot b_i - s^{-1}(b_i - p^{-1}\mu_i - \epsilon_i) \cdot Y \\ &= p \cdot b_i - (p \cdot s)^{-1}(pb_i - \mu_i - p\epsilon_i) \cdot Y\end{aligned}$$

  is a valid encryption of μ_i w.r.t secret $p \cdot s$.
- If the RLWE oracle outputs uniform samples, $\tilde{\mu}_i$ is uniform.

Thus, $\mathcal{A}'$ distinguishes RLWE samples with non-negligible probability, contradicting the $\mathsf{RLWE}(R, q, \mathcal{B}, \mathcal{E}, \mathcal{S}, m)$ assumption. The scheme is IND-CPA secure. □

5 Polynomial Decomposition

The interpolation algorithm described in Section 3 interpolates $\mathsf{DB} \in \mathbb{Z}_p^N$ as an m-variate polynomial of total degree at most D, satisfying $\binom{D+m}{m} \geq N$. Accommodating a large N necessitates either increasing m or D. In this section, we extend this algorithm to interpolate DB as $\sum_{i=0}^{\ell-1} f_i(X_1, \ldots, X_m) \cdot X_{m+1}^i$, where each $f_i \in \mathbb{Z}_p[X_1, \ldots, X_m]$ has a smaller total degree D satisfying $\binom{D+m}{m} \geq N/\ell$. Additionally, we describe how to privately evaluate this polynomial.

5.1 Interpolation

Theorem 3 (Polynomial decomposition). *Let $\mathbb{Z}_p$ be a prime field of order p, and let $m \in \mathbb{N}$. Given any set $\{y_i \in \mathbb{Z}_p\}_{i\in[N]}$, there exists an algorithm that constructs an $(m+1)$-variate polynomial*

$$f(X_1, \ldots, X_{m+1}) = \sum_{i=1}^{\ell} f_i(X_1, \ldots, X_m) \cdot X_{m+1}^{i-1} \in \mathbb{Z}_p[X_1, \ldots, X_{m+1}],$$

where each $f_i \in \mathbb{Z}_p[X_1, \ldots, X_m]$ *has a total degree bounded by* D *satisfying* $\binom{D+m}{m} \geq N/\ell$*, and for every* $i \in [N]$*, there exists* $[x_1, \ldots, x_{m+1}] \in \mathbb{Z}_p^{m+1}$ *s.t.* $f(x_1, \ldots, x_{m+1}) = y_i$*.*

Proof. Without loss of generality, assume N is a multiple of ℓ. If not, we can pad the set $\{y_i\}_{i\in[N]}$ with dummy values from $\mathbb{Z}_p$ to make N a multiple of ℓ. Let $N' = N/\ell$.

1. Partition the set $\{y_i\}_{i\in[N]}$ into ℓ chunks, each containing N' values. For each $k \in [\ell]$, define the chunk

$$\mathbf{z}_k = \{y_{(k-1)\cdot N'+1}, y_{(k-1)\cdot N'+2}, \ldots, y_{k\cdot N'}\}.$$

   Thus, $\{\mathbf{z}_k[j] = y_{(k-1)\cdot N'+j}\}_{j\in[N']}$.
2. For each $j \in [N']$, construct a univariate polynomial $g_j(X) \in \mathbb{Z}_p[X]$ of degree $\ell - 1$ such that

$$\{g_j(k) = \mathbf{z}_k[j]\}_{k\in[\ell]}.$$

3. For each $i \in [\ell]$, let $\mathbf{c}_i \in \mathbb{Z}_p^{N'}$ be the vector of coefficients of the X^{i-1} term across all polynomials $g_j(X)$. That is, for each $j \in [N']$, $g_j(X)$ can be expressed as

$$g_j(X) = \sum_{i=1}^{\ell} \mathbf{c}_i[j] X^{i-1}.$$

4. For each $i \in [\ell]$, apply Theorem 1 with input $\mathbf{c}_i$ to construct an m-variate polynomial $f_i(X_1, \ldots, X_m) \in \mathbb{Z}_p[X_1, \ldots, X_m]$ of degree bounded by D, satisfying $\binom{D+m}{m} \geq N'$. By Theorem 1, there is a unified $\{\mathbf{x}_j \in \mathbb{Z}_p^m\}_{j\in[N']}$,[7] such that

$$\{f_i(\mathbf{x}_j) = \mathbf{c}_i[j]\}_{i\in[\ell], j\in[N']}.$$

5. Define $f(X_1, \ldots, X_{m+1}) \in \mathbb{Z}_p[X_1, \ldots, X_{m+1}]$ as:

$$f(X_1, \ldots, X_{m+1}) = \sum_{i=1}^{\ell} f_i(X_1, \ldots, X_m) \cdot X_{m+1}^{i-1}.$$

Then, for each $j \in [N']$ and each $k \in [\ell]$, we have

$$f(\mathbf{x}_j[1], \ldots, \mathbf{x}_j[m], k) = \sum_{i=1}^{\ell} f_i(\mathbf{x}_j[1], \ldots, \mathbf{x}_j[m]) \cdot k^{i-1} = \sum_{i=1}^{\ell} \mathbf{c}_i[j] \cdot k^{i-1} = g_j(k)$$
$$= \mathbf{z}_k[j] = y_{(k-1)\cdot N'+j}.$$

□

[7] $\{\mathbf{x}_j[r] = (\omega^j)^{\varrho[r]}\}_{r\in[m]}$.

5.2 Evaluation

To privately evaluate the polynomial

$$f(X_1, \ldots, X_{m+1}) = \sum_{i=1}^{\ell} f_i(X_1, \ldots, X_m) \cdot X_{m+1}^{i-1},$$

the client and server first evaluate each f_i using the lookup tables (cf. Section 2.3). As all lookups access the same position across the ℓ tables, this process can be optimized by consolidating the ℓ tables into a single one and performing a single lookup. Let $\{\tilde{c}_i\}_{i\in[\ell]}$ denote the ciphertexts resulted from the lookups. The problem is transformed into evaluating a univariate polynomial

$$f'(X_{m+1}) = \sum_{i=1}^{\ell} \tilde{c}_i \cdot X_{m+1}^{i-1}.$$

Given a homomorphic encryption of X_{m+1} provided by the client, the server computes encryptions of $\{X_{m+1}^{i-1}\}_{i\in[3,\ell]}$ using $(\ell-2)$ ciphertext-ciphertext multiplications. Subsequently, $(\ell-1)$ additional ciphertext-ciphertext multiplications are required to linearly combine these terms and obtain the encryption of $f'(X_{m+1})$, resulting in a total of $(2\ell-3)$ ciphertext-ciphertext multiplications.

To optimize this process, we adopt techniques from Cong et al. [15]. Specifically, by applying the Paterson–Stockmeyer algorithm with parameters L and H satisfying $L \cdot H = \ell$, the polynomial f' is rewritten as

$$f'(X_{m+1}) = \sum_{j=1}^{H} \Big(\sum_{i=1}^{L} \tilde{c}_{(j-1)L+i} \cdot X_{m+1}^{i-1} \Big) X_{m+1}^{(j-1)L}.$$

The server first computes encryptions of $\{X_{m+1}^{i-1}\}_{i\in[3,L]}$ using $(L-2)$ ciphertext-ciphertext multiplications. Then, for each $j \in [H]$, it computes the inner sum $\sum_{i=1}^{L} \tilde{\mu}_{(j-1)L+i} \cdot X_{m+1}^{i-1}$ using L ciphertext-ciphertext multiplications. Subsequently, it computes encryptions of $\{X_{m+1}^{(j-1)L}\}_{j\in[2,H]}$ using $(H-1)$ ciphertext-ciphertext multiplications. Finally, another $(H-1)$ ciphertext-ciphertext multiplications are required to evaluate the outer sum and obtain the encryption of $f'(X_{m+1})$. This approach reduces the total number of ciphertext-ciphertext multiplications from $(2\ell-3)$ to $(\ell+L+2H-5)$. The multiplication depth can be further optimized by leveraging GPSP. We refer to [15] for more details.

Since X_{m+1} is not preprocessed, the homomorphic encryption scheme for X_{m+1} need not be ASHE; instead, a practical somewhat homomorphic encryption (SHE) scheme can be employed to achieve finer control over noise growth. In our implementation, we adopt the BGV scheme [8].

6 Polynomial Preprocessing with p-adic Numbers

In this section, we prove that the largest prime resulted from the KU method [28], described in Section 2.3, stabilizes at approximately $O(D \log D)$, regardless of

the number of recursions. To overcome this barrier, we propose a method based on BGGKU [6]. The key distinction between our method and BGGKU [6] is that we focuse on the preprocessing setting and its asymptotic analysis, whereas BGGKU does not consider preprocessing at all. Although extending BGGKU for preprocessing is straightforward, deriving the asymptotic results for data structures and online time is non-trivial.

We begin by outlining two properties of prime numbers that are extensively used in our proofs.

Fact 2 (Approximate formulas for some functions of prime numbers [44]). *For $x \in \mathbb{N}$ and prime numbers denoted by p, we have*

- $\pi(x) = \sum_{p \le x} 1 = (1 + o(1)) \frac{x}{\ln x}$;
- $\theta(x) = \sum_{p \le x} \ln p = (1 + o(1))x$ *(in particular, $\theta(x) \le 1.02x$; and $\theta(x) \ge 0.84x$ when $x \ge 101$).*

Based on this fact, we prove the following lemma.

Lemma 1. $\sum_{p \le B} p^m \ln p \le (1 + o(1)) \frac{B^{m+1}}{m+1}$.

The proof is deferred to Appendix B.

6.1 Stabilization of KU

Theorem 4 (Stabilization of KU [28]). *Let $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$. By repeatedly applying CRT, f stabilizes to an element in $\mathbb{Z}_p[X_1, \ldots, X_m]$, with*

$$p \ge \frac{(D+1)\ln 2}{1.02}\Big(\log \alpha(D+1) + \log\ln 2 + \log\log \alpha(D+1)\Big),$$

where $\alpha^{D+1} = 2N(1/2)^{D+1}$. Approximately, $p = O(D \cdot \log D)$

Proof. Let $p_0 = q$, and for each recursion $i \ge 1$, let p_i be the largest prime used for the i-th CRT round (cf. Section 2.3). Notice that CRT stabilizes when $p_i \ge p_{i+1}$.

The maximum value attainable by any lifted polynomial $f_i \in \mathbb{Z}_{p_i}[X_1, \ldots, X_m]$ is given by

$$M_i = 2N(p_i/2)^{D+1}.$$

Recall that p_{i+1} was chosen such that

$$\prod_{p \le p_{i+1}} p \ge M_i.$$

By Fact 2, we have

$$\theta(p_{i+1}) = \sum_{p \le p_{i+1}} \ln p \le 1.02 p_{i+1},$$

implying

$$M_i \leq \prod_{p \leq p_{i+1}} p = e^{\sum_{p \leq p_{i+1}} \ln p} \leq e^{1.02 p_{i+1}}.$$

Taking the natural logarithm, we obtain

$$p_{i+1} \geq \frac{\ln M_i}{1.02}. \tag{1}$$

Thus, CRT stabilizes when $p_i \leq \frac{\ln M_i}{1.02}$.

