Towards Practical Doubly-Efficient Private Information Retrieval

Hiroki Okada[2], Rachel Player[1], Simon Pohmann[1], and Christian Weinert[1]

[1] Royal Holloway, University of London, UK
[2] KDDI Research, Japan

Abstract. Private information retrieval (PIR) protocols allow clients to access database entries without revealing the queried indices. They have many real-world applications, including privately querying patent-, compromised credential-, and contact databases. While existing PIR protocols that have been implemented perform reasonably well in practice, they all have suboptimal asymptotic complexities.
A line of work has explored so-called doubly-efficient PIR (DEPIR), which refers to single-server PIR protocols with optimal asymptotic complexities. Recently, Lin, Mook, and Wichs (STOC 2023) presented the first protocol that completely satisfies the DEPIR constraints and can be rigorously proven secure. Unfortunately, their proposal is purely theoretical in nature. It is even speculated that such protocols are completely impractical, and hence no implementation of any DEPIR protocol exists.
In this work, we challenge this assumption. We propose several optimizations for the protocol of Lin, Mook, and Wichs that improve both asymptotic and concrete running times, as well as storage requirements, by orders of magnitude. Furthermore, we implement the resulting protocol and show that for batch queries it outperforms state-of-the-art protocols.

1 Introduction

Private information retrieval (PIR) refers to a subclass of multi-party computation protocols that allow clients to access database entries on a server without revealing the identity of the requested indices. A naive approach to realize PIR would be to have the client download the whole database, but in practice we are interested in protocols that have communication cost sublinear in the database size N. PIR has numerous real-world applications, for example, in anonymous communication [AS16; MOT+11], location privacy [FKP15; KSS08], querying patent databases [Aso04], compromised credential checking [GHPS22], and mobile contact discovery [KRS+19; HSW23; DRRT18].

PIR protocols can be classified into *single-server* and *multi-server* schemes. Multi-server schemes like [BFG06; BIKO12; BI01; BIKR02; Yek08; Efr09; DG16; GI14] tend to be more efficient and can provide information theoretic security guarantees. However, they inherently rely on non-collusion assumptions between servers, which are often unclear how to implement in practice.

As proven in [BIM00], a fundamental issue with single-server PIR schemes is that, without preprocessing, the server computation has to be linear in the

Table 1. Overview of single-server PIR protocols, comparing originally proposed ("classical") with state-of-the-art ("SOTA") protocols with optimal asymptotic cost.

Scheme	Offline (client-ind.)		Offline (client-dependent)			Online		Examples
	Comp.	Storage	Comp.	Comm.	Hint size	Comp.	Comm.	
Stateless PIR								
Classical	$\tilde{O}(N)$	$\tilde{O}(N)$	-	-	-	$\tilde{O}(N)$	$\tilde{O}(\sqrt{N})$	[MBFK16; ACLS18]
SOTA	$\tilde{O}(N)$	$\tilde{O}(N)$	-	-	-	$\tilde{O}(N)$	$\tilde{O}(1)$	[MCR21]
Stateful PIR								
Classical	$\tilde{O}(N)$	$\tilde{O}(N)$	$\tilde{O}(N)$	$\tilde{O}(N)$	$\tilde{O}(\sqrt{N})$	$\tilde{O}(\sqrt{N})$	$\tilde{O}(\sqrt{N})$	[CHK22; ZPSZ24]
SOTA	$\tilde{O}(N)$	$\tilde{O}(N)$	$\tilde{O}(N)$	$\tilde{O}(\sqrt{N})$	$\tilde{O}(\sqrt{N})$	$\tilde{O}(\sqrt{N})$	$\tilde{O}(1)$	[ZLTS23; LMW22]
DEPIR	$\tilde{O}(N)$	$\tilde{O}(N)$	-	-	-	$\tilde{O}(1)$	$\tilde{O}(1)$	[LMW23]

database size N. Intuitively, this should be clear: if the server does not "touch" an element of the database, it would learn that it was not requested. This has motivated variants of PIR that include an offline phase in which clients interact with the server to compute and store "hints" locally, as originally proposed by [CHK22]. Using these hints, they can then perform PIR queries in sublinear time and communication. A long list of work has explored this approach and introduced many tradeoffs between offline and online cost, and progressively gained better practical performance. Generally speaking, schemes like [CK20; CHK22; ZPSZ24] achieve a client-dependent offline phase of cost $O(N)$ that then supports $O(\sqrt{N})$ queries, where each query can be performed in $O(\sqrt{N})$ computational and communication complexity. Another class of schemes [MCR21; PPY18; DPC23; HHC+23] still has linear asymptotic server computation complexity, but very competitive practical running times. On the other hand, there are some schemes [ZLTS23; LMW22] that achieve $O(\text{polylog}(N))$ communication and $O(\sqrt{N})$ computation, again with $O(N)$ preprocessing complexity. This is unavoidable, as otherwise one could perform offline and online phase together, and get a standard stateless PIR protocol with computational complexity $O(N)$.

Doubly-efficient PIR. So far, we presented the landscape of practical and implemented single-server PIR protocols (cf. Table 1). The work of [BIM00] introduced the notion of *(unkeyed) doubly-efficient PIR (DEPIR)*, which avoids client-dependent preprocessing altogether and still provides polylogarithmic online complexities. DEPIR schemes circumvent the impossibility result with client-independent preprocessing, which then only has to be performed once for all clients. However, no construction was presented in [BIM00].

The first DEPIR constructions appeared in [CHR17; BIPW17]. However, their database preprocessing step still depends on a client's key (called *public-key DEPIR*). An additional problem with these protocols is that either they require this key to be kept from the server [CHR17] (and if any client is compromised, all security is lost); or they require non-standard hardness assumptions like ideal obfuscation, which would have to be heuristically instantiated with indistinguishability obfuscation [BIPW17]. Because of this, these works must be considered entirely theoretical, and it seems impossible to give any practical instantiations.

A breakthrough was achieved by Lin, Mook, and Wichs [LMW23], who constructed general *unkeyed DEPIR* from the standard Ring-LWE assumption. Their main tools are BV-style homomorphic encryption (HE) [BV11] and a data structure that can evaluate polynomials in polylogarithmic time w.r.t. their degree, as discovered earlier by [KU11]. Nevertheless, the protocol of [LMW23] is still a theoretical tool and, as with previous work, the heavy-weight building blocks used make the concrete performance worse than the asymptotically suboptimal protocols above. For example, the authors of [ZPSZ24] state that "within the limits of known techniques, we are still very far from making these cryptographic primitives practical (or even implementable)!".

Our contributions. Since we believe that DEPIR is the future, we try to quantify the "very far" in the above statement. In particular, we consider:

How far is doubly-efficient PIR from being practical?

We approach this question by trying to implement the protocol of [LMW23]. As pointed out in previous works, a naive implementation would require an astronomical amount of server-side storage: we estimate it to be at least 10^{78} elements (a few bytes each) for a database size of $N \approx 2^{20}$. As a first step, we propose a less naive implementation, which requires server-side storage of at least $3.0 \cdot 10^{19}$ elements. While much lower, it is clearly still impractical.

We therefore introduce a number of optimizations that significantly improve both the asymptotic and concrete running times as well as the required storage. Compared to the "less naive" implementation, we reduce the storage by more than three orders of magnitude and the running time by almost two orders of magnitude. For this, we exploit the ciphertext ring decomposition of the HE scheme under suitable parameters. Moreover, we manage to improve the storage cost by a further three orders of magnitude by choosing a different polynomial to represent the database. Overall, our implementation must store fewer than $5.3 \cdot 10^{11}$ elements on the server and perform $2.5 \cdot 10^{12}$ queries to these entries in order to answer a single query. Additionally, we use efficient techniques (Fourier transforms and simple modular arithmetic § 6) to efficiently perform all involved computations, to the point where basically all the runtime is spent on storage access.

Our results certainly debunk the second part of the statement, namely that DEPIR is not "even implementable" [ZPSZ24]. More concretely, our implementation of the optimized protocol in Rust runs on commodity hardware within minutes, though for very small database sizes where $N \approx 2^{13}$. For larger database sizes, we find that the main issue is increasing server storage requirements that soon exceed any practically available amount of memory. We conclude that DEPIR is not yet practical for large database sizes.

However, there is one particular setting where our protocol can compare to, and even outperform, the state of the art. In BV-style homomorphic encryption, a suitable choice of parameters induces a "slot-structure" on the plaintext space, which allows single-instruction-multiple-data (SIMD) parallelism. This allows us to encode up to 2^{16} indices within one query, such that PIR will be performed

for each of them. When amortizing costs over all queries, our implementation can compare to unbatched state-of-the-art protocols like [ACLS18; MCR21] in terms of runtime, and outperforms them in terms of communication. Compared to schemes with client-dependent preprocessing like [ZPSZ24], our running times are still inferior, but this is in exchange for superior communication cost.

2 Preliminaries

Notation. We denote the quotient ring $\mathbb{Z}/q\mathbb{Z}$ by $\mathbb{Z}_q$, and the coset of an element $a \in \mathbb{Z}$ in $\mathbb{Z}_q$ by $\overline{a}$. A map that will be used many times in this work is the *shortest-lift map* $\text{lift} : \mathbb{Z}_q \to \mathbb{Z}$ that maps each element $x \in \mathbb{Z}_q$ to its unique representative in $\{-\lfloor (q-1)/2 \rfloor, \ldots, \lceil (q-1)/2 \rceil\}$. Finally, we denote the ℓ_∞-norm over all coefficients of a polynomial $f \in \mathbb{Z}[T_1, ..., T_m]$ by $\|f\|_\infty$.

