Barely Doubly-Efficient SimplePIR

Keewoo Lee

Ethereum Foundation
keewoo.lee@ethereum.org

Abstract. A Private Information Retrieval (PIR) scheme allows a client to retrieve data from a database hosted on a remote server without revealing which location is being accessed. In Doubly-Efficient PIR (DEPIR), the server preprocesses the database offline into a data structure that enables it to answer any client query in sublinear time with respect to the database size N. The breakthrough work of Lin-Mook-Wichs (STOC'23) presented the first DEPIR construction from the Ring-LWE assumption. This remains essentially the only known construction to date, and realizing DEPIR from alternative foundations, particularly from plain LWE, has remained elusive.

In this work, we present a simple LWE-based construction of DEPIR in the CRS model. Our construction is inspired by SimplePIR (USENIX'23) and leverages the matrix–vector multiplication preprocessing of Williams (SODA'07). While our construction is only barely doubly-efficient, with server computation of $o(N/\log N)$, it was previously unknown whether even such modest sublinear efficiency could be achieved from unstructured, plain LWE.

1 Introduction

In (single-server) Private Information Retrieval (PIR) [CGKS95, KO97], a client wishes to retrieve an entry from a remote database maintained by an untrusted server, while ensuring that the query index remains private. A classic lower bound for PIR, due to Beimel-Ishai-Malkin [BIM00], states that the server must perform at least $\Omega(N)$ computation, where N is the size of the database. Otherwise, some entries of the database would be left untouched during computation, thereby compromising query privacy. This lower bound, however, assumes that the server accesses the database *as-is*.

In the same paper, Beimel-Ishai-Malkin [BIM00] demonstrated that sublinear server-side computation is possible if *preprocessing* is allowed. However, their construction is in the non-colluding multi-server setting, and they left open the question of whether such efficiency could be achieved in the single-server setting with preprocessing. This primitive was later termed Doubly-Efficient PIR (DEPIR) [CHR17]. The problem remained open for over two decades and was often regarded as too good to be true. Notably, [BIPW17] even devotes part of its discussion to explaining why proving the impossibility of DEPIR seems unlikely.

This changed with the breakthrough of Lin-Mook-Wichs [LMW23], which presented the first construction of DEPIR, based on the Ring-LWE (RLWE) assumption [LPR10], achieving $\mathsf{polylog}(N)$ server-side computation. While several optimizations were proposed [DMZ23, OPPW24, OPPW25], this remains essentially the only known construction to date, and realizing DEPIR from alternative foundations remains an open problem. In particular, constructing DEPIR from plain LWE [Reg05] is still open. While RLWE is widely regarded as a *de facto*[1] standard assumption, it remains intriguing whether the ring structure—central to the construction of [LMW23]—is truly necessary for DEPIR.

sk-DEPIR. The situation remains largely the same even when considering the weaker primitive sk-DEPIR (Definition 2.2), a keyed variant of DEPIR. All known sk-DEPIR constructions, except the one implied by [LMW23], rely on non-standard, code-based assumptions. Earlier works [BIPW17, CHR17] achieve $\mathsf{polylog}(N)$ server computation, but their security relies on ad hoc assumptions involving secretly permuted Reed–Muller codes. Although these assumptions have been examined in subsequent works [BHW19, BHMW21, BW21], their hardness remains unclear. More recent work [CIMR25] constructs sk-DEPIR from the Learning Subspace with Noise (LSN) assumption, introduced in [DKL09] in the context of leakage-resilient cryptography. While LSN can be viewed as a less structured variant of the earlier assumptions, the resulting scheme is only *barely* doubly-efficient, achieving server-side computation of $O(N/\log N)$.

1.1 Our Contribution

In this work, we present a DEPIR construction based on plain LWE in the Common Random String (CRS) model—arguably much simpler than that of [LMW23] and largely overlooked by the community. The CRS in our scheme can be readily compiled into a key for a keyed DEPIR scheme (Remark 2.1), yielding an LWE-based *keyed DEPIR with public preprocessing* in the standard model. We note that the notion of keyed DEPIR with public preprocessing is stronger than—that is, it implies—both sk-DEPIR and pk-DEPIR (Definition 2.2). While our constructions are only *barely* doubly-efficient, with server computation of $O(N/\log^2 N)\cdot\mathsf{polyloglog}(N)$, it was previously unknown whether even such modest sublinear efficiency could be achieved from unstructured, plain LWE.

[1] While RLWE enjoys an average-case to worst-case reduction due to [LPR10], the reduction is to lattice problems over *ideal lattices*, a class of structured lattices, whereas plain LWE reduces to problems over general lattices. However, despite over a decade of cryptanalytic efforts, no attack has been found that performs substantially better against RLWE than against plain LWE. (A line of work [ELOS15, CIV16, Pei16] shows certain instantiations of RLWE can be easily broken, but these examples are somewhat contrived.) On the other hand, we also note that, although it represents a minority opinion, some in the community express reservations about the long-term security of RLWE due to its additional algebraic structure [Ber14].

1.2 Technical Overview

In retrospect, the high-level template of [LMW23] is conceptually quite simple. The construction begins with the somewhat homomorphic encryption (SHE) scheme of Brakerski-Vaikuntanathan [BV11]. A distinctive feature of [BV11], compared to more modern SHE constructions such as [BGV12], is that the homomorphic evaluation of a polynomial on ciphertexts corresponds to the evaluation of some polynomial over the ciphertexts themselves. This property, referred to in [LMW23] as Algebraic SHE (ASHE), is central to their approach. By encoding the database as a polynomial, they are then able to leverage the fast polynomial evaluation with preprocessing due to Kedlaya-Umans [KU08] to achieve sublinear server-side computation.

Our construction resembles that of [LMW23], but instantiates it with SimplePIR [HHC+23] (Section 2.4) in place of [BV11], and employs matrix-vector multiplication preprocessing due to Williams [Wil07] (Section 2.5) instead of [KU08]. SimplePIR can be viewed as an instance of *algebraic linearly homomorphic encryption (ALHE)*, analogous to ASHE, where the homomorphic evaluation of a linear transformation on given ciphertexts itself corresponds to some linear transformation over the ciphertexts. After encoding the database as a matrix, we apply the result of [Wil07], which enables $m \times m$ matrix-vector multiplication to be performed in $O(m^2/\log^2 m)$ time, after preprocessing the matrix. The only technical caveat is that the result of [Wil07] in its original form depends too heavily on the base ring size for our purposes. However, this can be easily addressed using the Chinese Remainder Theorem (see Lemma A.1).