Define $\alpha^{D+1} = 2N(1/2)^{D+1}$, then

$$\frac{\ln M_i}{1.02} = \frac{\ln 2N(p_i/2)^{D+1}}{1.02} = \frac{(D+1)\ln \alpha p_i}{1.02}. \tag{2}$$

Define a function

$$g(x) = \frac{(D+1)\ln \alpha x}{1.02}.$$

We first seek a fixed point x_0 such that $g(x_0) = x_0$, which later we will prove to be the lower bound for p_i at stabilization.

To find x_0, we solve

$$\frac{(D+1)\ln \alpha x_0}{1.02} = x_0.$$

This implies

$$(D+1)\ln \alpha x_0 = 1.02 x_0 \Rightarrow \alpha x_0 = e^{\frac{1.02 x_0}{D+1}} \Rightarrow e^{-\frac{1.02 x_0}{D+1}} \alpha x_0 = 1$$

Rearranging, we get

$$\left(-\frac{1.02 x_0}{D+1}\right) e^{-\frac{1.02 x_0}{D+1}} = -\frac{1.02}{\alpha(D+1)}.$$

Let $y = -\frac{1.02 x_0}{D+1}$. Then, $y e^y = -\frac{1.02}{\alpha(D+1)}$. This equation has solutions given by the Lambert W function, $y = W(z)$, where $z = -\frac{1.02}{\alpha(D+1)}$. Since $z < 0$, there are two real branches: $W_0(z)$ and $W_{-1}(z)$. Thus,

$$y = -\frac{1.02 x_0}{D+1} = W_0\left(-\frac{1.02}{\alpha(D+1)}\right) \text{ or } W_{-1}\left(-\frac{1.02}{\alpha(D+1)}\right)$$
$$\Rightarrow x_0 = -\frac{D+1}{1.02} W_0\left(-\frac{1.02}{\alpha(D+1)}\right) \text{ or } -\frac{D+1}{1.02} W_{-1}\left(-\frac{1.02}{\alpha(D+1)}\right).$$

Since $W_{-1}(z) < W_0(z)$,

$$-\frac{D+1}{1.02} W_0\left(-\frac{1.02}{\alpha(D+1)}\right) < -\frac{D+1}{1.02} W_{-1}\left(-\frac{1.02}{\alpha(D+1)}\right).$$

Recall that we are going to prove x_0 to be the lower bound for p_i, hence x_0 should be as large as possible. To this end, we discard the W_0 branch and consider

$$x_0 = -\frac{D+1}{1.02} W_{-1}\left(-\frac{1.02}{\alpha(D+1)}\right).$$

As $z = -\frac{1.02}{\alpha(D+1)}$ is small, the following approximation [16] holds:

$$W_{-1}(z) \approx \ln(-z) - \ln(-\ln(-z)).$$

Then,

$$\begin{aligned} x_0 &\simeq \frac{D+1}{1.02}\ln\frac{\alpha(D+1)}{1.02} + \frac{D+1}{1.02}\ln\ln\frac{\alpha(D+1)}{1.02} \\ &= \frac{(D+1)\ln 2}{1.02}\log\alpha(D+1) + \frac{(D+1)\ln 2}{1.02}\log\ln 2\log\alpha(D+1) \\ &= \frac{(D+1)\ln 2}{1.02}\Big(\log\alpha(D+1) + \log\ln 2 + \log\log\alpha(D+1)\Big). \end{aligned}$$

Next, we prove that x_0 is the lower bound for p_i. Notice that $g(x) = \frac{(D+1)\ln\alpha x}{1.02}$ is monotonically increasing for $x > 0$. Since $g(x_0) = x_0$, we have $\{g(x) \geq x_0\}_{x \geq x_0}$. Given $p_0 = q \geq x_0$,[8] and assume $p_i \geq x_0$. Based on Equation 1 and 2, we have

$$p_{i+1} \geq \frac{\ln M_i}{1.02} = \frac{(D+1)\ln\alpha p_i}{1.02} = g(p_i) \geq g(x_0) = x_0.$$

By induction, we have $\{p_i \geq x_0\}_{i \geq 0}$. Thus, x_0 is a lower bound for p_i, and the CRT stabilizes at

$$p \geq \frac{(D+1)\ln 2}{1.02}\Big(\log\alpha(D+1) + \log\ln 2 + \log\log\alpha(D+1)\Big).$$

□

6.2 Preprocessing with p-adic Numbers

Next, we show how to overcome the above barrier by leveraging p-adic numbers. Specifically, Lemma 2 describes the preprocessing of a polynomial in $\mathbb{Z}_{p^k}[X_1, \ldots, X_m]$. Lemma 3 establishes the decomposition of a polynomial in $\mathbb{Z}_q[X_1, \ldots, X_m]$ into $\mathbb{Z}_{p_1^k}[X_1, \ldots, X_m], \ldots, \mathbb{Z}_{p_t^k}[X_1, \ldots, X_m]$, enabling preprocessing as in Lemma 2. Theorem 5 shows how recursive application of Lemma 3 further reduces the size of the largest prime resulted from this process.

Lemma 2 (Preprocessing a polynomial of p-adic coefficients). *Let p be a prime. There exists a preprocessing algorithm taking $f \in \mathbb{Z}_{p^k}[X_1, \ldots, X_m]$ as an input, outputs a data structure of $\binom{m+k-1}{m} p^m k \log p$ bits. Moreover, there exists an evaluation algorithm which, given $[x_1, \ldots, x_m] \in \mathbb{Z}_{p^k}^m$, computes $f(x_1, \ldots, x_m)$ in time $O(\binom{m+k-1}{m})$.*

[8] This assumption is justified because $x_0 \simeq \frac{(D+1)\ln 2}{1.02}\log\alpha(D+1) + \frac{(D+1)\ln 2}{1.02}\log\ln 2 + \frac{(D+1)\ln 2}{1.02}\log\log\alpha(D+1)$ is typically small for reasonable values of N and D; in contrast, q is the initial modulus, often a large integer or prime chosen to ensure a non-trivial CRT, making $q \geq x_0$ a natural condition.

Proof. For each X_i, express it as $X_i = \sum_{j=0}^{k-1} X_i^{(j)} p^j$. Then, we have

$$f(X_1, \ldots, X_m) = f\left(\sum_{j=0}^{k-1} X_1^{(j)} p^j, \ldots, \sum_{j=0}^{k-1} X_m^{(j)} p^j\right).$$

Let $\Phi_{m,V}$ denote all vectors $\mathbf{v} \in \mathbb{N}^m$ whose entries sum to $V \in \mathbb{N}$, i.e.,

$$\Phi_{m,V} = \left\{ \mathbf{v} \in \mathbb{N}^m \;\middle|\; \sum_{i=1}^{m} v_i = V \right\}.$$

By a multivariate Taylor expansion $[x_1^{(0)}, x_2^{(0)}, \ldots, x_m^{(0)}]$, we have

$$\begin{aligned} &f(X_1, \ldots, X_m) = f\left(\sum_{j=0}^{k-1} X_1^{(j)} p^j, \ldots, \sum_{j=0}^{k-1} X_m^{(j)} p^j\right) \\ &= \sum_{V=0}^{k-1} \sum_{\mathbf{v} \in \Phi_{m,V}} \Big(\frac{1}{v_1! \cdots v_m!} \cdot \frac{\partial^V f}{\partial x_1^{v_1} \cdots \partial x_m^{v_m}} (X_1^{(0)}, \ldots, X_m^{(0)}) \cdot \prod_{i=1}^{m} (\sum_{j=1}^{k-1} X_i^{(j)} p^j)^{v_i} \Big). \end{aligned}$$

For each $V \in [0, k-1]$ and each $\mathbf{v} \in \Phi_{m,V}$, define $g_{\mathbf{v}} \in \mathbb{Z}_{p^k}[X_1, \ldots, X_m]$ as:

$$\begin{aligned} &g_{\mathbf{v}}(X_1, \ldots, X_m) \\ &\quad = \frac{1}{v_1! \cdots v_m!} \cdot \frac{\partial^V f}{\partial X_1^{v_1} \cdots \partial X_m^{v_m}} (X_1^{(0)}, \ldots, X_m^{(0)}) \cdot \prod_{i=1}^{m} (\sum_{j=1}^{k-1} X_i^{(j)} p^j)^{v_i}. \end{aligned}$$

As $|\Phi_{m,V}| = \binom{m+V-1}{m-1}$, the total number of such polynomials is

$$\sum_{V=0}^{k-1} \binom{m+V-1}{m-1} = \binom{m+k-1}{m}.$$

For each $V \in [0, k-1]$ and each $\mathbf{v} \in \Phi_{m,V}$, define $h_{\mathbf{v}} \in \mathbb{Z}_{p^k}[X_1, \ldots, X_m]$ as:

$$h_{\mathbf{v}}(X_1, \ldots, X_m) = \frac{1}{v_1! \cdots v_m!} \cdot \frac{\partial^V f}{\partial x_1^{v_1} \cdots \partial x_m^{v_m}} (X_1, \ldots, X_m),$$

and use FFT to evaluate $h_{\mathbf{v}}$ on all points $\mathbf{x} \in \mathbb{Z}_p^m$, resulting in a table of evaluations, i.e.,

$$T_{\mathbf{v}} = \{h_{\mathbf{v}}(\mathbf{x})\}_{\mathbf{x} \in \mathbb{Z}_p^m}.$$

It is important to note that we only need to enumerate $(X_1^{(0)}, \ldots, X_m^{(0)})$ instead of $(X_1, \ldots, X_m)$. That means the input to $h_{\mathbf{v}}$ is in fact over $\mathbb{Z}_p^m$ (the output is over $\mathbb{Z}_{p^k}$).

The data structure, consisting of $\binom{m+k-1}{m}$ lookup tables, requires $\binom{m+k-1}{m}$ $p^m \log p^k$ bits of storage.

Given any point $\mathbf{x} \in \mathbb{Z}_{p^k}^m$, the following algorithm evaluates $f(\mathbf{x})$ using the above data structure:

1. For each $i \in [m]$, express x_i as $x_i = \sum_{j=0}^{k-1} x_i^{(j)} p^j$.
2. For each $V \in [0, k-1]$ and each $\mathbf{v} \in \Phi_{m,V}$, obtain the evaluation of $h_{\mathbf{v}}(x_1^{(0)}, \ldots, x_m^{(0)})$ by looking it up in table $T_{\mathbf{v}}$.
3. For each $i \in [m]$, compute $y_i := x_i - x_i^{(0)}$. Notice that y_i is equal to $\sum_{j=1}^{k-1} x_i^{(j)} p^j$.
4. For each $V \in [0, k-1]$ and each $\mathbf{v} \in \Phi_{m,V}$, compute
$$g_{\mathbf{v}}(\mathbf{x}) = h_{\mathbf{v}}(x_1^{(0)}, \ldots, x_m^{(0)}) \cdot \prod_{i=1}^{m} y_i^{v_i}.$$
5. Output
$$f(\mathbf{x}) = \sum_{V=0}^{k-1} \sum_{\mathbf{v}} g_{\mathbf{v}}(\mathbf{x}).$$

Step 4 clearly dominates the computation. A naive implementation would require $O(m\binom{m+k-1}{m})$ time, but we can optimize this by incrementally computing each $g_{\mathbf{v}}(\mathbf{x})$, e.g., $g_{v_1,\ldots,v_m}(\mathbf{x})$ can be derived from $g_{v_1-1,\ldots,v_m}(\mathbf{x})$ with a single multiplication by y_1. Applying this strategy across all terms reduces the total computational cost to $O(\binom{m+k-1}{m})$.