Homomorphic Encryption. Homomorphic encryption (HE) refers to the family of encryption schemes that allow performing computations on encrypted data without decrypting it. Traditionally, this is achieved by endowing the plaintext and ciphertext spaces with arithmetic structure.

Definition 2.1 (Somewhat homomorphic encryption). *A symmetric encryption scheme* $(\text{Gen}, \text{Enc}(m, \text{sk}), \text{Dec}(\text{ct}, \text{sk}))$ *is called* somewhat homomorphic encryption (SHE)*, if the plaintext space P is a ring and there is an additional operation* $\text{Eval}(f, \text{ct}_1, ..., \text{ct}_k)$ *that outputs a ciphertext encrypting* $f(x_1, ..., x_k)$ *where* $x_i = \text{Dec}(\text{ct}_i, \text{sk})$ *for all "allowed" polynomials* $f \in P[T_1, ..., T_m]$.

Since most HE constructions are built from Ring-LWE [LPR10; SSTX09], the ciphertexts are noisy, and this noise grows during homomorphic operations. If the noise grows too large, then decryption will fail. This leads to the restriction of "allowed" polynomials f. An SHE scheme that supports all polynomials f is called fully homomorphic encryption (FHE) [Gen09]. While it is trivial to build PIR from FHE, classical HE-based PIR schemes [MBFK16; ACLS18; MCR21] are built from SHE for performance reasons.

3 Related Work

The two most important techniques for constructing PIR schemes are HE and the hint sets of [CK20]. We now introduce both, and mention other research related to DEPIR. An overview of state-of-the-art PIR schemes is given in § 1.

3.1 Approaches to PIR

HE-based PIR. The guiding principle of HE-based PIR is to avoid ciphertext-ciphertext multiplications, as they are expensive, both in terms of performance and noise growth. Instead, most schemes [MBFK16; ACLS18; MCR21], and also hint-based PIR schemes like SimplePIR [HHC+23], arrange the server database

entries in form of a matrix $A \in \mathbb{F}_p^{k \times l}$ with $N = kl$. A client who wants to query the (u, v)-th entry of A now encrypts each component of the v-th unit vector e_v to get $(\mathrm{ct}_j)_j$. The server can then compute the "external" matrix-vector product $A(\mathrm{ct}_j)_j = (\mathrm{ct}'_i)_i$, i.e., $\mathrm{ct}'_i = \sum_{j=1}^{l} \mathrm{MulPlain}(A_{ij}, \mathrm{ct}_j)$. The client then finds the desired entry by decrypting ct'_u.

In the earliest work [MBFK16], k was chosen to be 1, with the caveat that the client's query is of size $O(N)$. Later work [ACLS18] focused on compressing the client's query, and [MCR21] introduced a very small number of repetitions of the matrix-multiplication step. The problem with these repetitions is that during later steps, the "database" is the output of the previous step, hence the entries of the matrix are ciphertexts. Thus, ciphertext-ciphertext multiplications are required. However, keeping the number of repetitions very low can decrease the reply size further without blowing up the parameters too much.

Hint-based PIR. Many PIR schemes that perform client-dependent preprocessing to compute and store hints [CK20; CHK22; ZLTS23; LMW22; ZPSZ24] are fundamentally based on [CK20]. This work proposed a nice approach based on $O(\sqrt{N})$ hint sets of size $O(\sqrt{N})$ that is very natural in a two-server setting, but can also be used as another approach to construct single-server PIR schemes with client-dependent preprocessing. However, after preprocessing, the client has to store one hint for each set, hence requires $O(\sqrt{N})$ client-side storage.

3.2 Other related notions

ORAM. The setting of DEPIR is similar to that of oblivious RAM (ORAM) [Gol87]. The main difference is that in ORAM, there is only one client who can read and write to the database. As opposed to DEPIR, practical implementations for ORAM exist [SDS+18; SS13; SSS12; BMP11; DMN11].

Used tools. Apart from the fundamental polynomial preprocessing techniques from [KU11], we also use some techniques that have been developed for improving FHE schemes. In particular, this includes fast algorithms for the residue number system [HPS19; BEHZ16] and Fourier decomposition techniques [GHS12].

4 The DEPIR Construction of [LMW23]

The basic idea of [LMW23] is to use a naive HE-based PIR protocol (shown in Fig. 1) in which the client encrypts the desired index with an HE scheme and the server formulates the database lookup as an arithmetic circuit that it can then evaluate on the encrypted index. This protocol already has very low communication cost for a fixed security parameter, being just polylogarithmic in N. However, it requires the server-side homomorphic evaluation of a polynomial with N monomials, which usually takes time $\Omega(N)$. There are also other more

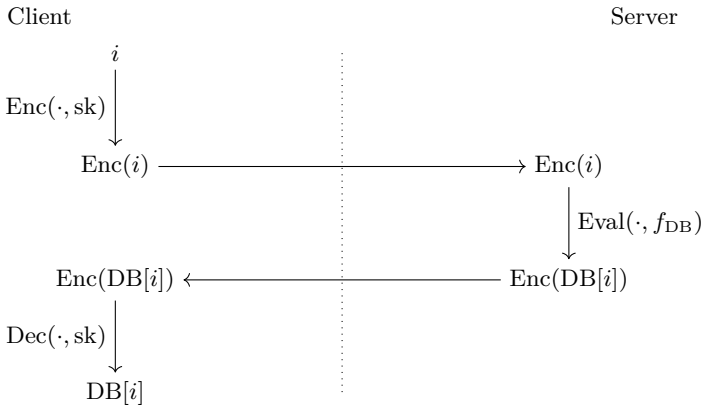

Fig. 1. The most naive HE-based PIR scheme. Here, f_{DB} is a polynomial given by the interpolation of all points $(i, \mathrm{DB}[i])$.

Algorithm 1: High-level overview of the DEPIR scheme.

	Preprocessing	
1	"Split" each index i into "parts" $i_1, ..., i_m$ (cf. § 6).	
2	Interpolate the points $(i_1, ..., i_m, \mathrm{DB}[i])$ into a polynomial $f_{\mathrm{DB}} \in \mathbb{Z}[T_1, ..., T_m]$	
3	Compute the evaluation datastructure D for f_{DB} as shown in § 4.1.	
	Client	**Server**
4	Split the input i into $i_1, ..., i_m$	
5	Encrypt each i_j as $\mathrm{ct}_j = \mathrm{Enc}(i_j)$	
6	**Send** $(\mathrm{ct}_1, ..., \mathrm{ct}_m)$	
7		Compute $\mathrm{ct}' = f_{\mathrm{DB}}(\mathrm{ct}_1, ..., \mathrm{ct}_m)$ using D
8		**Send** ct'
9	**return** $\mathrm{Dec}(\mathrm{ct}')$	

practical obstacles, e.g., the noise growth caused by evaluating a high-degree polynomial (which we mitigate by using multivariate polynomials, cf. § 6).

The groundbreaking idea of [LMW23] is to speed up the homomorphic evaluation by using preprocessing. The first main ingredient is the datastructure from [KU11] that encodes a multivariate polynomial $f \in \mathbb{Z}[T_1, ..., T_m]$ with N monomials and then allows computing evaluations of f in time $O(\mathrm{polylog}(N))$. However, this is not directly applicable to standard HE schemes, as their homomorphic operations cannot be (efficiently) expressed as polynomial evaluations. Hence, we need the second main ingredient, called *algebraic (somewhat) homomorphic encryption (ASHE)* by [LMW23], which is an HE scheme whose homomorphic (arithmetic) operations are the same arithmetic operations over the ciphertext ring. Combining these leads to Algorithm 1.

Algorithm 2: Datastructure query, using one step of reduction.

Input: Point $x = (x_1, ..., x_m) \in \mathbb{Z}_q^m$;
Tables tab_i containing all tuples $(z, f(z)) \in \mathbb{Z}_{p_i}^{m+1}$, where $p_1, ..., p_r$ are primes with $\prod p_i \geq 2N\|f\|_\infty (q/2)^{dm}$.

Output: $f(x)$

Compute $\hat{x} = (\text{lift}(x_1), ..., \text{lift}(x_m)) \in \mathbb{Z}^m$
for $i \in \{1, ..., r\}$ **do**
  Compute $\overline{x}^{(i)} \leftarrow \hat{x} \bmod p_i$
  Query precomputations $f(\overline{x}^{(i)}) := \text{tab}[\overline{x}^{(i)}]$
end
return $\text{lift}(\text{InvCRT}(f(\overline{x}^{(1)}), ..., f(\overline{x}^{(r)}))) \bmod q$

4.1 Preprocessing polynomials

First, we present the datastructure of [KU11]. For the remainder of this section, let $f \in \mathbb{Z}[T_1, ..., T_m]$ be a polynomial of individual degree $\leq d$ in each variable.

The most naive way of providing such a datastructure for fast polynomial evaluations would be to create a table of all tuples $(x, f(x))$ for all points $x = (x_1, ..., x_m)$ over a finite ring R. If $\#R$ is very small, this might already be sufficient. However, as we are interested in larger rings R, we want the datastructure size to be $\text{poly}(N, \log \#R)$ or, even better, $\text{poly}(N, \log\log \#R)$.

The case $R = \mathbb{Z}_q$. For now, assume that $R = \mathbb{Z}_q$. If q is a highly smooth number, we can easily reuse the above technique together with the CRT. In particular, if $q = p_1 \cdots p_r$, we have $\mathbb{Z}_q \cong \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_r}$ and hence, it suffices to store r tables, each containing all the tuples $(x, f(x))$ for $x \in \mathbb{Z}_{p_i}^m$.