Role of the CRS. While SimplePIR [HHC+23] (Section 2.4) introduces CRS for the sake of concrete efficiency, in our (unkeyed) DEPIR scheme, the CRS is essential for achieving sublinear server-side online computation. Without assuming CRS, the server would need to compute $(\widehat{\boldsymbol{D}}\boldsymbol{A}, \widehat{\boldsymbol{D}}\boldsymbol{v})$ from the LWE ciphertext $(\boldsymbol{A}, \boldsymbol{v}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$, where $\widehat{\boldsymbol{D}} \in \mathbb{Z}^{m\times m}$ is the encoded database. Here, n—a parameter of the underlying LWE assumption—is typically assumed to be $n = \mathsf{poly}(\lambda, \log(1/\alpha))$, with no stronger assumption, where $1/\alpha$ denotes the target modulus-to-noise ratio (Definition 2.4), which in our case is $\mathsf{poly}(N)$. As $n = \mathsf{polylog}(N)$, computing $\widehat{\boldsymbol{D}}\boldsymbol{A}$ online would nullify the $O(\log^2 N)$-factor savings afforded by [Wil07]. Therefore, we follow SimplePIR and treat $\boldsymbol{A}$ as a CRS, allowing the server to precompute $\widehat{\boldsymbol{D}}\boldsymbol{A}$ prior to the online phase.[2]

2 Preliminaries

2.1 Notations

We use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. The floor function is denoted by $\lfloor \cdot \rfloor$, and for $x \in \mathbb{R}$, we define the rounding function $\lfloor x \rceil$ as $\lfloor x + \frac{1}{2} \rfloor$. Whenever appli-

[2] Other possible, though less appealing, ways to mitigate this issue include: (a) assuming n can be set to $O_\lambda(\log^{2-\varepsilon}(1/\alpha))$ for some constant $\varepsilon > 0$, or (b) considering a *batched* setting as in [DMZ23], where the cost of computing $\widehat{\boldsymbol{D}}\boldsymbol{A}$ can be amortized across n queries using the batching technique from [PVW08].

cable, we identify $\mathbb{Z}_p$ with the set $\{0, 1, \cdots, p-1\} \subseteq \mathbb{Z}$. The security parameter is denoted by λ. Whenever the terms *negligible* or *overwhelming* are used, they are understood to be with respect to the security parameter λ, even when not stated explicitly.

2.2 Doubly-Efficient Private Information Retrieval

We begin by defining (unkeyed) DEPIR, following [LMW23].

Definition 2.1 (DEPIR). *A* Doubly-Efficient Private Information Retrieval (DEPIR) *scheme consists of a tuple of probabilistic algorithms* $(\mathtt{Setup}, \mathtt{Prep}, \mathtt{Query}, \mathtt{Answer}, \mathtt{Extract})$*, each described below.*

- $\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$*: The setup algorithm takes as input the security parameter* λ *and database size* N*, and outputs the public parameter* $\mathtt{pp}$*. All subsequent algorithms take* $\mathtt{pp}$ *as an implicit input.*
- $\widetilde{\mathtt{DB}} \leftarrow \mathtt{Prep}(\mathtt{DB})$*: The preprocessing algorithm takes as input a database* $\mathtt{DB} \in \{0,1\}^N$ *and outputs a preprocessed database* $\widetilde{\mathtt{DB}}$*.*
- $(\mathtt{q}, \mathtt{st}) \leftarrow \mathtt{Query}(i)$*: The query algorithm takes as input a query index* $i \in [N]$ *and outputs a query* $\mathtt{q}$ *along with a secret state* $\mathtt{st}$*.*
- $\mathtt{a} \leftarrow \mathtt{Answer}(\widetilde{\mathtt{DB}}, \mathtt{q})$*: The answer algorithm takes as input a preprocessed database* $\widetilde{\mathtt{DB}}$ *(stored in random-access memory) and a query* $\mathtt{q}$*, and outputs an answer* $\mathtt{a}$*.*
- $\mathtt{b} \leftarrow \mathtt{Extract}(\mathtt{st}, \mathtt{a})$*: The extract algorithm takes a secret state* $\mathtt{st}$ *and an answer* $\mathtt{a}$ *as input, and outputs a bit* $\mathtt{b} \in \{0,1\}$*.*

The algorithms should satisfy the following correctness, privacy, and efficiency.

***Correctness*:** *For any* $\mathtt{DB} \in \{0,1\}^N$ *and any* $i \in [N]$*, the following probability is overwhelming for all* $\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$*.*

$$\Pr\left[\mathtt{Extract}(\mathtt{st}, \mathtt{a}) = \mathtt{DB}[i] \;\middle|\; \begin{array}{r} \widetilde{\mathtt{DB}} \leftarrow \mathtt{Prep}(\mathtt{DB}) \\ (\mathtt{q}, \mathtt{st}) \leftarrow \mathtt{Query}(i) \\ \mathtt{a} \leftarrow \mathtt{Answer}(\widetilde{\mathtt{DB}}, \mathtt{q}) \end{array}\right]$$

***Privacy*:** *For any* $i_0, i_1 \in [N]$*, no probabilistic polynomial-time adversary can distinguish between* $\mathtt{Query}(i_0)$ *and* $\mathtt{Query}(i_1)$ *with non-negligible advantage, for an overwhelming fraction of* $\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$*.*

***Efficiency*:** *The* $\mathtt{Prep}$ *algorithm runs in* $\mathsf{poly}(\lambda, N)$ *time, while the* $\mathtt{Query}$*,* $\mathtt{Answer}$*, and* $\mathtt{Extract}$ *algorithms each run in* $o(N) \cdot \mathsf{poly}(\lambda)$ *time.*

We next define *keyed DEPIR*, following [BIPW17, CHR17]. The syntax for keyed DEPIR is the same as that of (unkeyed) DEPIR, except for the following modifications: (a) a key generation algorithm $\mathtt{KeyGen}$ is added, which outputs a key $\mathtt{k}$, and (b) a key $\mathtt{k}$ is provided as an additional input to $\mathtt{Prep}$, $\mathtt{Query}$, and $\mathtt{Extract}$. For keyed DEPIR, we consider three levels of privacy: sk-DEPIR, pk-DEPIR, and keyed DEPIR with public preprocessing.

