Doubly Efficient Batched Private Information Retrieval

Xiuquan Ding[1,2], Giulio Malavolta[3,4], and Tianwei Zhang[4,5]

[1]KLMM, Academy of Mathematics and Systems Science, Chinese Academy of Sciences
[2] School of Mathematical Sciences, University of Chinese Academy of Sciences
[3]Bocconi University
[4]Max Planck Institute for Security and Privacy
[5]Ruhr University Bochum

Abstract

Private information retrieval (PIR) allows a client to read data from a server, without revealing which information they are interested in. A PIR is *doubly efficient* if the server runtime is, after a one-time pre-processing, sublinear in the database size. A recent breakthrough result from Lin, Mook, and Wichs [STOC'23] proposed the first-doubly efficient PIR with (online) server computation poly-logarithmic in the size of the database, assuming the hardness of the standard Ring-LWE problem.

In this work, we consider the problem of doubly efficient *batched* PIR (DEBPIR), where the client wishes to download *multiple entries*. This problem arises naturally in many practical applications of PIR, or when the database contains large entries. Our main result is a construction of DEBPIR where the *amortized* communication and server computation overhead is $\tilde{O}(1)$, from the Ring-LWE problem. This represents an *exponential* improvement compared with known constructions, and it is optimal up to poly-logarithmic factors in the security parameter. Interestingly, the server's online operations are entirely combinatorial and all algebraic computations are done in the pre-processing or delegated to the client.

1 Introduction

Private information retrieval (PIR) [12] enables a client to download an element from a public database without revealing to the database server which record is being requested. Beyond its direct applications to private database queries, PIR serves as a core building block in a wide range of privacy-preserving applications such as anonymous messaging [1,2,21,24], contact discovery [7,13] and safe browsing [20]. As an illustration, suppose a client wishes to retrieve a video from YouTube without revealing which specific video they are interested in. The goal of PIR is to solve this problem with low communication complexity, sub-linear in the database size N, and ideally just polynomial in $\log N$. However, PIR comes with a major limitation: It inherently requires the server to read the entire database DB during every protocol execution. In our working example, YouTube would have to perform extensive computations across its entire video database merely to provide a single video to a client.

To overcome this limitation, the concept of doubly efficient PIR (DEPIR) was pioneered by Beimel, Ishai and Malkin [6]. In the DEPIR scheme [6, 8, 11, 22], the database $DB \in \{0,1\}^N$ is

preprocessed into some static data structure $\tilde{DB}$ that is stored by the server. Subsequently, any client can run a PIR protocol with the server to retrieve the value $DB[i]$ without revealing the index i, where both the communication and the server/client computation during the protocol are sub-linear in the database size N, and ideally polynomial in $\log N$. In particular, this means that during each PIR protocol execution, the server merely accesses a sub-linear number of locations in the data structure $\tilde{DB}$. Constructing a DEPIR with poly-logarithmic complexity has been a long-standing open problem for more than two decades, and it was recently solved in the breakthrough result of Lin, Mook and Wichs [22], who proposed a construction from the Ring-LWE assumption. More precisely, their scheme achieves server/client computation $poly(\lambda, \log N)$, where λ is the security parameter. For single-bit queries, the complexity of this scheme is optimal up to polynomial factors in λ.

In this work, we are interested in the settings of batched PIR where clients want to retrieve *multiple entries* from the same database [1, 2, 15, 16, 25]. Batched PIR matches the use case of many practical applications and holds the potential for substantial efficiency improvements over PIR in terms of amortized cost per query. Our motivations are two-fold: On the one hand, many applications of DEPIR naturally require multiple queries from the same database. For instance, fetching multiple messages from an anonymous messaging system [3], downloading multiple ads [26], checking which contacts signed up for a service [17], or checking all passwords at once against a database of breached passwords [20]. On the other hand, in the settings of multiple entries (or, equivalently, of large entries) we can hope to *amortize* the computation/communication of the server, to achieve even better asymptotics. However, the current best solution is to simply run the protocol from [22] in parallel. Motivated by this state of affairs, we ask the following question:

Can we construct a batched DEPIR with optimal amortized complexity?

1.1 Our Contributions

In this work, we construct the first doubly efficient *batched* PIR (DEBPIR) scheme with an amortized server response time and communication cost of $poly(\log \lambda)$, assuming the hardness of the standard Ring-LWE assumption [23], with quasi-polynomial approximation factors. Compared to the recent work of [22], which has a $poly(\lambda)$ overhead even in the amortized settings, our result represents an *exponential* improvement in the server efficiency and it is in fact optimal up to poly-logarithmic factors in λ. Similarly to [22], our protocol consists only of two messages from the client to the server and back. Below, we summarize the main result of this work.

Theorem 1.1 (Informal)**.** Assuming Ring-LWE with quasi-polynomial approximation factors, there exists a DEBPIR scheme such that, after offline preprocessing of the database, for security parameter λ and a database of size $N = poly(\lambda)$, the amortized response time of the server and the amortized communication cost are

$$poly(\log \lambda, \log N) = poly(\log \lambda) = \tilde{O}_\lambda(1).$$

At a technical level, our result builds on a new variant of (somewhat) homomorphic encryption, that we refer to as *Vectorized Algebraic Somewhat Homomorphic Encryption* (VASHE), which combines the Brakerski Gentry Vaikuntanathan (BGV) scheme [9] with plaintext packing techniques. We then obtain our main result by coupling this scheme with a new variant of the pre-processing algorithm introduced by Kedlaya and Umans [18, 19]. Interestingly, the online operations of the

server are entirely *combinatorial* in nature, whereas all algebraic operations are delegated to the client. Next, we give a more detailed overview of our techniques.

1.2 Technical Overview

The starting point of our construciton is the DEPIR scheme from [22], so we begin by recalling the high-level idea of their approch. Their scheme proceeds in three steps:

- *Pre-processing:* First, the server encodes the database DB into a m-variate polynomial f_{DB} of individual degree $< d$ over $\mathbb{Z}_d$ via multi-variate polynomial interpolation such that

$$f_{DB}(i_1, \ldots, i_m) = DB[i],$$

  where $(i_1, \ldots, i_m)$ is the base-d decomposition of i. Then it pre-processes the evaluation of this polynomial using the techniques introduced by Kedlaya and Umans [18, 19].

- *Query:* The client formulate its query by encrypting the base-d m-variate representation of its index under an *algebraic* homomorphic encryption scheme. The crucial property of this scheme is that the homomorphic evaluation of f_{DB} is itself an evaluation of a multi-variate polynomial (possibly over a different ring).

- *Response:* The server responds by simply evaluating f_{DB} on the point specified by the ciphertext sent by the client. Leveraging the result from [18,19], this step can be done in online time poly-logarithmic in the size of the database.

By a careful parameter selection, one can ensure that the server computational overhead is bounded by $poly(\lambda)$, when compared to the insecure scheme (where the clients simply send their indices in the plain). When considering n parallel queries, the trivial solution would be to run the DEPIR protocol multiple times in parallel. However, its amortized computation/communication of the server still remains at $poly(\lambda)$. A naive attempt to improve this, would be to couple DEPIR with a batching technique for homomorphic encryption, called *Single Instruction Multiple Data* (SIMD) [29] , which is able to reduce the amortized server response time by a factor n, by setting n to be also the number of slots in the SIMD encoding. Unfortunately, this approach falls short at achieving our goal: The response time of the server in DEPIR depends *polynomially* (as opposed to linearly) in the number of parallel queries n, thereby nullifying the savings of the SIMD technique.

To overcome this challenge, we proceed in two steps. First, we design a variant of the [18, 19] polynomial pre-processing algorithm, where the polynomial evaluation is split into three subroutines: CRT-decomposition, fast evaluation, and CRT-reconstruction. Then the decomposition and reconstruction subroutines are delegated to the client whereas the server only performes the fast evaluation. Interestingly, this is an entirely combinatorial subroutine, where the server simply fetches some values from the pre-processed database. Then, we combine it with a carefully crafted *vectorized* algebraic somewhat homomorphic encryption (VASHE) scheme that leverages the SIMD encoding to achieve an almost-optimal complexity of evaluation. We provide more details in the remaining part of the technical overview. Next, we review the results of [18, 19] on polynomial preprocessing for fast evaluation. Then, we discuss the construction of VASHE from Ring-LWE. Finally, we explain the construction of doubly efficient batched PIR using VASHE.

1.2.1 Fast Polynomials Evaluation with Preprocessing

Consider the problem of preprocessing multi-variate polynomials $f(X_1, \ldots, X_m)$ over a ring

$$R = \mathbb{Z}_q[Y, Z]/(E_1(Y), E_2(Z))$$

for monic polynomials E_1, E_2. The output of this procedure is a data structure T that enables highly efficient evaluation of the polynomial on any given input $(\alpha_1, \ldots, \alpha_m) \in R^m$, achieving an online time complexity that is sublinear in the description length of the polynomial. Let $r = |R|$ be the cardinality of the ring, and let e_1 and e_2 denote the degrees of E_1 and E_2, respectively. Assume that $f(X_1, \ldots, X_m)$ has an individual degree of less than d in each variable. Describing such a polynomial requires $N = d^m$ coefficients, so evaluating it naively would require at least $\Omega(N \log r)$ time. Kedlaya and Umans [19] demonstrate how to preprocess this polynomial in

$$S \coloneqq N \cdot O(m(\log m + \log d + \log\log r))^m \cdot \mathit{poly}(d, m, \log r)$$

time, resulting in a data structure T with size at most S. With this data structure, for any input $(\alpha_1, \cdots, \alpha_m) \in R^m$, one can evaluate $f(\alpha_1, \cdots, \alpha_m)$ in just $\mathit{poly}(d, m, e_1, e_2, \log q)$ time. Unfortunately, this is *insufficient* for us. In particular, constructing a doubly efficient batched PIR requires us to improve the dependency on e_1 and e_2 to be *quasi-linear*, as opposed to an arbitrary polynomial (see Section 5 for a more detailed analysis).

In a nutshell, we achieve this by splitting the polynomial evaluation into three separate algorithms: Decomp, Eval, and Reconst, where the running time of our Eval will have the desired efficiency. Next, we provide a more detailed sketch of our approach. Initially, we focus on the special case where $R = \mathbb{Z}_q$, and we will briefly discuss how to address the general case later in this outline. We choose M to be $d^m q^{m(d-1)+1}$, and let $p_1, ..., p_h$ be the distinct primes less than $16 \log M$ for some $h \in \mathbb{N}$. This choice ensures that

$$\prod_{i=1}^{h} p_i \geq M.$$

We reinterpret the polynomial $f \in \mathbb{Z}_q[X_1, ..., X_m]$ of individual degree less than d, as a polynomial over the integers $\mathbb{Z}$, where the coefficients and inputs are taken from the set $\{0, \ldots, q-1\}$. To evaluate the polynomial f over the integers, it suffices to evaluate it modulo each of the primes p_i separately and then reconstruct the answer over the integers using the Chinese Remainder Theorem (CRT). For each $i \in [h]$, we apply the fast Fourier transform (FFT) on $\mathbb{Z}_{p_i}^m$ to evaluate $f \pmod{p_i}$ on all points of $\mathbb{Z}_{p_i}^m$ and store the evaluations in a table T. In this way, we preprocess f into a data structure T.

The Decomp algorithm first lifts the input $\alpha \in \mathbb{Z}_q^m$ to $\overline{\alpha} \in \mathbb{Z}^m$ and then outputs the query vector

$$\boldsymbol{v}_\alpha = \{\overline{\alpha}_i \coloneqq \overline{\alpha} \pmod{p_i} | i \in [h]\},$$

which represents the CRT components of $\overline{\alpha}$ with respect to primes $p_1, \ldots, p_h$. Given the query vector $\boldsymbol{v}_\alpha = \{v_i\}_{i \in [h]}$, the Eval algorithm obtains $\overline{f}_i(v_i) \in \mathbb{Z}_{p_i}$ by performing a table lookup in T. The algorithm then outputs

$$\boldsymbol{v}_{f(\alpha)} \coloneqq \{\overline{f}_i(v_i) \mid i \in [h]\}.$$

Given an encoding vector $\boldsymbol{v}_\beta = \{v_{\beta_i}\}_{i \in [h]}$, the Reconst algorithm reconstructs the smallest $\beta \in \mathbb{Z}$ such that $\beta \equiv v_{\beta_i} \pmod{p_i}$ for all $i \in [h]$ and outputs β modulo q. To further decrease the runtime

of preprocessing and the size of the data structure T, we apply the aforementioned approach recursively to transform the problem of evaluating the polynomial f modulo each prime p_i into evaluating it modulo a different set of even smaller primes p'_j.

Finally, the result extends to rings $R = \mathbb{Z}_q[Y]/(E(Y))$ by reducing the problem over such rings to that over $\mathbb{Z}_r$, where r is much greater than q and depends on $|R|$. Instead of evaluating $f(\alpha_1, \ldots, \alpha_m)$ over R, we evaluate it over $\mathbb{Z}_r$ by substituting $Y = M$ for some sufficiently large integer $M \in \mathbb{Z}$ and performing the computation modulo r. This ensures there is no wrap-around. The output is an integer with base_M digits that correspond to the coefficients of Y in the correct evaluation of $f(\alpha_1, \ldots, \alpha_m)$ over R. We can extend the result to $R = \mathbb{Z}_q[Y, Z]/(E_1(Y), E_2(Z))$ analogously.

1.2.2 Constructing VASHE

We now turn to the problem of constructing VASHE, which we obtain by combining the BGV scheme [9, 22] with the SIMD encoding. Concretely, we take the most basic symmetric-key BGV of [22] and slightly modify it to handle a plaintext space $\mathbb{Z}_p[Z]/(Z^n + 1)$ for some prime p with $p \equiv 1 \pmod{2n}$, which can be decomposed into n copies of $\mathbb{Z}_p$ via the *Chinese Remainder Theorem* (CRT). SIMD, a CRT-based batching technique, allows us to perform multiplications and additions over $\mathbb{Z}_p[Z]/(Z^n + 1)$ as operations over n slots. We denote the rings

$$Q \coloneqq \mathbb{Z}_q[Z]/(Z^n + 1) \text{ and } R \coloneqq Q[Y]/(Y^D + 1) \cong \mathbb{Z}_q[Y, Z]/(Z^n + 1, Y^D + 1),$$

where $q \gg p$ is a relatively prime number to p. We describe the scheme as a symmetric encryption, which is anyway sufficient for our goal.