We further remark that the $\binom{m+k-1}{m}$ lookups in Step 2 can be optimized by consolidating the tables into a single one and performing a single lookup, as all lookups access the same position across the $\binom{m+k-1}{m}$ tables. □

Lemma 3 (Non-recursive CRT with p-adic numbers).
Let $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$ be a polynomial of total degree bounded by D with N monomials. For any fixed positive integer k, there exists a preprocessing algorithm taking f as an input, outputs a data structure of size

$$(1 + o(1)) \frac{\binom{m+k-1}{m}}{(m+1)k^m \log^m e} (D+1)^{m+1} \log^{m+1} q.$$

Moreover, there exists an evaluation algorithm, which, given $[x_1, \ldots, x_m] \in \mathbb{Z}_q^m$, computes $f(x_1, \ldots, x_m)$ in time

$$O(\binom{m+k-1}{m} \frac{(D+1)\log q}{k \log(D+1) + k \log\log q}).$$

Proof. The following algorithm preprocesses $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$ into a data structure:

1. Let M denote the maximum value that a lifted f can take (i.e., the coefficients of f are lifted from $\mathbb{Z}_q$ to $\mathbb{Z}$). Choose a set of distinct primes $p_1 = 2 < p_2 < \ldots < p_t$, and a power $k \in \mathbb{N}$, satisfying:

$$\prod_{i=1}^{t} p_i^k \geq M = 2N(q/2)^{D+1}. \tag{3}$$

2. For each $i \in [t]$, let $f_i \in \mathbb{Z}_{p_i^k}[X_1, \ldots, X_m]$ be the polynomial $f_i = f \pmod{p_i^k}$ (i.e., modulo every coefficient of f with p_i^k); apply Lemma 2 to generate a data structure for evaluating f_i.
3. The data structure for evaluating $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$ is assembled from the t individual data structures used for evaluating $f_1, \ldots, f_t$ respectively.

Given any point $\mathbf{x} \in \mathbb{Z}_q^m$, the following algorithm evaluates $f(\mathbf{x})$ using the above data structure:

1. For each $i \in [t]$, compute $\mathbf{x}_i := \mathbf{x} \pmod{p_i^k}$ and use Lemma 2 to obtain $f_i(\mathbf{x}_i) \in \mathbb{Z}_{p_i^k}$.
2. Use CRT to find the smallest $Z \in \mathbb{Z}$ s.t.

$$\{Z \equiv f_i(\mathbf{x}_i) \pmod{p_i^k}\}_{i \in [t]}.^{9}$$

3. Output $(Z \mod q)$ as the evaluation of $f(\mathbf{x})$.

The correctness of the above algorithm is argued as follows. Let $\bar{f} \in \mathbb{Z}[X_1, \ldots, X_m]$ be the lifted polynomial of $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$. Then, we have

$$\{\bar{f}(\mathbf{x}) \equiv f_i(\mathbf{x}_i) \pmod{p_i^k}\}_{i \in [t]}.$$

Base on the fact that $\prod_{i=1}^{s} p_i^k > M$, we have $\bar{f}(\mathbf{x}) < \prod_{i=1}^{s} p_i^k$. By CRT, there is a unique solution for $Z < \prod_{i=1}^{s} p_i^k$ s.t. $\{Z \equiv f_i(\mathbf{x}_i) \pmod{p_i^k}\}_{i \in [t]}$. Then, we have $Z = \bar{f}(\mathbf{x})$ by the uniqueness. Correctness follows since $\bar{f}(\mathbf{x}) \pmod{q} = f(\mathbf{x})$.

Storage Size. Define $B \geq p_t$ s.t. $\prod_{p \leq B} p = \prod_{p \leq p_t} p$. Equation 3 can be transformed into

$$\prod_{i=1}^{t} p_i^k \geq 2N(q/2)^{D+1} \Rightarrow \prod_{p \leq B} p \geq (2N(q/2)^{D+1})^{\frac{1}{k}}.$$

[9] This step, known as inverse-CRT, can be accelerated via a method described in Section 7.2.

By Fact 2, we have

$$\begin{aligned}
&\theta(B) = \sum_{p \le B} \ln p = (1 + o(1))B \\
&\Rightarrow \prod_{p \le B} p = e^{(1+o(1))B} \\
&\Rightarrow e^{(1+o(1))B} \ge (2N(q/2)^{D+1})^{\frac{1}{k}} \\
&\Rightarrow B \ge (1 + o(1))\frac{1}{k} \ln \left(2N(\frac{q}{2})^{D+1}\right) \\
&\Rightarrow B \ge (1 + o(1))\frac{D+1}{k} \ln q.
\end{aligned}$$

Since we want p_t to be as small as possible, we set

$$B = (1 + o(1))\frac{D+1}{k} \ln q. \tag{4}$$

Based on the storage size computed in Lemma 2, the required storage (in bits) here is:

$$\sum_{p \le B} \binom{m+k-1}{m} p^m \log p^k = \frac{1}{\ln 2} \binom{m+k-1}{m} \sum_{p \le B} p^m \ln p^k.$$

By Lemma 1, the storage size can be re-written as:

$$\frac{1}{\ln 2} \binom{m+k-1}{m} \sum_{p \le B} p^m \ln p^k \le (1 + o(1))\frac{k}{\ln 2} \binom{m+k-1}{m} \frac{B^{m+1}}{m+1}.$$

By Equation 4, the storage size is

$$\begin{aligned}
&(1 + o(1))\frac{k}{\ln 2} \binom{m+k-1}{m} \frac{B^{m+1}}{m+1} \\
=&(1 + o(1))\frac{k}{\ln 2} \binom{m+k-1}{m} \frac{(D+1)^{m+1} \ln^{m+1} q}{(m+1)k^{m+1}} \\
=&(1 + o(1))\frac{\binom{m+k-1}{m}}{(m+1)k^m \ln 2} (D+1)^{m+1} \ln^{m+1} q \\
=&(1 + o(1))\frac{\binom{m+k-1}{m}}{(m+1)k^m \ln 2} (D+1)^{m+1} (\ln 2 \log q)^{m+1} \\
=&(1 + o(1))\frac{\binom{m+k-1}{m} \ln^m 2}{(m+1)k^m} (D+1)^{m+1} \log^{m+1} q \\
=&(1 + o(1))\frac{\binom{m+k-1}{m}}{(m+1)k^m \log^m e} (D+1)^{m+1} \log^{m+1} q.
\end{aligned}$$

Runtime. By Fact 2,

$$t = \pi(B) = \sum_{p \le B} 1 = (1 + o(1))\frac{B}{\ln B}.$$

By Equation 4, we have

$$\begin{aligned} t &= (1 + o(1))\frac{(D+1)\ln q}{k \ln((D+1)/k \ln q)} \\ &= (1 + o(1))\frac{(D+1)\ln q}{k \ln(D+1) + k \ln\ln q} \\ &= (1 + o(1))\frac{(D+1)\log q}{k \log(D+1) + k \log\log q + k \log\ln 2} \\ &= (1 + o(1))\frac{(D+1)\log q}{k \log(D+1) + k \log\log q}. \end{aligned}$$

As we need to run the evaluation phase of Lemma 2 for each prime, the total runtime is

$$O(\binom{m+k-1}{m}\frac{(D+1)\log q}{k \log(D+1) + k \log\log q}).$$

□

We remark that the result in Lemma 3 is similar to that obtained in [6]; we extend and strengthen this result in the following theorem.

Theorem 5 (Recursive CRT with p-adic numbers).
Let $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$ be a polynomial of total degree bounded by D with N monomials. Then, there exists a preprocessing algorithm taking f as an input, outputs a data structure of size

$$(1 + o(1))\frac{\binom{m+k-1}{m}}{(m+1)k^m \log^m e}(D+1)^{m+1}(\log(D+1) + \log\log q)^{m+1}.$$

Moreover, there exists an evaluation algorithm, which, given $[x_1, \ldots, x_m] \in \mathbb{Z}_q^m$, computes $f(x_1, \ldots, x_m)$ in time

$$O(\frac{1}{k}\binom{m+k-1}{m}\frac{(D+1)^2 \log q}{\log(D+1) + \log\log(D+1) + \log\log\log q}).$$

Proof. The following algorithm preprocesses $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$ into a data structure:

1. Let M denote the maximum value that a lifted f can take (i.e., the coefficients of f are lifted from $\mathbb{Z}_q$ to $\mathbb{Z}$). Choose a set of distinct primes $p_1 = 2 < p_2 < \ldots < p_s < p_{s+1} < \ldots < p_t$, and a power $k \in \mathbb{N}$, satisfying:

$$\prod_{i=1}^{s} p_i^k \cdot \prod_{j=s+1}^{t} p_t \ge M = 2N(q/2)^{D+1} \tag{5}$$

and

$$\prod_{i=1}^{s} p_i^k \geq 2N(p_t/2)^{D+1} \tag{6}$$

2. For each $i \in [s]$, let $f_i \in \mathbb{Z}_{p_i^k}[X_1, \ldots, X_m]$ be the polynomial $f_i = f \pmod{p_i^k}$ (i.e., modulo every coefficient of f with p_i^k); apply Lemma 2 to generate a data structure for evaluating f_i.
3. The data structure for evaluating $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$ is assembled from the s individual data structures used for evaluating $f_1, \ldots, f_s$ respectively.

Given any point $\mathbf{x} \in \mathbb{Z}_q^m$, the following algorithm evaluates $f(\mathbf{x})$ using the above data structure:

1. For each $i \in [s]$, compute $\mathbf{x}_i := \mathbf{x} \pmod{p_i^k}$ and use Lemma 2 to obtain $f_i(\mathbf{x}_i) \in \mathbb{Z}_{p_i^k}$.
2. For each $j \in [s+1, t]$, compute $\mathbf{x}_j := \mathbf{x} \pmod{p_j}$, and use Lemma 3 to compute $f_j(\mathbf{x}_j) \in \mathbb{Z}_{p_j}$. As $\prod_{i=1}^{s} p_i^k > 2N(p_t/2)^{D+1}$ (cf. Equation 6), we could re-use the above data structure for this task.
3. Use CRT to find the smallest $Z \in \mathbb{Z}$ s.t.

$$\{Z \equiv f_i(\mathbf{x}_i) \pmod{p_i^k}\}_{i\in[s]} \text{ and } \{Z \equiv f_j(\mathbf{x}_j) \pmod{p_j}\}_{j\in[s+1,t]}.$$

4. Output $(Z \mod q)$ as the evaluation of $f(\mathbf{x})$.

The correctness of the above algorithm is argued as follows. Let $\bar{f} \in \mathbb{Z}[X_1, \ldots, X_m]$ be the lifted polynomial of $f \in \mathbb{Z}_q[X_1, \ldots, X_m]$. Then, we have

$$\{\bar{f}(\mathbf{x}) \equiv f_i(\mathbf{x}_i) \pmod{p_i^k}\}_{i\in[s]} \text{ and } \{\bar{f}(\mathbf{x}) \equiv f_j(\mathbf{x}_j) \pmod{p_j}\}_{j\in[s+1,t]}.$$

Base on the fact that $\prod_{i=1}^{s} p_i^k \cdot \prod_{j=s+1}^{t} p_t > M$, we have $\bar{f}(\mathbf{x}) < \prod_{i=1}^{s} p_i^k \cdot \prod_{j=s+1}^{t} p_t$. By CRT, there is a unique solution for $Z < \prod_{i=1}^{s} p_i^k \cdot \prod_{j=s+1}^{t} p_t$ s.t.

$$\{Z \equiv f_i(\mathbf{x}_i) \pmod{p_i^k}\}_{i\in[s]} \text{ and } \{Z \equiv f_j(\mathbf{x}_j) \pmod{p_j}\}_{i\in[s+1,t]}.$$

Then, we have $Z = \bar{f}(\mathbf{x})$ by the uniqueness. Correctness follows since $\bar{f}(\mathbf{x}) \pmod{q} = f(\mathbf{x})$.