The fascinating idea of [KU11] is now that for any q, we can find a surjection

$$\phi_{\text{lift}} : \mathbb{Z}_{p_1 \dots p_r} \twoheadrightarrow \mathbb{Z}_q, \quad x \mapsto \overline{\text{lift}(x)},$$

where $\text{lift}(\cdot)$ refers to the shortest lift function. This map is "sufficiently homomorphic", i.e., homomorphic on small values that do not cause a "wrap around" modulo $p_1 \cdots p_r$. Now we use it to evaluate $f(x_1, ..., x_m)$ with $x_i \in \mathbb{Z}_q$ via $\phi_{\text{lift}}(f(\overline{\text{lift}(x_1)}, ..., \overline{\text{lift}(x_m)})) = f(x_1, ..., x_m)$. For ϕ_{lift} to be sufficiently homomorphic, we require that $p_1 \cdots p_r \geq 2 \max_{|x_i| \leq q/2} |f(x_1, ..., x_r)|$, as this prevents a wrap around $p_1 \cdots p_r$. As this expression is somewhat unwieldy, we will require the stronger bound $p_1 \cdots p_r \geq N\|f\|_\infty (q/2)^{dm}$ instead. Choosing $p_1, ..., p_r$ to be the smallest r primes for a sufficiently large r, one can show using the prime number theorem (PNT) that the resulting database size is $\text{poly}(d, m, \log q)^m$. This method is shown in Algorithm 2.

However, there is no reason to stop here. We can repeat the procedure and reduce each evaluation of f in $\mathbb{Z}_{p_i}$ to r_i' evaluations of f in $\mathbb{Z}_{p'_{ij}}$, as shown in Algorithm 3 and Fig. 2. This results in a storage size depency on $\#R$ of only $\log\log \#R$, at the cost of a higher running time, bounded by $r \cdot \max_i r_i'$. As we will see, this is a runtime change from linear in d to quadratic in d. We could

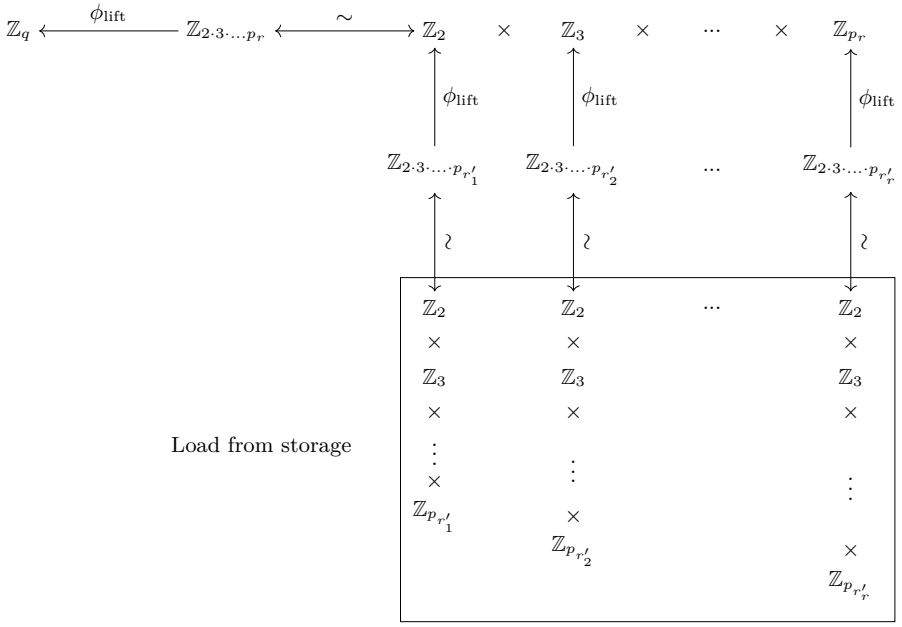

Fig. 2. The basic idea of the two-reduction-steps approach. An evaluation of f in $\mathbb{Z}_q$ is transformed via the lift-map and the CRT-isomorphism into many evaluations modulo small primes, which are precomputed and stored.

continue the process and reduce the dependency on $\#R$, at the cost of increased runtime. However, the current state is sufficient for [LMW23] and for this work.

The general case. This approach does not require the special properties of $\mathbb{Z}$ and can easily be generalized to other settings. In § C, we show how to do so in the case of number fields. However, as we later show how we can improve performance while avoiding going beyond $\mathbb{Z}$ (cf. § 6), we omit the details here. We just remark that [LMW23] explicitly built a "sufficiently homomorphic" map $\mathbb{Z}_Q \twoheadrightarrow \mathbb{Z}_q[X,Y]/(F(X),G(Y))$ for some very large Q and monic polynomials F and G. With this, they prove the following theorem.

Theorem 4.1 ([LMW23, Thm. 2.1], [KU11, Thm. 5.1]). *Let F, G be monic polynomials and R the quotient ring $\mathbb{Z}_q[X,Y]/(F(X),G(Y))$. Let further $f \in R[T_1,...,T_m]$ be a polynomial of individual degree $< d$ in every variable. Then there is an algorithm that, on input f, produces a datastructure of size $S \in \text{poly}(m,d,\log \#R)(dm \log\log \#R)^m$ (in time linear in S). Furthermore, there is an algorithm, that on input $(x_1,...,x_m) \in R^m$ and the above datastructure, computes $f(x_1,...,x_m)$ in time $\text{poly}(d,m,\log \#R)$.*

Algorithm 3: Datastructure query, using two steps of reduction.

Input: Point $x = (x_1, ..., x_m) \in \mathbb{Z}_q^m$;
Tables tab_i containing all tuples $(z, f(z)) \in \mathbb{Z}_{p_i}^{m+1}$, where $p_1, ..., p_r$ "sufficiently many" primes (see Proposition 5.4 for the exact number).

Output: $f(x)$

Compute $\hat{x} = (\text{lift}(x_1), ..., \text{lift}(x_m)) \in \mathbb{Z}^m$
Set $i \leftarrow 1$ and current $\leftarrow 1$
while current $\leq 2N\|f\|_\infty (q/2)^{dm}$ **do**
  Compute $\overline{x}^{(i)} \leftarrow \hat{x} \bmod p_i$
  Call Algorithm 2 with $\overline{x}^{(i)}$ and $\text{tab}_1, ..., \text{tab}_r$ to get the output $f(\overline{x}^{(i)})$
  Update $i \leftarrow i + 1$ and current $\leftarrow$ current $\cdot\, p_i$
end
return $\text{lift}(\text{InvCRT}(f(\overline{x}^{(1)}), ..., f(\overline{x}^{(i-1)}))) \bmod q$

4.2 Algebraic Somewhat Homomorphic Encryption

As mentioned before, we cannot speed up the homomorphic evaluation of standard FHE schemes like BGV/BFV [BGV11; FV12], CKKS [CKKS17], or GSW [GSW13] using the above techniques, as their homomorphic operations are not polynomial evaluations. Therefore, we need a new encryption scheme whose operations are given by polynomials. The authors of [LMW23] call such schemes algebraic somewhat homomorphic encryption (ASHE) schemes.

Definition 4.2 (ASHE). *A somewhat homomorphic encryption (SHE) scheme given by algorithms* $(\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ *is called* algebraic *or* ASHE*, if in addition to the plaintext ring P also the ciphertext space C is a ring and we have* $\text{Eval}(f, \text{ct}_1, ..., \text{ct}_r) = f(\text{ct}_1, ..., \text{ct}_r)$ *for all allowed polynomials f.*

Constructing ASHE. It remains to actually construct an ASHE scheme from the Ring-LWE assumption. As in [LMW23], we use the SHE scheme of Brakerski and Vaikuntanathan [BV11]. The cryptosystem is parameterized by a plaintext modulus t, a ciphertext modulus Q, a cyclotomic ring $R = \mathbb{Z}[X]/(X^n + 1)$ with n a power of two, and a narrow error distribution χ on R. A ciphertext is a polynomial $\text{ct}(Y) \in R_Q[Y]$ and encrypts a message $m \in R_t$ if $\text{lift}(\text{ct}(\text{sk})) \equiv m \mod t$. The scheme is then as follows.

Plaintext ring	$P := R_t = \mathbb{Z}_t[X]/(X^n + 1)$
Ciphertext ring	$C := R_Q[Y] = \mathbb{Z}_Q[X, Y]/(X^n + 1)$
KeyGen()	Output s where $s \leftarrow \$R_Q$
$\text{Enc}(s, m)$	Output $-as + te + \text{lift}(m) + aY$ where $a \leftarrow \$R_Q$, $e \leftarrow \chi$
$\text{Dec}(s, \text{ct})$	Output $\text{lift}(\text{ct}(s)) \bmod t$

Since the scheme is an ASHE scheme, Eval is just polynomial evaluation in the ciphertext ring C. We remark that we can easily make it a finite ring by taking the quotient modulo a monic polynomial $G(Y)$ of sufficient degree, i.e., using $C' := R_Q[Y]/(G(Y))$. The degree of G must be larger than the number of

supported ciphertext-ciphertext multiplications, but this is not very large, as the noise term te grows multiplicatively with each ciphertext-ciphertext multiplication.