- The secret key is a random ring element $s \leftarrow Q$.
- Use SIMD encoding to encode n messages $m_1, \cdots, m_n \in \mathbb{Z}_p^n$ to a plaintext $\mu \in \mathbb{Z}_p[Z]/(Z^n+1)$. The decoding is just the inverse operation.
- To encrypt $\mu \in \mathbb{Z}_p[Z]/(Z^n + 1)$, choose a random $a \leftarrow Q$, sample e from an error distribution χ and output
$$b - a \cdot Y \in R,$$
where $b = a \cdot s + p \cdot e + \mu$.
- To decrypt $ct(Y) \in R$, evaluate it on the secret key s and output $ct(s)$ modulo p.

In the VASHE scheme, the ciphertexts are polynomials of a formal variable Y instead of tuples of elements in Q. By adding and multiplying such ciphertext polynomials, we can add and multiply the corresponding encrypted messages component-wise. The degree of the ciphertext as a polynomial over $Q[Y]$ and the magnitude of the error grow with the number of multiplications.

To handle the error growth of homomorphically evaluating polynomials of total degree $< D$, we will set $q = 2^{poly(D, \log p)}$. For polynomials of degree $< D$, modding out by (Y^D+1) does not have any affect on the homomorphic polynomial evaluation. In detail, for any polynomial $f(X_1, \ldots, X_m)$ over $\mathbb{Z}_p$ of total degree $< D$, we first lift $f \in \mathbb{Z}_p[X_1, \ldots, X_m]$ to $\overline{f} \in R[X_1, \ldots, X_m]$. Given ciphertexts $ct_1, \ldots, ct_m \in R$ encrypting plaintext $\mu_1, \ldots, \mu_m \in \mathbb{Z}_p$, respectively, the homomorphic polynomial evaluation EvalP simply evaluates $\overline{f}$ over the ciphertexts, resulting in $ct^* = \overline{f}(ct_1, \ldots, ct_m)$ such that $ct^* \in R$ is an encryption of $f(\mu_1, \cdots, \mu_m)$.

The encryption, decryption, encode and decode time and the bit-size of ring elements can be bound by $poly(\lambda, p, D)$, growing polynomially with the security parameter λ, the size of the plaintext space $\mathbb{Z}_p$, and the total degree D. However, the runtime of EvalP is $O(d^m) \cdot poly(\lambda, d, m)$, which is not efficient enough for the construction of DEBPIR. Fortunately, we can leverage the preprocessing algorithm as outlined above, to derive a more efficient alternative, that we refer to as EvalFP, where the computation complexity is bounded by $\tilde{O}_\lambda(n)$, as desired.

1.2.3 Putting Things Together

We conclude this overview by presenting a sketch of our DEBPIR protocol, combining the tools that we built to far. In a preprocessing phase, the server encodes the database DB into a m-variate polynomial f_{DB} of individual degree $< d$ over $\mathbb{F}_p$ such that $f_{DB}(i_1, \ldots, i_m) = DB[i]$, where $(i_1, \ldots, i_m)$ is the base-d decomposition of i and p is the plaintext modulus of the VASHE. In the basic batched PIR scheme, the query of the client consists of n indices $i^{(1)}, \ldots, i^{(n)} \in [\![N]\!]$ written in base-d as $i^{(j)} = (i_1^{(j)}, \ldots, i_m^{(j)})$ for $j \in [n]$. The digits $(i_k^{(1)}, \ldots, i_k^{(n)})$ can be encoded into a plaintext and then encrypted into a single ciphertext ct_k for $k \in [m]$. To give a PIR response, the server computes

$$ct^* \leftarrow \text{VASHE.EvalP}(\{ct_1, \ldots, ct_m\}, f_{DB}).$$

The client decrypts ct^* and then decodes the plaintext to message vector $\boldsymbol{b} \in \{0,1\}^n$, where $\boldsymbol{b}[j] = f_{DB}(i_1^{(j)}, \ldots, i_m^{(j)}) = DB[i^{(j)}]$ since VASHE satisfies the SIMD property.

Preprocessing the polynomial f_{DB} into a data structure $\tilde{DB}$ allows us to replace VASHE.EvalP with VASHE.EvalFP, i.e. fast polynomial evaluation with preprocessing [19], which upgrades the base scheme into DEBPIR. To be more specific, the client breaks $ct_1, \ldots, ct_m$ down into CRT components[1] instead of sending the query ciphertexts to the server directly, the server obtains the CRT encoding $\boldsymbol{v}_{ct^*}$ of ct^* by looking up the preprocessed data structure $\tilde{DB}$ in the online response phase and the client performs the CRT reconstruction to recover ct^* from its CRT form before the decryption.

A careful balancing of the parameters allows us to take advantage of the SIMD encoding, to pack multiple plaintext query indices into a single encrypted query vector. Overall, we obtain an amortized server complexity of $poly(\log \lambda) = \tilde{O}_\lambda(1)$, which is an exponential improvement over prior work.

1.3 Related Works

We compare DEBPIR with some previous PIR protocols.

Multi-server PIR. A series of works [5, 14, 27] considered multi-server PIR protocols where the database is replicated across multiple servers. In general settings, multi-server constructions have reduced computational overhead and can often achieve information-theoretic security. However, the drawback is their reliance on having multiple non-colluding servers; this assumption can be challenging to realize in practice. Conversely, single-server protocol DEBPIR assumes a client-server architecture, provides computational security guarantees and do not assume non-colluding servers.

[1]Technically, we first turn $ct_i \in R$ into an element of $\mathbb{Z}_r$ for some large r and then break it down into CRT components.

DEPIR. The seminal work of Lin, Mook and Wichs [22] constructed DEPIR under the standard Ring-LWE [23] assumption with quasi-polynomial approximation factors. For any constant $\epsilon > 0$, they get a scheme where, for a database $DB \in \{0,1\}^N$, the preprocessing run-time and the size of the preprocessed data structure $\tilde{DB}$ are $O(N^{1+\epsilon})$, while the client and server run-times as well as the communication-complexity of each PIR query are just $poly(\lambda, \log N)$. Compared with DEPIR [22], our DEBPIR allows the client query several indices of the dababase in one batch and is able to achieve $poly(\log \lambda, \log N)$ amortized server online response time and amortized communication cost.

Batched PIR. A series of works [16, 25] considered batched PIR protocols where the client wants to download multiple entries from the server in one batch. However, the communication and computation cost of their schemes are still $poly(\sqrt{N})$, while the amortized server online response time and amortized communication cost of our DEBPIR scheme are $poly(\log \lambda, \log N)$, which is an exponential improvement compared with previous batched PIR protocols.

2 Preliminaries

We use the following notations for sets of numbers: $\mathbb{N}$ for positive integers, $\mathbb{Z}$ for integers, and $\mathbb{R}$ for real numbers. For an integer $n \in \mathbb{N}$, we write $[n] = \{1, \ldots, n\}$, and $[\![n]\!] = \{0, \ldots, n-1\}$. We treat numbers such as $n^{1/2}$ and n/k as integers, ignoring issues of integrality, since we can always round numbers with negligible asymptotic loss. We use $poly(\cdot)$ to denote a fixed polynomial in its argument. For a finite set X, the notation $x \leftarrow X$ denotes an independent and uniform random draw from X. Unless specifically stated otherwise, all logarithms mentioned are in base 2, and $\log n$ denotes $\log_2 n$. We define the norm of $a = \sum_{i=0}^{n-1} a_i Z^i \in \mathbb{Z}_q[Z]$ as $\|a\|_\infty \coloneqq \max\{|a_i|\}$, where we identify $a_i \in \mathbb{Z}_q$ with its integer representative in the range $(-q/2, q/2]$.

We index an array $A \in \{0,1\}^n$ starting from 0, such that $A[i]$ denotes the i-th bit in the array. For any $q \in \mathbb{N}$, we let $\mathbb{Z}_q$ denote the ring $\mathbb{Z}/q\mathbb{Z}$, and for a prime p, we let $\mathbb{F}_p$ denote the finite field of order p. A function $v : \mathbb{N} \to \mathbb{N}$ is called negligible, denoted as $v(n) = \text{negl}(n)$, if for every positive polynomial $p(\cdot)$ and sufficiently large n, it holds that $v(n) < 1/p(n)$. We use the abbreviation PPT for probabilistic polynomial time, and denote the security parameter as λ. For two distributions X and Y that are parameterized by λ, we say that they are computationally indistinguishable, denoted by $X \approx_c Y$, if for every PPT distinguisher D, we have

$$|\Pr[D(X) = 1] - \Pr[D(Y) = 1]| = \text{negl}(\lambda).$$

2.1 Ring-LWE

The security of many lattice-based cryptographic schemes is based on the hardness of the decisional *Ring Learning With Errors* (Ring-LWE) problem, which has become a standard building block in cryptography. The Ring-LWE problem is a mathematical problem proposed by Lyubashevsky et al. [23], which is an algebraic variant of the *Learning With Errors* (LWE) problem [28].

Let $Q \coloneqq \mathbb{Z}_q[Z]/(Z^n + 1)$ for some integer q. A distribution χ over Q is said to be β-bounded if $Pr\left[\|e\|_\infty \leq \beta : e \leftarrow \chi\right] = 1$. Ring-LWE essentially states that given a secret $s \leftarrow Q$, the following two distributions are computationally indistinguishable: the distribution of pairs $(a, b \coloneqq as + e) \in Q^2$, where $a \leftarrow Q$ and $e \leftarrow \chi$, and the distribution of uniformly sampled pairs $(a, b) \leftarrow Q^2$. We now formally define the computational problems that we discussed above.

Definition 2.1 (Ring-LWE Distributions)**.** For $s \in Q$, a sample from the Ring-LWE distribution $A_{n,q,\chi}$ over Q^2 is obtained by uniformly sampling an $a \in Q$, sampling $e \in Q$ from χ, and outputting the pair $(a, as + e)$.

Definition 2.2 (Decisional Ring-LWE$_{n,q,\chi}$ Problem)**.** Given access to polynomially many samples from Q^2, the decisional Ring-LWE$_{n,q,\chi}$ problem is to distinguish between the distributions $A_{n,q,\chi}$ and the uniform distribution of Q^2.

There is a quantum reduction from the *approximate shortest vector problem* (SVP) on ideal lattices to the Ring-LWE problem [23]. Informally, this means that the decisional Ring-LWE problem is hard, assuming that worst-case problems on ideal lattices are hard for quantum computers. To be more specific, the decisional Ring-LWE problem with some β-bounded error distribution χ is implied by the worst-case hardness of the approximate shortest vector problem in an ideal lattice with approximation factor $\approx q/\beta$.

Theorem 2.3 ([23])**.** For any n that is a power of two, ring $Q = \mathbb{Z}[Z]/(Z^n + 1)$, prime integer $q \equiv 1 \pmod{n}$, and $\beta = \omega(\sqrt{n \log n})$, there is an efficiently samplable β-bounded distribution χ over Q, such that there is a quantum reduction from the $n^{\omega(1)} \cdot (q/\beta)$-approximate worst-case SVP in ideal lattices over Q to Ring-LWE$_{n,q,\chi}$, where the reduction runs in time $poly(n, q)$.

We let Ring-LWE assumption with quasi-polynomial approximation factors refer to the assumption that, given any security parameter λ and any gap parameter t, we can, in $poly(\lambda, t)$ deterministic time, find parameters: $n = poly(\lambda, t)$, $\beta = poly(\lambda, t)$, a β-bounded distribution χ that is efficiently samplable in $poly(\lambda, t)$ time, and $q = \lambda^{poly(t)}$ with $q > (2\beta n)^t$, such that for any $t(\lambda) = poly(\log \lambda)$ the corresponding Ring-LWE$_{n,q,\chi}$ assumption holds. The Ring-LWE assumption with quasi-polynomial approximation factors is implied by the worstcase $2^{poly(\log n)}$ quantum hardness of the approximate SVP in an ideal lattice with approximation factors $2^{poly(\log n)}$.

The Ring-LWE$_{n,q,\chi}$ assumption is known to be equivalent to a *Hermite normal form* variant where we sample $s \leftarrow \chi$ from the error distribution rather than uniformly at random from the ring [4, 23]. It is also equivalent to a scaled error variant where instead of adding the errors $e_i \leftarrow \chi$ we add $p \cdot e_i$ for some integer p relatively prime to q [10].

2.2 Single Instruction Multiple Data (SIMD)

In [29], Smart and Vercauteren introduced a packing technique called *Single Instruction Multiple Data* (SIMD) for *homomorphic encryption* (HE) schemes. The technique decomposes the plaintext space using the *Chinese Remainder Theorem* (CRT), which allows multiple values to be encrypted simultaneously into one ciphertext and enables arithmetic operations to be performed homomorphically. Specifically, by applying a single instruction to a ciphertext, we can perform computations on large amounts of data simultaneously. Since HE schemes typically embed relatively small data within large ciphertexts, allowing each ciphertext to represent multiple independent data enables a more efficient use of both space and computational resources.

We denote $R_p \coloneqq \mathbb{Z}_p[Z]/(\Phi_m(Z))$, where $\Phi_m(Z)$ is the reduction modulo p of the m-th cyclotomic polynomial. They proposed a CRT-based packing method that decomposes the ring into smaller components, enabling parallel execution of multiple computations. If p and m are co-prime, then the m-th cyclotomic polynomial $\Phi_m(Z)$ can be decomposed into irreducible coprime factors modulo

p

$$\Phi_m(Z) \equiv \prod_{i=1}^{l} f_i(Z) \pmod{p},$$

where $\deg_Z(f_i) \coloneqq d = \phi(m)/l$ for all $i \in [l]$. Thus,

$$R_p \cong \mathbb{Z}_p[Z]/(f_1(Z)) \times \mathbb{Z}_p[Z]/(f_2(Z)) \times \cdots \times \mathbb{Z}_p[Z]/(f_l(Z)). \tag{2.1}$$

To construct a doubly efficient batched PIR, our focus is on the case where the m-th cyclotomic polynomial $\Phi_m(Z)$ splits completely modulo p. To begin with, we will consider Lemma 2.4, which provides the necessary condition for $\Phi_m(Z)$ to completely split modulo p.

