

Barely Sublinear RAM-FHE from Plain LWE

Keewoo Lee

Ethereum Foundation
keewoo.lee@ethereum.org

Abstract. Fully Homomorphic Encryption for RAM programs (RAM-FHE), introduced by Hamlin, Holmgren, Weiss, and Wichs (Crypto 2019), allows a server to evaluate a RAM program on encrypted data at a per-step cost sublinear in the memory size N , whereas standard FHE must read the entire memory for every memory access. In a celebrated work, Lin, Mook, and Wichs (STOC 2023) gave the first, and to date the only, construction from a concrete assumption, namely Ring-LWE. In this work, we construct a RAM-FHE scheme from plain LWE that evaluates each step of a RAM program in $o(N/\log N)$ time. While the scheme is barely sublinear, even such a slight saving over the linear cost of standard FHE was not known to be achievable from unstructured, plain LWE. As Lin, Mook, and Wichs did, we build RAM-FHE by extending a Doubly-Efficient Private Information Retrieval (DEPIR) scheme. Since the only DEPIR from plain LWE, due to Lee (Crypto 2026), relies on a Common Reference String (CRS) that is incompatible with the extension, we show how to remove the CRS and obtain a DEPIR from plain LWE in the standard model, which may be of independent interest.

1 Introduction

Fully Homomorphic Encryption (FHE) [RAD78, Gen09] allows a server to perform arbitrary computations on encrypted data without ever decrypting. Standard FHE schemes, however, operate in the circuit model, in which the program is given as a circuit and the server’s work is proportional to its size. In particular, a single random access to a memory of size N requires a circuit of size linear in N , since a circuit takes the whole memory as input and its wiring cannot depend on the address. For encrypted data stored cell by cell, this cost cannot be avoided [BIM00], since a server that could perform a random access correctly without reading some cells would learn that the address is none of those cells, which contradicts privacy.

Hamlin, Holmgren, Weiss, and Wichs [HHWW19] introduced Fully Homomorphic Encryption for RAM programs (RAM-FHE), which avoids this cost by preprocessing the encrypted memory once: afterwards the server evaluates each step of a RAM program in time sublinear in N . As evidence that RAM-FHE is plausible at all, they gave a construction from non-standard assumptions and a heuristic use of virtual black-box obfuscation for specific circuits. In a celebrated work, Lin, Mook, and Wichs [LMW23] gave the first, and to date the only, construction from a concrete assumption, namely Ring-LWE [LPR10], and their

scheme performs a random access at a cost that scales as a mere $\text{polylog}(N)$. While Ring-LWE is widely regarded as a *de facto*¹ standard assumption, this construction hinges on the algebra of polynomial rings, so whether RAM-FHE can be based on plain LWE [Reg05] remained an intriguing open question.

1.1 Our Contribution

RAM-FHE from Plain LWE. In this work, we construct RAM-FHE from plain LWE with a polynomial modulus-to-noise ratio (Theorem 7.5). After a public preprocessing of the encrypted memory by the server, every step of a RAM program, and in particular every memory access, costs $o(N/\log N)$ time, where N is the memory size. Our scheme is only barely sublinear, but no sublinear RAM-FHE at all was known from plain LWE prior to this work. The result shows that RAM-FHE does not hinge on the algebra of polynomial rings, and it narrows the gap in feasibility results between Ring-LWE and plain LWE.

LWE-Based DEPIR in the Standard Model. At a high level, our construction follows [LMW23], which builds RAM-FHE from Doubly-Efficient Private Information Retrieval (DEPIR) [BIM00, CHR17, BIPW17]. In a DEPIR scheme, the server preprocesses a public database once, after which any client can retrieve an entry without revealing its index, and the server answers each query in time sublinear in the database size. However, the only DEPIR from plain LWE, due to Lee [Lee26], relies on a Common Reference String (CRS) that is incompatible with this approach. We address this by constructing a DEPIR from plain LWE in the standard model (Theorem 4.1). Like the scheme of [Lee26], on which it builds, it is barely doubly-efficient: the server answers a query in $o(N/\log N)$ time for a database of N bits. This lifts the result of [Lee26], which narrowed the gap between Ring-LWE and plain LWE for DEPIR only in the CRS model, to the standard model, and may be of independent interest.

1.2 Technical Overview

Review: LWE-Based DEPIR in the CRS Model. The DEPIR scheme of [Lee26] starts from the PIR template of Kushilevitz and Ostrovsky [KO97],

¹ Module-LWE [LS15], which interpolates between Ring-LWE and plain LWE, underlies the NIST standard ML-KEM [NIS24]. Despite more than fifteen years of cryptanalytic efforts, no attack has been found that performs substantially better against Ring-LWE than against plain LWE. A line of work [ELOS15, CIV16] shows that certain instantiations of Ring-LWE can be easily broken, but these examples are somewhat contrived [Pei16], and hybrid attacks that exploit the ring structure gain only a modest number of bits [HJO26]. Ring-LWE does enjoy a worst-case to average-case reduction [LPR10], but the worst-case problems behind it are on *ideal lattices*, a class of structured lattices, whereas those behind plain LWE are on general lattices. We also note that, although it represents a minority opinion, some in the community express reservations about the long-term security of Ring-LWE due to its additional algebraic structure [Ber14].

instantiated with Regev encryption [Reg05]. The database is arranged as a matrix $\mathbf{D}$, and the query is a Regev ciphertext $(\mathbf{A}, \mathbf{v})$ encrypting the unit vector that selects the requested column. The server multiplies the query by $\mathbf{D}$ homomorphically, which amounts to computing $\mathbf{DA}$ and $\mathbf{Dv}$. The scheme of [Lee26] combines Williams’s preprocessing for matrix-vector multiplication [Wil07], with which the server computes $\mathbf{Dv}$ in $O(N/\log^2 N)$ time, and SimplePIR [HHC⁺23], which treats $\mathbf{A}$ as a CRS. The CRS lets the server precompute $\mathbf{DA}$, which matters because $\mathbf{A}$ has as many columns as the LWE dimension $n = \text{poly}(\lambda, \log N)$: computing $\mathbf{DA}$ online would cost n times as much as $\mathbf{Dv}$ and nullify the $\log^2 N$ saving unless one makes an extra assumption on how n depends on N .

CRS-Free DEPIR from LWE. Perhaps the most natural approach to removing the CRS is to compute $\mathbf{D}[\mathbf{A} \mid \mathbf{v}]$ for a fresh $\mathbf{A}$ in every query. Concretely, the goal would be to multiply a preprocessed matrix by a thin matrix of $n + 1$ columns in about the time of a single matrix-vector product with Williams’s preprocessing. However, to our knowledge, no prior work addresses this problem. Multiplying a fixed matrix by many vectors has been studied extensively, but the known speedups beyond Williams’s [LW17] are tailored to the Boolean semiring, whereas the products here are over $\mathbb{Z}_q$ for the ciphertext modulus q . On the other hand, fast matrix-matrix multiplication [Str69], in particular by a thin matrix [Cop82], takes both matrices at once, so its running time is at least linear in the size of $\mathbf{D}$, while DEPIR aims at sublinear time.

We therefore design the algorithm we need ourselves (Section 3). Our key observation is that Coppersmith’s algorithm [Cop82] composes with Williams’s preprocessing [Wil07]: as a *bilinear* algorithm (Definition 2.5), it can be applied to block matrices, where each block product can be instantiated with any algorithm, possibly one that leverages preprocessing. Coppersmith’s algorithm computes the product of a $k \times k$ matrix and a $k \times k^\alpha$ matrix with only $O(k^2 \log^2 k)$ multiplications for $\alpha \approx 0.17$.² Applied to $\mathbf{D}$ split into a $k \times k$ grid of blocks, with k just large enough that $n + 1 \leq k^\alpha$ (Theorem 3.4), it turns the product with $[\mathbf{A} \mid \mathbf{v}]$ into $O(k^2 \log^2 k)$ blockwise matrix-vector products, each of which can leverage Williams’s preprocessing. The product with $[\mathbf{A} \mid \mathbf{v}]$ thus costs only $\text{polylog}(n) = \text{poly}(\log \lambda, \log \log N)$ times a single run of Williams’s algorithm, whereas naive repetition takes $n + 1 = \text{poly}(\lambda, \log N)$ runs. Finally, following [Lee26], we run the whole computation modulo several primes via the Chinese Remainder Theorem, which mitigates the dependence on the ciphertext modulus q . Altogether, the server can afford a fresh $\mathbf{A}$ in every query, which yields a DEPIR satisfying Definition 2.1 without a CRS (Figure 1).

² Later work [Cop97, LG12, LGU18, VXXZ24] focuses on increasing α and bounds the number of multiplications only by $k^{2+o(1)}$. This is not the right trade-off in our setting. A larger α only allows a smaller k , which affects lower-order terms, whereas the extra factor $k^{o(1)}$ is only shown to be subpolynomial, so it would weaken the $\text{polylog}(n)$ factor in our bounds.

From DEPIR to RAM-FHE. To construct RAM-FHE, we follow [LMW23] and let the server perform each read of the memory as a DEPIR lookup under FHE. We instantiate the FHE with the GSW scheme [GSW13], whose ciphertexts are stacks of Regev ciphertexts under the same secret. This allows the server to turn GSW ciphertexts of the address into a DEPIR query for the requested column under the FHE secret key.³ The server answers the query with the algorithm of Section 3 and converts the answer back into a GSW ciphertext of the requested bit (Figure 3). This DEPIR lookup is performed on the memory itself, which the server holds in encrypted form. Stored as GSW ciphertexts, however, the N bits of the memory would occupy $N \cdot \text{poly}(\lambda)$ bits and put a $\text{poly}(\lambda)$ factor into the leading term of the cost per read.⁴ To avoid this factor, we use the compressible variant of GSW due to Gentry and Halevi [GH19] and store the memory in its constant-rate compressed form (Figure 2), as a public database of $O(N)$ bits that the server can preprocess without any key material. A read then obtains a double encryption of the requested word, namely GSW ciphertexts of its compressed ciphertext, and removes the inner layer by homomorphic decryption (Figure 3). For writes, we follow [LMW23] in a simpler form, since we only target a barely sublinear cost per step. Instead of their hierarchy of levels, two levels suffice: a preprocessed snapshot of the memory and a buffer of recent writes (Figure 4).

1.3 Organization

Section 2 collects the preliminaries. Section 3 presents the algorithm for multiplying a preprocessed matrix by a thin matrix, on which our main results rest. Section 4 gives the LWE-based DEPIR in the standard model. A reader interested only in DEPIR can stop there. A reader interested only in RAM-FHE can skip Section 4, since the later sections use the algorithm of Section 3 directly rather than the DEPIR scheme. Section 5 recalls the constant-rate compressible FHE scheme of Gentry and Halevi [GH19]. This section contains no new ideas, but we use the scheme in a non-black-box way and need explicit parameter conditions, so we restate it in full. Section 6 extends this scheme with a homomorphic database lookup, which retrieves the entry at an encrypted index from a preprocessed encrypted database in sublinear time. Section 7 adds writes and completes the RAM-FHE scheme.