Theorem 4.3 ([LMW23, Thm. 3.2]). *The above scheme is a secure ASHE scheme under the Ring-LWE assumption for the chosen parameters n, q, χ.*

5 Precise Performance Estimates

We believe that for designing and evaluating this work, it is very useful to have precise formulas for estimating the performance on concrete parameters. While the behavior of Algorithm 2 and Algorithm 3 can be simulated with a script, this does not scale to very large parameter settings, and gives less intuition on the asymptotic growth. However, the previous asymptotic calculations as presented in Theorem 4.1 do not capture constants, and thus are not sufficient for this purpose. Hence, in this section, we derive suitable formulas. We denote by D the total degree of f so that $N \leq \binom{D+m}{m} \leq D^m$.

Our approach is to estimate all metrics using a precise asymptotic analysis, depending only on a term $o(1)$, and then assume for concrete parameters that $o(1) = 0$. The main results are Proposition 5.3 (for Algorithm 2) and Proposition 5.4 (for Algorithm 3), with proofs based on the PNT. Since the proofs are technical and largely irrelevant for this work, we defer them to the appendix § A.

Lemma 5.1. *An asymptotic expression for the product of all primes less than or equal to B is $\prod_{p \leq B} p = \exp((1 + o(1))B)$.*

Lemma 5.2. *An asymptotic expression for the sum of the m^{th} power of all primes less than or equal to B is $\sum_{p \leq B} p^m = (1 + o(1))\frac{B^{m+1}}{(m+1)\log B}$.*

Proposition 5.3. *Assume that f is a polynomial in m variables of total degree bounded by D with N monomials. Assume further that $m \leq \log(N) \leq D$ and $\|f\|_\infty \in o(q^D)$. Then, in Algorithm 2, the required number of primes r is*

$$(1 + o(1))\frac{D \log q}{\log D + \log\log q}.$$

This implies that the required storage size is $(1 + o(1))^m \frac{D^{m+1} \log(q)^{m+1}}{(m+1)(\log D + \log\log q)}$ and the runtime consists of $(1 + o(1))\frac{D \log q}{\log D + \log\log q}$ random storage accesses.

Proposition 5.4. *Assume that f is a polynomial in m variables of total degree bounded by D with N monomials. Assume further that $m \leq \log(N) \leq D$ and $\|f\|_\infty \in o(q^D)$. Then, in Algorithm 3, the required number of primes r is*

$$(1 + o(1))(\log \|f\|_\infty + D \log D + D \log\log q)/C.$$

Hence, we need storage of size $\frac{(1+o(1))^m}{(m+1)C}(\log(\|f\|_\infty)^{m+1} + D^{m+1} \log(D \log q)^{m+1})$. The runtime consists of $\frac{1+o(1)}{\log(D \log q)C}(D^2 \log q(\log D + \log\log q) + D \log q \log \|f\|_\infty)$ random storage accesses. Here, $C := \log(\log \|f\|_\infty + D \log D + D \log\log q)$ is upper and lower bounded by a nontrivial polynomial in $\log D$, $\log\log q$ and $\log\log \|f\|_\infty$.

To demonstrate that these asymptotic expressions also give a good estimate for concrete parameters, we plot a comparison in Fig. 3. We believe the correlation is good enough to measure the impact of our optimizations.

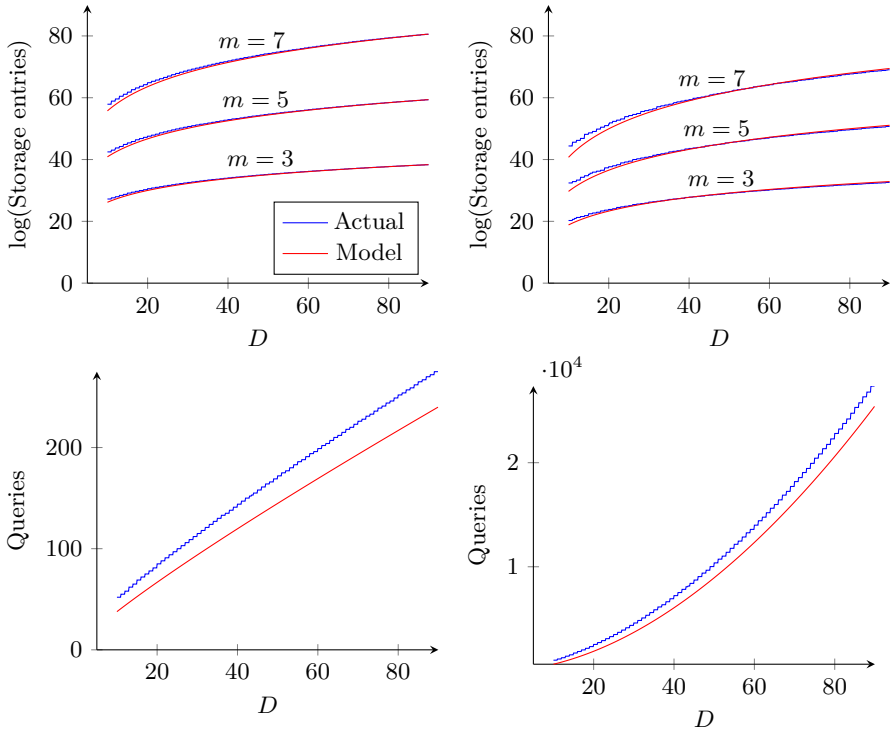

Fig. 3. Comparison of model and actual performance characteristics, assuming $q = 2^{20}$, $\|f\|_\infty = q/2$, and $N = \binom{D+m}{m}$ for Algorithm 2 (left) and Algorithm 3 (right). As the required storage size also depends on the number of variables m of the polynomial, we include plots for different values of m. Actual performance was computed with a SAGE script using the smallest possible values for r.

6 Optimizations and Tradeoffs

In this section, we introduce several optimizations that we combine in Algorithm 4. We illustrate the improvement of these optimizations over the original work [LMW23] in Table 2. We also discuss tradeoffs and the choices made in our implementation, which is a slight variant of the described algorithm that also includes other technical, low-impact optimizations.

Recall that the ciphertext ring is $R = \mathbb{Z}_Q[X, Y]/(X^n + 1, G(Y))$, where Q is the ciphertext modulus, n a power of two, and G an additional monomial of degree $> D = \deg_{\text{total}}(f)$. The plaintext ring is then $P = \mathbb{Z}_t[X]/(X^n + 1)$.

Table 2. Performance characteristics after different optimizations, using the estimates given by Proposition 5.3 and Proposition 5.4. The improvement column gives a lower bound for the improvement factor compared to the "no optimization" case. Parameters are chosen as $N \approx 2^{20}$, $Q \approx 2^{1000}$, and $n = 2^{16}$.

Optimization	m	Queries	Entries	Improvement	
				Queries	Size
One step (i.e. Algorithm 2)					
None [LMW23]	5	$> 6.9 \cdot 10^{11}$	$> 3.3 \cdot 10^{78}$	1	1
+ Fourier decomposition	5	$4.2 \cdot 10^{10}$	$3.4 \cdot 10^{18}$	16.4	$1.0 \cdot 10^{60}$
+ total degree interpolation	5	$1.1 \cdot 10^{10}$	$4.4 \cdot 10^{16}$	62.7	$7.5 \cdot 10^{61}$
	4	$3.1 \cdot 10^{10}$	$7.5 \cdot 10^{14}$	22.3	$4.3 \cdot 10^{63}$
Two steps (i.e. Algorithm 3)					
None (Natural variant of [LMW23])	5	$> 2.1 \cdot 10^{14}$	$> 3.0 \cdot 10^{19}$	1	1
+ Fourier decomposition	5	$3.8 \cdot 10^{12}$	$9.1 \cdot 10^{15}$	55.3	$3.3 \cdot 10^{3}$
+ total degree interpolation	5	$5.0 \cdot 10^{11}$	$7.6 \cdot 10^{13}$	420.0	$3.9 \cdot 10^{5}$
	4	$2.5 \cdot 10^{12}$	$5.3 \cdot 10^{11}$	84.0	$5.7 \cdot 10^{7}$

Concrete performance of the base case. In the original work [LMW23], the approach was to first use "sufficiently homomorphic" maps

$$\mathbb{Z}_{Q''} \twoheadrightarrow \mathbb{Z}_{Q'}[X]/(F(X)) \twoheadrightarrow \mathbb{Z}_{Q}[X,Y]/(F(X), G(Y)) = R$$

given in each case by $y \mapsto \sum a_i X^i$, where $\mathrm{lift}(y) = \sum a_i B^i$. This requires that B is larger than the coefficients can become, i.e., greater than $2(Q/2)^{D+1}\delta_R^{D-1}$ resp. $2(Q'/2)^{D+1}\delta_{\mathbb{Z}[X]/(F)}^{D-1}$, where δ_R is the ring expansion factor.

We will now show how to avoid these maps altogether, and hence are satisfied with the following crude estimates instead of a lengthy and precise analysis. We require $B \geq 2(Q/2)^D \deg(G)^{D-1} \geq 2(Q/2)^D D^{D-1}$, $B' \geq 2(Q'/2)^D n^{D-1}$, $Q'' = F(B') \geq (B')^n$, and $Q = G(B) \geq B^D$. This leads to $Q'' \geq (Q/2)^{D^3 n} D^{(D-1)D^2 n} \cdot n^{(D-1)n} \geq (Q/2)^{D^3 n} D^{(D-1)D^2 n}$. Reasonable parameters for the HE scheme are $Q \approx 2^{1000}$ and $n = 2^{16}$. Furthermore, we choose $m = 5$, as it seems to give a somewhat realistic runtime/speed tradeoff (cf. Fig. 3). This now implies that $d = N^{1/m} = 16$, $D = dm = 80$, and $q = (Q/2)^{D^3 n} D^{(D-1)D^2 n}$, and then the formulas of Proposition 5.3 and Proposition 5.4 give the corresponding values in Table 2.