Definition 2.2 (Keyed DEPIR). *A* Keyed DEPIR *scheme consists of a tuple of probabilistic algorithms* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Prep}, \mathtt{Query}, \mathtt{Answer}, \mathtt{Extract})$ *with the following syntax.*

- $\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$
- $\mathtt{k} \leftarrow \mathtt{KeyGen}(\mathtt{pp})$
- $\widetilde{\mathtt{DB}} \leftarrow \mathtt{Prep}(\mathtt{k}, \mathtt{DB})$
- $(\mathtt{q}, \mathtt{st}) \leftarrow \mathtt{Query}(\mathtt{k}, i)$
- $\mathtt{a} \leftarrow \mathtt{Answer}(\widetilde{\mathtt{DB}}, \mathtt{q})$
- $\mathtt{b} \leftarrow \mathtt{Extract}(\mathtt{k}, \mathtt{st}, \mathtt{a})$

The algorithms should satisfy the following correctness, privacy, and efficiency.

***Correctness*:** *For any* $\mathtt{DB} \in \{0,1\}^N$ *and any* $i \in [N]$*, the following probability is overwhelming for all* $\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$ *and* $\mathtt{k} \leftarrow \mathtt{KeyGen}(\mathtt{pp})$*.*

$$\Pr\left[\mathtt{Extract}(\mathtt{k}, \mathtt{st}, \mathtt{a}) = \mathtt{DB}[i] \;\middle|\; \begin{array}{r} \widetilde{\mathtt{DB}} \leftarrow \mathtt{Prep}(\mathtt{k}, \mathtt{DB}) \\ (\mathtt{q}, \mathtt{st}) \leftarrow \mathtt{Query}(\mathtt{k}, i) \\ \mathtt{a} \leftarrow \mathtt{Answer}(\widetilde{\mathtt{DB}}, \mathtt{q}) \end{array}\right]$$

***Privacy*:** *No probabilistic polynomial-time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *has a non-negligible advantage in the following game, where the adversary's advantage is defined as* $\Pr[b' = b] - 1/2$*.*

1. $b \leftarrow \{0,1\}$*: The challenger samples a random bit.*
2. $\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$*: The challenger sets up public parameter* $\mathtt{pp}$*.*
3. $\mathtt{k} \leftarrow \mathtt{KeyGen}(\mathtt{pp})$*: The challenger generates key* $\mathtt{k}$*.*
4. $(i_0, i_1, \mathtt{aux}) \leftarrow \mathcal{A}_1^{\mathtt{Query}(\mathtt{k},\cdot)}(\mathtt{pp})$*: The adversary chooses challenge indices* $i_0, i_1 \in [N]$ *and outputs auxiliary information* $\mathtt{aux}$*, given oracle access to* $\mathtt{Query}(\mathtt{k}, \cdot)$*.*
5. $(\mathtt{q}, \mathtt{st}) \leftarrow \mathtt{Query}(\mathtt{k}, i_b)$*: The challenger generates a challenge query* $\mathtt{q}$ *corresponding to the challenge index* i_b*.*
6. $b' \leftarrow \mathcal{A}_2^{\mathtt{Query}(\mathtt{k},\cdot)}(\mathtt{aux}, \mathtt{q})$*: The adversary, given the auxiliary information* $\mathtt{aux}$*, the challenge query* $\mathtt{q}$*, and oracle access to* $\mathtt{Query}(\mathtt{k}, \cdot)$*, outputs a guess* $b' \in \{0,1\}$*.*

We say that a keyed DEPIR scheme is an sk-DEPIR *scheme if the adversary is given only oracle access to* $\mathtt{Query}(\mathtt{k}, \cdot)$*, as in the security game described above. If the adversary is also given the key* $\mathtt{k}$*, the scheme is called a* pk-DEPIR *scheme. Finally, if the adversary is given the randomness used in* $\mathtt{KeyGen}$ *in addition to the key, the scheme is referred to as* keyed DEPIR with public preprocessing*.*

***Efficiency*:** *The* $\mathtt{Prep}$ *algorithm runs in* $\mathsf{poly}(\lambda, N)$ *time, while the* $\mathtt{Query}$*,* $\mathtt{Answer}$*, and* $\mathtt{Extract}$ *algorithms each run in* $o(N) \cdot \mathsf{poly}(\lambda)$ *time.*

It is straightforward that a keyed DEPIR scheme with public preprocessing is a pk-DEPIR scheme, and a pk-DEPIR scheme is an sk-DEPIR scheme. We also have the following equivalence.

Remark 2.1. An unkeyed DEPIR scheme in the CRS model implies a keyed DEPIR scheme with public preprocessing in the standard model, and vice versa, with both having the same online efficiency. This equivalence holds because the CRS in the unkeyed scheme can simply be interpreted as the key in the keyed setting, and we can append the key to the output of `Prep`.

2.3 Learning with Errors

The Learning with Errors (LWE) assumption [Reg05] is defined as follows.

Definition 2.3 (LWE). *Given dimension n, number of samples m, modulus q, and error distribution χ over $\mathbb{Z}$, the $\mathsf{LWE}_{n,m,q,\chi}$ assumption states that no probabilistic polynomial-time adversary can distinguish between the following two distributions with non-negligible advantage.*

- $(\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$, *where* $\boldsymbol{A} \leftarrow \mathbb{Z}_q^{m \times n}$, $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$, *and* $\boldsymbol{e} \leftarrow \chi^m$
- $(\boldsymbol{A}, \boldsymbol{r})$, *where* $\boldsymbol{A} \leftarrow \mathbb{Z}_q^{m \times n}$ *and* $\boldsymbol{r} \leftarrow \mathbb{Z}_q^m$

We also formally define the *LWE with polynomial modulus-to-noise ratio* assumption, in a manner similar to the *Ring-LWE with quasi-polynomial approximation factor* assumption defined in [LMW23]. We say that a distribution χ over $\mathbb{Z}$ is *B-bounded* if $|e| \leq B$ holds with overwhelming probability for $e \leftarrow \chi$.