2 Preliminaries

Notation. We write $[n] = \{0, 1, \dots, n-1\}$, and all indices start from 0. For a list or vector L , we write $L[i]$ for its i -th entry and $L[:\ell]$ for the list of its first

³ The DEPIR scheme of [Lee26] seems inapplicable, since it fixes $\mathbf{A}$ in advance as a CRS to precompute $\mathbf{DA}$, whereas here $\mathbf{A}$ is part of the query and is determined only during the computation.

⁴ Such a cost would still be sublinear in N , which is interesting in its own right. However, already a quadratic factor λ^2 exceeds $\log^2 N$ for every $N < 2^\lambda$ and nullifies the saving in any reasonable range of parameters.

ℓ entries. For a matrix $\mathbf{A}$, we write $\mathbf{A}[i]$ for its i -th row and $\mathbf{A}[i, j]$ for its (i, j) entry. The terms *negligible* and *overwhelming* are with respect to the security parameter λ .

2.1 Doubly-Efficient Private Information Retrieval

Definition 2.1 (DEPIR). A Doubly-Efficient Private Information Retrieval (DEPIR) scheme consists of algorithms (Setup, Prep, Query, Answer, Extract).

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

The algorithms must satisfy the following requirements.

Correctness: For $N = \text{poly}(\lambda)$, any $\text{DB} \in \{0, 1\}^N$, and any $i \in [N]$,

$$\Pr \left[\text{Extract}(\text{st}, \mathbf{a}) = \text{DB}[i] \mid \begin{array}{l} \widetilde{\text{DB}} \leftarrow \text{Prep}(\text{DB}) \\ (\mathbf{q}, \text{st}) \leftarrow \text{Query}(i) \\ \mathbf{a} \leftarrow \text{Answer}(\widetilde{\text{DB}}, \mathbf{q}) \end{array} \right] \geq 1 - \text{negl}(\lambda).$$

Privacy: For $N = \text{poly}(\lambda)$ and any $i_0, i_1 \in [N]$, no probabilistic polynomial-time adversary can distinguish between the queries $\mathbf{q}$ output by $\text{Query}(i_0)$ and by $\text{Query}(i_1)$ with non-negligible advantage.

Efficiency: For all λ and N , Setup runs in $\text{poly}(\lambda, \log N)$ and Prep in $\text{poly}(\lambda, N)$ time, while Query, Answer, and Extract each run in $o(N) \cdot \text{poly}(\lambda)$ time.⁵

2.2 Learning with Errors

Definition 2.2 (LWE [Reg05]). Given dimension n , number of samples m , modulus q , and error distribution χ over $\mathbb{Z}$, the $\text{LWE}_{n,m,q,\chi}$ assumption states that no probabilistic polynomial-time adversary can distinguish between the following two distributions with non-negligible advantage.

⁵ On a word RAM with $\Omega(\log N)$ -bit words, the trivial scheme, in which the server returns the whole database, already runs in $O(N/\log N)$ time. A meaningful DEPIR in this model must therefore run in $o(N/\log N)$ time, which our scheme achieves.

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

Definition 2.3 (LWE with Polynomial Modulus-to-Noise Ratio). A distribution χ over $\mathbb{Z}$ is B -bounded if $|e| \leq B$ holds with overwhelming probability for $e \leftarrow \chi$. We refer to the following as the LWE with polynomial modulus-to-noise ratio assumption:

There is a deterministic algorithm that, given 1^λ and a modulus-to-noise ratio $1/\alpha$, runs in $\text{poly}(\lambda, \log(1/\alpha))$ time and outputs an integer $B \geq 1$, parameters $n = \text{poly}(\lambda, \log(1/\alpha))$ and $q = \text{poly}(\lambda, 1/\alpha)$, and a description of a B -bounded distribution χ sampleable in $\text{poly}(\lambda, \log(1/\alpha))$ time, such that $\alpha q > B$. Moreover, for any polynomial modulus-to-noise ratio $1/\alpha = \text{poly}(\lambda)$, the $\text{LWE}_{n,m,q,\chi}$ assumption holds for any $m = \text{poly}(\lambda)$.

2.3 Fast Matrix-Vector Multiplication with Preprocessing

Williams [Wil07] proved the following for matrices over any finite semiring, and we instantiate it over $\mathbb{Z}_q$.

Theorem 2.4 ([Wil07]). For all $\varepsilon \in (0, 1)$ and $m \geq 2$, every $\mathbf{D} \in \mathbb{Z}_q^{m \times m}$ can be preprocessed in $O(m^{2+\varepsilon}) \cdot \text{polylog}(q)$ time so that afterwards, for every $\mathbf{v} \in \mathbb{Z}_q^m$, the product $\mathbf{D}\mathbf{v}$ can be computed in $O(m^2/(\varepsilon \log m)^2) \cdot \text{polylog}(q)$ time on an $O(\log m + \log \log q)$ -word RAM.

2.4 Bilinear Algorithms for Rectangular Matrix Multiplication

We recall Coppersmith’s algorithm for rectangular matrix multiplication (Theorem 2.6), which is a *bilinear algorithm*. As in Strassen’s recursion [Str69], a bilinear algorithm remains correct when the entries of the two matrices are matrix blocks of compatible sizes, and Lemma 3.1 uses this property with the block products computed by Theorem 2.4.

Definition 2.5 (Bilinear Algorithm). Let $\mathbb{F}$ be a field. A bilinear algorithm of rank r for $\langle k_1, k_2, k_3 \rangle$ -matrix multiplication over $\mathbb{F}$, the multiplication of a $k_1 \times k_2$ matrix by a $k_2 \times k_3$ matrix, consists of linear maps $f_\ell: \mathbb{F}^{k_1 \times k_2} \rightarrow \mathbb{F}$ and $g_\ell: \mathbb{F}^{k_2 \times k_3} \rightarrow \mathbb{F}$, and matrices $\mathbf{W}_\ell \in \mathbb{F}^{k_1 \times k_3}$, indexed by $\ell \in [r]$, such that

$$\mathbf{AB} = \sum_{\ell \in [r]} f_\ell(\mathbf{A}) \cdot g_\ell(\mathbf{B}) \cdot \mathbf{W}_\ell, \quad \text{for all } \mathbf{A} \in \mathbb{F}^{k_1 \times k_2} \text{ and } \mathbf{B} \in \mathbb{F}^{k_2 \times k_3}.$$

Theorem 2.6 ([Cop82] and [Wil14, App. C]). Let $\alpha = \frac{2 \log 2}{5 \log 5} \approx 0.17$. For every $k \geq 2$ and every field $\mathbb{F}$ with $|\mathbb{F}| \geq k$, there is a bilinear algorithm of rank $O(k^2 \log^2 k)$ for $\langle k, k, \lfloor k^\alpha \rfloor \rangle$ -matrix multiplication over $\mathbb{F}$. Such an algorithm can be constructed with $\text{poly}(k)$ field operations.

3 Rectangular Matrix Multiplication with Preprocessing

In this section, we show that after preprocessing a matrix $\mathbf{D} \in \mathbb{Z}_q^{m \times m}$, its product with a thin matrix $\mathbf{X} \in \mathbb{Z}_q^{m \times n}$ can be computed in time whose leading term grows only polylogarithmically in n (Theorem 3.4). The main step is Lemma 3.1, which splits $\mathbf{D}$ into a $k \times k$ grid of blocks and runs a bilinear algorithm with these blocks as entries, so that $\mathbf{D}\mathbf{X}$ reduces to r matrix-vector products whose matrices depend only on $\mathbf{D}$ and can thus be preprocessed with Theorem 2.4.

Lemma 3.1. *Let $k \geq 1$, $b \geq 2$, and $n \geq 1$ be integers, let $m := kb$, let q be a prime, and suppose that a bilinear algorithm of rank r for $\langle k, k, n \rangle$ -matrix multiplication over $\mathbb{F}_q$ is given. Then for every $\varepsilon \in (0, 1)$, every $\mathbf{D} \in \mathbb{F}_q^{m \times m}$ can be preprocessed in $O(r \cdot m^{2+\varepsilon}) \cdot \text{polylog}(q)$ time so that afterwards, for every $\mathbf{X} \in \mathbb{F}_q^{m \times n}$, the product $\mathbf{D}\mathbf{X}$ can be computed in*

$$O\left(\frac{r \cdot b^2}{\varepsilon^2 \cdot \log^2 b}\right) \cdot \text{polylog}(q) + O(r \cdot n \cdot m) \cdot \text{polylog}(q)$$

time on an $O(\log m + \log \log q)$ -word RAM.

Proof. Let $(f_\ell, g_\ell, \mathbf{W}_\ell)_{\ell \in [r]}$ be the given bilinear algorithm and let $\otimes$ denote the Kronecker product, which identifies $\mathbb{F}_q^{m \times m}$ with $\mathbb{F}_q^{k \times k} \otimes \mathbb{F}_q^{b \times b}$ and $\mathbb{F}_q^{m \times n}$ with $\mathbb{F}_q^{k \times n} \otimes \mathbb{F}_q^b$. Let $\hat{f}_\ell := f_\ell \otimes \text{id}: \mathbb{F}_q^{m \times m} \rightarrow \mathbb{F}_q^{b \times b}$ and $\hat{g}_\ell := g_\ell \otimes \text{id}: \mathbb{F}_q^{m \times n} \rightarrow \mathbb{F}_q^b$ be the linear maps with $\hat{f}_\ell(\mathbf{A} \otimes \check{\mathbf{D}}) = f_\ell(\mathbf{A}) \cdot \check{\mathbf{D}}$ and $\hat{g}_\ell(\mathbf{B} \otimes \check{\mathbf{x}}) = g_\ell(\mathbf{B}) \cdot \check{\mathbf{x}}$ for all $\mathbf{A} \in \mathbb{F}_q^{k \times k}$, $\check{\mathbf{D}} \in \mathbb{F}_q^{b \times b}$, $\mathbf{B} \in \mathbb{F}_q^{k \times n}$, and $\check{\mathbf{x}} \in \mathbb{F}_q^b$. We claim that

$$\mathbf{D}\mathbf{X} = \sum_{\ell \in [r]} \mathbf{W}_\ell \otimes (\hat{f}_\ell(\mathbf{D})\hat{g}_\ell(\mathbf{X})), \quad (1)$$

that is, the (i, j) block of $\mathbf{D}\mathbf{X}$ is $\sum_{\ell} \mathbf{W}_\ell[i, j] \cdot \hat{f}_\ell(\mathbf{D})\hat{g}_\ell(\mathbf{X})$. Both sides of (1) are bilinear in $(\mathbf{D}, \mathbf{X})$, and the matrices of the form $\mathbf{A} \otimes \check{\mathbf{D}}$ and $\mathbf{B} \otimes \check{\mathbf{x}}$ span $\mathbb{F}_q^{m \times m}$ and $\mathbb{F}_q^{m \times n}$, respectively, so it suffices to verify (1) for $\mathbf{D} = \mathbf{A} \otimes \check{\mathbf{D}}$ and $\mathbf{X} = \mathbf{B} \otimes \check{\mathbf{x}}$. For these, the left side of (1) is $\mathbf{D}\mathbf{X} = (\mathbf{A}\mathbf{B}) \otimes (\check{\mathbf{D}}\check{\mathbf{x}})$ by the mixed-product property of $\otimes$, and the right side is

$$\sum_{\ell \in [r]} \mathbf{W}_\ell \otimes (f_\ell(\mathbf{A})g_\ell(\mathbf{B}) \cdot \check{\mathbf{D}}\check{\mathbf{x}}) = \left(\sum_{\ell \in [r]} f_\ell(\mathbf{A})g_\ell(\mathbf{B})\mathbf{W}_\ell \right) \otimes (\check{\mathbf{D}}\check{\mathbf{x}}) = (\mathbf{A}\mathbf{B}) \otimes (\check{\mathbf{D}}\check{\mathbf{x}})$$

by the definition of $\hat{f}_\ell$ and $\hat{g}_\ell$, the bilinearity of $\otimes$, and Definition 2.5. The two sides agree, which proves (1).