Fourier decomposition of ciphertexts. As demonstrated above, the increase in the ring size introduced by the "sufficiently homomorphic" maps significantly harms efficiency. Fortunately, this can be avoided altogether when the ciphertext space is a quotient of some ring modulo a product of small prime ideals. This is standard for implementations of certain FHE schemes as it allows much faster arithmetic on ciphertexts (known as double-CRT format [GHS12]). The idea is to choose $Q = p_1 \cdots p_r$ with each $p_i \equiv 1 \bmod 2n$. In this case, the

ideal $(p_i) \subseteq \mathbb{Z}[X]/(X^n+1)$ splits into n degree-1 prime ideals, and so

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

We now propose to choose $G = Y^k - 1$ where $k > D$ is a prime, in which case

$$R \cong \mathbb{Z}_Q[Y]/(Y^k-1) \otimes \mathbb{Z}_Q[X]/(X^n+1) \cong \mathbb{Z}_Q[X]/(X^{nk}+1) \cong \bigoplus_{i=0}^{r} \bigoplus_{j=1}^{nk} \mathbb{F}_{p_i}$$

has a similar decomposition, assuming that every $p_i \mid Q$ satisfies $p_i \equiv 1 \bmod 2nk$. Moreover, this isomorphism can be computed very efficiently using a fast Fourier transform (FFT). It thus suffices to use Algorithm 2 or Algorithm 3 rnk times on much smaller primes $q \in \{p_1, ..., p_r\}$. Primes p with $p \equiv 1 \bmod 2nk$ are relatively common, and for the parameters $N = 2^{20}$, $m = 5$, $n = 2^{16}$, $Q \approx 2^{1000}$, it suffices to choose $k = D + 3 = 83$, $r = 35$, and $\max_i p_i = 2023489537$. This results in the corresponding values in Table 2.

Total degree instead of individual degree. Proposition 5.3 and Proposition 5.4 show that the performance depends only on D and m. Hence, D and m should be as small as possible, while maximizing N.

The choice of f as the polynomial of individual degree d interpolating $N = d^m$ points is suboptimal. Instead, we can choose f as the polynomial of total degree D interpolating $N = \binom{D+m}{m}$ points. This requires a different approach to compute f, which is discussed in § B. For $N = 2^{20}$, we can choose relatively small $m = 5$ and $D = 39$, and still find that $\binom{D+m}{m} \geq 2^{20}$ is large enough. Furthermore, it is sufficient to choose $k = 41$, $r = 35$, and $\max_i p_i = 1493630977$, which gives the corresponding entries of Table 2.

Computing lift(InvCRT(·)) mod q fast. In Algorithm 2 and Algorithm 3, we need to compute the CRT-induced map

$$\mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_r} \xrightarrow{\text{InvCRT}} \mathbb{Z}_{p_1 \cdots p_r} \xrightarrow{\text{lift}} \mathbb{Z} \xrightarrow{\text{mod } q} \mathbb{Z}_q\,.$$

In practice, the product $p_1 \cdots p_r$ will be quite large (thousands of bits), and so arithmetic in $\mathbb{Z}_{p_1 \cdots p_r}$ would require "slow" arbitrary precision integers. Alternatives for this have been proposed in the context of FHE [BEHZ16], and our method is inspired by that work. However, our approach is a variant of these techniques as these usually only give approximate results.

As in [BEHZ16], the idea is to consider $\phi : \mathbb{Z}_{p_1} \times ... \times \mathbb{Z}_{p_r} \to \mathbb{Z}$ which maps $(x_i)_i \mapsto \sum_i \text{lift}\left(x_i \frac{p_i}{P}\right) \frac{P}{p_i}$ for $P = p_1 \cdots p_r$. Clearly $\phi(x_1, ..., x_r) \equiv x_i \bmod p_i$, and furthermore, the images of ϕ are "almost" reduced, i.e., $|\phi(x_1, ..., x_r)| \leq \frac{Pr}{2}$. This implies that we can perform the whole sum directly in modular arithmetic, without being "far" from the result. Concretely, for $\delta \in \{-\lfloor r/2 \rfloor, ..., \lfloor r/2 \rfloor\}$, we

find that $\sum_i \text{lift}\left(x_i \frac{p_i}{P}\right) \frac{P}{p_i} \bmod q$ is equal to $\text{lift}(\text{InvCRT}(x_1, ..., x_r)) \bmod q$ except for an "error term" δP with $|\delta| \leq \lfloor r/2 \rfloor$.

In [BEHZ16], the authors show how to approximate δ up to ± 1 using only modular arithmetic, which suffices for their purpose. We need an exact result, but can also assume that the input is slightly smaller than $P/2$, by choosing the bound $2N\|f\|_\infty (q/2)^D$ in Algorithm 2 and Algorithm 3 slightly higher. More concretely, we assume $|\text{lift}(\text{InvCRT}(X_1, ..., x_r))| < P/(2 + 2\epsilon)$ for some $\epsilon > 0$. In this situation, we can exactly compute δ by using only the highest significant bits of the sum. In other words, we choose $\gamma \geq (\sqrt{1 + \frac{\epsilon}{2r}} - 1)^{-1}$ and compute both

$$R' := \sum_i \text{lift}\left(x_i \frac{p_i}{P}\right) \frac{P}{p_i} \bmod q \quad \text{and} \quad \delta := \left\lfloor \frac{1}{\gamma^2} \sum_i \left\lfloor \frac{\text{lift}\left(x_i \frac{p_i}{P}\right)}{P/\gamma} \right\rfloor \cdot \left\lfloor \frac{\gamma}{p_i} \right\rfloor \right\rceil.$$

The result is then $R' - \delta P \bmod q$. In our implementation, we can choose γ small enough to compute δ using 32-bit integer arithmetic.

Due to its complexity, we did not implement the "naive" base protocol with arbitrary precision integers and thus cannot measure the speedup achieved by this optimization. Nevertheless, we expect it to be significant, and profiling shows that the computational cost of Algorithm 2 resp. Algorithm 3 is extremely low (only storage access time matters).

Size/Runtime tradeoffs. Apart from whether to take one or two reduction steps (as in Algorithm 2 or Algorithm 3, respectively), the choice of m gives us the most important speed/size tradeoff. In [LMW23], m was chosen as $m = \frac{\epsilon}{2} \log(N)/\log\log(N)$, which has the advantage of giving polylogarithmic runtime and a storage size very close to linear (i.e., $O(N^{1+\epsilon})$). However, the constant in the required storage size is still huge, and, in practice, storage larger than a few terabytes is difficult to procure. Hence, it is necessary to choose smaller values for m. For our experiments, $m = 4$ seemed a reasonable tradeoff. Unfortunately, choosing m to be constant significantly increases asymptotic complexity.

Remark 6.1. By Proposition 5.3, if we use one reduction step, the runtime of the online phase of is $knrD \log(q)/(\log D + \log\log q)$, since we execute Algorithm 2 a total of knr times. We note that $knr \log(q) = \log \#R$.

This holds also for other rings R, and we note that the largest factor in the runtime will always be $\log \#R$, even if we use Algorithm 3 to get a storage size depending only on $\log\log \#R$. This is in some sense fundamental, as we have to at least read the whole ciphertext.

Applied to our case, we have at least a cubic runtime dependency on D. This is because clearly $k > D$, and, additionally, we must have $r = \Theta(D)$ to prevent noise overflow in the HE scheme. In other words, the runtime dependency in the one-reduction-step case is $\tilde{O}(N^{3/m})$. Letting $m = \log(N)/\log(\text{polylog}(N))$ would thus give us DEPIR, but since feasible storage constraints force us to take $m \leq 5$, the dependency on N in this setting looks more like $N^{3/4}$ or $N^{3/5}$.

Algorithm 4: The optimized DEPIR scheme.

Input: Index i (**Client**)
The database DB of $N = \binom{D+m}{m}$ entries in $\mathbb{Z}_t$ with $t > D$ (**Server**)
Output: DB$[i]$ (**Client**)

Preprocessing

Compute the total degree interpolation $f_{\mathrm{DB}} \in \mathbb{Z}_t[T_1, ..., T_m]$ as in § 6
Set $f = f_{\mathrm{DB}}^{\mathrm{lift}} \in \mathbb{Z}[T_1, ..., T_m]$
Compute the minimal number r_1 such that $\prod_{i=1}^{r_1} p_i \geq 2\|f\|_\infty N(q/2)^D$
if *two reduction steps are used (i.e. Algorithm 3)* **then**
  Compute the minimal number r such that $\prod_{i=1}^{r} p_i \geq 2\|f\|_\infty N(p_{r_1}/2)^m$
else
  $r := r_1$
end
for $i \in \{1, ..., r\}$ **do**
  Initialize table tab_i
  for $z_1, ..., z_m \in \mathbb{Z}_{p_i}$ **do**
    Compute $f(z_1, ..., z_m)$
    Add $(z_1, ..., z_m, f(z_1, ..., z_m))$ to tab_i
  end
end

Client	Server
$(i_1, ..., i_m) := \sigma(i)$ with σ as in § 6	
Generate a secret key s	
Encrypt $\mathrm{ct}_j = \mathrm{Enc}(i_j, s)$	
Send $(\mathrm{ct}_1, ..., \mathrm{ct}_m)$	
	Set $C_i := \mathrm{ct}_i \bmod (Y^k - 1) \in R$ where $R = \mathbb{Z}_Q[X, Y]/(X^n + 1, Y^k - 1)$
	Use FFT to get $\gamma_{iqj} := \phi(C_i)_{qj}$ where $\phi : R \xrightarrow{\sim} \bigoplus_{q \mid Q \text{ prime}} \bigoplus_{j=1}^{kn} \mathbb{F}_q$
	for $q \mid Q$ *prime* **do**
	for $j \in \{1, ..., kn\}$ **do**
	Use Alg. 2 or 3 to compute $\gamma'_{qj} := f_{\mathrm{DB}}(\gamma_{1qj}, ..., \gamma_{mqj})$
	end
	end
	Use inv. FFT to set $C' := \phi^{-1}((\gamma'_{qj}))$
	Lift C' to get $\mathrm{ct}_{\mathrm{res}} \in \mathbb{Z}_Q[X, Y]/(X^n + 1)$
	Send $\mathrm{ct}_{\mathrm{res}}$
return $\mathrm{Dec}(\mathrm{ct}_{\mathrm{res}}, s)$	

Table 3. Experimental and extrapolated results for databases ranging from "tiny" ($N \approx 2^{13}$) to "large" ($N \approx 2^{28}$). Entries marked by * are extrapolated.