Definition 2.4. *We refer to the following as the* LWE with polynomial modulus-to-noise ratio *assumption:*

> *For any security parameter λ and any polynomial modulus-to-noise ratio $1/\alpha = \mathsf{poly}(\lambda)$, one can, in deterministic $\mathsf{poly}(\lambda, \log(1/\alpha))$ time, find parameters $n = \mathsf{poly}(\lambda, \log(1/\alpha))$, $q = \mathsf{poly}(\lambda, 1/\alpha)$, and a B-bounded distribution χ that is efficiently sampleable in $\mathsf{poly}(\lambda, \log(1/\alpha))$ time, such that $\alpha \cdot q > B$ and $\mathsf{LWE}_{n,m,q,\chi}$ assumption holds for any $m = \mathsf{poly}(\lambda)$.*

2.4 SimplePIR

In this section, we review SimplePIR [HHC+23], on which our DEPIR construction is based. For more detailed discussions, we refer the reader to the original paper. (See also its concurrent work FrodoPIR [DPC23].)

At a high level, SimplePIR follows the well-established template of Kushilevitz-Ostrovsky [KO97], which constructs PIR from linearly homomorphic encryption (LHE) by reshaping the database into a matrix, and instantiates it using Regev's LWE-based cryptosystem [Reg05]. However, directly following this approach is far from being *concretely* efficient. The bottleneck lies in transmitting $\boldsymbol{A}$ and computing the matrix multiplication $\widehat{\boldsymbol{D}}\boldsymbol{A}$ when the server evaluates $(\widehat{\boldsymbol{D}}\boldsymbol{A}, \widehat{\boldsymbol{D}}\boldsymbol{v})$

from the LWE ciphertext $(\boldsymbol{A}, \boldsymbol{v}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m$ for a homomorphic linear transformation, where $\widehat{\boldsymbol{D}} \in \mathbb{Z}^{m \times m}$ is the reshaped database and n is typically set to be larger than a thousand to ensure the hardness of LWE.

The simple yet powerful idea behind SimplePIR is to observe that $\widehat{\boldsymbol{D}}\boldsymbol{A}$ is independent of the query index, allowing $\boldsymbol{A}$ to be fixed as a CRS and shared across all queries. The server then preprocesses $\boldsymbol{H} \leftarrow \widehat{\boldsymbol{D}}\boldsymbol{A}$ as a *hint*, which is delivered to the client during the offline phase. Meanwhile, for security, the secret key $\boldsymbol{s}$ must be freshly sampled for each query. This approach significantly improves the server's online runtime and communication, as the bulk of the computation and communication that depends on the dimension n is shifted to the offline phase. In the online phase, the server only performs a matrix-vector multiplication $\widehat{\boldsymbol{D}}\boldsymbol{v}$.

The protocol is detailed in Fig. 1. Note the syntax of $\texttt{Extract}$, which takes $\widetilde{\texttt{DB}}_{\texttt{c}}$, the output of $\texttt{Prep}$, as input. Therefore, $\widetilde{\texttt{DB}}_{\texttt{c}}$ must be delivered to the client during the offline phase.

Analysis. Here, we briefly note the correctness and privacy of SimplePIR. We omit its efficiency analysis, as it is not directly used in our work.

- **Correctness**: Correctness, defined similarly to Definition 2.1, holds as long as the modulus-to-noise ratio satisfies $1/\alpha \geq 2m \cdot p^3$. This follows from a standard calculation.
- **Privacy**: Privacy, defined as in Definition 2.1, is guaranteed by the $\mathsf{LWE}_{n,m,q,\chi}$ assumption. This guarantee stems from the fact that $\texttt{q} = \boldsymbol{v}$, the output of $\texttt{Query}$, hides the query index i—specifically, the vector $\lfloor q/p \rfloor \cdot \boldsymbol{u}_{\hat{\jmath}}$—by masking it with the pseudorandomness of the LWE sample $(\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e})$.

2.5 Fast Matrix-Vector Multiplication with Preprocessing

We recall the sub-quadratic matrix–vector multiplication with preprocessing due to Williams [Wil07]. For the proof, we refer readers to the original paper.

Theorem 2.1 ([Wil07]). *Let R be a finite (semi)-ring. For all $\varepsilon \in (0, 1)$, every $m \times m$ matrix over R can be processed in $O(m^{2+\varepsilon \log |R|})$ time such that every subsequent matrix-vector multiplication can be performed in $O(m^2/(\varepsilon \log m)^2)$ steps on a pointer machine or a $(\log m)$-word RAM, assuming operations in R take constant time.*

In this work, we use the following instantiation of the theorem, which is obtained by setting $R = \mathbb{Z}_q$ and replacing ε with $\varepsilon / \log q$.

Corollary 2.1. *For all $\varepsilon \in (0, 1)$, every $m \times m$ matrix over $\mathbb{Z}_q$ can be processed in $O(m^{2+\varepsilon}) \cdot \mathsf{polylog}(q)$ time so that every subsequent matrix-vector multiplication can be performed in $O(m^2/(\varepsilon \log m)^2) \cdot \mathsf{polylog}(q)$ steps on a pointer machine or a $(\log m + \log \log q)$-word RAM.*

SimplePIR [HHC+23]

$\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$:

1. Set (m, p) so that $m^2 \cdot \lfloor \log p \rfloor \geq N$.
2. Choose parameters (n, q, χ) such that $\mathsf{LWE}_{n,m,q,\chi}$ holds, and the modulus-to-noise ratio $1/\alpha$ is at least $2m \cdot p^3$. (See Section 2.3.)
3. Sample $\boldsymbol{A} \leftarrow \mathbb{Z}_q^{m \times n}$ as CRS.
4. Output $\mathtt{pp} = (m, p, n, q, \chi, \boldsymbol{A})$.