Now note that the matrices $\hat{f}_\ell(\mathbf{D})$ depend on $\mathbf{D}$ only. In preprocessing, compute these r matrices, each a linear combination of the k^2 blocks of $\mathbf{D}$, and apply Theorem 2.4 to every $\hat{f}_\ell(\mathbf{D})$, in $O(r \cdot m^{2+\varepsilon}) \cdot \text{polylog}(q)$ time in total. To compute $\mathbf{D}\mathbf{X}$ for a given $\mathbf{X}$, first form the r vectors $\hat{g}_\ell(\mathbf{X})$, which are linear combinations of the kn blocks of $\mathbf{X}$ and cost $O(rnm)$ field operations. The r products $\hat{f}_\ell(\mathbf{D})\hat{g}_\ell(\mathbf{X})$ then take $r \cdot O(b^2/(\varepsilon^2 \cdot \log^2 b)) \cdot \text{polylog}(q)$ time by Theorem 2.4,

and assembling $\mathbf{DX}$ from them by (1) takes another $O(rnm)$ field operations. Field operations cost $\text{polylog}(q)$ time, and the word size $O(\log b + \log \log q)$ that Theorem 2.4 needs for $b \times b$ matrices is clearly $O(\log m + \log \log q)$. $\square$

Over $\mathbb{F}_q$, the time bounds of Lemma 3.1 carry a $\text{polylog}(q)$ factor. To tame this factor, we lift $\mathbf{D}$ and $\mathbf{X}$ to the integers, compute their product modulo several small primes q_i , and recombine the results by the Chinese Remainder Theorem, as in [Lee26]. The primes must satisfy $q_i \geq k$, since Theorem 2.6 needs a field of size at least k . The following two lemmas provide such primes.

Lemma 3.2 ([KU08, Lem. 2.3]). *For all integers $M \geq 2$, the product of primes less than $16 \log M$ is greater than M .*

Lemma 3.3. *For all integers $M, k \geq 2$, there exist an integer t and primes $k \leq q_0 < \dots < q_{t-1}$ such that $\prod_{i \in [t]} q_i > M$, with $t \leq \log M / \log k + 1$ and $q_{t-1} < 16(\log M + k \log k)$. Such primes can be found in $\text{poly}(k, \log M)$ time.*

Proof. By Lemma 3.2 applied to $M \cdot k^k \geq 2$, the product of all primes less than $16(\log M + k \log k)$ exceeds $M \cdot k^k$. The product of the primes below k is clearly less than k^k , so the primes in the interval $[k, 16(\log M + k \log k))$ have product greater than M . Let $k \leq q_0 < q_1 < \dots$ be the primes in this interval, listed in $\text{poly}(k, \log M)$ time, and let t be the smallest integer with $\prod_{i \in [t]} q_i > M$. Then $k^{t-1} \leq \prod_{i \in [t-1]} q_i \leq M$, and thus $t \leq \log M / \log k + 1$. $\square$

Theorem 3.4. *Let $m, n, q \geq 2$ be integers. For every $\varepsilon \in (0, 1)$, every $\mathbf{D} \in \mathbb{Z}_q^{m \times m}$ can be preprocessed in $m^{2+\varepsilon} \cdot \text{poly}(n, \log(mq))$ time so that afterwards, for every $\mathbf{X} \in \mathbb{Z}_q^{m \times n}$, the product $\mathbf{DX} \in \mathbb{Z}_q^{m \times n}$ can be computed in*

$$O\left(\frac{m^2 \log(mq)}{\varepsilon^2 \cdot \log^2 m}\right) \cdot \text{poly}(\log n, \log \log(mq)) + m \cdot \text{poly}(n, \log(mq))$$

time on an $O(\log m + \log \log q)$ -word RAM.

Proof. Let α be the constant of Theorem 2.6, and let $k := \lceil n^{1/\alpha} \rceil = \text{poly}(n)$, so that $k \geq 2$ and $\lfloor k^\alpha \rfloor = n$. If $m < k^2 = \text{poly}(n)$, the preprocessing merely stores $\mathbf{D}$, and each product $\mathbf{DX}$ is computed entry by entry in $O(m^2 n) \cdot \text{polylog}(q) = m \cdot \text{poly}(n, \log(mq))$ time, so the theorem holds in this case. Otherwise pad $\mathbf{D}$ and $\mathbf{X}$ with zeros so that $m = kb$ for an integer b , at most doubling m . Since $m \geq k^2$, we have $b \geq \sqrt{m} \geq k \geq 2$ and $\log b \geq \frac{1}{2} \log m$.

Now lift $\mathbf{D}$ and $\mathbf{X}$ to integer matrices $\widehat{\mathbf{D}}$ and $\widehat{\mathbf{X}}$ with entries in $\{0, 1, \dots, q-1\}$, so that $\|\widehat{\mathbf{D}}\widehat{\mathbf{X}}\|_\infty \leq M := m(q-1)^2$. Applying Lemma 3.3 to M and k gives, in $\text{poly}(n, \log(mq))$ time, primes $k \leq q_0 < \dots < q_{t-1}$ with $\prod_{i \in [t]} q_i > M$, $t \leq \log M / \log k + 1 = O(\log(mq))$, and $q_{t-1} < 16(\log M + k \log k)$, so that $\log q_i = O(\log n + \log \log(mq))$. By the Chinese Remainder Theorem, $\widehat{\mathbf{D}}\widehat{\mathbf{X}}$, and hence $\mathbf{DX} = \widehat{\mathbf{D}}\widehat{\mathbf{X}} \bmod q$, is determined by $\widehat{\mathbf{D}}\widehat{\mathbf{X}} \bmod q_i$ for $i \in [t]$. In total over the t primes, reducing the m^2 entries of $\widehat{\mathbf{D}}$ modulo the q_i costs $m^2 \cdot t \cdot \text{polylog}(q) = m^2 \cdot \text{polylog}(mq)$ in preprocessing, and reducing the mn entries of $\widehat{\mathbf{X}}$, reconstructing

the mn entries of $\widehat{\mathbf{D}}\widehat{\mathbf{X}}$ from their residues, and reducing them modulo q cost $mn \cdot \text{poly}(t, \log q_{t-1}, \log q) = m \cdot \text{poly}(n, \log(mq))$ online.

Since $q_i \geq k$, Theorem 2.6 gives, in $\text{poly}(n)$ field operations per prime, a bilinear algorithm for $\langle k, k, \lfloor k^\alpha \rfloor \rangle = \langle k, k, n \rangle$ -matrix multiplication over $\mathbb{F}_{q_i}$ of rank $r_i = O(k^2 \log^2 k) = O(k^2 \log^2 n)$, as $k \leq 2n^{1/\alpha}$. For each $i \in [t]$, by Lemma 3.1 the matrix $\widehat{\mathbf{D}} \bmod q_i$ can be preprocessed in $O(r_i \cdot m^{2+\varepsilon}) \cdot \text{polylog}(q_i) = m^{2+\varepsilon} \cdot \text{poly}(n, \log \log(mq))$ time and then multiplied by $\widehat{\mathbf{X}} \bmod q_i$ in

$$\begin{aligned} & O\left(\frac{r_i \cdot b^2}{\varepsilon^2 \cdot \log^2 b}\right) \cdot \text{polylog}(q_i) + O(r_i \cdot n \cdot m) \cdot \text{polylog}(q_i) \\ &= O\left(\frac{m^2}{\varepsilon^2 \cdot \log^2 m}\right) \cdot \text{poly}(\log n, \log \log(mq)) + m \cdot \text{poly}(n, \log \log(mq)) \end{aligned}$$

time, where the equality uses $b^2/\log^2 b \leq 4m^2/(k^2 \log^2 m)$, $r_i/k^2 = O(\log^2 n)$, and $\text{polylog}(q_i) = \text{poly}(\log n, \log \log(mq))$. Summing over the $t = O(\log(mq))$ primes gives $m^{2+\varepsilon} \cdot \text{poly}(n, \log(mq))$ preprocessing time and

$$O\left(\frac{m^2 \log(mq)}{\varepsilon^2 \cdot \log^2 m}\right) \cdot \text{poly}(\log n, \log \log(mq)) + m \cdot \text{poly}(n, \log(mq))$$

online time, which are the bounds of the theorem. The costs of the prime search, the construction of the bilinear algorithms, the reductions, and the reconstruction are within these bounds. The word size $O(\log m + \log \log q_i)$ that Lemma 3.1 needs is clearly $O(\log m + \log \log q)$. $\square$

4 LWE-Based DEPIR in the Standard Model

Theorem 4.1. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), for every $\varepsilon \in (0, 1)$ there exists a DEPIR scheme with the following efficiency on an $O(\log N + \log \lambda)$ -word RAM, where $N \geq 2$ is the database size and λ is the security parameter.*

- Preprocessing Runtime: $\widetilde{O}(N^{1+\varepsilon}) \cdot \text{poly}(\lambda)$
- Online Runtime: $O(N/(\varepsilon^2 \cdot \log^2 N)) \cdot \text{poly}(\log \lambda, \log \log N) + \widetilde{O}(\sqrt{N}) \cdot \text{poly}(\lambda)$
- Online Communication: $\widetilde{O}(\sqrt{N}) \cdot \text{poly}(\lambda)$

Proof. We show that the scheme of Figure 1 satisfies the theorem.

CORRECTNESS. Let B be the bound from Definition 2.3, so that χ is B -bounded and $B < \alpha q$. We have $\mathbf{z} = \mathbf{w} - \mathbf{H}\mathbf{s} = \mathbf{D}(\mathbf{v} - \mathbf{A}\mathbf{s}) = (\mathbf{D}\mathbf{e} + \lfloor q/p \rfloor \cdot \mathbf{D}\mathbf{u}_j) \bmod q$, so $\widehat{\mathbf{z}}[i] = z \bmod q$ for the integer $z := (\mathbf{D}\mathbf{e})[i] + \lfloor q/p \rfloor \cdot \mathbf{D}[i, j]$. Since $\widehat{\mathbf{z}}[i]$ and z differ by a multiple of q , the integers $\lfloor \frac{p}{q} \cdot \widehat{\mathbf{z}}[i] \rfloor$ and $\lfloor \frac{p}{q} \cdot z \rfloor$ differ by a multiple of p and thus have the same $\hat{k}$ -th bit, p being a power of two. Writing $\lfloor q/p \rfloor = q/p - \delta$ with $0 \leq \delta < 1$,

$$\frac{p}{q} \cdot z = \mathbf{D}[i, j] + \frac{p}{q} ((\mathbf{D}\mathbf{e})[i] - \delta \cdot \mathbf{D}[i, j]).$$

LWE-Based DEPIR in the Standard Model

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

1. Set $m = \lceil \sqrt{N/\log N} \rceil$ and $p = 2^{\lceil \log N \rceil}$, so that $m^2 \cdot \log p \geq N$.
2. Obtain parameters (n, q, χ) from Definition 2.3 for $1/\alpha = 4m \cdot p^2$.
3. Output $\text{pp} = (m, p, n, q, \chi)$.