Parameter		tiny	small	medium	large
N		7315	46376	15020334	185250786
Ring degree n		2^{15}	2^{15}	2^{16}	2^{16}
Polynomial degree D		18	30	68	68
Variable number m		4	4	5	6
Ciphertext modulus q (bits)		518	804	1817	1817
# level 1 primes l_1		32	30	145	145
Communication	Query (MB)	10	14.5	75	90
	Reply (MB)	19	54.25	483	483
Used Storage	RAM (GB)	10	36	253	253
	Disk (TB)	0	1.0	873	411744
Performed Queries	RAM ($\cdot 10^9$)	8.7	31.2	1177.3	622.9
	Disk ($\cdot 10^9$)	0	0.9	2086.3	2607.3
Time	Preprocessing (h)	0.2	46.3	$1.3 \cdot 10^7$*	$7.4 \cdot 10^{10}$*
	Runtime (s)	236	1789	$2.1 \cdot 10^6$*	$2.6 \cdot 10^6$*
	Amortized (ms)	7.2	54.6	$32.3 \cdot 10^3$*	$4.0 \cdot 10^3$*

7 Implementation and Evaluation

Due to our optimizations, the running times and required storage sizes were reduced to a level that is manageable for modern hardware. Therefore, we are able to provide the first ever implementation of a DEPIR scheme. In this section, we report on the concrete performance of our implementation.

7.1 Setup and results

Our experiments were run on a machine with an Intel Core i9-10900K, 80GB RAM, and a 4TB PCIe Gen4 SSD. Our implementation exploits multithreading, using the available 16 logical cores both for RAM and Disk reads. For RAM reads, the speedup is close to linear in the number of cores, while the speedup for Disk accesses is sublinear. The code is written in Rust and contains both the polynomial evaluation datastructure as well as the ASHE scheme described in § 4.2. We used the `feanor_math` library for the Cooley-Tuckey and Bluestein FFT algorithms. The results are shown in Table 3.

For large parameter sets, our benchmarking setup does not have sufficient storage to run experiments. Nevertheless, our implementation can count the exact amount of storage accesses it would make in that setting. To obtain the extrapolated runtimes for medium and large database sizes in Table 3, we estimate that our system has RAM read speed of $4 \cdot 10^7$ reads per second and a disk read speed of $1 \cdot 10^6$ reads per second. However, it is likely that if distributed high-performance storage systems are used, higher access speeds are possible.

Amortization. Recall that the plaintext space is $\mathbb{Z}_t[X]/(X^n+1)$, which decomposes into s "slots" $S_j \cong \mathbb{F}_{t^{n/s}}$ that correspond to the factorization of X^n+1 modulo t. Hence, one evaluation at the server side can be used for s queries. Computing the runtime per query results in the listed "amortized time".

Additional parameters in Table 3. Our implementation allows us to "interpolate" between Algorithm 2 and Algorithm 3, i.e., have fine-grained control of whether we perform one or two reduction steps. Concretely, we choose l_0 primes $p_1, ..., p_{l_0}$ and l_1 further primes $p_{l_0+1}, ..., p_{l_0+l_1}$ where the latter are the "level 1 primes". We also introduce one more parameter, the "reply ciphertext modulus". To reduce the reply size, we modulus-switch the result to this smaller modulus before we send it to the client.

Results of Profiling. Profiling shows that the computational workload of our algorithm is very low, and the absolute majority of time is spent on storage access. The reads performed by the algorithm are very small (only one or two bytes) and very random, which is very different from the access patterns usual hardware is optimized for. This is especially the case in the SSD setting. Standard SSD hardware usually has a fixed block size of 4KB, which means that every read request must read at least that much data. Hence, the effective bandwidth utilization is fundamentally limited to at most 2B/4KB = 0.5%. Our implementation comes close to achieving that.

Therefore, modifying the algorithm to read larger chunks from the storage, or finding hardware that better supports many random and small reads might be a practical approach to improving the performance of this algorithm.

7.2 Comparison with other PIR protocols

The total running times of our implementation are much larger than the ones of any other modern protocol in the literature. However, the amortized running times are competitive. We present a comparison in Table 4, which shows that our protocol has the lowest amortized communication costs for a single query. As in SimplePIR [HHC+23], we compare protocols by total database size, i.e., parameters with the same $N \cdot t$, where t is the entry size.

We note that a fair comparison between the protocols is not trivial. In particular, the classical HE-based PIR protocols usually achieve good performance only for "small" databases with very large entries, e.g., $N = 2^{16}$ entries of size 30KB. Clearly, in this setting, communication costs of less than 30KB per query (as for our protocols) are impossible, and in fact the 192KB per query are only a factor of 6.4× higher than the unencrypted baseline. On the other hand, the hint-based approach following [CK20] is usually best suited for very large databases with entries as small as a single bit. The communication cost of these protocols usually scales with $O(\sqrt{N})$, so here the overhead is large, in exchange for fast running times. Our protocol has the natural database entry size given by $t = 65537$, or approximately two bytes.

Table 4. Comparison of amortized timings with other PIR protocols. The values presented are from the corresponding papers, which is why we compare our protocol with different protocols for different parameter sizes. For protocols with offline phase, only the client-dependent offline phase is considered. The entries for our protocol are extrapolated, as in Table 3.

Scheme	Storage	Am. queries	Am. runtime	Am. comm.	N	Entry size
Small DB						
Ours	870 TB	2^{16}	32.3 s	8.9 KB	15020334	2 B
FrodoPIR [DPC23]	-	500	192 ms	271 KB	65536	1 KB
Large DB						
Ours	412 PB	2^{16}	40.0 s	9.1 KB	185250786	2 B
PianoPIR [ZPSZ24]	-	$2.2 \cdot 10^5$	14.5 ms	68.9 KB	134217728	8 B
OnionPIR [MCR21]	-	1	24.9 s	192 KB	65536	30 KB

8 Conclusion and Future Work

As we answer our main question on the practicality of DEPIR with a somewhat unsatisfactory "almost", we want to comment on how future research might make it a "yes!". In our opinion, the polynomial evaluation datastructure is far less inefficient (even including constants) than one might think at first glance. In fact, the runtime dependency on the degree D is linear. The main runtime contribution stems from the linear dependency on $\log q$, or respectively $\log \#R$ (cf. Remark 6.1). However, the total runtime clearly cannot be lower than the length of the representation of a query, which is also $\log \#R$. Therefore, we strongly believe that the way forward is to improve the ASHE scheme.

Namely, the BV-style ASHE scheme needs an incredibly large ciphertext space (many millions of bits) for parameters that support a sufficient amount of multiplications. This is similar to the early state of FHE schemes, but their noise growth and thus ring size has been significantly decreased by techniques like modulus-switching and relinearization. However, these are not algebraic in nature, hence cannot be used to build ASHE schemes. It is an interesting question for further research to construct ASHE schemes with better noise growth and ring size, if this is even possible. If an ASHE scheme with sufficiently small ring size could be found, we believe that this could immediately lead to practical instantiations of DEPIR.

Of course, in addition to the high number of storage accesses, also the size of the storage is an issue. While decreasing it will not directly reduce the runtime, it might for example make it possible to store all data directly in RAM, thus improving performance. From the estimates in § 5, it is clear that, on a theoretical level, this size depends on the number of small prime numbers. Following the generalization to number fields as in § C, it then depends on the number of "small" prime ideals (meaning that $\#(\mathcal{O}_K/\mathfrak{p})$ is small). For example, if the ciphertext ring was a quotient of $\mathcal{O}_K$ with sufficiently many prime ideals $\mathfrak{p}_i$ of norm 2, then we could use the number field analogue of Algorithm 2 to reduce an evaluation to $\mathcal{O}_K/\prod \mathfrak{p}_i \cong \bigoplus \mathbb{F}_2$ – in other words, we would only need to store 2^m tuples (note that this situation is impossible). However, for all the rings traditionally considered in cryptography (polynomial rings, rings

of integers), the number of small prime ideals basically matches the number of small prime numbers in $\mathbb{Z}$ (compare, e.g., the Landau prime ideal theorem). We take this as evidence of a "fundamental mathematical barrier", and believe that significant (asymptotic) optimizations to the polynomial evaluation datastructure are impossible.