$(\widetilde{\mathtt{DB}}_\mathtt{s}, \widetilde{\mathtt{DB}}_\mathtt{c}) \leftarrow \mathtt{Prep}(\mathtt{DB})$:

1. Reshape $\mathtt{DB} \in \{0,1\}^N$ into $\boldsymbol{D} \in \mathbb{Z}_p^{m \times m}$ so that $\mathtt{DB}[i]$ equals $\hat{k}$-th bit (counting from the least significant bit) of $\boldsymbol{D}[\hat{\imath}, \hat{\jmath}]$, where $0 \leq \hat{\imath}, \hat{\jmath} < m$ and $0 \leq \hat{k} < \lfloor \log p \rfloor$ are uniquely determined by the relation $i = \hat{\imath} + \hat{\jmath} \cdot m + \hat{k} \cdot m^2$. Let $\widehat{\boldsymbol{D}}$ denote the natural lifting of $\boldsymbol{D}$ to $\mathbb{Z}^{m \times m}$ by interpreting elements of $\mathbb{Z}_p$ as their canonical representatives in $\{0, 1, \cdots, p-1\} \subseteq \mathbb{Z}$.
2. Compute $\boldsymbol{H} \leftarrow \widehat{\boldsymbol{D}}\boldsymbol{A}$.
3. Output $\widetilde{\mathtt{DB}}_\mathtt{s} = \widehat{\boldsymbol{D}}$ and $\widetilde{\mathtt{DB}}_\mathtt{c} = \boldsymbol{H}$.

$(\mathtt{q}, \mathtt{st}) \leftarrow \mathtt{Query}(i)$:

1. Compute $0 \leq \hat{\imath}, \hat{\jmath} < m$ and $0 \leq \hat{k} < \lfloor \log p \rfloor$ such that $i = \hat{\imath} + \hat{\jmath} \cdot m + \hat{k} \cdot m^2$.
2. Sample $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$ and $\boldsymbol{e} \leftarrow \chi^m$.
3. Compute $\boldsymbol{v} \leftarrow \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e} + \lfloor q/p \rfloor \cdot \boldsymbol{u}_{\hat{\jmath}}$, where $\boldsymbol{u}_{\hat{\jmath}} \in \mathbb{Z}_q^m$ denotes the unit vector with a one in the $\hat{\jmath}$-th position and zeros elsewhere.
4. Output $\mathtt{q} = \boldsymbol{v}$ and $\mathtt{st} = (\boldsymbol{s}, \hat{\imath}, \hat{k})$.

$\mathtt{a} \leftarrow \mathtt{Answer}(\widetilde{\mathtt{DB}}_\mathtt{s}, \mathtt{q})$:

1. Parse $\widetilde{\mathtt{DB}}_\mathtt{s}$ as $\widehat{\boldsymbol{D}}$ and $\mathtt{q}$ as $\boldsymbol{v}$.
2. Evaluate $\boldsymbol{w} \leftarrow \widehat{\boldsymbol{D}}\boldsymbol{v}$.
3. Output $\mathtt{a} = \boldsymbol{w}$.

$\mathtt{b} \leftarrow \mathtt{Extract}(\widetilde{\mathtt{DB}}_\mathtt{c}, \mathtt{st}, \mathtt{a})$:

1. Parse $\widetilde{\mathtt{DB}}_\mathtt{c}$, $\mathtt{st}$, and $\mathtt{a}$ as $\boldsymbol{H}$, $(\boldsymbol{s}, \hat{\imath}, \hat{k})$, and $\boldsymbol{w}$, respectively.
2. Compute $\boldsymbol{z} \leftarrow \boldsymbol{w} - \boldsymbol{H}\boldsymbol{s}$.
3. Output the $\hat{k}$-th bit of $\left\lfloor \frac{p}{q} \cdot \widehat{\boldsymbol{z}}[\hat{\imath}] \right\rceil$ as $\mathtt{b}$.

Fig. 1. A description of SimplePIR [HHC+23]. The public parameter $\mathtt{pp}$ is assumed to be an implicit input to all algorithms other than $\mathtt{Setup}$.

3 LWE-based DEPIR

In this section, we present our LWE-based DEPIR construction (Fig. 2, Theorem 3.1, and Corollary 3.1). The construction relies on the following refinement of Corollary 2.1, which incorporates the Chinese Remainder Theorem (CRT) to mitigate dependence on the modulus in the evaluation time.

Lemma 3.1. *For all $\varepsilon \in (0,1)$, every $m \times m$ matrix over $\mathbb{Z}_q$ can be processed so that every subsequent matrix-vector multiplication can be performed with the following performance on a pointer machine or a $(\log m + \log\log q)$-word RAM.*

- *Preprocessing time:* $\widetilde{O}(m^{2+\varepsilon}) \cdot \mathsf{polylog}(q)$
- *Evaluation time:* $O(m^2/(\varepsilon \log m)^2) \cdot \widetilde{O}(\log m + \log q) + \widetilde{O}(m) \cdot \mathsf{polylog}(q)$

Proof. We defer the proof to Section A. □

Our protocol is detailed in Fig. 2. It is largely identical to SimplePIR (Section 2.4 and Fig. 1), with the following modifications:

- Instead of performing matrix-vector multiplication $\widehat{\boldsymbol{D}}\boldsymbol{v}$ naively in `Answer`, the server now preprocesses $\widehat{\boldsymbol{D}}$ (Step 2 of `Prep` in Fig. 2) and performs fast evaluation (Step 2 of `Answer` in Fig. 2) using Lemma 3.1.
- Instead of sending the precomputed $\boldsymbol{H}$ to the client during the offline phase, the server now includes $\boldsymbol{H}$ as part of the answer in the online phase. Note that the syntax of `Prep` and `Extract` has been updated accordingly.

Analysis. The privacy of our scheme (Fig. 2) follows directly from that of SimplePIR (Section 2.4), as both `Setup` and `Query` algorithms are identical to those in SimplePIR. The correctness of our scheme is also straightforward, as the only modifications from SimplePIR lie in how $\boldsymbol{w}$ is computed—namely, via Lemma 3.1—and when $\boldsymbol{H}$ is delivered to the client. We now turn to the efficiency analysis of our scheme. As our construction is based on LWE with polynomial modulus-to-noise ratio (Definition 2.4), we plug in $n = \mathsf{poly}(\lambda, \log(1/\alpha))$ and $q = \mathsf{poly}(\lambda, 1/\alpha)$ whenever applicable.