Lemma 2.4. If $\gcd(m, p) = 1$ and p is prime, then the following are equivalent:

1. $p \equiv 1 \pmod{m}$.
2. $\Phi_m(Z)$ splits completely in $\mathbb{F}_p$.
3. $\Phi_m(Z)$ has a root in $\mathbb{F}_p$.

Furthermore when these conditions are satisfied, the roots of $\Phi_m(Z)$ in $\mathbb{F}_p$ consist of the primitive m-th roots of unity.

Therefore, we choose a prime number p and set $n = 2^r$ in a way that ensures $p \equiv 1 \pmod{2n}$. According to Lemma 2.4, we obtain

$$\Phi_{2n}(Z) = Z^n + 1 \equiv \prod_{i=1}^{n} f_i(Z) \pmod{p},$$

where $f_i(Z)$ is a linear polynomial for $i \in [n]$. Therefore, it follows from the CRT isomorphism (2.1) that $R_p \cong \mathbb{F}_p^n$. In this way, we can pack n elements into these n slots, and multiplications and additions over R_p correspond to component-wise operations over the n slots. We now describe the encoding and decoding algorithms in the SIMD scheme.

- SIMD. Encode: for $\mathbf{msg} = (m_1, m_2, \ldots, m_n) \in \mathbb{F}_p^n$, encode $\mathbf{msg}$ into $\mu \in R_p$ with the inverse of the CRT isomorphism (2.1).
- SIMD. Decode: for $\mu \in R_p$, decode μ into $\mathbf{msg} \in \mathbb{F}_p^n$ with the CRT isomorphism (2.1).

3 Multi-variate Polynomial Evaluation and Interpolation

3.1 Fast Polynomial Evaluation with Preprocessing

Our work, following [22], builds upon a technique developed by Kedlaya and Umans [19]. Their method involves preprocessing a multivariate polynomial f into a static data structure T that allows for efficient evaluation of $f(\alpha)$ for any input α provided later. The evaluation time is sublinear with respect to the length of the polynomial's description. In this work, we will rely on this result for multivariate polynomials f over rings of the form $R \coloneqq \mathbb{Z}_q[Y, Z]/\langle E_1(Y), E_2(Z)\rangle$, where $q \in \mathbb{N}$ and $E_1(Y)$ and $E_2(Z)$ are arbitrary non-constant monic polynomials of degree e_1 and e_2, respectively. Note that this result includes the rings $\mathbb{Z}_q$ and $\mathbb{Z}_q[Y]/(E(Y))$ as a special case.

We now describe the syntax and the efficiency requirement for a *fast polynomial evaluation with preprocessing* (FPE) scheme. An aspect that is new to our work, is the fact that we split the evaluation of the polynomial in three subroutines (Decomp, Eval, and Reconst), and we impose stronger efficiency requirements for Eval.

Definition 3.1. The fast polynomial evaluation with preprocessing (FPE) scheme is a tuple of deterministic algorithms (Setup, Prep, Decomp, Eval, Reconst) with the following syntax:

- $\text{aux} \leftarrow \text{Setup}(R)$: Output some public auxiliary information aux.
- $T \leftarrow \text{Prep}(f, \text{aux}; R)$: Given auxiliary information aux and a polynomial $f \in R[X_1, ..., X_m]$ with $deg_{X_i}(f) < d$, outputs a data structure T.
- $\boldsymbol{v}_\alpha \leftarrow \text{Decomp}(\alpha, \text{aux}; R)$: Given an evaluation point $\alpha \in R^m$, outputs an encoding vector $\boldsymbol{v}_\alpha$.
- $\boldsymbol{v}_{f(\alpha)} \leftarrow \text{Eval}(\boldsymbol{v}_\alpha, T, \text{aux}; R)$: Given random access to T and an encoding vector $\boldsymbol{v}_\alpha$, outputs an encoding vector $\boldsymbol{v}_{f(\alpha)}$ of $f(\alpha)$.
- $\beta \leftarrow \text{Reconst}(\boldsymbol{v}, \text{aux}; R)$:Given an encoding vector $\boldsymbol{v}$ as the input, outputs $\beta \in R$.

The algorithms $(\text{Setup}, \text{Prep}, \text{Decomp}, \text{Eval}, \text{Reconst})$ should satisfy the following properties:

- **Correctness:** Given a fixed polynomial f,

  $$\text{Reconst}(\text{Eval}(\text{Decomp}(\alpha, \text{aux}; R), T, \text{aux}; R), \text{aux}; R) = f(\alpha)$$

  for all $\alpha \in R^m$ and $T \leftarrow \text{Prep}(f, R)$.
- **Efficiency:** We require that the run-time of Decomp and Reconst are both bounded by $poly(m, d, e_1, e_2, \log q)$, the run-time of Eval is bounded by $\tilde{O}(e_1 e_2) \cdot poly(m, d, \log q)$, and the output size of Decomp, Eval are bounded by $\tilde{O}(e_1 e_2) \cdot poly(m, d, \log q)$. Additionally, we require that the run-time and space complexity of Prep is bounded by

  $$d^m \cdot poly(m, d, \log |R|) \cdot O(m(\log m + \log d + \log\log |R|))^m.$$

The FPE algorithm of [19], and used in [22], is not directly applicable for our construction of doubly efficient batched PIR, as we require the runtime of evaluation algorithm to be quasi-linear in e_1 and e_2. Therefore, we modify the FPE algorithms in [19] and [22] by breaking their evaluation algorithm into three separate algorithms: Decomp, Eval, and Reconst. This allows us to delegate some of the server's work to the client, which ultimately reduces the server's run time and communication cost and resulted in a more efficient scheme. Before giving a precise statement of our results we recall a Lemma from [19].

Lemma 3.2 (Product of small primes [19])**.** For all integers $M \geq 2$, the product of the primes less than or equal to $16 \log M$ is greater than M. That is,

$$M < \prod_{p \leq 16 \log M,\ p \in \mathsf{PRIME}} p.$$

We now summarize our results about the FPE scheme into the following theorem.

Theorem 3.3. Let the ring $R := \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle$ for two monic polynomials $E_1(Y)$ and $E_2(Z)$ with the degree e_1 and e_2 respectively. Let $f(X_1,\ldots,X_m) \in R[X_1,\ldots,X_m]$ be a fixed polynomial with $deg_{X_i}(f) < d$. Then, there exists an FPE scheme satisfies the following properties. The preprocessing algorithm Prep runs in time

$$S = d^m \cdot poly(m, d, \log|R|) \cdot O(m(\log m + \log d + \log\log|R|))^m$$

and outputs a data structure T of size at most of S. The running time of Decomp is bounded by

$$O(e_1e_2(\log q + \log e_1 + \log e_2)^2 \cdot poly(d, m, \log\log q)).$$

The running time of Eval is bounded by

$$O(e_1e_2(\log q + \log e_1 + \log e_2))poly(m, d).$$

The running time of Reconst is bounded by

$$poly(m, d, e_1, e_2, \log q).$$

Furthermore, the output size of Setup is bounded by $O(e_1^2e_2^2 \cdot (\log q + \log e_1 + \log e_2)) \cdot poly(m, d)$, and the output size of Decomp and Eval are bounded by

$$O(e_1e_2(\log q + \log e_1 + \log e_2)) \cdot poly(m, d).$$

In the following we only provide a sketch of the proof of Theorem 3.3, but we refer the reader to Appendix A for a complete argument. In Table 1 and 2 we provide a complete overview of the time and space complexity of our algorithms (for several types of rings $R = \mathbb{Z}_q$, $R = \mathbb{Z}_q[Y]/(E(Y))$ and $R = \mathbb{Z}_q[Y,Z]/(E_1(Y), E_2(Z))$).

Proof. Let us first consider the special case $R = \mathbb{Z}_q$, and we will show how to handle the general case towards the end of this outline. We now give a high-level overview of the algorithms, along with their complexities.

- In the Setup algorithm, we choose M to be $d^m q^{m(d-1)+1}$, and let $p_1, ..., p_h$ be the distinct primes less than $16 \log M$ for some $h \in \mathbb{N}$. We then output the auxiliary information $aux = (M, \{p_1, ..., p_h\})$. The size of aux is bounded by

$$\log M + \sum \log p_i \leq O(d \cdot m \cdot \log q).$$

- In the Prep algorithm, we preprocess the coefficients of a polynomial f into a data structure T. We then lift $f \in \mathbb{Z}_q[X_1,\ldots,X_m]$ to $\overline{f} \in \mathbb{Z}[X_1,\ldots,X_m]$ by lifting the corresponding coefficients of f from $\mathbb{Z}_q$ to $\mathbb{Z}$. For each $i \in [h]$, we take every coefficient of $\overline{f}$ modulo p_i and get $\overline{f_i} \in \mathbb{Z}_{p_i}[X_1,\ldots,X_m]$ such that $\overline{f_i} = \overline{f}$ modulo p_i. Moreover, we use the *fast Fourier transform* (FFT) on $\mathbb{Z}_{p_i}^m$ to evaluate $\overline{f_i}(a)$ on all points $a \in \mathbb{Z}_{p_i}^m$ and store the evaluations in a table

$$T_i \coloneqq \{\overline{f_i}(a) : a \in \mathbb{Z}_{p_i}^m\}.$$

  The data structure T consists of $p_1, p_2, \ldots, p_h, T_1, T_2, \ldots, T_h$. This algorithm is unchanged from [19, 22], and therefore the running time and output size of Prep are bounded by

$$O(md\log q)^m \cdot poly(m, d, \log q).$$

- Given any point $\alpha \in \mathbb{Z}_q^m$, the Decomp algorithm lifts α to $\overline{\alpha} \in \mathbb{Z}^m$ and outputs the CRT form of $\overline{\alpha}$ with respect to primes $p_1, \ldots, p_h$, i.e. we output the encoding vector $\boldsymbol{v}_\alpha = \{\overline{\alpha}_i \coloneqq \overline{\alpha} \pmod{p_i} | i \in [h]\}$. The time complexity of Decomp is bounded by

$$\sum_{i=1}^{h} m \cdot O(\log p_i \log q) \leq O((\log q)^2) \cdot poly(m, d).$$

  The size of output is bounded by

$$m \cdot \sum_i \log p_i \leq O(\log q) \cdot poly(m, d).$$

- Given the encoding vector $\boldsymbol{v}_\alpha = \{v_i\}_{i \in [h]}$, the Eval algorithm obtains $\overline{f}_i(v_i) \in \mathbb{Z}_{p_i}$ by looking it up in table T_i and outputs $\boldsymbol{v}_{f(\alpha)} \coloneqq \{\overline{f}_i(v_i) \mid i \in [h]\}$. The time complexity of Eval is bounded by

$$O(h) + m \cdot \sum_i \log p_i \leq O(\log q) \cdot poly(m, d).$$

  The size of the output of Eval is bounded by

$$\sum_i \log p_i \leq O(\log q) \cdot poly(m, d).$$

- Given an encoding vector $\boldsymbol{v} = \{v_i\}$, the Reconst algorithm uses CRT reconstruction to find the smallest $\beta \in \mathbb{Z}$ such that $\beta \equiv v_i \pmod{p_i}$ for all $i \in [h]$ and outputs β modulo q. The running time of Reconst is bounded by $poly(m, d, \log q)$.

The correctness of the above algorithm is seen as follows. Because the coefficients of the lifted polynomial $\overline{f}$ and the components of the lifted point $\overline{\alpha}$ are in $\{0, \ldots, q-1\}$, and $\overline{f}$ has individual degree $< d$, we can bound the lifted evaluation over $\mathbb{Z}$ by $0 \leq \overline{f}(\overline{\alpha}) < M$. By Lemma 3.2, we then have

$$\overline{f}(\overline{\alpha}) < M < \prod_{i \in [h]} p_i.$$

Note that $\overline{f}(\overline{\alpha}) \equiv \overline{f_i}(\overline{\alpha_i}) \pmod{p_i}$ for all $i \in [h]$. Also, by CRT, there is a unique solution for

$$z < \prod_{i \in [h]} p_i \text{ s.t. } z \equiv \overline{f_i}(\overline{\alpha_i}) \pmod{p_i}, \forall\ i \in [h].$$

Recalling that $\overline{f_i}(\overline{\alpha_i})$ are correctly computed by the FFT evaluation, we have $z = \overline{f}(\overline{\alpha})$ by the uniqueness. Correctness follows since $z = \overline{f}(\overline{\alpha}) \mod q = f(\alpha)$.

An important aspect that we have glossed over, is that the size of the data structure T is $O(md \log q)^m \cdot poly(m, d, \log q)$, which does not satisfy our efficiency requirements. To fix this, we can apply the above idea recursively to replace the $(\log q)^m$ factor in the preprocessing run-time and output size by a $(\log \log q)^m$ factor.