Acknoledgments. Simon Pohmann thanks his father Peter Pohmann and his friend Walter Hoos for helpful discussions on high-storage SSD and HDD drives.

References

[ACLS18] S. Angel, H. Chen, K. Laine, S. T. V. Setty. "PIR with Compressed Queries and Amortized Query Processing". In: *S&P*. IEEE Computer Society, 2018, pp. 962–979.

[AS16] S. Angel, S. T. V. Setty. "Unobservable Communication over Fully Untrusted Infrastructure". In: *OSDI*. USENIX Association, 2016, pp. 551–569.

[Aso04] D. Asonov. *Querying Databases Privately: A New Approach to Private Information Retrieval*. Springer, 2004.

[BEHZ16] J. Bajard, J. Eynard, M. A. Hasan, V. Zucca. "A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes". In: *SAC*. Springer, 2016, pp. 423–442.

[BFG06] R. Beigel, L. Fortnow, W. I. Gasarch. "A tight lower bound for restricted pir protocols". In: *Comput. Complex.* 15.1 (2006), pp. 82–91.

[BI01] A. Beimel, Y. Ishai. "Information-Theoretic Private Information Retrieval: A Unified Construction". In: *ICALP*. Springer, 2001, pp. 912–926.

[BIKO12] A. Beimel, Y. Ishai, E. Kushilevitz, I. Orlov. "Share Conversion and Private Information Retrieval". In: *CCC*. IEEE Computer Society, 2012, pp. 258–268.

[BIKR02] A. Beimel, Y. Ishai, E. Kushilevitz, J. Raymond. "Breaking the O(n1/(2k-1)) Barrier for Information-Theoretic Private Information Retrieval". In: *FOCS*. IEEE Computer Society, 2002, pp. 261–270.

[BIM00] A. Beimel, Y. Ishai, T. Malkin. "Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing". In: *CRYPTO*. Springer, 2000, pp. 55–73.

[BMP11] D. Boneh, D. Mazieres, R. A. Popa. "Remote oblivious storage: Making oblivious RAM practical". In: (2011).

[BIPW17] E. Boyle, Y. Ishai, R. Pass, M. Wootters. "Can We Access a Database Both Locally and Privately?" In: *TCC*. Springer, 2017, pp. 662–693.

[BGV11] Z. Brakerski, C. Gentry, V. Vaikuntanathan. "Fully Homomorphic Encryption without Bootstrapping". In: *IACR Cryptol. ePrint Arch.* (2011), p. 277.

[BV11] Z. Brakerski, V. Vaikuntanathan. “Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages”. In: *CRYPTO*. Springer, 2011, pp. 505–524.

[CHR17] R. Canetti, J. Holmgren, S. Richelson. “Towards Doubly Efficient Private Information Retrieval”. In: *TCC*. Springer, 2017, pp. 694–726.

[CKKS17] J. H. Cheon, A. Kim, M. Kim, Y. S. Song. “Homomorphic Encryption for Arithmetic of Approximate Numbers”. In: *ASIACRYPT*. Springer, 2017, pp. 409–437.

[CHK22] H. Corrigan-Gibbs, A. Henzinger, D. Kogan. “Single-Server Private Information Retrieval with Sublinear Amortized Time”. In: *EUROCRYPT*. Springer, 2022, pp. 3–33.

[CK20] H. Corrigan-Gibbs, D. Kogan. “Private Information Retrieval with Sublinear Online Time”. In: *EUROCRYPT*. Springer, 2020, pp. 44–75.

[DMN11] I. Damgård, S. Meldgaard, J. B. Nielsen. “Perfectly Secure Oblivious RAM without Random Oracles”. In: *TCC*. Springer, 2011, pp. 144–163.

[DPC23] A. Davidson, G. Pestana, S. Celi. “FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval”. In: *PoPETS* 2023.1 (2023), pp. 365–383.

[DRRT18] D. Demmler, P. Rindal, M. Rosulek, N. Trieu. “PIR-PSI: Scaling Private Contact Discovery”. In: *PoPETS* 2018.4 (2018), pp. 159–178.

[DG16] Z. Dvir, S. Gopi. “2-Server PIR with Subpolynomial Communication”. In: *J. ACM* 63.4 (2016), 39:1–39:15.

[Efr09] K. Efremenko. “3-query locally decodable codes of subexponential length”. In: *STOC*. ACM, 2009, pp. 39–44.

[FV12] J. Fan, F. Vercauteren. “Somewhat Practical Fully Homomorphic Encryption”. In: *IACR Cryptol. ePrint Arch.* (2012), p. 144.

[FKP15] E. Fung, G. Kellaris, D. Papadias. “Combining Differential Privacy and PIR for Efficient Strong Location Privacy”. In: *SSTD*. Springer, 2015, pp. 295–312.

[Gen09] C. Gentry. “A fully homomorphic encryption scheme”. PhD thesis. Stanford University, USA, 2009.

[GHS12] C. Gentry, S. Halevi, N. P. Smart. “Homomorphic Evaluation of the AES Circuit”. In: *CRYPTO*. Springer, 2012, pp. 850–867.

[GSW13] C. Gentry, A. Sahai, B. Waters. “Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based”. In: *CRYPTO*. Springer, 2013, pp. 75–92.

[GI14] N. Gilboa, Y. Ishai. “Distributed Point Functions and Their Applications”. In: *EUROCRYPT*. Springer, 2014, pp. 640–658.

[Gol87] O. Goldreich. “Towards a Theory of Software Protection and Simulation by Oblivious RAMs”. In: *STOC*. ACM, 1987, pp. 182–194.

[GHPS22] D. Günther, M. Heymann, B. Pinkas, T. Schneider. "GPU-accelerated PIR with Client-Independent Preprocessing for Large-Scale Applications". In: *USENIX Security Symposium*. USENIX Association, 2022, pp. 1759–1776.

[HPS19] S. Halevi, Y. Polyakov, V. Shoup. "An Improved RNS Variant of the BFV Homomorphic Encryption Scheme". In: *CT-RSA*. Springer, 2019, pp. 83–105.

[HHC+23] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, V. Vaikuntanathan. "One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval". In: *USENIX Security Symposium*. USENIX Association, 2023.

[HSW23] L. Hetz, T. Schneider, C. Weinert. "Scaling Mobile Private Contact Discovery to Billions of Users". In: *ESORICS*. Springer, 2023.

[KRS+19] D. Kales, C. Rechberger, T. Schneider, M. Senker, C. Weinert. "Mobile Private Contact Discovery at Scale". In: *USENIX Security Symposium*. USENIX Association, 2019, pp. 1447–1464.

[KU11] K. S. Kedlaya, C. Umans. "Fast Polynomial Factorization and Modular Composition". In: *SIAM J. Comput.* 40.6 (2011), pp. 1767–1802.

[KSS08] A. Khoshgozaran, H. Shirani-Mehr, C. Shahabi. "SPIRAL: A scalable private information retrieval approach to location privacy". In: *MDMW*. IEEE. 2008, pp. 55–62.

[LMW22] W. Lin, E. Mook, D. Wichs. "Near-Optimal Private Information Retrieval with Preprocessing". In: *IACR Cryptol. ePrint Arch.* (2022), p. 830.

[LMW23] W. Lin, E. Mook, D. Wichs. "Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring LWE". In: *STOC*. ACM, 2023, pp. 595–608.

[LPR10] V. Lyubashevsky, C. Peikert, O. Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *EUROCRYPT*. Springer, 2010, pp. 1–23.

[MBFK16] C. A. Melchor, J. Barrier, L. Fousse, M. Killijian. "XPIR : Private Information Retrieval for Everyone". In: *PoPETS* 2016.2 (2016), pp. 155–174.

[MOT+11] P. Mittal, F. G. Olumofin, C. Troncoso, N. Borisov, I. Goldberg. "PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval". In: *USENIX Security Symposium*. USENIX Association, 2011.

[MCR21] M. H. Mughees, H. Chen, L. Ren. "OnionPIR: Response Efficient Single-Server PIR". In: *CCS*. ACM, 2021, pp. 2292–2306.

[PPY18] S. Patel, G. Persiano, K. Yeo. "Private Stateful Information Retrieval". In: *CCS*. ACM, 2018, pp. 1002–1019.

[SDS+18] E. Stefanov, M. van Dijk, E. Shi, T. H. Chan, C. W. Fletcher, L. Ren, X. Yu, S. Devadas. "Path ORAM: An Extremely Simple Oblivious RAM Protocol". In: *J. ACM* 65.4 (2018), 18:1–18:26.

[SS13] E. Stefanov, E. Shi. “ObliviStore: High Performance Oblivious Cloud Storage”. In: *S&P*. IEEE Computer Society, 2013, pp. 253–267.

[SSS12] E. Stefanov, E. Shi, D. X. Song. “Towards Practical Oblivious RAM”. In: *NDSS*. The Internet Society, 2012.

[SSTX09] D. Stehlé, R. Steinfeld, K. Tanaka, K. Xagawa. “Efficient Public Key Encryption Based on Ideal Lattices”. In: *ASIACRYPT*. Springer, 2009, pp. 617–635.

[Yek08] S. Yekhanin. “Towards 3-query locally decodable codes of subexponential length”. In: *J. ACM* 55.1 (2008), 1:1–1:16.

[ZLTS23] M. Zhou, W. Lin, Y. Tselekounis, E. Shi. “Optimal Single-Server Private Information Retrieval”. In: *EUROCRYPT*. Springer, 2023, pp. 395–425.