- *Preprocessing Runtime*: Step 2 takes $\widetilde{O}(m^{2+\varepsilon}) \cdot \mathsf{polylog}(q)$ time (Lemma 3.1), and Step 3 takes $m^2 \cdot n \cdot \mathsf{polylog}(q)$ time using textbook matrix multiplication. Plugging in $n = \mathsf{poly}(\lambda, \log(1/\alpha))$ and $q = \mathsf{poly}(\lambda, 1/\alpha)$ (Definition 2.4), the overall preprocessing runtime is $\widetilde{O}(m^{2+\varepsilon}) \cdot \mathsf{poly}(\lambda, \log(1/\alpha))$.
- *Online Communication*: $|\mathsf{q}| + |\mathsf{a}| = O(m \cdot n \cdot \log q) = m \cdot \mathsf{poly}(\lambda, \log(1/\alpha))$.
- *Server's Online Runtime*: Note that $m \le q$ by `Setup`. Then, by Lemma 3.1, Step 2 of `Answer` takes $O(m^2/(\varepsilon \log m)^2) \cdot \widetilde{O}(\log q) + \widetilde{O}(m) \cdot \mathsf{polylog}(q)$. Plugging in $q = \mathsf{poly}(\lambda, 1/\alpha)$ and accounting for the time required to output $\boldsymbol{H}$, which takes $m \cdot \mathsf{poly}(\lambda, \log(1/\alpha))$, the server's online runtime is:

$$O(m^2/(\varepsilon \log m)^2) \cdot \widetilde{O}(\log \lambda + \log(1/\alpha)) + \widetilde{O}(m) \cdot \mathsf{poly}(\lambda, \log(1/\alpha)).$$

LWE-based DEPIR in the CRS Model

$\mathtt{pp} \leftarrow \mathtt{Setup}(1^\lambda, 1^N)$: Same as SimplePIR (Fig. 1).

$\widetilde{\mathtt{DB}} \leftarrow \mathtt{Prep}(\mathtt{DB})$:

1. Reshape $\mathtt{DB} \in \{0,1\}^N$ into $\boldsymbol{D} \in \mathbb{Z}_p^{m \times m}$, and let $\widehat{\boldsymbol{D}}$ denote the natural lifting of $\boldsymbol{D}$ to $\mathbb{Z}^{m \times m}$, following the convention in Fig. 1.
2. Preprocess $\widehat{\boldsymbol{D}} \pmod q$ into $\widetilde{\boldsymbol{D}}$ using Lemma 3.1.
3. Compute $\boldsymbol{H} \leftarrow \widehat{\boldsymbol{D}}\boldsymbol{A}$.
4. Output $\widetilde{\mathtt{DB}} = (\widetilde{\boldsymbol{D}}, \boldsymbol{H})$.

$(\mathtt{q}, \mathtt{st}) \leftarrow \mathtt{Query}(i)$: Same as SimplePIR (Fig. 1).

$\mathtt{a} \leftarrow \mathtt{Answer}(\widetilde{\mathtt{DB}}, \mathtt{q})$:

1. Parse $\widetilde{\mathtt{DB}}$ as $(\widetilde{\boldsymbol{D}}, \boldsymbol{H})$ and $\mathtt{q}$ as $\boldsymbol{v}$.
2. Evaluate $\boldsymbol{w} \leftarrow \widehat{\boldsymbol{D}}\boldsymbol{v}$ with $\widetilde{\boldsymbol{D}}$ using Lemma 3.1.
3. Output $\mathtt{a} = (\boldsymbol{w}, \boldsymbol{H})$.

$\mathtt{b} \leftarrow \mathtt{Extract}(\mathtt{st}, \mathtt{a})$:

1. Parse $\mathtt{st}$ as $(\boldsymbol{s}, \hat{\imath}, \hat{k})$ and $\mathtt{a}$ as $(\boldsymbol{w}, \boldsymbol{H})$.
2. Compute $\boldsymbol{z} \leftarrow \boldsymbol{w} - \boldsymbol{H}\boldsymbol{s}$.
3. Output the $\hat{k}$-th bit of $\left\lfloor \frac{p}{q} \cdot \widehat{\boldsymbol{z}}[\hat{\imath}] \right\rceil$ as $\mathtt{b}$.

Fig. 2. A description of our LWE-based DEPIR in the CRS model. The public parameter $\mathtt{pp}$ is assumed to be an implicit input to all algorithms other than $\mathtt{Setup}$.

– *Client's Online Runtime*: Aside from Step 1 of $\mathtt{Query}$, which takes $\mathsf{polylog}(N)$ time, the bottleneck in client's online runtime lies in the matrix-vector multiplication $\boldsymbol{A}\boldsymbol{s}$ (in Step 3 of $\mathtt{Query}$), which requires $m \cdot n \cdot \mathsf{polylog}(q)$ time. Thus, plugging in $n = \mathsf{poly}(\lambda, \log(1/\alpha))$ and $q = \mathsf{poly}(\lambda, 1/\alpha)$, the client's overall online runtime is $\mathsf{polylog}(N) + m \cdot \mathsf{poly}(\lambda, \log(1/\alpha))$.

If we set $m = \left\lceil \sqrt{\frac{N}{\log N}} \right\rceil$ and $p = 2^{\lceil \log N \rceil}$, which implies $1/\alpha = \mathsf{poly}(N)$, we obtain the following theorem:

Theorem 3.1. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.4), there exists a DEPIR scheme in the CRS model with the following efficiency, on a pointer machine or a* $(\log N + \log\log\lambda)$*-word RAM, for any* $0 < \varepsilon < 1/2$*, where* N *is the database size and* λ *is the security parameter.*

– *Preprocessing Runtime:* $\widetilde{O}(N^{1+\varepsilon}) \cdot \mathsf{poly}(\lambda)$
– *Online Runtime:* $O(N/(\varepsilon \log N)^2) \cdot \mathsf{polyloglog}(N) \cdot \widetilde{O}(\log \lambda) + \widetilde{O}(\sqrt{N}) \cdot \mathsf{poly}(\lambda)$
– *Online Communication:* $\widetilde{O}(\sqrt{N}) \cdot \mathsf{poly}(\lambda)$

Proof. Correctness, privacy, and efficiency follow directly from the previous discussions. The only remaining step is to check the assumption that $1/\alpha = \mathsf{poly}(\lambda)$. This holds since we set $1/\alpha = \mathsf{poly}(N)$ and $N = \mathsf{poly}(\lambda)$. We note that the fact $N = \mathsf{poly}(\lambda)$ has not been used elsewhere, as doing so would allow N-dependent factors to be hidden within the $\mathsf{poly}(\lambda)$ term. □

As discussed in Remark 2.1, a DEPIR scheme in the CRS model implies a keyed DEPIR scheme with public preprocessing, hence both pk-DEPIR and sk-DEPIR, in the standard model with the same online efficiency.