We then extend the result to rings $R = \mathbb{Z}_q[Y]/(E(Y))$, by reducing the problem over such rings to that over $\mathbb{Z}_r$ for some $r \gg q$ that depends on $|R|$. Instead of evaluating $f(\alpha_1, \ldots, \alpha_m)$ over R, we evaluate it over $\mathbb{Z}_r$ by taking all the coefficient/input ring elements and substituting $Y = M$ for

some sufficiently large integer $M \in \mathbb{Z}$ and doing all the computation modulo r for some sufficiently large $r \gg M$, such that there is no wrap-around. The output is an integer whose base_M digits correspond to the coefficients of Y in the correct evaluation of $f(\alpha_1, \ldots, \alpha_m)$ over R. We can further extend the result to $R = \mathbb{Z}_q[Y, Z]/(E_1(Y), E_2(Z))$ analogously. □

	$\mathbb{Z}_q$	$\mathbb{Z}_q[Y]/\langle E_1(Y)\rangle$	$\mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle$
Setup	$O(md \log q)$	$O(e^2(\log qe))poly(m,d)$	$O(e_1^2 e_2^2 \cdot (\log qe_1e_2)) \cdot poly(m,d)$
Prep	$d^m \cdot poly(m,d,\log q) \cdot O(m(\log d \log q))^m$	$d^m \cdot poly(m,d,\log \lvert R\rvert)O(m(\log(md \log \lvert R\rvert)))^m$	$d^m \cdot poly(m,d,\log \lvert R\rvert)O(m(\log(m \log \lvert R\rvert)))^m$
Decomp	$O(\log q)poly(m,d)$	$O(e(\log qe))poly(m,d)$	$O(e_1e_2 \log(qe_1e_2))poly(m,d)$
Eval	$O(\log q)poly(m,d)$	$O(e(\log qe))poly(m,d)$	$O(e_1e_2 \log(qe_1e_2))poly(m,d)$

Table 1: Space Complexity

	$\mathbb{Z}_q$	$\mathbb{Z}_q[Y]/\langle E_1(Y)\rangle$	$\mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle$
Prep	$d^m \cdot poly(m,d,\log q) \cdot O(m(\log md \log q)^m)$	$d^m \cdot poly(m,d,\log \lvert R\rvert)O(m(\log(md \log \lvert R\rvert)))^m$	$d^m \cdot poly(m,d,\log \lvert R\rvert)O(m(\log(m \log \lvert R\rvert)))^m$
Decomp	$O(\log^2 q)poly(m,d)$	$O(e^2(\log e)^2)poly(m,d)$	$O((e_1e_2 \log(qe_1e_2))^2)poly(m,d)$
Eval	$O(\log q)poly(m,d)$	$O(e(\log qe))poly(m,d)$	$O(e_1e_2 \log(qe_1e_2))poly(m,d)$
Reconst	$poly(m,d,\log q)$	$poly(m,d,e,\log q)$	$poly(m,d,e_1,e_2,\log q)$

Table 2: Time Complexity

3.2 Multivariate Polynomial Interpolation

Assume we are provided with a set of d^m values $y(x_1, ..., x_m) \in \mathbb{F}_p$ for all $x_i \in [\![d]\!]$. Under these conditions, there exists a distinct m-variables polynomial denoted as $f(X_1, ..., X_m) \in \mathbb{F}_p[X_1, ..., X_m]$, where the degree of each variable is strictly less than d, satisfying the equation $f(x_1, ..., x_m) = y(x_1, ..., x_m)$. This is possible due to the existence of a one-to-one correspondence between the d^m coefficients of such polynomials and the d^m evaluations obtained for all possible inputs. Furthermore, it is feasible to devise an algorithm that efficiently determines the coefficients of the polynomial $f(X_1, ..., X_m)$ based on the d^m evaluations in quasi-linear time.

A similar statement is proven in [22]. However, their result requires the degree d to equal the prime modulus p, which is too restrictive for us, as we need to choose p such that $p \equiv 1$ modulo a power of two. In the following we generalize their result and provide a simple proof. The algorithm is recursive and uses single-variate polynomial interpolation in the base case.

Lemma 3.4. (Univariate polynomial interpolation). Let $\mathbb{F}_p$ be a field of prime order p, then there is an interpolation algorithm such that takes as input any distinct $a_1, \ldots, a_n \in \mathbb{F}_p$, and any $b_1, \ldots, b_n \in \mathbb{F}_p$ for $n \leq p$, outputs the n coefficients of a univarite polynomial g of degree $< n$, such that $g(a_i) = b_i$ for all $i \in [n]$ in time $n \cdot poly(\log n, \log p)$.

Proof. See Corollary 10.12 of [30]. □

Lemma 3.5. (Multivariate polynomial interpolation). Let $\mathbb{F}_p$ be a field of prime order p, and let $m \in \mathbb{N}$ be an integer. Let $S = \{(x_1, ..., x_m) \mid x_i \in [\![d]\!]\} \subset \mathbb{F}_p^m$ and let $\{y_{(x_1,...,x_m)} \in \mathbb{F}_p\}_{(x_1,...,x_m)\in S}$ be any set of d^m values. Then there is an algorithm that runs in quasi-linear time

$$O(d^m \cdot m \cdot poly(\log p, \log d))$$

and recovers the coefficients of a polynomial $f(X_1, ..., X_m) \in \mathbb{F}_p[X_1, ..., X_m]$ with $deg_{X_i}(f) < d$ such that

$$f(x_1, ..., x_m) = y_{x_1,...,x_m}$$

for all $(x_1, ..., x_m) \in S$.

Proof. We use the following algorithm to interpolate f given values $\{y_{(x_1,...,x_m)} \in \mathbb{F}_p\}_{(x_1,...,x_m)\in S}$:

1. Base case: if $m = 1$, run the univariate polynomial interpolation (Lemma 3.4) to recover the coefficients of the univariate polynomial $f(X)$ of degree $< d$ such that $f(x_1) = y_{x_1}$ for all $x_1 \in [\![d]\!]$.
2. Otherwise, if $m > 1$:
   (a) For each $(x_1, ..., x_{m-1}) \in \mathbb{F}_p^{m-1}$ with $x_i \in [\![d]\!]$, run the univariate polynomial interpolation (Lemma 3.4) to recover the coefficients of a univariate polynomial $g_{x_1,...,x_{m-1}}(X) = \sum_{i=0}^{d-1} c_{x_1,...,x_{m-1},i} X^i$ of degree d such that
   $$g_{x_1,...,x_{m-1}}(x_m) = y_{(x_1,...,x_m)}$$
   for all $x_m \in [\![d]\!]$.
   (b) For each $i \in [\![d]\!]$, recursively call this algorithm with input
   $$\{c_{x_1,...,x_{m-1},i}\}_{(x_1,...,x_{m-1})\in\mathbb{F}_p^{m-1}, x_i\in[d]}$$
   to interpolate a polynomial $f_i(X_1, ..., X_{m-1})$ of individual degree $< d$ such that
   $$f_i(x_1, ..., x_{m-1}) = c_{x_1,...,x_{m-1},i}$$
   for all $(x_1, ..., x_{m-1}) \in \mathbb{F}_p^{m-1}$ with $x_i \in [\![d]\!]$.
   (c) Output f as
   $$f(X_1, ..., X_m) = \sum_{i=0}^{d-1} f_i(X_1, ..., X_{m-1}) X_m^i.$$

To see the efficiency, let $T(m)$ be the running time for m-variate polynomials. We have

$$T(m) = \begin{cases} d \cdot poly(\log p, \log d) & m = 1 \\ d^m \cdot poly(\log p, \log d) + d \cdot T(m-1) & m > 1 \end{cases}$$

Solving the recursion equation, we get $T(m) = d^m \cdot m \cdot poly(\log p, \log d)$ as claimed. □

4 Vectorized Algebraic Somewhat Homomorphic Encryption

In this section, we define and construct a modified homomorphic encryption scheme, called *Vectorized Algebraic Somewhat Homomorphic Encryption* (VASHE). Our construction is based on the BGV [9] scheme combined with the SIMD encoding, to achieve multiplication and addition operations with good amortized complexity. For convenience, in this work we focus on the simplified variant where (i) encryption requires a secret key, and (ii) we only homomorphically evaluate polynomials of some fixed degree (i.e., we only construct a symmetric-key somewhat homomorphic encryption). Both restrictions can be easily avoided with standard key-switching and bootstrapping techniques. However, in this work, we chose to concentrate on the simplified variant, since it suffices for our application.

4.1 Definition

In the following, we define the notion of Vectorized Algebraic Somewhat Homomorphic Encryption.

Definition 4.1 (VASHE)**.** A Vectorized Algebraic Somewhat Homomorphic Encryption (VASHE) is a symmetric-key CPA-secure encryption scheme consisting of a tuple of PPT algorithms (Setup, KeyGen, Encode, Decode, Encrypt, Decrypt, EvalP) with the following syntax:

- params $\leftarrow$ Setup$(1^\lambda, 1^d, 1^m)$: Given a security parameter λ, individual degree d, and number of variables m, the algorithm deterministically fixes certain public parameters params, which implicitly define the total degree $D = dm$, the plaintext space $\mathbb{Z}_p[Z]/(Z^n + 1)$ for a power-of-two n and prime p satisfying $p \equiv 1 \pmod{2n}$, as well as a ring R of the form $R = \mathbb{Z}_q[Y, Z]/(Y^D + 1, Z^n + 1)$ for some $q \in \mathbb{N}$. All other algorithms implicitly take params as input even when not explicitly stated.
- $sk \leftarrow$ KeyGen(params): Output a secret key sk.
- $\mu \leftarrow$ Encode$(m_1, \cdots, m_n)$: Given $m_i \in \mathbb{Z}_p$ for $i \in [n]$, deterministically outputs a plaintext $\mu \in \mathbb{Z}_p[Z]/(Z^n + 1)$.
- $(m_1, \cdots, m_n) \leftarrow$ Decode(μ): Given $\mu \in \mathbb{Z}_p[Z]/(Z^n + 1)$, deterministically outputs $m_i \in \mathbb{Z}_p$ for $i \in [n]$.
- $ct \leftarrow$ Encrypt(sk, μ): Given a secret key sk and a plaintext $\mu \in \mathbb{Z}_p[Z]/(Z^n + 1)$, outputs a ciphertext $ct \in R$.
- $\mu \leftarrow$ Decrypt(sk, ct): Given a secret key sk and a ciphertext $ct \in R$, deterministically outputs a plaintext $\mu \in \mathbb{Z}_p[Z]/(Z^n + 1)$.
- $ct_f \leftarrow$ EvalP$(\boldsymbol{ct}, f)$: Given a polynomial $f \in \mathbb{Z}_p[X_1, \cdots, X_m]$ with individual degree less than d and $\boldsymbol{ct} \in R^m$, deterministically outputs a ciphertext $ct_f \in R$.

We require that the scheme satisfies the following properties:

- **Correctness**: For all params in the support of Setup and all sk in the support of KeyGen(params):

  1. For all messages $\mathbf{msg} = (m_1, \ldots, m_n) \in \mathbb{Z}_p^n$,

  $$Pr\left[\text{Decode}(\text{Decrypt}(sk, ct)) = \mathbf{msg} : \begin{array}{rcl} \mu & \leftarrow & \text{Encode}(\mathbf{msg}) \\ ct & \leftarrow & \text{Encrypt}(sk, \mu) \end{array}\right] = 1.$$

  2. For all messages $\mathbf{msg_1}, \cdots, \mathbf{msg_m} \in \mathbb{Z}_p^n$ where $\mathbf{msg_j} = (m_{j1}, \ldots, m_{jn})$ for $j \in [m]$ and for any polynomial $f(X_1, \ldots, X_m)$ over $\mathbb{Z}_p$ with individual degree less than d. Furthermore, we denote $m'_i = f(m_{1i}, ..., m_{mi})$ for $i \in [n]$. Then it holds that

  $$Pr\left[\begin{array}{c} \text{Decrypt}(sk, ct_f) = f(\mu_1, ..., \mu_m) \wedge \\ \text{Decode}(f(\mu_1, ..., \mu_m)) = (m'_1, ..., m'_n) \end{array} : \begin{array}{rcl} \mu_j & \leftarrow & \text{Encode}(\mathbf{msg_j}) \\ ct_j & \leftarrow & \text{Encrypt}(sk, \mu_j) \\ ct_f & \leftarrow & EvalP(\{ct_j\}, f) \end{array}\right] = 1.$$

- **Security**: We require the standard symmetric-key IND-CPA security for the encryption scheme (KeyGen, Enc, Dec) when params $\leftarrow$ Setup$(1^\lambda, 1^d, 1^m)$ for any $p = poly(\lambda)$, $d = poly(\log \lambda)$, $m = poly(\log \lambda)$.

- **Efficiency**: We require that the description length of ring elements, the run-time of the ring operations, and the run-time of Setup, KeyGen, Encode, Decode, Encrypt, Decrypt are all bounded by $poly(\lambda, d, m)$. Furthermore, the runtime of EvalP is bounded by $O(d^m) \cdot poly(\lambda, d, m)$.[2]

4.2 Construction

Overview. The starting point of our construction is the leveled FHE from Ring-LWE of Brakerski, Gentry and Vaikuntanathan (BGV) [9,22]. We only make a slight modification to the BGV scheme: the plaintext space of the BGV scheme is $\mathbb{Z}_d$, but we choose a prime p such that $p \equiv 1 \pmod{2n}$ and set the plaintext space to be $\mathbb{Z}_p[Z]/(Z^n+1)$, in order to make it compatible with SIMD. In BGV, a ciphertext is of the form $ct = (c_0, c_1)$ where $c_0, c_1 \in Q \coloneqq \mathbb{Z}_q[Z]/(Z^n+1)$, whereas we introduce a formal varible Y to interpret $ct = c_0 + c_1 \cdot Y$ as a linear polynomial of Y, and general ciphertexts as polynomials in $Q[Y]$ of degree $< D$, i.e. elements of the ring $R \coloneqq Q[Y]/(Y^D+1) = \mathbb{Z}_q[Y,Z]/(Z^n+1, Y^D+1)$.

In BGV scheme, homomorphic multiplication consists of two steps, after the first step, the ciphertext ct'_{mult} is a degree-2 polynomial in Y. Then the second step key-switching/relinearization is required to convert ct'_{mult} to a degree-1 polynomial ct_{mult}, using some evaluation keys. However, the relinearization step is not an "algebraic" operation, which is crucial to do fast polynomial evaluation with preprocessing and benefit from the FPE algorithm from Theorem 3.3.

Fortunately for us, the relinearization algorithm is not necessary for the special task of EvalP, which involves homomorphic low-degree multi-variate polynomial evaluation. By additionally harnessing the efficiency of SIMD encoding, we can perform simultaneous computations on multiple plaintext data for each instruction applied to a single ciphertext, thereby enhancing both space and computational efficiency.

Our Construction. For the construction of VASHE, we will identify elements of $\mathbb{Z}_q$ with their representative in the range $(-q/2, ..., q/2] \cap \mathbb{Z}$, and similarly for $\mathbb{Z}_p$. This allows us to reinterpret an element $\mu \in \mathbb{Z}_p$ as an element of $\mathbb{Z}_q$ by taking the representative of μ and reducing it modulo q (and vice versa). Similarly, we can naturally reinterpret elements $\mu \in \mathbb{Z}_p$ as elements of $Q = \mathbb{Z}_q[Z]/(Z^n+1)$ or $R = \mathbb{Z}_q[Y,Z]/(Z^n+1, Y^D+1)$ by first reinterpreting μ as an element of $\mathbb{Z}_q$ and then interpreting it as a constant polynomial in Q or R respectively. We now describe the construction in detail.