Proofs for storage size and runtime are in Appendix C. □

The derivation in Appendix C, Equation 17, implies that the largest prime from Theorem 5 is

$$\frac{1+o(1)}{k}(D+1)(\ln(D+1) + \ln\ln q),$$

which surpasses the $O(D \cdot \log D)$ barrier established by Theorem 4 for large k. Table 3 empirically compares three polynomial preprocessing methods, demonstrating the clear superiority of our method for large N and q.

Setup. The server runs as follows:

1. Run $\mathsf{ASHE.Setup}(1^\lambda, D, p, N) \to n, Q, \chi, R_p, R_Q, R_R, [b_1, \ldots, b_m]$:
   - $Q = q_1 \cdots q_r$ with each prime $q_i \equiv 1 \pmod{2n}$ (cf. Section 2.4);
   - $R_p = \mathbb{Z}_p[Z]/(Z^n + 1)$, $R_Q = \mathbb{Z}_Q[Z]/(Z^n + 1)$, $R_R = R_Q[Y]/(Y^D + 1)$.
2. Use 3-way cuckoo hashing to encode N DB elements into $\mathbf{D} \in \mathbb{Z}_p^{n \times N'}$ with $N' = 3N/n$.
3. For each $i \in [n]$, apply Theorem 3 to interpolate the i-th row of $\mathbf{D}$ as a polynomial in $\mathbb{Z}_p[X_1, \ldots, X_{m+1}]$:

   $$f_i(X_1, \ldots, X_{m+1}) = \textstyle\sum_{j=1}^{\ell} f_{i,j}(X_1, \ldots, X_m) \cdot X_{m+1}^{j-1},$$

   where each $f_{i,j} \in \mathbb{Z}_p[X_1, \ldots, X_m]$ is of total degree bounded by D with $\binom{D+m}{m} \geq N'/\ell$. Let $\omega \in \mathbb{Z}_p$ be the (N'/ℓ)-th root of unity and ϱ be the extremal basis (cf. Theorem 3).
4. For each $j \in [l]$ and $k \in [D]$, construct $G_{j,k} \in R_Q[X_1, \ldots, X_{m-1}]$:
   (a) for each $i \in [n]$, write $f_{i,j}$ as: $\sum_{\mathbf{e}} c_{i,j,\mathbf{e}} X_1^{e_1} X_2^{e_2} \cdots X_m^{e_m}$, where each $\mathbf{e} = [e_1, e_2, \ldots, e_m]$ represents an exponent vector of a monomial with $e_1 + e_2 + \cdots + e_m \leq D$.
   (b) for each $\mathbf{e}$, run $\mathsf{ASHE.Encode}([c_{1,j,\mathbf{e}}, c_{2,j,\mathbf{e}}, \ldots, c_{n,j,\mathbf{e}}]) \to C_{j,\mathbf{e}} \in R_p$ (cf. Section 4), and reinterpret $C_{j,\mathbf{e}}$ as an element of R_R.
   (c) construct $F_j \in R_R[X_1, \ldots, X_m]$:

   $$\begin{aligned} F_j(X_1, \ldots, X_m) &= \sum_{\mathbf{e}} C_{j,\mathbf{e}} (X_1)^{e_1} (X_2)^{e_2} \cdots (X_m)^{e_m} \\ &= \sum_{\mathbf{e}} C_{j,\mathbf{e}} (b_1 + YZ_1)^{e_1} (b_2 + YZ_2)^{e_2} \cdots (b_m + YZ_m)^{e_m} \\ &= \sum_{k=1}^{D} Y^k Z_m^k G_{j,k}(Z_1/Z_m, \ldots, Z_{m-1}/Z_m) + G_0(b_1, \ldots, b_m), \end{aligned}$$

   where each $G_{j,k} \in R_Q[X_1, \ldots, X_{m-1}]$.
5. For each $j \in [l]$, $k \in [D]$, run double-CRT for $G_{j,k}$, generating $\{g_{j,k,i,i'} \in \mathbb{Z}_{q_i}[X_1, \ldots, X_{m-1}]\}_{i \in [r], i' \in [n]}$ (cf. Section 2.4).
6. For each $j \in [l]$, $k \in [D]$, $i \in [r]$ and $i' \in [n]$, apply Theorem 5 to build a data structure for evaluating $g_{j,k,i,i'}(X_1, \ldots, X_{m-1})$.
7. Publish $\{n, Q, \chi, R_p, R_Q, R_R, [b_1, \ldots, b_m], \omega, \varrho\}$.

Fig. 2. The setup phase of our DEPIR.

Query for indices $\{\alpha_i \in [N]\}_{i\in[L]}$**.**

1. The client runs as follows to generate a query:
   (a) Use 3-way cuckoo hashing to encode $\{\alpha_i \in [N]\}_{i\in[L]}$ into $\{\beta_i \in [N']\}_{i\in[n]}$.
   (b) For each $i \in [n]$, compute $\gamma_i := \beta_i \pmod{\ell}$, and $\kappa_i := \lfloor \beta_i/\ell \rfloor$.
   (c) For each $i \in [n]$ and $j \in [m]$, compute $x_{i,j} := (\omega^{\gamma_i})^{g[j]} \in \mathbb{Z}_p$ (cf. Theorem 3).
   (d) For each $j \in [m]$, run $\mathsf{ASHE.Encode}([x_{1,j}, \ldots, x_{n,j}]) \to \mu_j \in R_p$
   (e) Run $\mathsf{ASHE.Enc}([b_1, \ldots, b_m], [\mu_1/\mu_m, \mu_2/\mu_m, \ldots, \mu_{m-1}/\mu_m, \mu_m]) \to [\tilde{\mu}_1, \ldots, \tilde{\mu}_m] \in R_R^m, s \in R_Q$.
   (f) Run $\mathsf{BGV.Enc}([\kappa_1, \ldots, \kappa_n]) \to \tilde{\mu}_{m+1} \in R_R$ (cf. Theorem 3 and Section 5.2).
   (g) Send $\{\tilde{\mu}_i \in R_R\}_{i\in[m+1]}$ to the server.
2. The server runs as follows to answer a query:
   (a) For each $i'' \in [m]$, compute $a_{i''} := b_{i''} - \tilde{\mu}_{i''}(1) \in R_Q$.
   (b) For each $i'' \in [m-1]$, run double-CRT for $a_{i''}$ to generate $\{x_{i'',i,i'} \in \mathbb{Z}_{q_i}\}_{i\in[r],i'\in[n]}$ (cf. Section 2.4).
   (c) For each $j \in [l]$, $k \in [D]$, $i \in [r]$ and $i' \in [n]$, apply Theorem 5 to evaluate $y_{j,k,i,i'} := g_{j,k,i,i'}(x_{1,i,i'}, \ldots, x_{m-1,i,i'}) \in \mathbb{Z}_{q_i}$.
   (d) For each $j \in [l]$ and $k \in [D]$, use double-CRT to recover $G_{j,k}(a_1, \ldots, a_{m-1}) \in R_Q[X_1, \ldots, X_m]$ (cf. Section 2.4).
   (e) For each $j \in [l]$, compute $F_j(\tilde{\mu}_1, \ldots, \tilde{\mu}_m) := \sum_{k=1}^{D} Y^k a_m^k G_{j,k}(a_1, \ldots, a_{m-1}) + G_0(b_1, \ldots, b_m) \in R_R$.
   (f) Return $\tilde{\mu} := \sum_{j=1}^{\ell} F_j(\tilde{\mu}_1, \ldots, \tilde{\mu}_m) \cdot \tilde{\mu}_{m+1}^{j-1} \in R_R$ (cf. Section 5.2).
3. The client runs as follows to recover an element:
   (a) Run $\mathsf{ASHE.Dec}(\tilde{\mu}, s) \to \mu \in R_p$.
   (b) Run $\mathsf{ASHE.Decode}(\mu) \to [y_1, \ldots, y_n] \in \mathbb{Z}_p^n$.
   (c) Use cuckoo hashing to recover $[z_1, \ldots, z_L] \in \mathbb{Z}_p^L$.

Fig. 3. The query phase of our DEPIR.

Table 3. Comparison of different polynomial preprocessing methods.

N	495	3003	54264	1221759	218618940
q	847594697	569397427	903609989	637273877	560922889
(m, D)	(4, 8)	(5, 10)	(6, 15)	(5, 40)	(6, 70)
LMW (Theorem A.4 in [33])					
Largest prime	73	89	127	293	523
Storage (GB)	0.1	19.0	13 153	19 264	268 898 888
Read count	444	592	1 078	4 663	11 636
OPPW (Section 5 in [40])					
Largest prime	41	47	67	157	271
Storage (GB)	0.04	4.7	2 234	5 119	39 985 701
Read count	342	439	782	3 291	8 247
# multiplications	1 368	2 165	4 692	16 455	49 482
Our work (Theorem 5)					
k	4	3	4	4	6
Largest prime	19	31	31	73	89
Storage (GB)	0.02	2.1	324	1 227	3 204 477
Read count	179	273	436	1 662	2 916
# multiplications	6 264	5 732	36 623	93 071	1 347 191

7 Further Optimizations

7.1 Handling Batch Queries with Cuckoo Hashing

Cuckoo hashing is commonly used for batch queries in PIR. Specifically, to retrieve L elements in a single batch, the server employs 3-way cuckoo hashing to encode its N DB elements into $3N$ elements, distributed across $B = 1.5L$ buckets, with each bucket containing $N' = 3N/B$ elements. By retrieving one element per bucket through a black-box PIR, the client can recover all L desired elements with a small failure probability of 2^{-40}. When a traditional PIR, which requires touching the entire database, is employed as the black-box PIR, the computational cost is exactly three times that of retrieving a single element, yielding an amortized cost reduced by a factor of $L/3$.

To our knowledge, we are the first to integrate cuckoo hashing with DEPIR, demonstrating that this approach yields greater cost savings than combining cuckoo hashing with traditional PIR. We describe the integration of cuckoo hashing with DEPIR through our complete protocol, detailed in Figures 2 and 3. By applying cuckoo hashing, we reduce the database size by a factor of $n/3$, thereby decreasing both m and D. This reduction significantly lowers preprocessing time, server storage requirements, online processing time, and communication costs, independent of amortization.