Corollary 3.1. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.4), there exists a keyed DEPIR scheme with public preprocessing in the standard model, achieving the same efficiency stated in Theorem 3.1.*

Proof. The only remaining step is to verify that appending the key (i.e., the CRS) to the preprocessed database does not increase the asymptotic preprocessing runtime. This holds because the size of the CRS, $\boldsymbol{A} \in \mathbb{Z}_q^{m \times n}$, is $O(m \cdot n \cdot \log q) = \widetilde{O}(\sqrt{N}) \cdot \mathsf{poly}(\lambda)$. □

4 Open Problems

A significant milestone would be the construction of LWE-based (unkeyed) DEPIR scheme in the standard model with server-side computation of $O(N^{1-\varepsilon})$ for some constant $\varepsilon > 0$, ideally $\mathsf{polylog}(N)$. As intermediate steps toward this goal, we identify the following open problems:

– LWE-based (unkeyed) *barely* doubly-efficient PIR in the standard model, i.e. with server-side computation of $o(N/\log N)$.
– LWE-based (unkeyed) DEPIR with server-side computation of $O(N^{1-\varepsilon})$ for some constant $\varepsilon > 0$, in the CRS model or even using any black-box crypto [LMW25] not currently known to be constructible from the LWE assumption.
– LWE-based sk-DEPIR with server-side computation of $O(N^{1-\varepsilon})$ for some constant $\varepsilon > 0$.

Acknowledgments. We would like to thank Daniel Wichs for helpful discussions.

References

Ber14. Daniel J. Bernstein. A subfield-logarithm attack against ideal lattices, Feb 2014. URL: https://blog.cr.yp.to/20140213-ideal.html. 2

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012. doi:10.1145/2090236.2090262. 3

BHMW21. Elette Boyle, Justin Holmgren, Fermi Ma, and Mor Weiss. On the security of doubly efficient PIR. Cryptology ePrint Archive, Report 2021/1113, 2021. URL: https://eprint.iacr.org/2021/1113. 2

BHW19. Elette Boyle, Justin Holmgren, and Mor Weiss. Permuted puzzles and cryptographic hardness. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 465–493. Springer, Cham, December 2019. doi:10.1007/978-3-030-36033-7_18. 2

BIM00. Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 55–73. Springer, Berlin, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_4. 1

BIPW17. Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Kalai and Reyzin [KR17], pages 662–693. doi:10.1007/978-3-319-70503-3_22. 1, 2, 4

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Berlin, Heidelberg, August 2011. doi:10.1007/978-3-642-22792-9_29. 3

BW21. Keller Blackwell and Mary Wootters. A note on the permuted puzzles toy conjecture, 2021. URL: https://arxiv.org/abs/2108.07885, arXiv:2108.07885. 2

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995. doi:10.1109/SFCS.1995.492461. 1

CHR17. Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Kalai and Reyzin [KR17], pages 694–726. doi:10.1007/978-3-319-70503-3_23. 1, 2, 4

CIMR25. Caicai Chen, Yuval Ishai, Tamer Mour, and Alon Rosen. Secret-key PIR from random linear codes. Cryptology ePrint Archive, Paper 2025/646, 2025. URL: https://eprint.iacr.org/2025/646. 2

CIV16. Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Provably weak instances of ring-LWE revisited. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 147–167. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49890-3_6. 2

DKL09. Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 621–630. ACM Press, May / June 2009. doi:10.1145/1536414.1536498. 2

DMZ23. Xiuquan Ding, Giulio Malavolta, and Tianwei Zhang. Doubly efficient batched private information retrieval. Cryptology ePrint Archive, Report 2023/1552, 2023. URL: https://eprint.iacr.org/2023/1552. 2, 3

DPC23. Alex Davidson, Gonçalo Pestana, and Sofía Celi. FrodoPIR: Simple, scalable, single-server private information retrieval. *PoPETs*, 2023(1):365–383, January 2023. doi:10.56553/popets-2023-0022. 6

ELOS15. Yara Elias, Kristin E. Lauter, Ekin Ozman, and Katherine E. Stange. Provably weak instances of ring-LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 63–92. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-47989-6_4. 2

FF25. Serge Fehr and Pierre-Alain Fouque, editors. *EUROCRYPT 2025, Part VI*, volume 15606 of *LNCS*. Springer, Cham, May 2025. 13

HHC+23. Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 3889–3905. USENIX Association, August 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger. 3, 6, 8

KO97. Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997. doi:10.1109/SFCS.1997.646125. 1, 6

KR17. Yael Kalai and Leonid Reyzin, editors. *TCC 2017, Part II*, volume 10678 of *LNCS*. Springer, Cham, November 2017. 12

KU08. Kiran S. Kedlaya and Christopher Umans. Fast modular composition in any characteristic. In *49th FOCS*, pages 146–155. IEEE Computer Society Press, October 2008. doi:10.1109/FOCS.2008.13. 3, 14

LMW23. Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 595–608. ACM Press, June 2023. doi:10.1145/3564246.3585175. 2, 3, 4, 6

LMW25. Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Black box crypto is useless for doubly efficient PIR. In Fehr and Fouque [FF25], pages 65–93. doi:10.1007/978-3-031-91095-1_3. 11

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010. doi:10.1007/978-3-642-13190-5_1. 2

OPPW24. Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. Towards practical doubly-efficient private information retrieval. In Jeremy Clark and Elaine Shi, editors, *FC 2024, Part II*, volume 14745 of *LNCS*, pages 264–282. Springer, Cham, March 2024. doi:10.1007/978-3-031-78679-2_14. 2