- params $\leftarrow$ Setup$(1^\lambda, 1^d, 1^m)$: Denote $D = dm$, and $N = d^m$. We set the gap parameter $t \coloneqq D \log p + d \log m + \log p + 1$ and choose $n = poly(\lambda, t)$, $q = \lambda^{poly(t)}$ and a β-bounded error distribution χ so that $q > (2\beta n)^t > 2Nd(d(\beta+1)^n)^D$ and q is relatively prime to p. Define the rings

$$Q \coloneqq \mathbb{Z}_q[Z]/(Z^n+1), R \coloneqq Q[Y]/(Y^D+1) = \mathbb{Z}_q[Y,Z]/(Z^n+1, Y^D+1).$$

- $sk \leftarrow$ KeyGen(params): Sample $sk \leftarrow Q$ uniformly at random.

- $\mu \leftarrow$ Encode$(m_1, \ldots, m_n)$: Given $(m_1, ..., m_n) \in \mathbb{Z}_p^n$, outputs

$$\mu \in \mathbb{Z}_p[Z]/(Z^n+1) \leftarrow \text{SIMD.Encode}(m_1, ..., m_n).$$

[2]EvalP is not efficient enough for the construction of DEBPIR, hence we propose a more efficient alternative called EvalFP at the end of this section.

- $m_1, \ldots, m_n \leftarrow \text{Decode}(\mu)$: Given $\mu \in \mathbb{Z}_p[Z]/(Z^n+1)$, outputs

$$(m_1, ..., m_n) \in \mathbb{Z}_p^n \leftarrow \text{SIMD.Decode}(\mu).$$

- $ct \leftarrow \text{Encrypt}(sk, \mu)$: Reinterpret $\mu \in \mathbb{Z}_p[Z]/(Z^n+1)$ as an element of Q. Sample $a \leftarrow Q, e \leftarrow \chi$. Let

$$b = a \cdot s + p \cdot e + \mu \in Q.$$

  Define $ct \in R$ as the formal polynomial with a symbolic variable Y via:

$$ct(Y) = -a \cdot Y + b$$

- $\mu \leftarrow \text{Decrypt}(sk, ct)$: Interpret ciphertext $ct \in R$ as a formal polynomial $ct(Y) \in Q[Y]/(Y^D+1)$, evaluate $ct(Y)$ at $s \in Q$ and get $g = ct(sk) \in Q$. Interpret $g \in Q$ as a formal polynomial $g(Z) \in \mathbb{Z}_q[Z]/(Z^n+1)$. Reinterpret g as an element of $\mathbb{Z}_p[Z]/(Z^n+1)$ and output it.

- $ct_f \leftarrow \text{EvalP}(\boldsymbol{ct}, f)$: Given a polynomial $f \in \mathbb{Z}_p[X_1, \cdots, X_m]$ with individual degree less than d and $\boldsymbol{ct} = (ct_1, \cdots, ct_m) \in R^m$, we first lift every coefficient of f from $\mathbb{Z}_p$ to R and get $\overline{f}$. Outputs the result ciphertext of homomorphic multi-variate polynomial evaluation $ct_f = \overline{f}(ct_1, \cdots, ct_m) \in R$.

Theorem 4.2 (VASHE from Ring-LWE)**.** The above scheme is a VASHE for poly-logarithmic degree under Ring-LWE with quasi-polynomial approximation factors.

Proof. We argue that the scheme satisfies the correctness, security and efficiency properties of Definition 4.1.

Correctness: Let $f \in \mathbb{Z}_p[X_1, \ldots, X_m]$ be any polynomial of total degree $< D$ with individual degree $< d$ over $\mathbb{Z}_p$. Lift f to $\overline{f} \in R[X_1, \ldots, X_m]$. Fresh encryptions $ct_i = ct_i(Y)$ outputted by VASHE.Encrypt are degree-1 polynomials in Y. Without loss of generality, we analyze correctness assuming all operations are done in $Q[Y]$. This is because $ct_f(Y) = \overline{f}(ct_1, ..., ct_m)$ has degree $< D$ in Y, hence modding out by (Y^D+1) does nothing, and ct_f is the same whether we do the computation over $R = Q[Y]/(Y^D+1)$ or simply over $Q[Y]$.

We say that a ciphertext $ct(Y)$ is an encryption of $\mu \in \mathbb{Z}_p[Z]/\langle Z^n+1\rangle$ with noise γ if $\|ct(s)\|_\infty \leq \gamma$ and $Dec(s, ct) = \mu$. A freshly encrypted ciphertext $ct(Y)$ of a message μ has noise $\gamma = p(\beta+1) < q/2$, hence ct can be correctly decrypted. We can show the correctness property (1) by additionally noting the fact that SIMD encoding and decoding are ring isomorphisms between $\mathbb{Z}_p[Z]/(Z^n+1)$ and $\mathbb{Z}_p^n$.

Assume ct_1, ct_2 are γ_1, γ_2 noisy encryptions of μ_1, μ_2 respectively. Then $ct_1 + ct_2$ is an encryption of $\mu_1 + \mu_2$ with noise $\gamma^+ = (\gamma_1 + \gamma_2)$ as long as $\gamma^+ < q/2$ since

$$(ct_1 + ct_2)(s) = ct_1(s) + ct_2(s) = (pe_1 + \mu_1) + (pe_2 + \mu_2) = p(e_1 + e_2) + (\mu_1 + \mu_2).$$

Similarly, $ct_1 \cdot ct_2$ is an encryption of $\mu_1 \cdot \mu_2$ with noise $\gamma^\times = n\gamma_1\gamma_2$ as long as $\gamma^\times < q/2$. For the above, we rely on the fact that for $a, b \in \mathbb{Z}_q[Z]/(Z^n+1)$ with $\|a\|_\infty < \gamma_a$ and $\|b\|_\infty < \gamma_b$, we have $\|a \cdot b\|_\infty < n\gamma_a\gamma_b$. Lastly, if $a \in \mathbb{Z}_p$ is a constant then $a \cdot ct$ is an encryption of $a\mu$ with noise $\gamma_C = p\gamma$ as long as $\gamma_C < q/2$. Therefore, if ct_i are fresh encryptions of μ_i, then $ct_f = \text{EvalP}(\{(ct_1, ..., ct_m)\}, f)$ is an encryption of $f(\mu_1, ..., \mu_m)$ with noise $\gamma_f = N \cdot p(pn(\beta+1))^D$, as long as $\gamma_f < q/2$, which is ensured by our choice of parameters. The SIMD encoding and decoding make sure that the

multiplications and additions over ciphertext ring R correspond to the component-wise operations over n slots. The correctness property (2) then follows from the above discussion.

Security: Security follows directly from the (scaled error variant of the) Ring-LWE assumption. In particular, the ciphertexts consist of $a_i \leftarrow Q$, $b_i = a_i \cdot s + p \cdot e_i + \mu_i$ which is indistinguishable from uniform under RingLWE. In general, we can bound $t = D \log p + d \log m + \log p + 1 = poly(\log \lambda)$ and therefore we need to rely on RingLWE with quasi-polynomial approximation factors.

Efficiency: By the definition of Ring-LWE, we can determine the parameters n, q, χ in $poly(\lambda, t) = poly(\lambda, d, m)$ time, which also bounds the efficiency of sampling from λ. We have $q = \lambda^{poly(t)} = \lambda^{poly(d,m,\log \lambda)}$ and $n = poly(\lambda, t) = poly(\lambda, d, m, \log p)$. The cardinality of the ring R is q^{Dn} and therefore the ring elements have description length $Dn \log q = poly(\lambda, d, m, \log p)$ bits. The run-time of ring operations and the algorithms Setup, KeyGen, Encode, Decode, Encrypt, Decrypt are therefore also bounded by $poly(\lambda, d, m)$. The run-time of EvalP is bounded by $O(d^m) \cdot poly(\lambda, d, m)$. □

Homomorphic Polynomial Evaluation with Preprocessing. For a fixed polynomial $f \in \mathbb{Z}_p[X_1, \cdots, X_m]$ with individual degree less than d, we note that EvalP first lifts f to $\overline{f} \in R[X_1, \cdots, X_m]$ and outputs $ct_f = \overline{f}(ct_1, \cdots, ct_m) \in R$, which is nothing but multi-variate polynomial evaluation over R. It turns out that the batched PIR problem can be converted into the evaluation of a multivariate polynomial for a fixed polynomial f_{DB} that is only related to the database DB. However, the efficiency of the multi-variate polynomial evaluation algorithm EvalP is not powerful enough to meet the efficiency requirement of DEBPIR. To overcome this challenge, we leverage the FPE algorithm developed in Theorem 3.3, to speed up the evaluation via pre-processing and by delegating some of the work to the client.

For the fixed polynomial f, the computation efficiency of homomorphic low-degree multi-variate polynomial evaluation can be improved by applying the fast multi-variate polynomial evaluation with preprocessing. In detail, we lift f to $\overline{f}$ and then use the Prep algorithm (Theorem 3.3) during the offline phase to obtain the preprocessed data structure T_f. The client applies the FPE. Decomp algorithm to the ciphertexts $\{ct_1, \ldots, ct_m\} \in R^m$ and sends the resulted query vector $\boldsymbol{v_{ct}}$ to the server.

The server invokes FPE. Eval to obtain an encoding vector of $\boldsymbol{v}_{\overline{f}(\boldsymbol{ct})}$ by looking up the data structure T_f. We denote this subroutine by

$$\boldsymbol{v}_{\overline{f}(\boldsymbol{ct})} \leftarrow \text{EvalFP}(T_f, \boldsymbol{v_{ct}}).$$

Finally, the client receives $\boldsymbol{v}_{\overline{f}(\boldsymbol{ct})}$ and performs the CRT reconstruction to obtain $\overline{f}(\boldsymbol{ct})$ by calling FPE. Reconst, which is exactly the output of EvalP. Through the use of FPE, some work is delegated to the client, thus reducing the run-time of the server during the online phase to $O(n(\log q + \log n + \log D)) \cdot poly(m, d)$. In particular, the amortized time complexity of EvalFP becomes $O(\log q + \log n + \log D) \cdot poly(m, d)$. We then apply the above choices of parameters, resulting in $poly(\log \lambda)$ amortized time complexity of EvalFP.

5 Doubly Efficient Batched PIR

In this section, we formally define a batched PIR scheme called *doubly efficient batched private information retrieval* (DEBPIR) and give a generic construction from VASHE, via fast polynomial evaluation with preprocessing.

5.1 Definition

At high level, a DEBPIR scheme is a protocol between a server and a client. The protocol consists of four algorithms: Prep, Query, Resp and Dec, and the algorithms are performed in two phases, offline and online, illustrated as follows.

Offline: The server has a database $DB = \{0,1\}^N$. In the offline preprocessing phase, server runs the deterministic algorithm $\tilde{\text{DB}} \leftarrow \text{Prep}(1^\lambda, DB)$ and stores $\tilde{\text{DB}}$ in the random-access memory.

Online: The client holds indices $\{i^{(j)} \in [N]\}_{j\in[n]}$, and wants to learn $DB[i^{(j)}]$ without revealing $i^{(j)}$ for $j \in [n]$.

1. The client runs $(\boldsymbol{v_{ct}}, s) \leftarrow \text{Query}(1^\lambda, N, \{i^{(1)}, ..., i^{(n)}\})$ to generate encrypted query vector $\boldsymbol{v_{ct}}$ (i.e. CRT form of query ciphertext) that it sends to Server, and a secret key s that it keeps locally.

2. The server responds with the answer $\text{ans} \leftarrow \text{Resp}(\tilde{DB}, \boldsymbol{v_{ct}})$.

3. The client decrypts the answer using the algorithm $\text{Dec}(s, \text{ans})$ to learn $DB[i^{(j)}]$ for all $j \in [n]$.

We next describe the definition formally. Our definition is based on those of [22], with two main differences. First of all, DEBPIR is a batched PIR scheme where the client can query the database at n indices at the same time, hence for efficiency we also consider the amortized response time of the server for a single index. Secondly, although the online phase of DEBPIR also consists of three subroutines $(\text{Query}, \text{Resp}, \text{Dec})$, the input and output is not a HE ciphertext, but encrypted vectors which are of CRT form with respect to some public parameters.

Definition 5.1. (DEBPIR). A doubly efficient batched private information retrieval scheme (DEBPIR) is a tuple of algorithms (Prep, Query, Resp, Dec) with the following syntax.

1. $\tilde{DB} \leftarrow \text{Prep}(1^\lambda, DB)$ takes the security parameter 1^λ and a database $DB = \{0,1\}^N$, and deterministically outputs a preprocessed database $\tilde{DB}$.

2. $(\boldsymbol{v_{ct}}, s) \leftarrow \text{Query}(1^\lambda, N, \{i^{(1)}, ..., i^{(n)}\})$ takes the security parameter 1^λ, a database size N and n indices $\{i^{(j)} \in [N]\}_{j\in[n]}$. It outputs an encrypted query vector $\boldsymbol{v_{ct}}$ and a secret key s.

3. $\text{ans} \leftarrow \text{Resp}(\tilde{DB}, \boldsymbol{v_{ct}})$ takes the preprocessed database $\tilde{DB}$ stored in a random-access memory and a query $\boldsymbol{v_{ct}}$. It responds with an answer ans.

4. $\boldsymbol{b} \leftarrow \text{Dec}(s, \text{ans})$ takes the secret key s and the answer ans and outputs decode bits $\boldsymbol{b} \in \{0,1\}^n$.

The algorithms should satisfy the following properties.