7.2 Preprocessing Inverse-CRT

In Step 2(c) of Figure 3, the server needs to compute:

1. $Z := \mathsf{CRT}^{-1}_{p_1,p_2,\ldots,p_t}(z_1, z_2, \ldots, z_t) \in \mathbb{Z}$, where $z_i \in \mathbb{Z}_{p_i}$[10] $\forall i \in [t]$;
2. $z := Z \pmod q$.

Naively, $\mathsf{CRT}^{-1}_{p_1,p_2,\ldots,p_t}$ is computed as:

$$\mathsf{CRT}^{-1}_{p_1,p_2,\ldots,p_t}(z_1, z_2, \ldots, z_t) = \sum_{i=1}^{t} z_i \cdot P_i \cdot P'_i \pmod P, \tag{7}$$

where $P = \prod_{i=1}^{t} p_i$, and $\forall i \in [t]$, $P_i = \frac{P}{p_i}$, $P'_i = P_i^{-1} \pmod{p_i}$. Such naive computation is expensive when P is large.

We adopt an optimization inspired by HPS19 [23]. By Equation 7, we have

$$\sum_{i=1}^{t} z_i \cdot P_i \cdot P'_i = Z + u \cdot P \;\Rightarrow\; u = \frac{\sum_{i=1}^{t} z_i \cdot P_i \cdot P'_i - Z}{P}.$$

As $Z \in [P-1]$, we have

$$u = \lfloor \frac{\sum_{i=1}^{t} z_i \cdot P_i \cdot P'_i}{P} \rfloor = \lfloor \sum_{i=1}^{t} z_i \cdot \frac{P'_i}{p_i} \rfloor.$$

Then,

$$\begin{aligned} z &= Z \mod q \\ &= (\sum_{i=1}^{t} z_i \cdot P_i \cdot P'_i - u \cdot P) \mod q \\ &= (\sum_{i=1}^{t} z_i \cdot P_i \cdot P'_i - \lfloor \sum_{i=1}^{t} z_i \cdot \frac{P'_i}{p_i} \rfloor \cdot P) \mod q. \end{aligned} \tag{8}$$

Notice that the following components are independent of z_i and hence can be precomputed:

- $P_i \cdot P'_i \pmod q \in \mathbb{Z}_q$, $\forall i \in [t]$,
- $\frac{P'_i}{p_i} \in (0, 1)$, $\forall i \in [t]$, and
- $P \pmod q \in \mathbb{Z}_q$.

Then, in the online phase, Equation 8 can be computed efficiently as q is significantly smaller than P. Furthermore, since $P_i \cdot P'_i \pmod q$ is known, we could adopt Shoup's optimization of barrett reduction [4] to accelerate the computation of $z_i \cdot P_i \cdot P'_i \pmod q$. While Okada et al. [41] employ a similar optimization, our method is both simpler and faster.

[10] In the context of Lemma 3, p_i here represents p_i^k.

Table 4. Comparison of our DEPIR with OPPW [40].

N level	2^{17}	2^{23}	2^{27}
OPPW (cf. Table 3 in [40])			
N	142506	15020334	185250786
(m, D)	(5, 25)	(5, 68)	(6, 68)
n	2^{15}	2^{16}	2^{16}
$(\lfloor \log Q \rceil, p, r)$	(721, 30, 24)	(1894, 30, 63)	(1924, 30, 64)
Security (bits)	156	118	116
Storage (GB)	2 927	750 922	394 474 940
# RAM/Disk reads	$5.1 \cdot 2^{30}$	$133.9 \cdot 2^{30}$	$335.1 \cdot 2^{30}$
Query time (s)	6 267.5	cannot run	cannot run
Our work			
N	222938	9116891	137478300
(m, D)	(2, 3)	(3, 5)	(3, 5)
n	2^{13}	2^{13}	2^{14}
$(\lfloor \log Q \rceil, p, r)$	(178, 65537, 9)	(245, 65537, 12)	(361, 65537, 17)
Security (bits)	160	112	159
(k, ℓ)	(1, 12)	(2, 64)	(2, 462)
Storage (GB)	0.39	171.8	3 640
# Disk reads	$3.6 \cdot 2^{21}$	$1.3 \cdot 2^{24}$	$3.8 \cdot 2^{24}$
Query time (s)	10	112	843
Amort. query time (ms)	1.8	21	77
Bandwidth (query+response) (MB)	1.2	2.4	7
Amort. bandwidth (KB)	0.23	0.46	0.66

8 Implementation and Evaluation

We fully implemented our DEPIR scheme, as depicted in Fig. 2 and 3. Our implementation consists of approximately 2 000 lines of C++ code. We utilized the Microsoft SEAL library [45] (with Intel HEXL [25] support) for homomorphic encryption, the Microsoft Kuku library [30] for cuckoo hashing, the FLINT library [18] for algebraic operations, and the io-uring library [27] for asynchronous I/O.

We benchmarked our implementation in a setting similar to that of OPPW [40], enabling a direct comparison with the results reported therein (cf. Table 3 in [40]). Specifically, we ran our implementation on a system equipped with an Intel(R) Xeon(R) Platinum 8160 CPU, 32GB DDR4 RAM (2400MHz), and a 4TB Samsung 990 PRO NVMe SSD. Our setup has two disadvantages compared to that of OPPW [40]: (1) they employed multi-threading (utilizing all 24 cores), whereas we used single-threading; (2) they allocated 26GB of RAM to store portions of the data structure, whereas we stored all data structures on disk. These differences ensure that the comparison is fair, or even conservative.

Table 4 shows the chosen parameters and experimental results. We selected encryption parameters to ensure at least 112 bits of security, as verified using

the LWE-Estimator [36]. The plaintext modulus p was chosen to satisfy $p \equiv 1 \pmod{2n}$, enabling SIMD operations. Specifically, we set $p = 65\,537$, resulting in each DB element being 2 bytes in size. The DB sizes in the OPPW experiments were cherry-picked to favor their construction, thus, we adopted only the corresponding order-of-magnitude levels (i.e., $N = 2^{17}$, 2^{23}, and 2^{27}) rather than their exact values. Notice that the database to be interpolated is in fact $\frac{3N}{\ell \cdot n}$, which is relatively small even for the largest $N = 2^{27}$. Therefore, we adopted the total degree interpolation approach, which requires $O((\frac{3N}{\ell \cdot n})^2)$ time with a larger D satisfying $\binom{m+D}{m} \geq \frac{3N}{\ell \cdot n}$. The noise growth (i.e., critical quantity) during ASHE is approximately

$$e_{\text{exp}} = \sqrt{\frac{3N}{\ell \cdot n}} \left(\text{erf}^{-1}\left(2^{-1/(nm)}\right) p \cdot n \cdot \sigma \right)^D$$

in canonical L^∞-norm, i.e., $\|e\|_{\text{can},\infty} = \max |\sigma(e)|$ where $\sigma : R \to \mathbb{C}$ runs through all complex embeddings. This then determines the size of Q via $Q > e_{\text{exp}}$.

For $N = 2^{17}$, our scheme processes 5 461 queries in a single batch with 0.39GB of server state and 2^{24} online disk reads, yielding a 10s total query time and a 1.8ms amortized query time. Compared with OPPW, this represents a 7 505× reduction in server state and a 3 481 944× reduction in query time. Importantly, even without considering amortization, our scheme remains 637× faster than OPPW; these gains arise from polynomial decomposition and preprocessing with p-adic numbers. For $N = 2^{23}$, a batch query of 5 461 elements requires 171.8GB of server state and 2^{24} online disk reads, with a total query time of 112s (21ms amortized). Here, our scheme maintains a 4 371× reduction in server state over OPPW; the reduction in query time is unknown since OPPW is unable to run at this scale. For $N = 2^{27}$ and a larger batch of 10 922 queries, our scheme requires 3 640GB server state (a 108 372× reduction over OPPW) and 843s total query time (77ms amortized). Notably, this performance is comparable to the state-of-the-art in batch PIR [35].

9 Related Work

The research on DEPIR is still in its early stages. To the best of our knowledge, the schemes presented in Section 1 are the only single-server DEPIR schemes developed to date. In this section, we examine other PIR schemes beyond single-server DEPIR.

Early Single-Server PIR. Kushilevitz and Ostrovsky [31] propose the first single-server PIR protocol, which is based on additively homomorphic encryption, achieving a query size of $O(\sqrt{n} \log n)$ and a response size of $O(\sqrt{n})$. Cachin et al. [10] utilized the ϕ-hiding assumption (instead of homomorphic encryption) to reduce the query size to $O(\log^4 n)$ and the response size to $O(\log^d n)$, representing the database as a d-dimensional cube. Based on the decision subgroup problem (a variant of the ϕ-hiding assumption), Gentry and Ramazan [19]

made further strides by decreasing the query size of Cachin et al.'s protocol to $O(\log^{3-o(1)} n)$. Chang [13] instantiates Kushilevitz-Ostrovsky's approach with Paillier homomorphic encryption to retain an $O(\sqrt{n} \log n)$ query size while reducing the response size to $O(\log n)$. This was later generalized by Lipmaa [34], using Damgard-Jurik encryption [17], achieving $O(\log^2 n)$ query size and $O(\log n)$ response size. However, all these protocols requires $O(n)$ public-key operations to answer a single query. As pointed out by Sion and Carbunar [46], such protocols often perform slower than simply downloading the entire database.

FHE-Based PIR. Most FHE-based PIR schemes build upon the oblivious expansion technique, first introduced in SealPIR [3]. This technique enables the server to expand an encrypted index into a selection vector, which is then multiplied with the hypercube-structured database. To avoid the high cost of ciphertext-ciphertext multiplications, SealPIR [3] treats encrypted payloads as "plaintexts", replacing a single expensive ciphertext-ciphertext multiplication with multiple plaintext-ciphertext multiplications. However, this comes at the cost of significantly larger response sizes. OnionPIR [39] and MulPIR [2] instead use the external product technique [14] to perform ciphertext-ciphertext multiplications, reducing response sizes but introducing substantial computational overhead. Spiral [37] refines this approach by combining Regev encryption [42] with GSW encryption [20], enabling a more efficient external product. Respire [9] further optimizes Spiral for databases with small records.

The SIMD feature of an FHE scheme can be leveraged to achieve batch PIR [35,38]. Specifically, each slot in the SIMD ciphertext can be used to encrypt a separate query, which allows the server's computation cost to be amortized to $O(n/N)$, even though it still performs $O(n)$ computation for the entire batch. However, this approach requires the client to issue at least N parallel queries simultaneously to amortize the cost. In real-world scenarios, however, the client may prefer to adaptively decide their subsequent queries based on the results of previous ones.