[ZPSZ24] M. Zhou, A. Park, E. Shi, W. Zheng. “Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation”. In: *S&P*. IEEE Computer Society, 2024.

A Omitted Proofs

Proof of Lemma 5.1. It is well-known that the PNT implies

$$\lim_{B\to\infty}\left(\prod_{p\leq B} p\right)^{\frac{1}{B}} = e$$

Taking logarithms on either side shows then that

$$\log\left(\prod_{p\leq B} p\right)/B = 1 + o(1)$$

and the claim follows. □

Proof of Lemma 5.2. Again using the PNT, we find that

$$\sum_{p\leq B} p^m = o(B^m) + \sum_{0<n\leq B/\log(B)} p_n^m = o(B^{m+1}/\log(B)) + \sum_{0<n\leq B/\log(B)} (n\log n)^m$$

We can estimate this with an integral to get

$$\begin{aligned}
&\sum_{1\le n\le B/\log(B)} (n\log n)^m = O(B^m) + \int_1^{B/\log(B)} (t\log t)^m dt \\
=&O(B^m) + \frac{1}{m+1}(B/\log B)^{m+1}\log(B/\log(B))^m - \int_1^{B/\log B} t^m \log(t)^{m-1} dt \\
=&O(B^m) + \frac{B^{m+1}(\log B - \log\log B)^m}{(m+1)\log(B)^{m+1}} - \int_1^{B/\log B} O(t^m\log(t)^{m-1}) dt \\
=&O(B^m) + \frac{B^{m+1}}{(m+1)\log B} + O((B/\log B)^{m+1}\log(B/\log(B))^{m-1}) \\
=&\frac{B^{m+1}}{(m+1)\log B} + O(B^{m+1}/\log(B)^2) = (1+o(1))\frac{B^{m+1}}{(m+1)\log B}
\end{aligned}$$

□

Proof of Proposition 5.3. We need to find primes $p_1, ..., p_r$ such that $\prod p_i \ge 2N\|f\|_\infty (q/2)^D$. Using Lemma 5.1, we find that the set of primes $p_i \le B$ with

$$\begin{aligned}
B &= \frac{1}{1+o(1)}\log(2N\|f\|_\infty (q/2)^D) \\
&= (1+o(1))(\log N + \log\|f\|_\infty + D\log q) = (1+o(1))(\log\|f\|_\infty + D\log q) \\
&= (1+o(1))D\log q
\end{aligned}$$

is sufficient.

Now, by the prime number theorem, there are $\pi(B) = (1+o(1))B/\log B$ primes $\le B$. This gives

$$(1+o(1))\frac{D\log q}{\log(D\log q)} = (1+o(1))\frac{D\log q}{\log D + \log\log q}$$

The number of queries is clearly equal to that number.

Furthermore, the precomputations consist of that many tables, with size ranging from p_1^m to p_r^m. By Lemma 5.2, we get

$$\begin{aligned}
\sum_{p\le B} p^m =&(1+o(1))^m \frac{(D\log q)^{m+1}}{(m+1)\log(D\log q)} \\
=&(1+o(1))^m \frac{D^{m+1}\log(q)^{m+1}}{(m+1)(\log D + \log\log q)}
\end{aligned}$$

□

Proof of Proposition 5.4. We already know from Proposition 5.3 that after the first reduction, we are left with

$$(1+o(1))\frac{D\log q}{\log D + \log\log q}$$

primes of size at most

$$(1+o(1))D\log q$$

Applying Lemma 5.1 again, we see that the second-level primes are bounded by

$$\begin{aligned} B =&(1+o(1))\log\left(2N\|f\|_\infty\left((1+o(1))D\log q/2\right)^D\right)\\ =&(1+o(1))\left(\log N+\log\|f\|_\infty+D\log\left(D\log q\right)\right)\\ =&(1+o(1))\left(\log N+\log\|f\|_\infty+D\log D+D\log\log q\right)\\ =&(1+o(1))\left(D\log D+D\log\log q+\log\|f\|_\infty\right)\end{aligned}$$

Using the prime number theorem to estimate $\pi(B)$, we find that the number of second-level primes is

$$(1+o(1))\frac{D\log D+D\log\log q+\log\|f\|_\infty}{\log\left(D\log D+D\log\log q+\log\|f\|_\infty\right)}$$

The total number of queries is now the product

$$(1+o(1))\frac{D\log D+D\log\log q+\log\|f\|_\infty}{\log(D\log D+D\log\log q+\log\|f\|_\infty)}\cdot\frac{D\log q}{\log D+\log\log q}$$

We find that

$$\frac{D\log D+D\log\log q+\log\|f\|_\infty}{\log D+\log\log q}=\frac{D\log(D\log q)+\log\|f\|_\infty}{\log(D\log q)}$$

and the expression for the number of queries follows. Finally, we again use Lemma 5.2 and find the required storage size

$$(1+o(1))^m\frac{\log(\|f\|_\infty)^{m+1}+D^{m+1}\log(D\log q)^{m+1}}{(m+1)\log(\log\|f\|_\infty+D\log(D\log q))}$$ □

B Computing the total degree interpolation polynomial

In the original work [LMW23], a Fourier transform-based method was used to compute the polynomial f from the database entries via interpolation, with asymptotic complexity only $O(N\log N)$. This approach is not compatible with the total degree optimization we introduced in § 6. However, there still is a faster approach than using a generic algorithm to solve the linear system

$$\forall i\leq\binom{D+m}{m}:\quad\sum_{\substack{I\in\{0,\ldots,D\}^m\\ \sum I_j\leq D}}X_I\prod_j x_{ji}^{I_j}=y_i$$

given points $(x_{1i},\ldots,x_{mi},y_i)$. In particular, we can choose the interpolation points $(x_{1i},\ldots,x_{mi})$ with $i\leq\binom{D+m}{m}$ to be the set of points

$$\left\{(z_{0i_1},\ldots,z_{mi_m})\in\mathbb{F}_t^m\;\middle|\;\sum_j i_j\leq D\right\}$$

for arbitrary distinct values $z_{j0}, ..., z_{jD} \in \mathbb{F}_t$ for each j. In this case, we can arrange the rows and columns of the corresponding matrix to have a block form. More precisely, we consider the bijection

$$\sigma : \left\{ I \in \{0, ..., D\}^m \;\middle|\; \sum_j I_j \leq D \right\} \xrightarrow{\sim} \left\{ 1, ..., \binom{D+m}{m} \right\}$$

induced by the enumeration of the tuples I in lexicographic order. The matrix is given as $A = (a_{ij})$ with $a_{ij} = \prod_{l=1}^{m} z_{l,\sigma^{-1}(i)_l}^{\sigma^{-1}(j)_l}$. All blocks have different size, but it is still possible to adjust the explicit formula for the entries of the LU-decomposition of the Vandermonde matrix to this setting, which then gives an explicit $O(N^2)$-algorithm for inverting the matrix. Depending on the choice of m, n, and Q, this can dominate the asymptotic preprocessing time. In practice, it is absolutely negligible.

C The reduction step for number fields

The idea described in § 4.1 does not require the special properties of $\mathbb{Z}$, but can be generalized. We consider a natural generalizations to number fields, as these already have a "natural" norm and their quotients yield all finite fields.

In detail, we replace the map

$$\mathbb{Z}_{p_1 \cdots p_r} \twoheadrightarrow \mathbb{Z}_q, \quad x \mapsto \overline{\mathrm{lift}(x)}$$

with

$$\mathcal{O}_K/J \twoheadrightarrow \mathcal{O}_K/I, \quad x \mapsto \overline{\mathrm{lift}(x)}$$

where $\mathcal{O}_K$ is the ring of integers in a number field K, and $\mathrm{lift}(\cdot)$ refers to any one of the shortest representatives $a \in \mathcal{O}_K$ of some coset $a + J$ w.r.t. a norm $\|\cdot\|$ on K. Usually, we would take the canonical norm $\|\cdot\|$ on K, which is given by

$$\|x\| := \sqrt{\sum_{\sigma \in \mathrm{Hom}(K,\mathbb{C})} |\sigma x|^2}.$$

Doing this, the constraint $p_1 \cdots p_r \geq 2N\|f\|_\infty (q/2)^D$ that ensures that the computations do not wrap around $p_1 \cdots p_r$ in the scalar case has to be replaced by

$$\lambda(J) \geq 2N\|f\|_\infty \mu(I)^D.$$

Here, $\lambda(J)$ stands for the length of a shortest nonzero element of J, and $\mu(I)$ stands for the maximal distance of an element from I, i.e.,

$$\lambda(J) = \min_{x \in J \setminus \{0\}} \|x\| \quad \text{and} \quad \mu(I) = \sup_{x \in K} \min_{a \in I} \|x - a\|.$$

Note now that if we choose $J = \mathfrak{p}_1 \cdots \mathfrak{p}_r$ a product of prime ideals $\mathfrak{p}_i \leq \mathcal{O}_K$ over primes p_i that split completely in $\mathcal{O}_K$, then each $\mathcal{O}_K/\mathfrak{p}_i \cong \mathbb{F}_{p_i}$. In particular, this implies that

$$\mathcal{O}_K/J = \bigoplus_{i=1}^{r} \mathbb{F}_{p_i} = \mathbb{Z}_{p_1 \cdots p_r}.$$

This naturally allows us to reduce the evaluation in finite rings of the form $\mathbb{Z}_q[X]/(F(X))$ to prime fields $\mathbb{F}_p$ using exactly the same approach (i.e., Algorithm 2). Thus, this implies Theorem 4.1 for a suitable choice of parameters.