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

1. Reshape $\text{DB} \in \{0, 1\}^N$ into $\mathbf{D} \in \{0, 1, \dots, p-1\}^{m \times m}$ so that $\text{DB}[i]$ equals the $\hat{k}$ -th bit of $\mathbf{D}[\hat{i}, \hat{j}]$, the zeroth bit being the least significant, where $\hat{i}, \hat{j} \in [m]$ and $\hat{k} \in [\log p]$ are uniquely determined by $i = \hat{i} + \hat{j} \cdot m + \hat{k} \cdot m^2$.
2. Preprocess $\mathbf{D} \bmod q$ into $\widetilde{\mathbf{D}}$ using Theorem 3.4.
3. Output $\widetilde{\text{DB}} = \widetilde{\mathbf{D}}$.

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

1. Compute $\hat{i}, \hat{j} \in [m]$ and $\hat{k} \in [\log p]$ such that $i = \hat{i} + \hat{j} \cdot m + \hat{k} \cdot m^2$.
2. Sample $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and $\mathbf{e} \leftarrow \chi^m$.
3. Compute $\mathbf{v} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e} + \lfloor q/p \rfloor \cdot \mathbf{u}_{\hat{j}}$, where $\mathbf{u}_{\hat{j}} \in \mathbb{Z}_q^m$ denotes the unit vector with a one in position $\hat{j}$ and zeros elsewhere.
4. Output $\mathbf{q} = (\mathbf{A}, \mathbf{v})$ and $\text{st} = (\mathbf{s}, \hat{i}, \hat{k})$.

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

1. Parse $\widetilde{\text{DB}}$ as $\widetilde{\mathbf{D}}$ and $\mathbf{q}$ as $(\mathbf{A}, \mathbf{v})$.
2. Compute $[\mathbf{H} \mid \mathbf{w}] \leftarrow \mathbf{D}[\mathbf{A} \mid \mathbf{v}]$ from $\widetilde{\mathbf{D}}$ using Theorem 3.4.
3. Output $\mathbf{a} = (\mathbf{H}, \mathbf{w})$.

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

1. Parse st as $(\mathbf{s}, \hat{i}, \hat{k})$ and $\mathbf{a}$ as $(\mathbf{H}, \mathbf{w})$.
2. Compute $\mathbf{z} \leftarrow \mathbf{w} - \mathbf{H}\mathbf{s}$ and lift it to $\widehat{\mathbf{z}} \in \mathbb{Z}^m$ with entries in $\{0, 1, \dots, q-1\}$.
3. Output the $\hat{k}$ -th bit of $\left\lfloor \frac{p}{q} \cdot \widehat{\mathbf{z}}[\hat{i}] \right\rfloor$ as $\mathbf{b}$.

Fig. 1. LWE-based DEPIR in the standard model.

By a union bound over the $m \leq N = \text{poly}(\lambda)$ entries of $\mathbf{e}$, we have $\|\mathbf{e}\|_\infty \leq B$ with overwhelming probability, and then $|(\mathbf{De})[i]| \leq m(p-1)B$. Since $\delta \cdot \mathbf{D}[\hat{i}, \hat{j}] < p$ as well, the second term has absolute value less than

$$\frac{p}{q} \cdot (m(p-1)B + p) \leq \frac{2mp^2B}{q} < 2mp^2 \cdot \alpha = \frac{1}{2}$$

by $B < \alpha q$ and $1/\alpha = 4m \cdot p^2$. Hence $\lfloor \frac{p}{q} \cdot z \rfloor = \mathbf{D}[\hat{i}, \hat{j}]$, whose $\hat{k}$ -th bit is $\text{DB}[i]$.

PRIVACY. For $N = \text{poly}(\lambda)$, both $1/\alpha = 4m \cdot p^2$ and m are $\text{poly}(\lambda)$, so Definition 2.3 ensures that the $\text{LWE}_{n,m,q,\chi}$ assumption holds. Under this assumption, the query $\mathbf{q} = (\mathbf{A}, \mathbf{As} + \mathbf{e} + \lfloor q/p \rfloor \cdot \mathbf{u}_j)$ is computationally indistinguishable from $(\mathbf{A}, \mathbf{r})$ with uniform $\mathbf{r}$, whose distribution does not depend on $\hat{j}$, while $\hat{i}$ and $\hat{k}$ never leave the client.

EFFICIENCY. We have $m^2 = \Theta(N/\log N)$ and $\log m = \Theta(\log N)$. Definition 2.3 with $1/\alpha = \text{poly}(N)$ gives $n = \text{poly}(\lambda, \log N)$ and $\log q = O(\log N + \log \lambda)$ in $\text{poly}(\lambda, \log N)$ time, so Setup runs in $\text{poly}(\lambda, \log N)$ time. Theorem 3.4 is applied to $[\mathbf{A} \mid \mathbf{v}]$, which has $n+1$ columns, with $\log(mq) = O(\log N + \log \lambda)$, and the word size $O(\log m + \log \log q)$ that it needs is clearly $O(\log N + \log \lambda)$.

- *Preprocessing Runtime:* By Theorem 3.4, Prep takes $m^{2+\varepsilon} \cdot \text{poly}(n, \log(mq)) = \tilde{O}(N^{1+\varepsilon}) \cdot \text{poly}(\lambda)$ time.
- *Server's Online Runtime:* By Theorem 3.4, Answer takes

$$\begin{aligned} & O\left(\frac{m^2 \log(mq)}{\varepsilon^2 \cdot \log^2 m}\right) \cdot \text{poly}(\log n, \log \log(mq)) + m \cdot \text{poly}(n, \log(mq)) \\ &= O(N/(\varepsilon^2 \cdot \log^2 N)) \cdot \text{poly}(\log \lambda, \log \log N) + \tilde{O}(\sqrt{N}) \cdot \text{poly}(\lambda) \end{aligned}$$

time, since $m^2 \log(mq)/\log^2 m = O(N(\log N + \log \lambda)/\log^3 N)$, which is $O(N \log \lambda/\log^2 N)$, and $\text{poly}(\log n, \log \log(mq)) = \text{poly}(\log \lambda, \log \log N)$.

- *Client's Online Runtime:* Sampling $\mathbf{A}$ and computing $\mathbf{As}$ in Query and $\mathbf{Hs}$ in Extract take $m \cdot n \cdot \text{polylog}(q)$ time, and sampling $\mathbf{e} \leftarrow \chi^m$ takes $m \cdot \text{poly}(\lambda, \log N)$ time by Definition 2.3. Both are $\tilde{O}(\sqrt{N}) \cdot \text{poly}(\lambda)$, and the remaining steps are within these bounds.
- *Online Communication:* $|\mathbf{q}| + |\mathbf{a}| = O(m \cdot n \cdot \log q) = \tilde{O}(\sqrt{N}) \cdot \text{poly}(\lambda)$.

□

5 Constant-Rate Compressible FHE

In this section, we recall the constant-rate compressible FHE scheme of [GH19]. The later sections use it in a non-black-box way, so we restate it in full, together with the parameter guarantees they rely on.

5.1 Definition

We define constant-rate compressible FHE following [GH19]. Our schemes require multi-hop correctness [GHV10], and we state it in Micciancio's formulation [Mic25], which captures it most cleanly.

Definition 5.1 (Constant-Rate Compressible FHE [GH19, adapted]).

A constant-rate compressible FHE scheme consists of polynomial-time algorithms (Setup, KeyGen, Enc, Eval, Comp, Dec), where KeyGen and Enc are randomized, and the others are deterministic. Random coins r are omitted unless explicitly needed.

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$: The setup algorithm takes as input the security parameter λ and outputs public parameters pp . All subsequent algorithms take pp as an implicit input.
- $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp}; r)$: The key generation algorithm takes as input public parameters pp and outputs a public key pk and a secret key sk .
- $C \leftarrow \text{Enc}_{\text{pk}}(\mu; r)$: The encryption algorithm takes as input a public key pk and a bit $\mu \in \{0, 1\}$, and outputs a ciphertext C . If $\text{pk} = \emptyset$, it deterministically outputs a trivial encryption of μ .
- $C_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(f, C_{\text{in}})$: The evaluation algorithm takes as input a public key pk , a Boolean circuit f , and a list C_{in} of input ciphertexts, and outputs a list C_{out} of ciphertexts.
- $\bar{C} \leftarrow \text{Comp}(C_0, \dots, C_{t-1})$: The compression algorithm takes as input a list $C_0, \dots, C_{t-1}$ of $t \geq 1$ ciphertexts and outputs a compressed ciphertext $\bar{C}$.
- $\mu \leftarrow \text{Dec}_{\text{sk}}(\bar{C}, t)$: The decryption algorithm takes as input a secret key sk , a compressed ciphertext $\bar{C}$, and a plaintext length $t \geq 1$, and outputs a plaintext $\mu \in \{0, 1\}^t$.

The algorithms must satisfy the following requirements.

Correctness: For ciphertext sets $\mathcal{C}_0, \mathcal{C}_1$, write $\mathcal{C}_\mu := \prod_{i \in [t]} \mathcal{C}_{\mu[i]}$ for $\mu \in \{0, 1\}^t$. Define the correctness predicates $\text{Corr}_1, \text{Corr}_2, \text{Corr}_3$, with implicit arguments $(\text{pp}, \text{pk}, \text{sk}, \mathcal{C}_0, \mathcal{C}_1)$:

$$\forall \mu \in \{0, 1\}, \forall r : \quad \text{Enc}_{\text{pk}}(\mu; r), \text{Enc}_{\emptyset}(\mu) \in \mathcal{C}_\mu. \quad (\text{Corr}_1)$$

$$\forall s, t \geq 1, \forall f : \{0, 1\}^s \rightarrow \{0, 1\}^t, \forall \mu \in \{0, 1\}^s, \forall \mathbf{C} \in \mathcal{C}_\mu : \quad \text{Eval}_{\text{pk}}(f, \mathbf{C}) \in \mathcal{C}_{f(\mu)}. \quad (\text{Corr}_2)$$

$$\forall t \geq 1, \forall \mu \in \{0, 1\}^t, \forall \mathbf{C} \in \mathcal{C}_\mu : \quad \text{Dec}_{\text{sk}}(\text{Comp}(\mathbf{C}), t) = \mu. \quad (\text{Corr}_3)$$

The following probability is overwhelming:

$$\Pr \left[\exists \mathcal{C}_0, \mathcal{C}_1 : \text{Corr}_1 \wedge \text{Corr}_2 \wedge \text{Corr}_3 \mid (\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp}) \right].$$

Privacy: We require IND-CPA security: for $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, no probabilistic polynomial-time adversary given pk can distinguish between $\text{Enc}_{\text{pk}}(0)$ and $\text{Enc}_{\text{pk}}(1)$ with non-negligible advantage.