Sublinear PIR with Client-Side State. Recent work also considers sublinear PIR with client-side state [22, 26, 43, 47, 49], where each client performs a separate preprocessing phase with the server and stores a hint related to the database. For example, Piano [49] has the client to stream the entire database to generate $O(n^{1/2})$ hints, which allow for efficient processing of $O(n^{1/2})$ online queries, with each query requiring $O(n^{1/2})$ communication and $O(n^{1/2})$ plaintext operations. Ren et al. [43], Plinko [26] and Wang et al. [47] achieve similar asymptotic efficiency as Piano, but with better concrete performance. Plinko [26] also supports updating the hints when the database changes. Ghoshal et al. [22] reduce the online communication of Piano to $O(n^{1/4})$, with a trade-off of introducing the "broken hint" issue. Compared to this line of work, DEPIR offers the advantage of amortizing the same preprocessing across an unlimited number of clients, hence the total preprocessing effort and space consumption are independent of the number of clients.

Multi-Server DEPIR. Recent work on multi-server DEPIR schemes [21, 32] utilizes secret-sharing techniques to achieve information-theoretic security. Building on the multi-server PIR protocol of [48], which evaluates a polynomial and its derivatives at distinct points, these schemes incorporate the KU method to enable sublinear evaluation time. Despite not relying on ASHE or FHE, their dependence on multiple non-colluding servers maintaining synchronized database copies often renders them impractical for real-world applications.

10 Conclusion

This paper presents algorithmic innovations that move single-server DEPIR significantly closer to being practical. Addressing the prohibitive concrete efficiency gap in prior work, our approach yields substantial gains. Specifically, compared to the state-of-the-art, we achieve a reduction of up to 4 orders of magnitude in server state and 6 orders of magnitude in query time. These advancements bridge the gap between theoretical promise and real-world feasibility.

Acknowledgments. We thank Simon Pohmann for answering our questions. This work was supported by the National Natural Science Foundation of China (No. U25A20425).

References

1. Agarwal, R., Burrus, C.: Number theoretic transforms to implement fast digital convolution. Proceedings of the IEEE **63**(4), 550–560 (1975). https://doi.org/10.1109/PROC.1975.9791
2. Ali, A., Lepoint, T., Patel, S., Raykova, M., Schoppmann, P., Seth, K., Yeo, K.: Communication-computation trade-offs in PIR. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, August 11-13, 2021. pp. 1811–1828. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/ali
3. Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 962–979. IEEE Computer Society (2018). https://doi.org/10.1109/SP.2018.00062
4. Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 311–323. Springer (1986)
5. Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers' computation in private information retrieval: PIR with preprocessing. J. Cryptol. **17**(2), 125–151 (Mar 2004). https://doi.org/10.1007/s00145-004-0134-y
6. Bhargava, V., Ghosh, S., Guo, Z., Kumar, M., Umans, C.: Fast multivariate multipoint evaluation over all finite fields. Journal of the ACM **71**(3), 1–32 (2024)
7. Boyle, E., Ishai, Y., Pass, R., Wootters, M.: Can we access a database both locally and privately? In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. pp. 662–693. Springer International Publishing, Cham (2017)

8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. p. 309–325. ITCS '12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2090236.2090262
9. Burton, A., Menon, S.J., Wu, D.J.: Respire: High-rate PIR for databases with small records. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024. pp. 1463–1477. ACM (2024). https://doi.org/10.1145/3658644.3690328
10. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. Lecture Notes in Computer Science, vol. 1592, pp. 402–414. Springer (1999). https://doi.org/10.1007/3-540-48910-X_28
11. Canetti, R., Holmgren, J., Richelson, S.: Towards doubly efficient private information retrieval. In: Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II. p. 694–726. Springer-Verlag, Berlin, Heidelberg (2017). https://doi.org/10.1007/978-3-319-70503-3_23
12. Challis, M.F., Robinson, J.P.: Some extremal postage stamp bases. Journal of Integer Sequences **13**(2), 3 (2010)
13. Chang, Y.: Single database private information retrieval with logarithmic communication. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3108, pp. 50–61. Springer (2004). https://doi.org/10.1007/978-3-540-27800-9_5
14. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2020). https://doi.org/10.1007/s00145-019-09319-x
15. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. p. 1135–1150. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3460120.3484760
16. Corless, R.M., Gonnet, G.H., Hare, D.E., Jeffrey, D.J., Knuth, D.E.: On the Lambert W function. Advances in Computational mathematics **5**(1), 329–359 (1996)
17. Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of Paillier's public-key system with applications to electronic voting. Int. J. Inf. Sec. **9**(6), 371–385 (2010). https://doi.org/10.1007/s10207-010-0119-9
18. FLINT, https://github.com/flintlib/flint
19. Gentry, C., Ramzan, Z.: Single-database private information retrieval with constant communication rate. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005. Lecture Notes in Computer Science, vol. 3580, pp. 803–815. Springer (2005). https://doi.org/10.1007/11523468_65
20. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti,

R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference. Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_5
21. Ghoshal, A., Li, B., Ma, Y., Dai, C., Shi, E.: Scalable multi-server private information retrieval. Cryptology ePrint Archive, Paper 2024/765 (2024), https://eprint.iacr.org/2024/765
22. Ghoshal, A., Zhou, M., Shi, E.: Efficient pre-processing PIR without public-key cryptography. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI. Lecture Notes in Computer Science, vol. 14656, pp. 210–240. Springer (2024). https://doi.org/10.1007/978-3-031-58751-1_8
23. Halevi, S., Polyakov, Y., Shoup, V.: An improved RNS variant of the BFV homomorphic encryption scheme. In: Cryptographers' Track at the RSA Conference. pp. 83–105. Springer (2019)
24. Henzinger, A., Hong, M.M., Corrigan-Gibbs, H., Meiklejohn, S., Vaikuntanathan, V.: One server for the price of two: simple and fast single-server private information retrieval. In: Proceedings of the 32nd USENIX Conference on Security Symposium. SEC '23, USENIX Association, USA (2023)
25. HEXL, https://github.com/IntelLabs/hexl
26. Hoover, A., Patel, S., Persiano, G., Yeo, K.: Plinko: Single-server PIR with efficient updates via invertible PRFs. In: Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, Springer (2025), https://eprint.iacr.org/2024/318
27. io-uring, https://github.com/axboe/liburing
28. Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM J. Comput. **40**(6), 1767–1802 (Dec 2011). https://doi.org/10.1137/08073408X
29. Kirfel, C.: Extremale asymptotische reichweitenbasen. Acta Arithmetica **60**(3), 279–288 (1992), http://eudml.org/doc/206438
30. Kuku, https://github.com/microsoft/Kuku
31. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science. p. 364. FOCS '97, IEEE Computer Society, USA (1997)
32. Lazzaretti, A., Liu, Z., Fisch, B., Miao, P., Papamanthou, C.: Multi-server doubly efficient PIR in the classical model and beyond. Cryptology ePrint Archive, Paper 2024/829 (2024), https://eprint.iacr.org/2024/829
33. Lin, W.K., Mook, E., Wichs, D.: Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In: Proceedings of the 55th Annual ACM Symposium on Theory of Computing. p. 595–608. STOC 2023, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3564246.3585175
34. Lipmaa, H.: An oblivious transfer protocol with log-squared communication. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) Information Security, 8th International Conference, ISC 2005, Singapore, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3650, pp. 314–328. Springer (2005). https://doi.org/10.1007/11556992_23

35. Liu, J., Li, J., Wu, D., Ren, K.: PIRANA: Faster Multi-query PIR via Constant-weight Codes. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4315–4330. IEEE Computer Society, Los Alamitos, CA, USA (May 2024). https://doi.org/10.1109/SP54263.2024.00039
36. LWE-Estimator, https://bitbucket.org/malb/lwe-estimator
37. Menon, S.J., Wu, D.J.: SPIRAL: fast, high-rate single-server PIR via FHE composition. In: 43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022. pp. 930–947. IEEE (2022). https://doi.org/10.1109/SP46214.2022.9833700
38. Mughees, M., Ren, L.: Vectorized batch private information retrieval. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1812–1827. IEEE Computer Society, Los Alamitos, CA, USA (may 2023). https://doi.org/10.1109/SP46215.2023.00104
39. Mughees, M.H., Chen, H., Ren, L.: OnionPIR: Response efficient single-server PIR. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security 2021. pp. 2292–2306. ACM (2021). https://doi.org/10.1145/3460120.3485381
40. Okada, H., Player, R., Pohmann, S., Weinert, C.: On algebraic homomorphic encryption and its applications to doubly-efficient PIR. In: Advances in Cryptology – EUROCRYPT 2025: 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4–8, 2025, Proceedings, Part VI. p. 34–64. Springer-Verlag, Berlin, Heidelberg (2025). https://doi.org/10.1007/978-3-031-91095-1_2
41. Okada, H., Player, R., Pohmann, S., Weinert, C.: Towards practical doubly-efficient private information retrieval. In: Financial Cryptography and Data Security: 28th International Conference, FC 2024, Willemstad, Curaçao, March 4–8, 2024, Revised Selected Papers, Part II. p. 264–282. Springer-Verlag, Berlin, Heidelberg (2025). https://doi.org/10.1007/978-3-031-78679-2_14
42. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005. pp. 84–93. ACM (2005). https://doi.org/10.1145/1060590.1060603
43. Ren, L., Mughees, M.H., Sun, I.: Simple and practical amortized sublinear private information retrieval using dummy subsets. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. p. 1420–1433. CCS '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3658644.3690266
44. Rosser, J.B., Schoenfeld, L.: Approximate formulas for some functions of prime numbers. Illinois Journal of Mathematics **6**(1), 64–94 (1962)
45. SEAL, https://github.com/microsoft/SEAL
46. Sion, R., Carbunar, B.: On the practicality of private information retrieval. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, USA, 28th February - 2nd March 2007. The Internet Society (2007), https://www.ndss-symposium.org/ndss2007/practicality-private-information-retrieval/
47. Wang, Z., Ren, L.: Single-server client preprocessing PIR with tight space-time trade-off. In: Fehr, S., Fouque, P.A. (eds.) Advances in Cryptology – EUROCRYPT 2025. pp. 94–122. Springer Nature Switzerland, Cham (2025)
48. Woodruff, D.P., Yekhanin, S.: A geometric approach to information-theoretic private information retrieval. In: 20th Annual IEEE Conference on Computational

Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA. pp. 275–284. IEEE Computer Society (2005). https://doi.org/10.1109/CCC.2005.2
49. Zhou, M., Park, A., Zheng, W., Shi, E.: Piano: Extremely simple, single-server PIR with sublinear server computation. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 4296–4314 (2024)

A Frequently used notations

Table 5. Frequently used notations.