OPPW25. Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. On algebraic homomorphic encryption and its applications to doubly-efficient PIR. In Fehr and Fouque [FF25], pages 34–64. doi:10.1007/978-3-031-91095-1_2. 2

Pei16. Chris Peikert. How (not) to instantiate ring-LWE. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 411–430. Springer, Cham, August / September 2016. doi:10.1007/978-3-319-44618-9_22. 2

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Berlin, Heidelberg, August 2008. doi:10.1007/978-3-540-85174-5_31. 3

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. doi:10.1145/1060590.1060603. 2, 6

Wil07. Ryan Williams. Matrix-vector multiplication in sub-quadratic time: (some preprocessing required). In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *18th SODA*, pages 995–1001. ACM-SIAM, January 2007. 3, 7

A Proof of Lemma 3.1

In this section, we provide a deferred proof of Lemma 3.1, which augments Corollary 2.1 with the Chinese Remainder Theorem (CRT) to mitigate dependence on the modulus. We will need the following fact:

Lemma A.1 ([KU08]). *For all integers $M \geq 2$, the product of primes less than or equal to $16 \log M$ is greater than M.*

We restate and prove Lemma 3.1 below.

Lemma 3.1. *For all $\varepsilon \in (0,1)$, every $m \times m$ matrix over $\mathbb{Z}_q$ can be processed so that every subsequent matrix-vector multiplication can be performed with the following performance on a pointer machine or a $(\log m + \log\log q)$-word RAM.*

- *Preprocessing time:* $\widetilde{O}(m^{2+\varepsilon}) \cdot \mathsf{polylog}(q)$
- *Evaluation time:* $O(m^2/(\varepsilon \log m)^2) \cdot \widetilde{O}(\log m + \log q) + \widetilde{O}(m) \cdot \mathsf{polylog}(q)$

Proof. Set $M := m \cdot (q-1)^2$. For any $\boldsymbol{A} \in \mathbb{Z}_q^{m \times m}$ and $\boldsymbol{v} \in \mathbb{Z}_q^m$, we can naturally lift them to $\widehat{\boldsymbol{A}} \in \mathbb{Z}^{m \times m}$ and $\widehat{\boldsymbol{v}} \in \mathbb{Z}^m$ by interpreting elements of $\mathbb{Z}_q$ as their canonical representatives in $\{0, 1, \cdots, q-1\} \subseteq \mathbb{Z}$. We then have $\|\widehat{\boldsymbol{A}}\widehat{\boldsymbol{v}}\|_\infty \leq M$ under this lifting. Hence, we can reconstruct $\boldsymbol{A}\boldsymbol{v} \in \mathbb{Z}_q^m$ from $\widehat{\boldsymbol{A}}\widehat{\boldsymbol{v}} \pmod{q_i}$ for all $i \in [t]$, as long as the product of distinct primes satisfies $\prod_{i=1}^t q_i > M$, using the Chinese Remainder Theorem (CRT). By Lemma A.1, we can choose such $\{q_1, \cdots, q_t\}$ with each $q_i = O(\log M)$ and $t = O(\log M)$. We then apply Corollary 2.1 to preprocess $\widehat{\boldsymbol{A}} \pmod{q_i}$ and to efficiently evaluate $\widehat{\boldsymbol{A}}\widehat{\boldsymbol{v}} \pmod{q_i}$ for each $i \in [t]$. The algorithm is detailed below. Its correctness follows directly from Corollary 2.1 and CRT, and the runtime analysis is incorporated into the algorithm's description.

Preprocessing

1. Compute a set of distinct primes $\{q_1, \cdots, q_t\}$ such that $q_i = O(\log M)$, $t = O(\log M)$, and $\prod_{i=1}^{t} q_i > M$. By Lemma A.1, this can be done in $\mathsf{polylog}(M) = \mathsf{poly}(\log m, \log q)$ time.
2. Given $\boldsymbol{A}$, for $i \in [t]$, compute $\widehat{\boldsymbol{A}}_i \leftarrow \widehat{\boldsymbol{A}} \pmod{q_i}$. This can be done in $t \cdot m^2 \cdot \mathsf{poly}(\log M, \log q) = \widetilde{O}(m^2) \cdot \mathsf{polylog}(q)$ time.
3. For $i \in [t]$, preprocess $\widehat{\boldsymbol{A}}_i \in \mathbb{Z}_{q_i}^{m \times m}$ with Corollary 2.1. This takes $t \cdot O(m^{2+\varepsilon}) \cdot \mathsf{polyloglog}(M) = O(m^{2+\varepsilon}) \cdot \widetilde{O}(\log M) = \widetilde{O}(m^{2+\varepsilon}) \cdot \widetilde{O}(\log q)$ time.

Evaluation

1. Given $\boldsymbol{v}$, for $i \in [t]$, compute $\widehat{\boldsymbol{v}}_i \leftarrow \widehat{\boldsymbol{v}} \pmod{q_i}$. This can be done in $t \cdot m \cdot \mathsf{poly}(\log M, \log q) = \widetilde{O}(m) \cdot \mathsf{polylog}(q)$ time.
2. For $i \in [t]$, evaluate $\widehat{\boldsymbol{A}}_i \widehat{\boldsymbol{v}}_i \in \mathbb{Z}_{q_i}^m$ using the preprocessed $\widehat{\boldsymbol{A}}_i$ with Corollary 2.1. This takes $t \cdot O(m^2/(\varepsilon \log m)^2) \cdot \mathsf{polyloglog}(M) = O(m^2/(\varepsilon \log m)^2) \cdot \widetilde{O}(\log M) = O(m^2/(\varepsilon \log m)^2) \cdot \widetilde{O}(\log m + \log q)$ time.
3. Reconstruct $\boldsymbol{A}\boldsymbol{v} \in \mathbb{Z}_q^m$ from $\widehat{\boldsymbol{A}}_i \widehat{\boldsymbol{v}}_i$ for all $i \in [t]$, using CRT. This can be done in $m \cdot \mathsf{polylog}(M) = \widetilde{O}(m) \cdot \mathsf{polylog}(q)$ time.

□