Compactness: For every honestly generated key pair, every ciphertext output by Enc_{pk} , $\text{Enc}_{\emptyset}$, or Eval_{pk} has length at most $\text{poly}(\lambda)$.

Constant-Rate: There is an absolute constant $\rho \geq 1$ such that, for every λ , every $t \geq 1$, and all ciphertexts $C_0, \dots, C_{t-1}$,

$$|\text{Comp}(C_0, \dots, C_{t-1})| \leq \rho \cdot t + \text{poly}(\lambda).$$

5.2 Construction

We use the constant-rate compressible FHE scheme of [GH19], which is based on GSW encryption [GSW13]. Figures 2 and 5 present it essentially unchanged, except that **Setup** takes an additional input N for the size of a database or memory in the later sections. Lemma 5.2 calibrates the parameters so that the noise of ciphertexts stays within the margin we need, and gives explicit ciphertext sets that satisfy the correctness predicates. Lemma 5.3 bounds the sizes and costs in terms of N , and Proposition 5.5 shows that the scheme satisfies Definition 5.1. The later sections rely only on Figures 2 and 5 and the statements of Lemmas 5.2 and 5.3 and Proposition 5.5, not on their proofs.

Lemma 5.2. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), there is a constant c such that, for every $N \geq 2$, the parameters of Figure 2 admit a noise bound $B \geq 1$ with $q/B > 2d^7 \ell^4 N^3$. That is, for $N = \text{poly}(\lambda)$, except with negligible probability over $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, the sets $\mathcal{C}_0$ and $\mathcal{C}_1$ satisfy Corr_1 , Corr_2 , and Corr_3 of Definition 5.1, where $\mathcal{C}_\mu$ is the set of matrices $C \in \mathbb{Z}_q^{d \times d\ell}$ with*

$$\mathbf{SC} = \mu \mathbf{SG} + \mathbf{E} \pmod{q}, \quad \|\mathbf{E}\|_\infty \leq B. \quad (2)$$

Proof. We defer the proof to Appendix A. $\square$

Lemma 5.3. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), for every $N \geq 2$, the parameters of Figure 2 satisfy*

$$d = \text{poly}(\lambda, \log N), \quad \log q = O(\log(\lambda + N)), \quad b_{\text{ct}}/b_{\text{pt}} = O(1),$$

where **Comp** compresses each block of $b_{\text{pt}} = n^2 \log p$ ciphertexts into a matrix of $b_{\text{ct}} = dn\ell$ bits. Moreover, keys and bit ciphertexts have length $\text{poly}(\lambda, \log N)$, and every algorithm of Figures 2 and 5 runs in $\text{poly}(\lambda, \log N)$ time, counted per gate for Eval_{pk} and per block for **Comp** and **Dec**.

Proof. We defer the proof to Appendix A. $\square$

Since the public key contains the encryptions $\mathbf{C}_{\mathbf{S}}$ of the secret key, the security of the scheme relies on the circular security of GSW encryption, that is, of the same scheme with $\mathbf{C}_{\mathbf{S}}$ removed from the public key.

Constant-Rate Compressible FHE (Part 1)

We use the standard gadget notation, with d , ℓ , and q from **Setup**. For the gadget matrix $\mathbf{G} = \mathbf{I}_d \otimes (1, 2, \dots, 2^{\ell-1}) \in \mathbb{Z}_q^{d \times d\ell}$ and every $t \geq 1$, the gadget decomposition $\mathbf{G}^{-1} : \mathbb{Z}_q^{d \times t} \rightarrow \{0, 1\}^{d\ell \times t}$ replaces each entry by the ℓ -bit binary expansion of its representative in $\{0, 1, \dots, q-1\}$, so that $\mathbf{G}\mathbf{G}^{-1}(\mathbf{X}) = \mathbf{X}$ for every $\mathbf{X} \in \mathbb{Z}_q^{d \times t}$.

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

1. Obtain parameters (n, q, χ) from Definition 2.3 for $1/\alpha = (\lambda + N)^c$, where c is the constant of Lemma 5.2.
2. Set $d = 2n$, $\ell = \lceil \log q \rceil$, $k = d\ell + 2\lambda$, and $p = 2^{\lceil \log(\lambda + N) \rceil}$.
3. Output $\text{pp} = (n, q, \chi, d, \ell, k, p)$.

$(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$:

1. Sample $\mathbf{S}_0 \leftarrow \mathbb{Z}_q^{n \times n}$, $\mathbf{A}_0 \leftarrow \mathbb{Z}_q^{n \times k}$, and $\mathbf{E}_0 \leftarrow \chi^{n \times k}$.
2. Set $\mathbf{A} = \begin{pmatrix} \mathbf{A}_0 \\ -\mathbf{S}_0\mathbf{A}_0 + \mathbf{E}_0 \end{pmatrix} \in \mathbb{Z}_q^{d \times k}$ and $\mathbf{S} = [\mathbf{S}_0 \mid \mathbf{I}_n] \in \mathbb{Z}_q^{n \times d}$.
3. Sample $\mathbf{R}_i \leftarrow \{0, 1\}^{k \times d\ell}$ for $i \in [nd\ell]$, and set $\mathbf{C}_\mathbf{S} = (\mathbf{A}\mathbf{R}_i + \mu_\mathbf{S}[i]\mathbf{G})_{i \in [nd\ell]}$, where $\mu_\mathbf{S} \in \{0, 1\}^{nd\ell}$ is the binary encoding of $\mathbf{S}$.
4. Output $\text{pk} = (\mathbf{A}, \mathbf{C}_\mathbf{S})$ and $\text{sk} = \mathbf{S}$.

$C \leftarrow \text{Enc}_{\text{pk}}(\mu)$:

1. If $\text{pk} = \emptyset$, output $C = \mu\mathbf{G}$.
2. Otherwise, sample $\mathbf{R} \leftarrow \{0, 1\}^{k \times d\ell}$ and output $C = \mathbf{A}\mathbf{R} + \mu\mathbf{G} \in \mathbb{Z}_q^{d \times d\ell}$.

$\overline{C} \leftarrow \text{Comp}(C_0, \dots, C_{t-1})$:

1. For $i, j \in [n]$, let $\mathbf{U}_{i,j} \in \{0, 1\}^{n \times n}$ be the matrix with a one in position (i, j) and zeros elsewhere.
2. Append copies of $\text{Enc}_\emptyset(0)$ to $C_0, \dots, C_{t-1}$ so that their number is a multiple of $b_{\text{pt}} = n^2 \log p$, and split them into blocks of b_{pt} ciphertexts.
3. For each block, index its ciphertexts as $C_{i,j,r}$ in lexicographic order of $(i, j, r) \in [n] \times [n] \times [\log p]$, and compute the matrix

$$\sum_{i,j \in [n]} \sum_{r \in [\log p]} C_{i,j,r} \mathbf{G}^{-1} \left(\lfloor q/p \rfloor 2^r \begin{pmatrix} \mathbf{0} \\ \mathbf{U}_{i,j} \end{pmatrix} \right) \in \mathbb{Z}_q^{d \times n}.$$

4. Output the concatenation of these matrices in order.

$\mu \leftarrow \text{Dec}_{\text{sk}}(\overline{C}, t)$:

1. For each matrix $\mathbf{X} \in \mathbb{Z}_q^{d \times n}$ in $\overline{C}$:
 - (a) Lift $\mathbf{S}\mathbf{X}$ to $\hat{\mathbf{Y}} \in \{0, 1, \dots, q-1\}^{n \times n}$, and set $\mathbf{Z} = \lfloor (p/q)\hat{\mathbf{Y}} \rfloor$.
 - (b) For $(i, j, r) \in [n] \times [n] \times [\log p]$ in lexicographic order, take the r -th bit of $\mathbf{Z}[i, j]$, the zeroth bit being the least significant.
2. Output the first t bits taken in step 1, in order.

Fig. 2. Constant-rate compressible FHE of [GH19]. The algorithm Eval_{pk} is given in Figure 5 of Appendix A.

Definition 5.4 (Single-Key Circular Security). For a public-key encryption scheme, sample $\text{pp} \leftarrow \text{Setup}(1^\lambda)$ and $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$. Let μ_{sk} be a fixed binary encoding of sk and set $\mathbf{C}_{\text{sk}} = (\text{Enc}_{\text{pk}}(\mu_{\text{sk}}[i]))_i$. The scheme is single-key circular secure if it remains IND-CPA secure when $\mathbf{C}_{\text{sk}}$ is included in the public key, where all encryptions use independent randomness.

Proposition 5.5. Let $N \geq 2$ be a function of λ with $N = \text{poly}(\lambda)$. Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), if GSW encryption is single-key circular secure (Definition 5.4), then the scheme of Figures 2 and 5 with $\text{Setup}(1^\lambda, N)$ as the setup algorithm is a constant-rate compressible FHE scheme (Definition 5.1).

Proof. We defer the proof to Appendix A. $\square$

6 FHE with Encrypted Database Lookup

In this section, we define and construct FHE with encrypted database lookup, which serves as a stepping stone toward the RAM-FHE scheme of Section 7. After a one-time preprocessing of an encrypted database, the scheme supports reading the entry at an encrypted index in time sublinear in the database size.

6.1 Definition

We consider a database $\text{DB} \in (\{0, 1\}^w)^M$ with M entries of w bits each, and encode it as a string of $N = Mw$ bits by concatenating its entries in index order. We encode an index $a \in [M]$ by $\lceil \log M \rceil$ bits.

Definition 6.1 (FHE with Encrypted Database Lookup). A scheme for FHE with encrypted database lookup consists of polynomial-time algorithms KeyGen , Enc , and Eval with the interfaces of Definition 5.1, together with the following polynomial-time algorithms, where $N = Mw$ for the database dimensions M and w given to Setup . The algorithms KeyGen and Enc are randomized, and the others are deterministic.

- $\text{pp} \leftarrow \text{Setup}(1^\lambda, M, w)$: The setup algorithm takes as input the security parameter λ and database dimensions $M, w \geq 1$ with $N \geq 2$, and outputs public parameters pp . All subsequent algorithms take pp as an implicit input.
- $\tilde{\mathbf{C}}_{\text{DB}} \leftarrow \text{Prep}(\mathbf{C}_{\text{DB}})$: The preprocessing algorithm takes as input a bitwise encryption $\mathbf{C}_{\text{DB}}$ of a database $\text{DB} \in (\{0, 1\}^w)^M$ and outputs a preprocessed database $\tilde{\mathbf{C}}_{\text{DB}}$. It takes no key material as input.
- $\mathbf{C}_v \leftarrow \text{Read}_{\text{pk}}(\tilde{\mathbf{C}}_{\text{DB}}, \mathbf{C}_a)$: The read algorithm takes as input a public key pk , a preprocessed database $\tilde{\mathbf{C}}_{\text{DB}}$, and a bitwise encryption $\mathbf{C}_a$ of an index $a \in [M]$, and outputs bitwise ciphertexts $\mathbf{C}_v$ of $\text{DB}[a]$.
- $\mu \leftarrow \text{Dec}_{\text{sk}}(C)$: The decryption algorithm takes as input a secret key sk and a ciphertext C , and outputs a bit μ .