- **Correctness:** Honest execution of Prep, Query, Resp and Dec successfully recovers requested data items with probability 1. That is, for every $DB \in \{0,1\}^N$ and every sequence $\{i^{(j)} \in [N]\}_{j=1,\cdots,n}$, it holds that:

$$Pr\left[\boldsymbol{b}[j] = DB[i^{(j)}] \ for\ j \in [n] : \begin{array}{c} \tilde{DB} \leftarrow \text{Prep}(1^\lambda, DB) \\ (\boldsymbol{v_{ct}}, s) \leftarrow \text{Query}(1^\lambda, N, \{i^{(1)}, ..., i^{(n)}\}) \\ \text{ans} \leftarrow \text{Resp}(\tilde{DB}, \boldsymbol{v_{ct}}) \\ \boldsymbol{b} \leftarrow \text{Dec}(s, \text{ans}) \end{array}\right] = 1.$$

- **Security:** No efficient adversary can distinguish the queries output by Query on input index $\{i^{(1)}, ..., i^{(n)}\}$ and $\{j^{(1)}, ..., j^{(n)}\}$. Namely, we define the following game between a challenger and an adversary $\mathcal{A}$.

  1. $(I := \{i^{(1)}, ..., i^{(n)}\}, J := \{j^{(1)}, ..., j^{(n)}\}, 1^N, \text{aux}) \leftarrow \mathcal{A}(1^\lambda)$: $\mathcal{A}$ selects a size N, a challenge index sequence I, J, and auxiliary information aux.
  2. $t \leftarrow \{0,1\}; (\boldsymbol{v_{ct}}, s) \leftarrow \text{Query}(1^\lambda, N, i_t)$: The challenger selects a random bit b and generates a sample encrypted query vector $\boldsymbol{v_{ct}}$ for the chosen index sequence (I if $t = 0$, otherwise J).
  3. $t' \leftarrow \mathcal{A}(\text{aux}, \boldsymbol{v_{ct}})$: $\mathcal{A}$ outputs a guess for t, given the encrypted query vector $\boldsymbol{v_{ct}}$.

  We require that for every PPT adversary $\mathcal{A}$, there exists a negligible function negl such that the distinguishing advantage of $\mathcal{A}$ in the above security game is $|Pr[t' = t] - 1/2| \leq \text{negl}(\lambda)$.

- **Efficiency:** Suppose that Resp is given random accesses to $\tilde{DB}$. We say the batched PIR scheme (Prep, Query, Resp, Dec) is doubly efficient if the running time of Prep is $poly(\lambda, N)$, and Query, Resp, Dec run in time $poly(\lambda, \log N)$. Furthermore, we require that the amortized Resp run-time and the amortized communication cost are $poly(\log \lambda, \log N)$.

5.2 Construction

We now construct the DEBPIR scheme. The construction relies on combining SIMD with the DEPIR scheme in [22]. We plan to construct DEBPIR in three steps:

- Express the function $f_{DB}(i) = DB[i]$ as an m-variate polynomial of individual degree $< d$ over $\mathbb{F}_p$, where the inputs are the base-d digits of the index i and p is the plaintext modulus of the VASHE scheme.
- Construct a basic batched PIR scheme by using VASHE.EvalP to homomorphically evaluate the polynomial f_{DB} over the encryptions of the base-d digits of n indices simultaneously.
- Preprocess the polynomial f_{DB} into a data structure $\tilde{DB}$, using Theorem 3.3 so that we can replace VASHE.EvalP with fast polynomial evaluation with preprocessing (VASHE.EvalFP), upgrading the basic batched PIR into a DEBPIR.

We first give a construction overview for each of the steps, introducing some useful notation and claims along the way. We choose parameters d, m such that $d^m > N$. Let $D = dm$. Let $R = \mathbb{Z}_q[Y, Z]/(Z^n + 1, Y^D + 1)$ and $p \equiv 1 \pmod{2n}$ is the plaintext modulus of VASHE scheme.

Encoding the Database as a Polynomial. The server encodes the database into a polynomial f_{DB} over $\mathbb{F}_p$ through multivariate polynomial interpolation. For $i \in [\![N]\!]$, let $(i_1, \cdots, i_m) = base_{d,m}(i)$ be the based-d representation of i such that

$$i = \sum_{j=1}^{m} i_j d^{j-1} \text{ with } i_j \in [\![d]\!].$$

The polynomial f_{DB} satisfies $f_{DB}(base_{d,m}(i)) = DB[i]$ and moreover we can find the coefficient representation of this polynomial efficiently. The following lemma is a slight modification of Claim 4.2.1 in [22], which shows the correctness of this step and bounds the efficiency of the encoding procedure.

Lemma 5.2. For any $DB \in \{0,1\}^N$, any d, m such that $d^m \geq N$, there exists a m-variate polynomial $f_{DB}(X_1, \cdots, X_m)$ with individual degree $< d$ in each variable over $\mathbb{F}_p$ for some prime $p > d$ such that for all $i \in [\![N]\!]$, it holds that $f_{DB}(i_1, \cdots, i_m) = DB[i]$ where $(i_1, \cdots, i_m) = base_{d,m}(i)$. Moreover, there is an multivariate polynomial interpolation algorithm that outputs the coefficients of f_{DB} in time $O(d^m \cdot m) \cdot poly(\log p, \log d)$.

Proof. We invoke multivariate polynomial interpolation from Lemma 3.5. Define the d^m evaluation points $\{y_{(x_1,\cdots,x_m)} \in \mathbb{F}_p\}_{(x_1,\cdots,x_m)\in\mathbb{F}_p^m, x_i\in[\![d]\!]}$ via $y_{(x_1,\cdots,x_m)} = DB[i]$ when $(x_1, \cdots, x_m) = base_{d,m}(i)$ for $i \in [N]$ and $y_{(x_1,\cdots,x_m)} = 0$ else. The lemma says that in time $O(d^m \cdot m \cdot poly(\log p, \log d))$ we can interpolate the coefficients of a polynomial $f_{DB}(X_1, \cdots, X_m)$ such that $f_{DB}(x_1, \cdots, x_m) = y_{(x_1,\cdots,x_m)}$ for all $(x_1, \cdots, x_m) \in \mathbb{F}_p^m$ and $x_i \in [\![d]\!]$. □

Basic Batched PIR. The client uses a VASHE scheme to encrypt m base-d digits of indices and sends the ciphertext vector to the server. The server responds by invoking VASHE.EvalP to homomorphically evaluate the polynomial f_{DB}, and sending the homomorphically evaluated ciphertext to the client. The client uses VASHE.Decrypt and VASHE.Decode to recover $DB[i^{(j)}] = f_{DB}(base_{d,m}(i^{(j)}))$ for $j \in [n]$. As mentioned above, this simple scheme does not satisfy the strong efficiency requirements of DEBPIR. Therefore, our next step is to preprocess the polynomial f_{DB}, and upgrade the basic batched PIR into a doubly efficient batched PIR by replacing VASHE.EvalP with VASHE.EvalFP.

Upgrading to DEBPIR. Lastly, we take the basic batched PIR scheme above and upgrade it to DEBPIR by preprocessing the polynomial f_{DB}. We improve the server's efficiency and accelerate the computation of $ct^* := \overline{f_{DB}}(ct_1, \cdots, ct_m)$ by preprocessing f_{DB} into some data structure $\tilde{DB}$ using Theorem 3.3. Furthermore, the server relegates some work of computation of ct^* to the client, which is a re-balance of the work in favor of the server.

In detail, the client first breaks the query ciphertext vector into CRT form as a encrypted query vector. Then the server sends the output[3] of VASHE.EvalFP instead of VASHE.EvalP to client. At the end, the client does the CRT reconstruction to get ct^*. In the process of homomorphically evaluating the polynomial f_{DB}, the server only does the VASHE.EvalFP, which is nothing but looking up some pre-computed tables. Consequently, the amortized run-time of Resp will become $poly(\log \lambda, \log N)$. We present the full construction below.

Algorithm 1: DEBPIR from VASHE

Parameters: The size of database is N, and it determines the parameters d and m such that $d^m \geq N$. Let $params \leftarrow \text{VASHE.Setup}(1^\lambda, 1^d, 1^m)$, which determines a ring $R = \mathbb{Z}_q[Y, Z]/(Z^n + 1, Y^D + 1)$, total degree $D = dm$, plaintext modulus p and the number of slots n. Without loss of generality, we implicitly assume all algorithms have access to these parameters, which they can derive from λ, N.

$\text{Prep}(1^\lambda, DB)$:

1. Use polynomial interpolation to obtain $f_{DB} \in \mathbb{F}_p[X_1, ..., X_m]$ such that $f(base_{d,m}(i)) = DB[i]$ for every $i \in [\![N]\!]$. Note that $deg_{X_i}(f_{DB}) < d$ for all X_i, hence the total degree of $f_{DB} < D$.
2. Lift f_{DB} to $\overline{f_{DB}}$.

[3]The output is also of CRT form.

3. aux $\leftarrow$ FPE. Setup(R), output $\tilde{DB} \leftarrow$ FPE. Prep($\overline{f_{DB}}$, aux; R).

Query($1^\lambda, N, \{i^{(1)}, ..., i^{(n)}\}$):

1. Let $(i_1^{(j)}, ..., i_m^{(j)}) = base_{d,m}(i^{(j)})$ be the based-d digits of $i^{(j)}$ for each $j \in [n]$.
2. Sample $s \leftarrow$ VASHE. KeyGen($params$).
3. For each $j \in [m]$, encode $(i_j^{(1)}, ..., i_j^{(n)})$ by invoking
$$\mu_j \leftarrow \text{VASHE. Encode}(i_j^{(1)}, ..., i_j^{(n)}).$$
4. For each $j \in [m]$, encrypt μ_j by invoking $ct_j \leftarrow$ VASHE. Encrypt(s, μ_j).
5. Output $\boldsymbol{v}_{ct} \leftarrow$ FPE. Decomp($ct = \{ct_1, ..., ct_m\}, aux$; R).

Resp($\tilde{DB}, \boldsymbol{v}_{ct}$):

1. Output $\boldsymbol{v}_{ct_f} \leftarrow$ FPE. Eval($\boldsymbol{v}_{ct}, \tilde{DB}$, aux; R).

Dec($s, \boldsymbol{v}_{ct_f}$):

1. $ct_f \leftarrow$ FPE. Reconst($\boldsymbol{v}_{ct_f}$).
2. $\overline{\mu} \leftarrow$ VASHE. Decrypt(s, ct_f).
3. Output VASHE. Decode($\overline{\mu}$).

Correctness. Consider any $DB \in \{0,1\}^N$ and any $i \in [\![N]\!]$ with $(i_1, \ldots, i_m) = base_{d,m}(i)$. Let $\tilde{DB} \leftarrow \text{Prep}(1^\lambda, DB)$, $(\boldsymbol{v_{ct}}, s) \leftarrow \text{Query}(1^\lambda, N, \{i^{(1)}, ..., i^{(n)}\})$, $\boldsymbol{v}_{ct_f} \leftarrow \text{Resp}(\tilde{DB}, \boldsymbol{v_{ct}})$, $\boldsymbol{b} = \text{Dec}(s, \boldsymbol{v}_{ct_f})$. By Lemma 5.2, we know that the polynomial f_{DB} computed during preprocessing satisfies $f_{DB}(i_1, \ldots, i_m) = DB[i]$. Also, by the correctness of fast polynomial evaluation with preprocessing (Theorem A.3), we have that the encrypted query vector $\boldsymbol{v_{ct}}$ computed during Query is the CRT form of the query ciphertexts $(ct_1, \ldots, ct_m)$, the result $\boldsymbol{v}_{ct_f}$ of Resp is the CRT encoding of $\overline{f_{DB}}(ct_1, \cdots, ct_m)$. Hence, by the definition of correctness for VASHE (Definition 4.1), we have that for $j \in [n]$,

$$\begin{aligned}\boldsymbol{b}[j] &= \text{VASHE. Decode}(\text{VASHE. Decrypt}(\text{s}, \text{FPE. Reconst}(\boldsymbol{v}_{\text{ct}_\text{f}})))[j] \\ &= f_{DB}(i_1^{(j)}, \ldots, i_m^{(j)}) \\ &= DB[i^{(j)}].\end{aligned}$$

Security. The security of the DEBPIR follows directly from that of VASHE, since the adversary sees m encrypted query vectors and can only reconstruct m VASHE ciphertexts with public parameters. Notice that the adversary runs in time $poly(\lambda)$ and chooses the bound 1^N , meaning that $N = poly(\lambda)$. Depending on the choice of d, m and q such that $d, m = poly(\log N) = poly(\log \lambda)$ and $q = \lambda^{poly(\log \lambda)}$, we can rely on VASHE security for polylogarithmic degree.

Efficiency. We calculate the computation time and output size for each algorithm. Recall that, by the definition of VASHE efficiency, the bit-length of ring elements and the run-time of the ring operations are bounded by $poly(\lambda, d, m)$. For any constant $\epsilon > 0$, we can choose the same d and m as in [22], such that $d = O(\log^{2/\varepsilon} N)$, $m = \lceil \log_d(N) \rceil = \frac{\epsilon}{2} \log N / \log \log N + O(1)$.

1. Prep: The multivariate polynomial interpolation takes time $d^m m \cdot poly(\log p, \log d)$.
   The preprocessing run-time and the server storage are bounded by
   $$\begin{aligned} &d^m \cdot m^m \cdot poly(m, d, |R|) \cdot O(\log m + \log d + \log\log |R|)^m \\ &= d^m \cdot m^m \cdot poly(m, d, \lambda) \cdot O(\log m + \log d + \log \lambda)^m \\ &+ O(N^{1+\epsilon}) poly(\lambda, \log N). \end{aligned}$$
   where we bound
   $$m^m \leq O(\log N)^m \leq (\log N)^{\frac{\epsilon}{2} \log N / \log\log N + O(1)} \leq N^{\frac{\epsilon}{2}} poly(\log N),$$
   and similarly
   $$O(\log m + \log d + \log \lambda)^m \leq O(\log N)^m \leq N^{\frac{\epsilon}{2}} poly(\log N).$$

2. Query: The run-time of the Query is bounded by that of VASHE. KeyGen, that of running m copies of VASHE. Encrypt and that of FPE. Decomp, which is bounded by
   $$\begin{aligned} &O((nD \cdot (\log q + \log n + \log D))^2) \cdot poly(d, m) + poly(d, m, log\ |R|) \\ &= poly(\lambda, \log N). \end{aligned}$$

3. Resp: The run-time is bounded by $O((nD \cdot (\log q + \log n + \log D))) \cdot poly(d, m)$. As DEBPIR satisfies SIMD and is a batched PIR scheme, one running of the Resp algorithm corresponds to the responses of n indices, hence the amortized run-time is $poly(\log \lambda, \log N)$.