Notation	Description
(a, b)	an RLWE ciphertext
c, C	coefficient
D	total degree
d	individual degree
e	exponent
$F()$, $f()$, $g()$, $G()$, $h()$	polynomial
ℓ	parameter for polynomial decomposition
L	# queries in a batch
m	# variables
M	the maximum value of $f()$
n	lattice dimension
N	# elements in the database
$\mathbb{Z}_p$	plaintext space
$\mathbb{Z}_q$, $\mathbb{Z}_Q$	ciphertext space
R, R_p, R_q, R_Q, R_R	ring
r	$Q = q_1 \cdots q_r$
s	encryption key
t	# primes

B Proof of Lemma 1

Proof. By Fact 2, we have

$$\pi(B) = \sum_{p \le B} 1 = (1 + o(1))\frac{B}{\ln B} = \frac{B}{\ln B} + o(1)\frac{B}{\ln B}.$$

Then,

$$\sum_{p \le B} p^m = \sum_{i=1}^{\frac{B}{\ln B}} p_i^m + \sum_{j=\frac{B}{\ln B}+1}^{\pi(B)} p_j^m \le \sum_{i=1}^{\frac{B}{\ln B}} p_i^m + o(1)\frac{B}{\ln B}B^m.$$

By the fact that the i-th prime is approximately equal to $i \ln i$. Then

$$\sum_{p \le B} p^m \le \sum_{i=1}^{\frac{B}{\ln B}} p_i^m + o(1)\frac{B}{\ln B}B^m = \sum_{i=1}^{\frac{B}{\ln B}} (i \ln i)^m + o(\frac{B^{m+1}}{\ln B}).$$

Then,

$$\begin{aligned}
&\sum_{p \le B} p^m \ln p \le \sum_{i=1}^{\frac{B}{\ln B}} (i \ln i)^m \ln(i \ln i) + o(\frac{B^{m+1}}{\ln B}) \ln B \\
&= \sum_{i=1}^{\frac{B}{\ln B}} (i \ln i)^m \ln i + \sum_{i=1}^{\frac{B}{\ln B}} (i \ln i)^m \ln \ln i + o(B^{m+1}) \\
&= (1 + o(1)) \sum_{i=1}^{\frac{B}{\ln B}} (i \ln i)^m \ln i + o(B^{m+1}) \\
&= (1 + o(1)) \int_1^{\frac{B}{\ln B}} x^m (\ln x)^{m+1} dx + o(B^{m+1}) \\
&= (1 + o(1)) \left. \frac{x^{m+1}}{m+1} (\ln x)^{m+1} \right|_1^{\frac{B}{\ln B}} \\
&\quad - (1 + o(1)) \frac{m+1}{m+1} \int_1^{\frac{B}{\ln B}} x^m (\ln x)^m dx + o(B^{m+1}) \\
&= (1 + o(1)) \frac{B^{m+1}}{m+1} - O((\frac{B}{\ln B})^{m+1} (\ln \frac{B}{\ln B}))^m + o(B^{m+1}) \\
&= (1 + o(1)) \frac{B^{m+1}}{m+1} - O(\frac{B^{m+1}}{\ln B}) + o(B^{m+1}) \\
&= (1 + o(1)) \frac{B^{m+1}}{m+1}.
\end{aligned}$$

□

C Proof of Theorem 5: storage and runtime

Storage size. By Fact 2, when $p_t \ge 101$ we have

$$0.84 p_t \le \sum_{p \le p_t} \ln p \le 1.02 p_t \;\Rightarrow\; e^{0.84 p_t} \le \prod_{p \le p_t} p \le e^{1.02 p_t}.$$

Define

$$e^{\beta p_t} = \prod_{p \le p_t} p \text{ with } 0.84 \le \beta \le 1.02,$$

$$b = \prod_{i=1}^{s} p_i^k, \tag{9}$$

and

$$\alpha^{D+1} = 2N \left(\frac{1}{2}\right)^{D+1}. \tag{10}$$

Then, Equation 5 can be transformed into

$$\begin{aligned}
&\prod_{i=1}^{s} p_i^k \cdot \prod_{j=s+1}^{t} p_j \geq 2N(q/2)^{D+1} \\
\Rightarrow &\prod_{i=1}^{s} p_i^{k-1} \prod_{p \leq p_t} p \geq \alpha^{D+1} q^{D+1} \\
\Rightarrow &b^{\frac{k-1}{k}} e^{\beta p_t} \geq \alpha^{D+1} q^{D+1} \\
\Rightarrow &p_t \geq \frac{D+1}{\beta} \ln \alpha q - \frac{k-1}{\beta k} \ln b.
\end{aligned} \tag{11}$$

Equation 6 can be transformed into

$$\begin{aligned}
&\prod_{i=1}^{s} p_i^k \geq 2N(p_t/2)^{D+1} \\
\Rightarrow &b \geq \alpha^{D+1} p_t^{D+1} \\
\Rightarrow &p_t \leq \alpha^{-1} b^{\frac{1}{D+1}}.
\end{aligned} \tag{12}$$

Combine Equation 11 and 12, we obtain

$$\alpha^{-1} b^{\frac{1}{D+1}} \geq \frac{D+1}{\beta} \ln \alpha q - \frac{k-1}{\beta k} \ln b.$$

Define

$$\gamma^{D+1} (\alpha q)^{D+1} = b, \tag{13}$$

then

$$\begin{aligned}
&\alpha^{-1} b^{\frac{1}{D+1}} \geq \frac{D+1}{\beta} \ln \alpha q - \frac{k-1}{\beta k} \ln b \\
\Rightarrow &\gamma q \geq \frac{D+1}{\beta} \ln \alpha q - \frac{k-1}{\beta k} (D+1) \ln \gamma \alpha q \\
\Rightarrow &\beta \gamma q + \frac{k-1}{k} (D+1) \ln \gamma \alpha q - (D+1) \ln \alpha q \geq 0 \\
\Rightarrow &\frac{\beta \gamma q k}{D+1} + (k-1) \ln \gamma \alpha q - k \ln \alpha q \geq 0 \\
\Rightarrow &\frac{\beta \gamma q k}{D+1} + (k-1) \ln \gamma + (k-1) \ln \alpha q - k \ln \alpha q \geq 0 \\
\Rightarrow &\frac{\beta \gamma q k}{D+1} + (k-1) \ln \gamma - \ln \alpha q \geq 0.
\end{aligned}$$

Define $(\alpha q)^{\frac{1}{k-1}}\delta = \gamma$, then

$$\begin{aligned}
&\frac{\beta\gamma qk}{D+1} + (k-1)\ln\gamma - \ln\alpha q \geq 0 \\
&\Rightarrow \frac{\beta qk}{D+1}(\alpha q)^{\frac{1}{k-1}}\delta + (k-1)\ln(\alpha q)^{\frac{1}{k-1}} + (k-1)\ln\delta - \ln\alpha q \geq 0 \\
&\Rightarrow \frac{\beta qk}{D+1}(\alpha q)^{\frac{1}{k-1}}\delta + (k-1)\ln\delta \geq 0 \Rightarrow \frac{k}{k-1}\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\delta + \ln\delta \geq 0 \\
&\Rightarrow e^{\frac{k}{k-1}\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\delta + \ln\delta} \geq 1 \Rightarrow \delta \cdot e^{\frac{k}{k-1}\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\delta} \geq 1 \\
&\Rightarrow \frac{k}{k-1}\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\delta \cdot e^{\frac{k}{k-1}\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\delta} \geq \frac{k}{k-1}\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}.
\end{aligned}$$

This inequality has solutions given by the Lambert W function:

$$\delta \geq \frac{k-1}{k}\cdot\frac{D+1}{\beta q}\cdot(\alpha q)^{-\frac{1}{k-1}}\cdot W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right).$$

By Equation 13, we obtain

$$\begin{aligned}
b = \gamma^{D+1}(\alpha q)^{D+1} &= ((\alpha q)^{\frac{1}{k-1}}\delta)^{D+1}(\alpha q)^{D+1} \qquad (14)\\
&\geq \left(\frac{k-1}{k}\cdot\frac{D+1}{\beta q}\right)^{D+1} W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)^{D+1}(\alpha q)^{D+1} \\
&= \left(\frac{k-1}{k}\cdot\frac{\alpha(D+1)}{\beta}\right)^{D+1} W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)^{D+1}. \qquad (15)
\end{aligned}$$

Define $B \geq p_s$ s.t. $\prod\limits_{p\leq B} p = \prod\limits_{p\leq p_s} p = \prod\limits_{i=1}^{s} p_i$. By Fact 2, we have

$$\theta(B) = \sum_{p\leq B}\ln p = (1+o(1))B \Rightarrow \prod_{p\leq B} p = e^{(1+o(1))B} \Rightarrow \prod_{i=1}^{s} p_i^k = e^{(1+o(1))kB}.$$

By Equation 9 and 15, we have

$$\begin{aligned}
&b = \prod_{i=1}^{s} p_i^k = e^{(1+o(1))kB} \qquad (16)\\
&\Rightarrow \left(\frac{k-1}{k}\cdot\frac{\alpha(D+1)}{\beta}\right)^{D+1} W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)^{D+1} \leq e^{(1+o(1))kB} \\
&\Rightarrow B \geq \frac{(1+o(1))}{k}\ln\left(\left(\frac{k-1}{k}\cdot\frac{\alpha(D+1)}{\beta}\right)^{D+1} W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)^{D+1}\right) \\
&\Rightarrow B \geq \frac{(1+o(1))}{k}(D+1)\ln\left(\left(\frac{k-1}{k}\cdot\frac{\alpha(D+1)}{\beta}\right) W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)\right).
\end{aligned}$$

Since we want p_s to be as small as possible, we set

$$B = \frac{(1+o(1))}{k}(D+1)\ln(\left(\frac{k-1}{k}\cdot\frac{\alpha(D+1)}{\beta}\right)W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)).$$

By the property of the Lambert W function:

$$W_0(z)e^{W_0(z)} = z \Rightarrow \ln W_0(z) + W_0(z) = \ln z.$$

By the fact $W_0(z) = \ln z - \ln\ln z + o(1)$, we have

$$\ln W_0(z) = \ln\ln z + o(1).$$

Then,

$$\begin{aligned}
B &= \frac{(1+o(1))(D+1)}{k}\ln(\left(\frac{k-1}{k}\cdot\frac{\alpha(D+1)}{\beta}\right)W_0\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)) \\
&= \frac{(1+o(1))(D+1)}{k}\Big(\ln(D+1) + \ln\frac{(k-1)\alpha}{k\beta} \\
&\quad + \ln\ln\left(\frac{k}{k-1}\cdot\frac{\beta q}{D+1}(\alpha q)^{\frac{1}{k-1}}\right)\Big) \\
&= \frac{(1+o(1))(D+1)}{k}\Big(\ln(D+1) + \ln\frac{(k-1)\alpha}{k\beta} \\
&\quad + \ln\left(\ln\frac{q^{\frac{k}{k-1}}}{D+1} + \ln\frac{k\beta\alpha^{\frac{1}{k-1}}}{k-1}\right)\Big) \\
&= \frac{1+o(1)}{k}(D+1)\left(\ln(D+1) + \ln\ln\frac{q^{\frac{k}{k-1}}}{D+1}\right) \\
&\leq \frac{1+o(1)}{k}(D+1)(\ln(D+1) + \ln\ln q^{\frac{k}{k-1}}) \\
&= \frac{1+o(1)}{k}(D+1)(\ln(D+1) + \ln\frac{k}{k-1} + \ln\ln q) \\
&= \frac{1+o(1)}{k}(D+1)(\ln(D+1) + \ln\ln q). \qquad (17)
\end{aligned}$$

Based on the storage size computed in Lemma 2, the required storage (in bits) here is:

$$\sum_{p\leq B}\binom{m+k-1}{m}p^m\log p^k = \frac{1}{\ln 2}\binom{m+k-1}{m}\sum_{p\leq B}p^m\ln p^k.$$