The algorithms must satisfy the following requirements.

Correctness: In addition to Corr_1 and Corr_2 of Definition 5.1, define the correctness predicates Corr'_3 and Corr_4 , with the same implicit arguments:

$$\forall \mu \in \{0, 1\}, \forall C \in \mathcal{C}_\mu : \text{Dec}_{\text{sk}}(C) = \mu. \quad (\text{Corr}'_3)$$

$$\forall \text{DB} \in (\{0, 1\}^w)^M, \forall \mathbf{C}_{\text{DB}} \in \mathcal{C}_{\text{DB}}, \forall a \in [M], \forall \mathbf{C}_a \in \mathcal{C}_a : \text{Read}_{\text{pk}}(\text{Prep}(\mathbf{C}_{\text{DB}}), \mathbf{C}_a) \in \mathcal{C}_{\text{DB}[a]}. \quad (\text{Corr}_4)$$

For M and w with $N = \text{poly}(\lambda)$, the following probability is overwhelming:

$$\Pr \left[\exists \mathcal{C}_0, \mathcal{C}_1 : \text{Corr}_1 \wedge \text{Corr}_2 \wedge \text{Corr}'_3 \wedge \text{Corr}_4 \mid (\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp}) \right].$$

Privacy: For M and w with $N = \text{poly}(\lambda)$, we require IND-CPA security as in Definition 5.1.

Compactness: For all λ , M , and w and every honestly generated key pair, every ciphertext C output by Enc_{pk} , $\text{Enc}_\emptyset$, Eval_{pk} , or Read_{pk} has length at most $\text{poly}(\lambda, \log N)$, and $\text{Dec}_{\text{sk}}(C)$ runs in $\text{poly}(\lambda, \log N)$ time.

6.2 Construction

Lemma 6.2. Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), the scheme of Figure 3 satisfies correctness (Definition 6.1).

Proof. We use the sets $\mathcal{C}_0, \mathcal{C}_1$ of Lemma 5.2 and condition on the event, of negligible failure probability, that they satisfy Corr_1 , Corr_2 , and Corr_3 of Definition 5.1. Since Enc and Eval of Figure 3 are those of Figures 2 and 5, Corr_1 and Corr_2 hold for Figure 3 as well. Since $\text{Dec}_{\text{sk}}(C)$ of Figure 3 applies the decryption of Figure 2 to $\text{Comp}(C)$ with $t = 1$, Corr_3 with $t = 1$ gives Corr'_3 .

It remains to show Corr_4 . Fix DB , $\mathbf{C}_{\text{DB}} \in \mathcal{C}_{\text{DB}}$, $a \in [M]$, and $\mathbf{C}_a \in \mathcal{C}_a$, and let $g = \lfloor a/\gamma \rfloor < \overline{M}$. Since $\mathbf{C}_{\mathbf{S}}$ consists of outputs of Enc_{pk} , Corr_1 and Corr_2 show that Read_{pk} returns ciphertexts in the ciphertext sets of the output of the circuit of step 5 on the underlying plaintexts. By Corr_1 and Corr_3 , decrypting $\overline{C}_g$ with length γw gives the entries of group g , whose $(a \bmod \gamma)$ -th one is the a -th entry. Since m and $g\overline{w} = \hat{i} + \hat{j}m + \hat{k}m^2$ are multiples of $\overline{w}$, the bits of $\overline{C}_g$ are stored in the entries $\mathbf{D}[\hat{i}, \hat{j}], \dots, \mathbf{D}[\hat{i} + \overline{w} - 1, \hat{j}]$, so bit r of $\overline{C}_g$ is the $\hat{k}$ -th bit of $\mathbf{D}[\hat{i} + r, \hat{j}]$ by construction. It therefore suffices to show that, for each $r \in [\overline{w}]$, the $\hat{k}$ -th bit of $\lfloor (\overline{p}/q)\hat{\mathbf{y}}[\hat{i} + r] \rfloor$ taken in step 5(b) agrees with that of $\mathbf{D}[\hat{i} + r, \hat{j}]$.

Since $\mathbf{C}_{\mathbf{u}_j} \in \mathcal{C}_{\mathbf{u}_j}$ by Corr_2 , relation (2) gives integers e_j with $|e_j| \leq d\ell B$ such that, for every $j \in [m]$,

$$\mathbf{Q}[j]\mathbf{s}^\top = \lfloor q/\overline{p} \rfloor \mathbf{u}_j[j] + e_j \pmod{q}.$$

Consequently, since $\mathbf{y} = \mathbf{H}\mathbf{s}^\top = \mathbf{D}\mathbf{Q}\mathbf{s}^\top$ and the entries of $\mathbf{D}$ are less than $\overline{p}$, for every $r \in [\overline{w}]$ there is an integer e' with $|e'| \leq m\overline{p}d\ell B$ such that

$$\mathbf{y}[\hat{i} + r] = \lfloor q/\overline{p} \rfloor \mathbf{D}[\hat{i} + r, \hat{j}] + e' \pmod{q}.$$

FHE with Encrypted Database Lookup

Let FHE_0 be the scheme of Figures 2 and 5. Reuse KeyGen , Enc , and Eval of FHE_0 , together with the following algorithms.

$\text{pp} \leftarrow \text{Setup}(1^\lambda, M, w)$:

1. Run $\text{FHE}_0.\text{Setup}(1^\lambda, Mw)$ to obtain $(n, q, \chi, d, \ell, k, p)$.
2. Set the number of entries per group to $\gamma = \max\{1, \lfloor b_{\text{pt}}/w \rfloor\}$, the number of groups to $\overline{M} = \lceil M/\gamma \rceil$, and the length of a compressed group to $\overline{w} = \lceil \gamma w / b_{\text{pt}} \rceil b_{\text{ct}}$, where $b_{\text{pt}} = n^2 \log p$ and $b_{\text{ct}} = dn\ell$.
3. Set $\overline{N} = \overline{M}\overline{w}$ and $\overline{p} = 2^{\lceil \log \overline{N} \rceil}$, and let m be the smallest multiple of $\overline{w}$ with $m^2 \log \overline{p} \geq \overline{N}$.

$\tilde{\mathbf{C}}_{\text{DB}} \leftarrow \text{Prep}(\mathbf{C}_{\text{DB}})$:

1. Pad $\mathbf{C}_{\text{DB}}$ with copies of $\text{Enc}_\emptyset(0)$ and split it into $\overline{M}$ groups of γw ciphertexts.
2. For each $g \in [\overline{M}]$, compress group g with $\text{FHE}_0.\text{Comp}$ into $\overline{C}_g$, and identify $\overline{C}_g$ with a $\overline{w}$ -bit string using ℓ bits per element of $\mathbb{Z}_q$.
3. Form $\mathbf{D} \in \{0, 1, \dots, \overline{p}-1\}^{m \times m}$ so that, for all $(\hat{i}, \hat{j}, \hat{k}) \in [m] \times [m] \times [\log \overline{p}]$, the $\hat{k}$ -th bit of $\mathbf{D}[\hat{i}, \hat{j}]$, the zeroth bit being the least significant, is the $(\hat{i} + \hat{j}m + \hat{k}m^2)$ -th bit of $\overline{C}_0 \parallel \dots \parallel \overline{C}_{\overline{M}-1}$, or 0 if there is no such bit.
4. Preprocess $\mathbf{D} \bmod q$ into $\tilde{\mathbf{C}}_{\text{DB}}$ via Theorem 3.4 if $m > \overline{w}$, or set $\tilde{\mathbf{C}}_{\text{DB}} = \mathbf{D}$ otherwise.

$\mathbf{C}_v \leftarrow \text{Read}_{\text{pk}}(\tilde{\mathbf{C}}_{\text{DB}}, \mathbf{C}_a)$:

1. Use Eval_{pk} on $\mathbf{C}_a$ to compute bitwise ciphertexts $\mathbf{C}_i$, $\mathbf{C}_j$, and $\mathbf{C}_{\hat{k}}$ of the unique $(\hat{i}, \hat{j}, \hat{k}) \in [m] \times [m] \times [\log \overline{p}]$ with $\lfloor a/\gamma \rfloor \overline{w} = \hat{i} + \hat{j}m + \hat{k}m^2$, the starting position of the compressed group $\overline{C}_{\lfloor a/\gamma \rfloor}$ containing the a -th entry.
2. Use Eval_{pk} on $\mathbf{C}_j$ to compute bitwise ciphertexts $\mathbf{C}_{\mathbf{u}_j}$ of the unit vector $\mathbf{u}_j \in \{0, 1\}^m$ with a one in position $\hat{j}$ and zeros elsewhere.
3. Let $\mathbf{Q} \in \mathbb{Z}_q^{m \times d}$ be the matrix with rows $\mathbf{Q}[j] = (\mathbf{C}_{\mathbf{u}_j}[j] \mathbf{G}^{-1}(\lfloor q/\overline{p} \rfloor \mathbf{u}_n))^T$ for $j \in [m]$, where $\mathbf{u}_n \in \{0, 1\}^d$ is the unit vector with a one in position n . This extracts a Regev ciphertext from each GSW ciphertext: each row $\mathbf{Q}[j]$ encrypts $\mathbf{u}_j[j]$ with scaling factor $\lfloor q/\overline{p} \rfloor$ under the secret $\mathbf{s}$, the first row of $\mathbf{S}$.
4. Compute $\mathbf{H} = \mathbf{D}\mathbf{Q}$ from $\tilde{\mathbf{C}}_{\text{DB}}$ via Theorem 3.4 if $m > \overline{w}$, or directly otherwise.
5. Use Eval_{pk} on $\mathbf{C}_s$, $\mathbf{C}_i$, $\mathbf{C}_{\hat{k}}$, and $\mathbf{C}_a$ to compute and output $\mathbf{C}_v$ by evaluating the following circuit, with public $\mathbf{H}$ hardwired:
 - (a) Compute $\mathbf{y} = \mathbf{H}\mathbf{s}^T$ and lift it to $\hat{\mathbf{y}} \in \mathbb{Z}^m$ with entries in $\{0, 1, \dots, q-1\}$.
 - (b) Take the $\hat{k}$ -th bit of $\lfloor (\overline{p}/q)\hat{\mathbf{y}}[\hat{i} + r] \rfloor$ for each $r \in [\overline{w}]$.
 - (c) Decrypt these $\overline{w}$ bits, which form $\overline{C}_{\lfloor a/\gamma \rfloor}$, with $\text{FHE}_0.\text{Dec}_{\text{sk}}(\cdot, \gamma w)$.
 - (d) Return the $(a \bmod \gamma)$ -th of the γ resulting entries.