4. Dec: The rum-time of Dec is bounded by $poly(\lambda, d, m) = poly(\lambda, \log N)$.

The communication cost is bounded by $O((nD \cdot (\log q + \log n + \log D))) \cdot poly(d, m)$. The amortized communication cost is $poly(\log \lambda, \log N)$.

Theorem 5.3. Assuming VASHE for polylogarithmic degree, for any constant $\epsilon > 0$, there is a DEBPIR scheme such that, for a database of size N and security parameter λ, the preprocessing run-time and the server storage are bounded by $O(N^{1+\epsilon}) poly(\lambda, \log N)$, the query time is bounded by $poly(\lambda, \log N)$, the amortized response time is bounded by $poly(\log \lambda, \log N)$ and the decrypt time is bounded by $poly(\lambda, \log N)$. The amortized communication cost is bounded by $poly(\log \lambda, \log N)$. In particular, such a scheme exists assuming Ring-LWE with quasi-polynomial approximation factors.

The server response time and the communication cost (in amortization) of DEPIR [22] are $poly(\lambda, \log N)$. This is in contrast with what we obtain in this work, which is $poly(\log \lambda, \log N) = poly(\log \lambda) = \tilde{O}_\lambda(1)$, since $N = poly(\lambda)$. Therefore the amortized response time of the server and the amortized communication cost become $\tilde{O}_\lambda(1)$. This is an exponential improvement from previous schemes and it is optimal up to some polylogarithmic terms.

Acknowledgments

Xiuquan Ding is supported by the National Natural Science Foundation of China under Grant No. 12171469 and National Key Research and Development Project 2020YFA0712300. Tianwei Zhang is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972.

References

[1] Angel, S., Chen, H., Laine, K., Setty, S.: Pir with compressed queries and amortized query processing. In: 2018 IEEE symposium on security and privacy (SP). pp. 962–979. IEEE (2018)

[2] Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 551–569. USENIX Association, Savannah, GA (Nov 2016), https://www.usenix.org/conference/osdi16/technical-sessions/presentation/angel

[3] Angel, S., Setty, S.T.: Unobservable communication over fully untrusted infrastructure. In: OSDI. vol. 16, pp. 551–569 (2016)

[4] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) Advances in Cryptology - CRYPTO 2009. pp. 595–618. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

[5] Beimel, A., Ishai, Y.: Information-theoretic private information retrieval: A unified construction. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) Automata, Languages and Programming. pp. 912–926. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)

[6] Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers computation in private information retrieval: Pir with preprocessing. In: Bellare, M. (ed.) Advances in Cryptology — CRYPTO 2000. pp. 55–73. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)

[7] Borisov, N., Danezis, G., Goldberg, I.: Dp5: A private presence service. Proceedings on Privacy Enhancing Technologies **2015** (06 2015)

[8] 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)

[9] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)

[10] Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. pp. 505–524. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[11] Canetti, R., Holmgren, J., Richelson, S.: Towards doubly efficient private information retrieval. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. pp. 694–726. Springer International Publishing, Cham (2017)

[12] Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Proceedings of IEEE 36th Annual Foundations of Computer Science. pp. 41–50 (1995)

[13] Demmler, D., Rindal, P., Rosulek, M., Trieu, N.: Pir-psi: Scaling private contact discovery. Proceedings on Privacy Enhancing Technologies **2018**, 159–178 (10 2018)

[14] Fanti, G., Ramchandran, K.: Efficient private information retrieval over unsynchronized databases. IEEE Journal of Selected Topics in Signal Processing **9**(7), 1229–1239 (2015)

[15] Henry, R.: Polynomial batch codes for efficient it-pir. Cryptology ePrint Archive (2016)

[16] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Batch codes and their applications. In: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing. pp. 262–271 (2004)

[17] Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: USENIX Security Symposium. pp. 1447–1464 (2019)

[18] Kedlaya, K.S., Umans, C.: Fast modular composition in any characteristic. In: 2008 49th Annual IEEE Symposium on Foundations of Computer Science. pp. 146–155 (2008)

[19] Kedlaya, K.S., Umans, C.: Fast polynomial factorization and modular composition. SIAM Journal on Computing **40**(6), 1767–1802 (2011)

[20] Kogan, D., Corrigan-Gibbs, H.: Mittal lookups with checklist. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 875–892. USENIX Association (Aug 2021), https://www.usenix.org/conference/usenixsecurity21/presentation/kogan

[21] Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: An efficient communication system with strong anonymity. Proceedings on Privacy Enhancing Technologies **2016** (08 2015)

[22] Lin, W.K., Mook, E., Wichs, D.: Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe. Cryptology ePrint Archive (2022)

[23] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. pp. 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[24] Mittal, P., Olumofin, F., Troncoso, C., Borisov, N., Goldberg, I.: PIR-Tor: Scalable anonymous communication using private information retrieval. In: 20th USENIX Security Symposium (USENIX Security 11). USENIX Association, San Francisco, CA (Aug 2011), https://www.usenix.org/conference/usenix-security-11/pir-tor-scalable-anonymous-communication-using-private-information

[25] Mughees, M., Ren, L.: Vectorized batch private information retrieval. In: 2023 2023 IEEE Symposium on Security and Privacy (SP) (SP). pp. 437–452. IEEE Computer Society, Los Alamitos, CA, USA (may 2023)

[26] Mughees, M.H., Pestana, G., Davidson, A., Livshits, B.: Privatefetch: Scalable catalog delivery in privacy-preserving advertising. arXiv preprint arXiv:2109.08189 (2021)

[27] Olumofin, F.G.: Practical Private Information Retrieval. Ph.D. thesis, University of Waterloo, Ontario, Canada (2011), https://hdl.handle.net/10012/6142

[28] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)

[29] Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**, 57–81 (2014)

[30] Von Zur Gathen, J., Gerhard, J.: Modern computer algebra. Cambridge university press (2013)

A Fast Polynomial Evaluation with Preprocessing

We prove Theorem 3.3, restated below.

Theorem 3.3. Let the ring $R := \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle$ for two monic polynomials $E_1(Y)$ and $E_2(Z)$ with the degree e_1 and e_2 respectively. Let $f(X_1,\ldots,X_m) \in R[X_1,\ldots,X_m]$ be a fixed polynomial with $deg_{X_i}(f) < d$. Then, there exists an FPE scheme satisfies the following properties. The preprocessing algorithm Prep runs in time

$$S = d^m \cdot poly(m, d, \log|R|) \cdot O(m(\log m + \log d + \log\log|R|))^m$$

and outputs a data structure of size at most of S. The running time of Decomp is bounded by

$$O(e_1e_2(\log q + \log e_1 + \log e_2)^2 \cdot poly(d, m, \log\log q)).$$

The running time of Eval is bounded by

$$O(e_1e_2(\log q + \log e_1 + \log e_2))poly(m, d).$$

The running time of Reconst is bounded by

$$poly(m, d, e_1, e_2, \log q).$$

Furthermore, the output size of Setup is bounded by $O(e_1^2e_2^2 \cdot (\log q + \log e_1 + \log e_2)) \cdot poly(m, d)$, and the output size of Decomp and Eval are bounded by

$$O(e_1e_2(\log q + \log e_1 + \log e_2)) \cdot poly(m, d).$$

We now give a full proof of the theorem. We first consider the special case $R = \mathbb{Z}_q$, and then we will show how to handle the general case.

A.1 Polynomials over $\mathbb{Z}_q$

We begin with a special case, where $R = \mathbb{Z}_q$. We fix the parameters d, m as additional input.

Theorem A.1. Let $R = \mathbb{Z}_q$. Let $f(X_1,...,X_m) \in R[X_1,...,X_m]$ be a fixed polynomial with $deg_{X_i}(f) < d$. Then, the preprocessing algorithm $\mathrm{Prep}(f, aux; \mathbb{Z}_q)$ runs in time

$$S = O(md\log q)^m \cdot poly(m, d, \log q).$$

and outputs a data structure of size at most S.

The running time of Decomp is bounded by

$$O((\log q)^2) \cdot poly(m, d).$$

The running time of Eval is bounded by

$$O(\log q) \cdot poly(m, d).$$

The running time of Reconst is bounded by

$$poly(m, d, \log q).$$

Moreover, the output size of Setup is bounded by $O(d \cdot m \cdot \log q)$, and the output size of Decomp and Eval are bounded by $O(\log q) \cdot poly(m, d)$.

Proof. Setup($\mathbb{Z}_q$) :

1. Let $M := d^m q^{m(d-1)+1}$, and let $p_1, ..., p_h$ be the distinct primes less than $16 \log M$ for some $h \in \mathbb{N}$.
2. Output aux $= (M, \{p_1, ..., p_h\})$.

The size of aux is bounded by $\log M + \sum \log p_i \leq O(d \cdot m \cdot \log q)$.

Prep(f, aux; $\mathbb{Z}_q$) :

1. Lift f to $\overline{f} \in \mathbb{Z}[X_1, ..., X_m]$, and denote $\overline{f}_i = \overline{f} \pmod{p_i}$ for $i \in [h]$.
2. Using FFT on $\mathbb{Z}_{p_i}^m$, evaluate $\overline{f_i}(a)$ on the all points $a \in \mathbb{Z}_{p_i}^m$. Store the evaluations in a table T_i, i.e. $T_i = \{\overline{f}_i(a) \mid a \in \mathbb{Z}_{p_i}^m\}$.
3. Output $T = \{T_i \mid i \in [h]\}$.

The running time of Prep is bounded by

$$S = O(md \log q)^m \cdot poly(m, d, \log q).$$

The size of output of Prep is bounded by S.

Decomp(α, aux; $\mathbb{Z}_q$) :

1. Lift $\alpha \in \mathbb{Z}_q^m$ to $\overline{\alpha} \in \mathbb{Z}^m$.
2. For $i \in [h]$, compute $\overline{\alpha}_i = \overline{\alpha} \pmod{p_i} \in \mathbb{Z}_{p_i}^m$.
3. Output $\boldsymbol{v}_\alpha = \{\overline{\alpha}_i | i \in [h]\}$.

The time complexity of Decomp is bounded by $\sum_{i=1}^{h} m \cdot O(\log p_i \log q) \leq O((\log q)^2) \cdot poly(m, d)$. The size of output is bounded by $m \cdot \sum \log p_i \leq O(\log q) \cdot poly(m, d)$.

Eval($\boldsymbol{v} = \{v_i\}_{i \in [h]}, aux; \mathbb{Z}_q$) :

1. Obtain $\overline{f}_i(v_i) \in \mathbb{Z}_{p_i}$ by looking it up in table T_i.
2. Output $\boldsymbol{v}_{f(\alpha)} := \{\overline{f}_i(v_i) \mid i \in [h]\}$.

The time complexity of Eval is bounded by $O(h) + m \cdot \sum \log p_i \leq O(\log q) \cdot poly(m, d)$. The size of the output of Eval is bounded by $\sum \log p_i \leq O(\log q) \cdot poly(m, d)$.

Reconst($\boldsymbol{v}_z$, aux; $\mathbb{Z}_q$) :

1. Use CRT to find the smallest $z \in \mathbb{Z}_+$ such that for all $i \in [h]$

$$z \equiv \boldsymbol{v}_z[i] \pmod{p_i}.$$

2. Output $z \pmod{q}$.

The running time of Reconst is bounded by $poly(m, d, \log q)$. □

We now apply the theorem A.1 recursively to replace the $(\log q)^m$ factor in the preprocessing run-time by a $(\log \log q)^m$ factor.

Theorem A.2. Let $R = \mathbb{Z}_q$. Let $f(X_1, ..., X_m) \in R[X_1, ..., X_m]$ be a fixed polynomial with $deg_{X_i}(f) < d$. Then, the preprocessing algorithm $Prep$ runs in time

$$S = d^m \cdot poly(m, d, \log q) \cdot O(m(\log m + \log d + \log\log q))^m$$

and outputs a data structure of size at most S.

The running time of Decomp is bounded by

$$O(\log q)^2 \cdot poly(d, m).$$

The running time of Eval is bounded by

$$O(\log q) \cdot poly(m, d).$$

The running time of Reconst us bounded by

$$poly(m, d, \log q).$$

Moreover, the output size of Setup is bounded by $O(\log q) \cdot poly(m, d)$, and the output size of Decomp and Eval are both bounded by

$$O(\log q) \cdot poly(m, d).$$

Proof. To compute the data structure from the coefficients of f, we introduce a level of recursion by applying the theorem A.1, instead of invoking FFT directly.

Setup($\mathbb{Z}_q$) :

1. Firstly, apply the algorithm in theorem A.1 by $pp \leftarrow \text{Setup}(\mathbb{Z}_q)$.
2. For each p_i, we compute $pp_i \leftarrow \text{Setup}(\mathbb{Z}_{p_1})$.
3. Output aux $\leftarrow \{pp, pp_i \mid i \in [h]\}$.

The size of aux is bounded by $dm(\log q + \sum \log p_i) \leq O(\log q) \cdot poly(m, d)$.

Prep($f, aux; \mathbb{Z}_q$) :

1. Lift f to $\overline{f} \in \mathbb{Z}[X_1, ..., X_m]$, and denote $\overline{f}_i = \overline{f} \pmod{p_i}$ for $i \in [h]$.
2. Use Theorem A.1 to construct the data structure for $\overline{f}_i$ over $\mathbb{Z}_{p_i}$: $T_i \leftarrow \text{Prep}(\overline{f}_i, \text{aux}; \mathbb{Z}_{p_i})$.
3. Output $T = \{T_i \mid i \in [h]\}$.

By Theorem A.1, the data structure T_i can be computed in time $S_i = O(md \log p_i)^m \cdot poly(m, d, \log p_i)$, and takes at most S_i space. Hence the running time of Prep is bounded by

$$S = d^m \cdot poly(m, d, \log q) \cdot O(m(\log m + \log d + \log\log q))^m.$$

The total size of all h data structures is bounded by S.