By Lemma 1, the storage size can be re-written as:

$$\frac{1}{\ln 2}\binom{m+k-1}{m}\sum_{p\leq B}p^m\ln p^k \leq (1+o(1))\frac{k}{\ln 2}\binom{m+k-1}{m}\frac{B^{m+1}}{m+1}.$$

By Equation 17, the storage size is

$$\begin{aligned}
&(1+o(1))\frac{k}{\ln 2}\binom{m+k-1}{m}\frac{B^{m+1}}{m+1}\\
=&(1+o(1))\frac{\binom{m+k-1}{m}}{(m+1)k^m\ln 2}(D+1)^{m+1}(\ln(D+1)+\ln\ln q)^{m+1}\\
=&(1+o(1))\frac{\binom{m+k-1}{m}\ln^m 2}{(m+1)k^m}(D+1)^{m+1}(\log(D+1)+\log\log q+\log\ln 2)^{m+1}\\
=&(1+o(1))\frac{\binom{m+k-1}{m}}{(m+1)k^m\log^m e}(D+1)^{m+1}(\log(D+1)+\log\log q)^{m+1}.
\end{aligned}$$

Runtime. By Fact 2, we have

$$t=\pi(p_t)=\sum_{p\le p_t}1=(1+o(1))\frac{p_t}{\ln p_t}.$$

Recall that the evaluation phase run Lemma 2 for each $i\in[s]$ and run Lemma 3 for each $j\in[s+1,t]$. Then, the total runtime is

$$\begin{aligned}
&s\cdot O(\binom{m+k-1}{m})+\sum_{j=s+1}^{t}O(\binom{m+k-1}{m}\frac{(D+1)\log p_j}{k\log(D+1)+k\log\log p_j})\\
\le&\sum_{i=1}^{t}O(\binom{m+k-1}{m}\frac{(D+1)\log p_i}{k\log(D+1)+k\log\log p_i})\\
\le&tO(\binom{m+k-1}{m}\frac{(D+1)\log p_t}{k\log(D+1)+k\log\log p_t})\\
=&O(\binom{m+k-1}{m}\frac{t(D+1)\log p_t}{k\log(D+1)+k\log\log p_t})\\
=&O(\binom{m+k-1}{m}\frac{\frac{p_t}{\ln p_t}(D+1)\log p_t}{k\log(D+1)+k\log\log p_t})\\
=&O(\frac{\log e}{k}\binom{m+k-1}{m}\frac{(D+1)p_t}{\log(D+1)+\log\log p_t}).
\end{aligned}$$

By Equation 12, 16 and 17, we have

$$\begin{aligned}
&p_t\le\alpha^{-1}b^{\frac{1}{D+1}}=\alpha^{-1}e^{\frac{1}{D+1}(1+o(1))kB}=\alpha^{-1}e^{(1+o(1))(\ln(D+1)+\ln\ln q)}\\
&=(1+o(1))\alpha^{-1}(D+1)\ln q.
\end{aligned}$$

The total runtime is

$$\begin{aligned}
&O(\frac{\log e}{k}\binom{m+k-1}{m}\frac{(D+1)p_t}{\log(D+1)+\log\log p_t})\\
\le&O(\frac{\log e}{k}\binom{m+k-1}{m}\frac{\alpha^{-1}(D+1)^2\ln q}{\log(D+1)+\log\log(\alpha^{-1}(D+1)\ln q)}).
\end{aligned}$$

Given $\alpha^{-1} = (\frac{1}{2N})^{\frac{1}{D}} \cdot 2 < 2$,

$$
\begin{aligned}
&O(\frac{\log e}{k}\binom{m+k-1}{m}\frac{\alpha^{-1}(D+1)^2\ln q}{\log(D+1)+\log\log(\alpha^{-1}(D+1)\ln q)})\\
\leq&O(\frac{\log e}{k}\binom{m+k-1}{m}\frac{2(D+1)^2\ln q}{\log(D+1)+\log\log\alpha^{-1}+\log\log(D+1)+\log\log\ln q})\\
=&O(\frac{\log e}{k}\binom{m+k-1}{m}\frac{(D+1)^2\ln q}{\log(D+1)+\log\log(D+1)+\log\log\log q})\\
=&O(\frac{1}{k}\binom{m+k-1}{m}\frac{(D+1)^2\log e\ln q}{\log(D+1)+\log\log(D+1)+\log\log\log q})\\
=&O(\frac{1}{k}\binom{m+k-1}{m}\frac{(D+1)^2\log q}{\log(D+1)+\log\log(D+1)+\log\log\log q}).
\end{aligned}
$$

D Hardness of our RLWE variant

We prove that the fixed-$\mathbf{b}$ variant used in our construction is hard for any fixed polynomial-length sequence of messages, assuming the hardness of Definition 1 with uniformly distributed $\mathbf{b}$. The proof follows the hybrid argument in Corollary C.3 of SimplePIR [24]. Whereas SimplePIR reuses the public LWE matrix, we reuse the public vector $\mathbf{b}$ and sample an independent secret and fresh errors for each batch.

Theorem 6 (RLWE variant used in our paper). *Let R be a number ring, q, m be positive integers, $\mathcal{B}$,$\mathcal{E}$ distributions over R_q, and $\mathcal{S}$ a distribution over the unit group R_q^*. Let λ be the security parameter and $T = \mathsf{poly}(\lambda)$. Define the following two distributions:*

- *In $\mathsf{Real}^T_{R,q,\$R_q,\mathcal{E},\mathcal{S},m}$, on input $(\boldsymbol{\mu}^{(1)},\ldots,\boldsymbol{\mu}^{(T)})$ with each $\boldsymbol{\mu}^{(t)} \in R_p^m$, the oracle returns $(\mathbf{b},\mathbf{a}^{(1)},\ldots,\mathbf{a}^{(T)})$, where $\mathbf{b} \xleftarrow{\$} R_q^m$, $\forall t \in [T]$, $s^{(t)} \leftarrow \mathcal{S}$, $\boldsymbol{\epsilon}^{(t)} \leftarrow \mathcal{E}^m$, and $\mathbf{a}^{(t)} := (s^{(t)})^{-1}(\mathbf{b} - \boldsymbol{\mu}^{(t)} - \boldsymbol{\epsilon}^{(t)})$.*
- *In $\mathsf{Rand}^T_{R,q,m}$, on input $(\boldsymbol{\mu}^{(1)},\ldots,\boldsymbol{\mu}^{(T)})$ with each $\boldsymbol{\mu}^{(t)} \in R_p^m$, the oracle returns $(\mathbf{b},\mathbf{r}^{(1)},\ldots,\mathbf{r}^{(T)})$, where $\mathbf{b} \xleftarrow{\$} R_q^m$, $\forall t \in [T]$, $\mathbf{r}^{(t)} \xleftarrow{\$} R_q^m$.*

If $\mathsf{RLWE}(R,q,\$R_q,\mathcal{E},\mathcal{S},m)$ is hard, then

$$\mathsf{Real}^T_{R,q,\$R_q,\mathcal{E},\mathcal{S},m} \approx_c \mathsf{Rand}^T_{R,q,m}.$$

More precisely, if a PPT distinguisher $\mathcal{D}$ distinguishes these two distributions with advantage ε, then there is a PPT algorithm $\mathcal{A}$ solving $\mathsf{RLWE}(R,q,\$R_q,\mathcal{E},\mathcal{S},m)$ with advantage ε/T.

Proof. For each $j \in \{0,\ldots,T\}$, define a hybrid distribution H_j as follows:

1. sample $\mathbf{b} \xleftarrow{\$} R_q^m$;

2. for each $t \le j$, sample $\mathbf{r}^{(t)} \xleftarrow{\$} R_q^m$;
3. for each $t > j$, sample $s^{(t)} \leftarrow \mathcal{S}$, $\boldsymbol{\epsilon}^{(t)} \leftarrow \mathcal{E}^m$, and set $\mathbf{a}^{(t)} := \big(s^{(t)}\big)^{-1}\big(\mathbf{b} - \boldsymbol{\mu}^{(t)} - \boldsymbol{\epsilon}^{(t)}\big)$.

Thus,

$$H_j = \big(\mathbf{b}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(j)}, \mathbf{a}^{(j+1)}, \ldots, \mathbf{a}^{(T)}\big).$$

Clearly,

$$H_0 = \mathsf{Real}^T_{R,q,\$R_q,\mathcal{E},\mathcal{S},m} \text{ and } H_T = \mathsf{Rand}^T_{R,q,m}.$$

Define

$$p_j := \Pr[\mathcal{D}(H_j) = 1].$$

If $\mathcal{D}$ distinguishes H_0 and H_T with advantage ε, then

$$\varepsilon = |p_T - p_0| = |\sum_{j=1}^{T}(p_j - p_{j-1})| \le \sum_{j=1}^{T}|p_j - p_{j-1}|$$

Thus, there exists some $j^* \in \{1, \ldots, T\}$ such that

$$|p_{j^*} - p_{j^*-1}| = |\Pr[\mathcal{D}(H_{j^*}) = 1] - \Pr[\mathcal{D}(H_{j^*-1}) = 1]| \ge \frac{\varepsilon}{T}$$

Next, we construct a PPT algorithm $\mathcal{A}$ for $\mathsf{RLWE}(R, q, \$R_q, \mathcal{E}, \mathcal{S}, m)$:

1. samples $\boldsymbol{\mu}^{(j^*)}$ and queries the $\mathsf{RLWE}(R, q, \$R_q, \mathcal{E}, \mathcal{S}, m)$ oracle on $\boldsymbol{\mu}^{(j^*)}$ and receives $(\mathbf{a}^*, \mathbf{b}^*)$;
2. for each $t < j^*$, samples $\mathbf{r}^{(t)} \xleftarrow{\$} R_q^m$;
3. places $\mathbf{a}^*$ at position j^*;
4. for each $t > j^*$, samples $s^{(t)} \leftarrow \mathcal{S}$, $\boldsymbol{\epsilon}^{(t)} \leftarrow \mathcal{E}^m$ and sets $\mathbf{a}^{(t)} := \big(s^{(t)}\big)^{-1}\big(\mathbf{b}^* - \boldsymbol{\mu}^{(t)} - \boldsymbol{\epsilon}^{(t)}\big)$.
5. applies $\mathcal{D}$ to the resulting transcript.

In Scenario 1 of Definition 1, the challenge $(\mathbf{a}^*, \mathbf{b}^*)$ is an honest answer for $\boldsymbol{\mu}^{(j^*)}$. Hence, the simulated transcript is distributed exactly as H_{j-1}. In Scenario 2, $\mathbf{a}^*$ and $\mathbf{b}^*$ are independent and uniform. Hence, the simulated transcript is distributed exactly as H_j. Therefore, $\mathcal{A}$ solves $\mathsf{RLWE}(R, q, \$R_q, \mathcal{E}, \mathcal{S}, m)$ with advantage at least $|p_{j^*} - p_{j^*-1}| \ge \frac{\varepsilon}{T}$.

Since $\mathsf{RLWE}(R, q, \$R_q, \mathcal{E}, \mathcal{S}, m)$ is hard and $T = \mathrm{poly}(\lambda)$, $\mathrm{Adv}_{\mathcal{A}}$ is negligible, and hence ε is also negligible.