$\mu \leftarrow \text{Dec}_{\text{sk}}(C)$:

1. Output $\text{FHE}_0.\text{Dec}_{\text{sk}}(\text{FHE}_0.\text{Comp}(C), 1)$.

Fig. 3. FHE with encrypted database lookup.

Writing $\lfloor q/\bar{p} \rfloor = q/\bar{p} - \delta$ with $0 \leq \delta < 1$, we have

$$\lfloor (\bar{p}/q)\hat{\mathbf{y}}[i+r] \rfloor = \mathbf{D}[i+r, j] + \lfloor (\bar{p}/q)(e' - \delta \mathbf{D}[i+r, j]) \rfloor \pmod{\bar{p}}.$$

Since $\bar{p}$ is a power of two and $\hat{k} < \log \bar{p}$, it suffices to show that the last rounding is 0, that is, that its argument has absolute value less than $1/2$. This argument has absolute value at most

$$\frac{\bar{p}}{q}(m\bar{p}d\ell B + \bar{p}) < \frac{d^7\ell^4 N^3 B}{q} < \frac{1}{2},$$

where the first inequality uses $m \leq \bar{N}$, $\bar{p} < 2\bar{N}$, and $\bar{N} \leq b_{\text{ct}}N = d^2\ell N/2$, and the second uses $q/B > 2d^7\ell^4 N^3$ from Lemma 5.2. $\square$

Proposition 6.3. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), if GSW encryption is single-key circular secure (Definition 5.4), then for every $\varepsilon \in (0, 1)$, the scheme of Figure 3 with parameter ε satisfies Definition 6.1. For $w = O(\log N)$, its algorithms have the following running times on an $O(\log N + \log \lambda)$ -word RAM.*

- Prep: $\tilde{O}(N^{1+\varepsilon}) \cdot \text{poly}(\lambda)$
- Read_{pk}: $O(N/(\varepsilon^2 \cdot \log^2 N)) \cdot \text{poly}(\log \lambda, \log \log N) + \tilde{O}(\sqrt{N}) \cdot \text{poly}(\lambda)$
- Eval_{pk}($f, \cdot$): $|f| \cdot \text{poly}(\lambda, \log N)$, where $|f|$ is the number of gates of f

Proof. The correctness, privacy, and compactness requirements of Definition 6.1 follow from Lemma 6.2, Proposition 5.5, and Lemma 5.3, respectively, so it remains to bound the running times. The word size $O(\log m + \log \log q)$ that Theorem 3.4 needs is $O(\log N + \log \lambda)$, since $m \leq \bar{N} \leq b_{\text{ct}}N$.

- Prep: First note that $\bar{w} \leq wb_{\text{ct}} = \text{poly}(\lambda, \log N)$. By the choice of m , we have $m < \sqrt{\bar{N}/\log \bar{p}} + \bar{w}$, so $m^2 = N \cdot \text{poly}(\lambda, \log N)$ since $\bar{N} \leq b_{\text{ct}}N$. Hence, by Theorem 3.4, Prep takes $m^{2+\varepsilon} \cdot \text{poly}(d, \log \log(mq)) = N^{1+\varepsilon/2} \cdot \text{poly}(\lambda, \log N)$ time, which is within $\tilde{O}(N^{1+\varepsilon}) \cdot \text{poly}(\lambda)$.
- Read_{pk}: Steps 1, 2, 3, and 5 take $m \cdot \text{poly}(\lambda, \log N)$ time, since the circuit of step 5 has $m \cdot \text{poly}(\lambda, \log N)$ gates. If $m = \bar{w}$, then $m \leq wb_{\text{ct}} = \text{poly}(\lambda, \log N)$ and step 4 computes $\mathbf{H}$ directly, so Read_{pk} takes $\text{poly}(\lambda, \log N)$ time, which is within the stated bound. Otherwise, Theorem 3.4 computes $\mathbf{H} = \mathbf{DQ}$ in

$$O\left(\frac{m^2 \log(mq)}{\varepsilon^2 \cdot \log^2 m}\right) \cdot \text{poly}(\log d, \log \log(mq)) + m \cdot \text{poly}(d, \log(mq))$$

time. Recall that m is the smallest multiple of $\bar{w}$ that is at least $m_0 = \sqrt{\bar{N}/\log \bar{p}}$. Since $m > \bar{w}$, we have $m_0 > \bar{w}$, so $m_0 \leq m < m_0 + \bar{w} < 2m_0$, and thus $m^2 = \Theta(\bar{N}/\log \bar{p})$. Moreover, $m_0 > \bar{w}$ means $\bar{N} > \bar{w}^2 \log \bar{p} \geq \bar{w}$, so $\bar{M} > 1$, that is, $M/\gamma > 1$. By the choice of γ , a group is either a single entry longer than a block or as many entries as fit into a block. Hence $\gamma w/b_{\text{pt}} > 1/2$, so each rounding in $\bar{N} = \lceil M/\gamma \rceil \lceil \gamma w/b_{\text{pt}} \rceil b_{\text{ct}}$ at most doubles its argument, and $\bar{N} \leq (4b_{\text{ct}}/b_{\text{pt}})N = O(N)$ by Lemma 5.3. Consequently, $m^2 = \Theta(N/\log N)$, and the stated bound follows as in the proof of Theorem 4.1.

- $\text{Eval}_{\text{pk}}(f, \cdot)$: By Lemma 5.3, each evaluated gate takes $\text{poly}(\lambda, \log N)$ time. $\square$

7 LWE-Based RAM-FHE

In this section, we present our main result, an LWE-based RAM-FHE scheme, which we obtain by adding writes to the FHE with encrypted database lookup of Section 6. We largely follow [LMW23], but since we only target a barely sublinear cost per step, we replace their hierarchy of levels by two levels, a preprocessed snapshot of the memory and a buffer of recent writes.

7.1 Definition

We first define RAM-FHE, following the step-circuit formulation of [LMW23] for the RAM model.

Definition 7.1 (RAM Program). *Let $M \geq 2$ be a power of two, let $w \geq 1$ be an integer, and let $N = Mw$. A RAM program Π with $T \geq 1$ steps over a memory of M words of w bits consists of a circuit Step_t of size $\text{poly}(\log N)$ for each step $t \in [T]$,*

$$(\sigma', a', v') \leftarrow \text{Step}_t(\sigma, a, v),$$

which maps a local state σ of a fixed length, an address $a \in [M]$ given by $\log M$ bits, and a word v to a next local state σ' of the same length, a next address a' , and a word v' to write. The execution of Π on an initial memory $\text{mem} \in (\{0, 1\}^w)^M$ starts with the all-zero local state σ and the address $a = 0$. For each $t \in [T]$, step t computes $(\sigma', a', v') \leftarrow \text{Step}_t(\sigma, a, \text{mem}[a])$ and sets $\text{mem}[a] \leftarrow v'$, $\sigma \leftarrow \sigma'$, and $a \leftarrow a'$. We write $\Pi(\text{mem})$ for the memory after the T steps.

Definition 7.2 (RAM-FHE). *A RAM-FHE scheme consists of polynomial-time algorithms KeyGen , Enc , and Dec with the interfaces of Definition 6.1, together with the following polynomial-time algorithms, where $N = Mw$ for the memory dimensions M and w given to Setup . The algorithms KeyGen and Enc are randomized, and the others are deterministic.*

- $\text{pp} \leftarrow \text{Setup}(1^\lambda, M, w)$: *The setup algorithm takes as input the security parameter λ and memory dimensions M, w as in Definition 7.1, and outputs public parameters pp . All subsequent algorithms take pp as an implicit input.*
- $\tilde{C}_{\text{mem}} \leftarrow \text{Prep}(C_{\text{mem}})$: *The preprocessing algorithm takes as input a bitwise encryption C_{mem} of a memory $\text{mem} \in (\{0, 1\}^w)^M$ and outputs a preprocessed memory $\tilde{C}_{\text{mem}}$. It takes no key material as input.*
- $\tilde{C}_{\Pi(\text{mem})} \leftarrow \text{Eval}_{\text{pk}}(\Pi, \tilde{C}_{\text{mem}})$: *The evaluation algorithm takes as input a public key pk , a RAM program Π over a memory of M words of w bits, and a preprocessed memory $\tilde{C}_{\text{mem}}$, and outputs a preprocessed memory $\tilde{C}_{\Pi(\text{mem})}$.*

- $C \leftarrow \text{Output}_{\text{pk}}(\tilde{C}_{\text{mem}}, \ell)$: The output algorithm takes as input a public key pk , a preprocessed memory $\tilde{C}_{\text{mem}}$, and an integer $1 \leq \ell \leq M$, and outputs bitwise ciphertexts C of the first ℓ words of the memory.

The algorithms must satisfy the following requirements.

Correctness: For M and w with $N = \text{poly}(\lambda)$ and for all $\text{mem} \in (\{0, 1\}^w)^M$, RAM programs Π , and integers $1 \leq \ell \leq M$, the following probability is overwhelming:

$$\Pr \left[\text{Dec}_{\text{sk}}(C) = \Pi(\text{mem})[:\ell] \mid \begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp}) \\ C_{\text{mem}} \leftarrow \text{Enc}_{\text{pk}}(\text{mem}) \\ \tilde{C}_{\text{mem}} \leftarrow \text{Prep}(C_{\text{mem}}) \\ \tilde{C}_{\Pi(\text{mem})} \leftarrow \text{Eval}_{\text{pk}}(\Pi, \tilde{C}_{\text{mem}}) \\ C \leftarrow \text{Output}_{\text{pk}}(\tilde{C}_{\Pi(\text{mem})}, \ell) \end{array} \right],$$

where mem is encrypted and C is decrypted bit by bit. The same holds when any bits of mem are encrypted with $\text{Enc}_{\emptyset}$ instead of Enc_{pk} .

Privacy: For M and w with $N = \text{poly}(\lambda)$, we require IND-CPA security as in Definition 6.1.⁶

Compactness: For all λ , M , and w and every honestly generated key pair, every ciphertext C output by $\text{Output}_{\text{pk}}$ has length at most $\text{poly}(\lambda, \log N)$, and $\text{Dec}_{\text{sk}}(C)$ runs in $\text{poly}(\lambda, \log N)$ time.

7.2 Construction

We present our construction in Figure 4. It represents the memory by a snapshot, preprocessed as in Figure 3, and a buffer of the writes since the snapshot was taken. A read looks up the snapshot with the read algorithm of Figure 3 and then scans the buffer, and a write appends a record to the buffer. Once the buffer holds $M_{\text{Buf}} = \lceil \sqrt{M} \rceil$ records, it is merged into the snapshot with the circuit of Lemma 7.3, and the new snapshot is preprocessed again. Thus each read scans fewer than M_{Buf} records, and each preprocessing is amortized over M_{Buf} writes.

Lemma 7.3. For $1 \leq \ell \leq M$ and $k \leq N$, there is a circuit **Apply** with $(\ell + k) \cdot \text{poly}(w, \log N)$ gates that, given a memory $\text{mem} \in (\{0, 1\}^w)^\ell$ and writes $(a_j, v_j) \in [M] \times \{0, 1\}^w$ for $j \in [k]$, returns $\text{mem}' \in (\{0, 1\}^w)^\ell$ such that, for every $a \in [\ell]$, $\text{mem}'[a] = v_j$ for the largest j with $a_j = a$, and $\text{mem}'[a] = \text{mem}[a]$ if there is no such j .