For the Decomp algorithm, we also need to recursively invoke the algorithm Decomp in Theorem A.1.

Decomp(α, aux; $\mathbb{Z}_q$) :

1. Lift $\alpha \in \mathbb{Z}_q^m$ to $\overline{\alpha} \in \mathbb{Z}^m$.

2. For $i \in [h]$, compute $\overline{\alpha}_i = \overline{\alpha} \pmod{p_i}$.

3. Output $\boldsymbol{v}_\alpha = \{\text{Decomp}(\overline{\alpha}_i, \text{aux}; \mathbb{Z}_{p_i}) | i \in [h]\}$.

The time complexity of Decomp is bounded by

$$\sum_{p_i} O((\log p_i)^2) \cdot poly(m, d) \leq O((\log q)^2) \cdot poly(m, d).$$

The output size is bounded by $\sum_{p_i} O(\log p_i) \cdot poly(m, d) \leq O(\log q) \cdot poly(m, d)$.

For the Eval algorithm, we need to recursively invoke the algorithm Eval in Theorem A.1 rather than looking up the table directly.

Eval$(\boldsymbol{v} = \{\boldsymbol{v}_i\}_{i \in [h]}, \text{aux}; \mathbb{Z}_q)$:

1. For each $i \in [h]$, we denote $\boldsymbol{v}_{\overline{f}_i(\alpha)} \leftarrow \text{Eval}(\boldsymbol{v}_i, \text{aux}; \mathbb{Z}_{p_i})$ by using the algorithm in Theorem A.1.

2. Output $\boldsymbol{v}_{f(\alpha)} := \{\boldsymbol{v}_{\overline{f}_i(\alpha)} \mid i \in [h]\}$.

The time complexity of Eval is bounded by

$$\sum_{p_i} O(\log p_i) \cdot poly(m, d) \leq O(\log q) \cdot poly(m, d).$$

The output size of Eval is bounded by $\sum_{p_i} O(\log p_i) \cdot poly(m, d) \leq O(\log q) \cdot poly(m, d)$.

Reconst$(\boldsymbol{v}_z = \{\boldsymbol{v}_i \ i \in [h]\}, \text{aux}; \mathbb{Z}_q)$:

1. Use the algorithm Reconst in Theorem A.1 to obtain $v_i \leftarrow \text{Reconst}(\boldsymbol{v}_i, aux; \mathbb{Z}_{p_i})$

2. Use CRT to find the smallest $z \in \mathbb{Z}_+$ such that for all $i \in [h]$

$$z \equiv \boldsymbol{v}_i \pmod{p_i}.$$

3. Output $z \pmod{q}$.

The running time of Reconst is bounded by $poly(m, d, \log q)$. □

A.2 Polynomials over Extension Rings with one variable

In this section, we will consider the case $R = \mathbb{Z}_q[Y]/\langle E(Y) \rangle$, where $E(Y)$ is a monic polynomial with degree e.

Theorem A.3. Let $R = \mathbb{Z}_q[Y, Z]/\langle E(Y) \rangle$ for a monic polynomial $E(Y)$ with the degree $e > 0$. Let $f(X_1, ..., X_m) \in R[X_1, ..., X_m]$ be a fixed polynomial with $deg_{X_i}(f) < d$. Then, the preprocessing algorithm Prep runs in time

$$S = d^m \cdot poly(m, d, \log |R|) \cdot O(m(\log m + \log d + \log \log |R|))^m.$$

and outputs a data structure of size at most S.

The running time of Decomp is bounded by

$$O(e^2(\log q + \log e)^2 \cdot poly(m, d)).$$

The running time of Eval is bounded by

$$O(e(\log q + \log e) \cdot poly(m, d)).$$

The running time of Reconst is bounded by

$$poly(m, d, e, \log q).$$

Moreover, the output size of Setup is bounded by $O(e^2 \cdot (\log q + \log e)) \cdot poly(m, d)$, and the output size of Decomp and Eval are both bounded by

$$poly(\log m, \log d, \log e, \log\log q).$$

Proof. Setup$(\mathbb{Z}_q[Y]/\langle E(Y)\rangle)$:

1. Let $M := d^m(e(q-1))^{(d-1)m+1} + 1$, $D := (e-1)((d-1)m+1)$ and let $r = M^{D+1}$.
2. Output aux $= \{\{M^i \mid i \in \{0, \cdots, e-1\}\}, D, r, Setup(\mathbb{Z}_r)\}$.

Thus, the output size of Setup$(\mathbb{Z}_q[Y]/\langle E(Y)\rangle)$ is bounded by

$$\begin{aligned} &O(e^2 \log M) + O(dem \log M) + O(dm \log r) \\ &\leq O(e^2 \cdot (\log q + \log e)) \cdot poly(m, d). \end{aligned}$$

Prep$(f, \text{aux}; \mathbb{Z}_q[Y]/\langle E(Y)\rangle)$:

1. Lift $f \in R[X_1, ..., X_m]$ to $\tilde{f} \in \mathbb{Z}[Y][X_1, ..., X_m]$.
2. Compute $\overline{f} \in \mathbb{Z}_r[X_1, ..., X_m]$ by reducing $\tilde{f}$ modulo the ideal $(r, Y - M)$.
3. Output $T \leftarrow \text{Prep}(\overline{f}, \text{aux}; \mathbb{Z}_r)$.

The output size and the running time are bounded by $S = d^m \cdot poly(m, d, \log|R|) \cdot O(m(\log m + \log d + \log\log|R|))^m$.

Decomp$(\alpha, \text{aux}; \mathbb{Z}_q[Y]/\langle E(Y)\rangle)$:

1. Lift $\alpha \in R^m$ to $\tilde{\alpha} \in \mathbb{Z}[Y]^m$.
2. Compute $\overline{\alpha}$ by reducing each coordinate of $\tilde{\alpha}$ modulo the ideal $(r, Y - M)$.
3. Output $\boldsymbol{v}_\alpha \leftarrow \text{Decomp}(\overline{\alpha}, \text{aux}; \mathbb{Z}_r)$.

The running time of Decomp$(\alpha, \text{aux}; \mathbb{Z}_q[Y]/\langle E(Y)\rangle)$ is bounded by

$$\begin{aligned} &me \cdot O(\log q) + O((\log r)^2) \cdot poly(m, d) \\ &\leq O(e^2 \cdot (\log q + \log e)^2) \cdot poly(m, d). \end{aligned}$$

The output size of Decomp is bounded by $O(e \cdot (\log e + \log q)) \cdot poly(m, d)$.

Eval$(\boldsymbol{v}, T, \text{aux}; \mathbb{Z}_q[Y]/\langle E(Y)\rangle)$:

1. Output $\mathrm{Eval}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_r)$.

The running time of $\mathrm{Eval}(\boldsymbol{v}, T, \mathrm{aux}; \mathbb{Z}_q[Y]/\langle E(Y)\rangle)$ is bounded by $O(e \cdot (\log e + \log q)) \cdot poly(m, d)$. The output size is bounded by $O(e \cdot (\log e + \log q)) \cdot poly(m, d)$.

$\mathrm{Reconst}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_q[Y]/\langle E(Y)\rangle)$:

1. $\overline{\beta} \leftarrow \mathrm{Reconst}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_r)$ and lift it to $\tilde{\beta} \in \mathbb{Z}$.
2. Let $\tilde{\beta}_0, \cdots, \tilde{\beta}_D \in [M]$ be the digits of $\tilde{\beta}$ written in base M so that $\tilde{\beta} = \sum_{i=0}^{D} \tilde{\beta}_i M^i$.
3. Construct the polynomial $Q(Y) = \sum_{i=0}^{i=D} \tilde{\beta}_i Y^i \in \mathbb{Z}[Y]$. Output $Q(Y) \mod (q, E(Y))$.

The running time of $\mathrm{Reconst}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_q[Y]/\langle E(Y)\rangle)$ is bounded by $poly(m, d, e, \log q)$. □

A.3 Polynomials over Extension Rings with two variables

Finally, we are ready to consider the main theorem. In this case, we denote $R = \mathbb{Z}_q[Y, Z]/\langle E_1(Y), E_2(Z)\rangle$ where E_1, E_2 are monic polynomials with degree e_1, e_2.

Theorem A.4. Let the ring $R := \mathbb{Z}_q[Y, Z]/\langle E_1(Y), E_2(Z)\rangle$ for two monic polynomials $E_1(Y)$ and $E_2(Z)$ with the degree e_1 and e_2 respectively. Let $f(X_1, ..., X_m) \in R[X_1, ..., X_m]$ be a fixed polynomial with $deg_{X_i}(f) < d$. Then, the preprocessing algorithm Prep runs in time

$$S = d^m \cdot poly(m, d, \log(|R|)) \cdot O(m(\log m + \log d + \log\log |R|))^m$$

and outputs a data structure of size at most of S.

The running time of Decomp is bounded by

$$O((e_1 e_2 \cdot (\log q + \log e_1 + \log e_2))^2) \cdot poly(d, m).$$

The running time of Eval is bounded by

$$O((e_1 e_2 \cdot (\log q + \log e_1 + \log e_2))) \cdot poly(d, m).$$

The running time of Reconst is bounded by

$$poly(m, d, e_1, e_2, \log q).$$

Moreover, the output size of Setup is bounded by $O(e_1^2 e_2^2 \cdot (\log q + \log e_1 + \log e_2)) \cdot poly(m, d)$, and the output size of *Decomp* and *Eval* are both bounded by

$$O((e_1 e_2 \cdot (\log q + \log e_1 + \log e_2))) \cdot poly(d, m).$$

Proof. $\mathrm{Setup}(\mathbb{Z}_q[Y, Z]/\langle E_1(Y), E_2(Z)\rangle)$:

1. Let $M := d^m (e_1 e_2 (q-1))^{(d-1)m+1} + 1$, $D_1 := (e_1 - 1)((d-1)m+1)$, $D_2 := (e_2 - 1)((d-1)m+1)$ and let $r := M^{D_2+1}$. Also let $R' = \mathbb{Z}_r[Z]/(Z^{D_2+1} + 1)$.
2. Output $\mathrm{aux} \leftarrow \{\{M^i \mid i \in [e_1]\}, D_1, D_2, r, \mathrm{Setup}(R')\}$.

The size of aux is bounded by

$$O(D_2^2(\log r + \log D_2)) \cdot poly(m,d) + e_1^2 \log M \leq O(e_1^2 e_2^2 \cdot (\log q + \log e_1 + \log e_2)) \cdot poly(m,d).$$

$\mathrm{Prep}(f, \mathrm{aux}; \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle)$:

1. Lift $f \in R[X_1, ..., X_m]$ to $\tilde{f} \in \mathbb{Z}[Y,Z][X_1, ..., X_m]$.
2. Compute $\overline{f} \in R'[X_1, ..., X_m]$ by reducing $\tilde{f}$ modulo the ideal $(r, Y - M, Z^{D_2+1} + 1)$.
3. Output $T \leftarrow \mathrm{Prep}(\overline{f}, \mathrm{aux}; \mathbb{Z}_r[Z]/\langle Z^{D_2+1} + 1\rangle)$.

The output size and the running time are bounded by $S = d^m \cdot poly(m, d, \log|R|) \cdot O(m(\log m + \log d + \log\log|R|))^m$.

$\mathrm{Decomp}(\alpha, \mathrm{aux}; \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle)$:

1. Lift $\alpha \in R^m$ to $\tilde{\alpha} \in \mathbb{Z}[Y,Z]^m$.
2. Compute $\overline{\alpha} \in (R')^m$ from $\tilde{\alpha}$ by reducing modulo the ideal $(r, Y - M, Z^{D_2+1} + 1)$.
3. Output $\mathrm{Decomp}(\overline{\alpha}, \mathrm{aux}; \mathbb{Z}_r[Z]/\langle Z^{D_2+1} + 1\rangle)$.

The running time of $\mathrm{Decomp}(\alpha, \mathrm{aux}; \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle)$ is bounded by

$$\begin{aligned} &O(me_1^2 e_2 \cdot \log q \cdot \log M) + O(D_2^2 \cdot (\log r + \log D_2)^2) \cdot poly(d,m) \\ &\leq O((e_1 e_2 \cdot (\log q + \log e_1 + \log e_2))^2) \cdot poly(d,m). \end{aligned}$$

The output size is bounded by $O(e_1 e_2 \cdot (\log e_1 + \log e_2 + \log q)) \cdot poly(m,d)$.

$\mathrm{Eval}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_2(Z)\rangle)$:

1. Output $\mathrm{Eval}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_r[Z]/\langle Z^{D_2+1} + 1\rangle)$.

The running time of $\mathrm{Eval}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_q[Y,Z]/\langle E_1(Z), E_2(Y)\rangle)$ is bounded by

$$O(e_1 e_2 \cdot (\log e_1 + \log e_2 + \log q)) \cdot poly(d,m).$$

The output size is bounded by

$$O(e_1 e_2 \cdot (\log e_1 + \log e_2 + \log q)) \cdot poly(d,m).$$

$\mathrm{Reconst}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_1(Z)\rangle, e_1, e_2)$:

1. $\overline{\beta} \leftarrow \mathrm{Reconst}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_r[Z]/\langle Z^{D_2+1} + 1\rangle)$ and lift it to $\tilde{\beta} = \sum_{i=0}^{D_1} c_i Z^i \in \mathbb{Z}[Z]$.
2. For each $i \in \{0, ..., D_2\}$, let $c_{i,0}, ..., c_{i,D_1}$ be the digits of c_i written in base M so that $c_i = \sum_{j=0}^{D_1} c_{i,j} M^j$.
3. Construct the polynomial $Q(Y,Z) = \sum_{i=0}^{D_1} \sum_{j=0}^{D_2} c_{i,j} Y^i Z^j$, and output

$$Q(Y,Z) \mod (q, E_1(Y), E_2(Z)).$$

The running time of $\mathrm{Reconst}(\boldsymbol{v}, \mathrm{aux}; \mathbb{Z}_q[Y,Z]/\langle E_1(Y), E_1(Z)\rangle, e_1, e_2)$ is bounded by

$$poly(m, d, e_1, e_2, \log q).$$

□