Proof. We build **Apply** from Batchers sorting networks [Bat68]. It forms a triple $(a, \text{mem}[a], 0)$ for each $a \in [\ell]$ and a triple $(a_j, v_j, j + 1)$ for each $j \in [k]$, each

⁶ Since preprocessing, evaluation, and output use only public information, this hides the memory from the evaluator's complete view, including its physical accesses.

LWE-Based RAM-FHE

Let FHE_1 be the scheme of Figure 3. Reuse **Setup**, **KeyGen**, **Enc**, and **Dec** of FHE_1 , together with the following algorithms, where $M_{\text{Buf}} = \lceil \sqrt{M} \rceil$.

$\tilde{\mathcal{C}}_{\text{mem}} \leftarrow \text{Prep}(\mathcal{C}_{\text{mem}})$:

1. Set $\mathcal{C}_{\text{snap}} = \mathcal{C}_{\text{mem}}$ and compute $\tilde{\mathcal{C}}_{\text{snap}} \leftarrow \text{FHE}_1.\text{Prep}(\mathcal{C}_{\text{snap}})$.
2. Set $\text{Snap} = (\mathcal{C}_{\text{snap}}, \tilde{\mathcal{C}}_{\text{snap}})$ and $\text{Buf} = \emptyset$.
3. Output $\tilde{\mathcal{C}}_{\text{mem}} = (\text{Snap}, \text{Buf})$.

$\tilde{\mathcal{C}}_{\Pi(\text{mem})} \leftarrow \text{Eval}_{\text{pk}}(\Pi, \tilde{\mathcal{C}}_{\text{mem}})$:

1. Set $\tilde{\mathcal{C}} = \tilde{\mathcal{C}}_{\text{mem}}$, and set $\mathcal{C}_\sigma$ and $\mathcal{C}_a$ to the trivial encryptions of the all-zero state and of address 0.
2. For $t \in [T]$:
 - (a) Set $\mathcal{C}_v \leftarrow \text{Read}_{\text{pk}}(\tilde{\mathcal{C}}, \mathcal{C}_a)$.
 - (b) Set $(\mathcal{C}_{\sigma'}, \mathcal{C}_{a'}, \mathcal{C}_{v'}) \leftarrow \text{FHE}_1.\text{Eval}_{\text{pk}}(\text{Step}_t, (\mathcal{C}_\sigma, \mathcal{C}_a, \mathcal{C}_v))$.
 - (c) Set $\tilde{\mathcal{C}} \leftarrow \text{Write}_{\text{pk}}(\tilde{\mathcal{C}}, \mathcal{C}_a, \mathcal{C}_{v'})$ and $(\mathcal{C}_\sigma, \mathcal{C}_a) \leftarrow (\mathcal{C}_{\sigma'}, \mathcal{C}_{a'})$.
3. Output $\tilde{\mathcal{C}}_{\Pi(\text{mem})} = \tilde{\mathcal{C}}$.

$\mathcal{C} \leftarrow \text{Output}_{\text{pk}}(\tilde{\mathcal{C}}, \ell)$:

1. Output $\text{FHE}_1.\text{Eval}_{\text{pk}}(\text{Apply}, (\mathcal{C}_{\text{snap}}[:\ell w], \text{Buf}))$, where **Apply** is the circuit of Lemma 7.3, which returns the given memory after the given writes, applied in order, ignoring writes to addresses outside the given memory.

$\mathcal{C}_v \leftarrow \text{Read}_{\text{pk}}(\tilde{\mathcal{C}}, \mathcal{C}_a)$:

1. Initialize $\mathcal{C}_v \leftarrow \text{FHE}_1.\text{Read}_{\text{pk}}(\tilde{\mathcal{C}}_{\text{snap}}, \mathcal{C}_a)$.
2. For $j \in [|\text{Buf}|]$:
 - (a) Let $(\mathcal{C}_{a_j}, \mathcal{C}_{v_j}) = \text{Buf}[j]$.
 - (b) Set $\mathcal{C}_v \leftarrow \text{FHE}_1.\text{Eval}_{\text{pk}}(\text{Upd}, (\mathcal{C}_a, \mathcal{C}_{a_j}, \mathcal{C}_v, \mathcal{C}_{v_j}))$, where the circuit $\text{Upd}(x, y, z_0, z_1)$ returns z_1 if $x = y$ and z_0 otherwise.
3. Output $\mathcal{C}_v$.

$\tilde{\mathcal{C}} \leftarrow \text{Write}_{\text{pk}}(\tilde{\mathcal{C}}, \mathcal{C}_a, \mathcal{C}_v)$:

1. Set $\text{Buf} \leftarrow \text{Buf} \parallel (\mathcal{C}_a, \mathcal{C}_v)$.
2. If $|\text{Buf}| = M_{\text{Buf}}$, set $\text{Snap} \leftarrow \text{Merge}_{\text{pk}}(\text{Snap}, \text{Buf})$ and $\text{Buf} \leftarrow \emptyset$.
3. Output $\tilde{\mathcal{C}} = (\text{Snap}, \text{Buf})$.

$\text{Snap} \leftarrow \text{Merge}_{\text{pk}}(\text{Snap}, \text{Buf})$:

1. Set $\mathcal{C}'_{\text{snap}} \leftarrow \text{FHE}_1.\text{Eval}_{\text{pk}}(\text{Apply}, (\mathcal{C}_{\text{snap}}, \text{Buf}))$.
2. Compute $\tilde{\mathcal{C}}'_{\text{snap}} \leftarrow \text{FHE}_1.\text{Prep}(\mathcal{C}'_{\text{snap}})$.
3. Output $(\mathcal{C}'_{\text{snap}}, \tilde{\mathcal{C}}'_{\text{snap}})$.

Fig. 4. LWE-based RAM-FHE.

consisting of an address, a word, and a timestamp. It sorts the triples by address, breaking ties by timestamp, marks the last triple of each address by comparing adjacent addresses, sorts the triples again so that the marked ones come first in address order, and outputs the words of the first ℓ triples. For each $a \in [\ell]$, the marked triple with address a is that of the latest write to a , or $(a, \text{mem}[a], 0)$ if there is none, and these ℓ triples come first, since the other marked triples have larger addresses. Each sort uses $O(n \log^2 n)$ comparators for $n = \ell + k \leq 2N$, and each comparator and each comparison in the marking has $\text{poly}(w, \log N)$ gates on triples of $w + O(\log N)$ bits, which gives $(\ell + k) \cdot \text{poly}(w, \log N)$ gates. $\square$

Lemma 7.4. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), the scheme of Figure 4 satisfies correctness (Definition 7.2).*

Proof. By Lemma 6.2, there are sets $\mathcal{C}_0, \mathcal{C}_1$ that satisfy $\text{Corr}_1, \text{Corr}_2, \text{Corr}'_3$, and Corr_4 except with negligible probability over $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, and we fix such sets. Let mem_t be the memory after the first t steps of the execution of Π on mem . We show by induction on t the invariant that, after t iterations of step 2 of Eval_{pk} , $\mathcal{C}_\sigma$ and $\mathcal{C}_a$ lie in the ciphertext sets of the state and the address after t steps, and $\tilde{\mathcal{C}} = (\text{Snap}, \text{Buf})$ with $\text{Snap} = (\mathcal{C}_{\text{snap}}, \tilde{\mathcal{C}}_{\text{snap}})$ satisfies the following for $s = t - |\text{Buf}|$: $\mathcal{C}_{\text{snap}} \in \mathcal{C}_{\text{mem}_s}$, $\tilde{\mathcal{C}}_{\text{snap}}$ is the output of Prep of Figure 3 on $\mathcal{C}_{\text{snap}}$, and Buf consists of records in the ciphertext sets of the writes of the last $|\text{Buf}|$ steps, in order. For $t = 0$, this follows from Corr_1 . If it holds after t iterations, then in the next iteration, for the current address a , step 1 of Read_{pk} returns ciphertexts in $\mathcal{C}_{\text{mem}_s[a]}$ by Corr_4 . Each evaluation of Upd in step 2 replaces the current word by the written word exactly when the record is a write to a , so by Corr_2 , the result lies in $\mathcal{C}_{\text{mem}_t[a]}$. Hence, by Corr_2 , the outputs of the evaluation of Step_t lie in the ciphertext sets of the next state, the next address, and the word to write, Write_{pk} appends the record of this write, and a merge preserves the invariant by Corr_2 and Lemma 7.3. Finally, by the invariant for $t = T$, Corr_2 , and Lemma 7.3, $\text{Output}_{\text{pk}}$ returns ciphertexts in $\mathcal{C}_{\Pi(\text{mem})[:\ell]}$, which Dec_{sk} decrypts to $\Pi(\text{mem})[:\ell]$ by Corr'_3 . $\square$

Theorem 7.5. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), if GSW encryption is single-key circular secure (Definition 5.4), then for every $\varepsilon \in (0, 1/2)$ there exists a RAM-FHE scheme (Definition 7.2) such that, for RAM programs over a memory of $N = Mw$ bits in M words of $w = O(\log N)$ bits, its algorithms have the following running times on an $O(\log N + \log \lambda)$ -word RAM.*

- *Preprocessing Runtime:* $\tilde{O}(N^{1+\varepsilon}) \cdot \text{poly}(\lambda)$
- *Online Runtime (Amortized)⁷:*

$$O\left(\frac{N}{\varepsilon^2 \cdot \log^2 N}\right) \cdot \text{poly}(\log \lambda, \log \log N) + \tilde{O}(N^{1/2+\varepsilon}) \cdot \text{poly}(\lambda) \quad (3)$$

per RAM step, plus $\ell \cdot \text{poly}(\lambda, \log N)$ for outputting ℓ words.

⁷ By Corollary 7.6, this bound also holds in the worst case.

Proof. We show that the scheme of Figure 4 with parameter ε satisfies Definition 7.2 with these bounds. The correctness requirement of Definition 7.2 follows from Lemma 7.4. The privacy and compactness requirements follow from Proposition 6.3, since Figure 4 reuses **KeyGen**, **Enc**, and **Dec** of Figure 3 and **Output_{pk}** returns the output of **Eval_{pk}** of Figure 3. Since **Prep** is one database preprocessing of Figure 3, Proposition 6.3 also gives the preprocessing runtime. It remains to bound the online runtime.

We first bound the running times of **Read_{pk}**, **Write_{pk}**, and **Output_{pk}**. Recall that, by Proposition 6.3, evaluating a circuit f takes $|f| \cdot \text{poly}(\lambda, \log N)$ time, and ciphertexts have length $\text{poly}(\lambda, \log N)$. Moreover, $\text{poly}(w, \log N) = \text{poly}(\log N)$, since $w = O(\log N)$. Step 1 of **Read_{pk}** is a read of Figure 3, and step 2 scans at most $M_{\text{Buf}} = O(\sqrt{N})$ records in $\text{poly}(\lambda, \log N)$ time each, so by Proposition 6.3, **Read_{pk}** takes time

$$O\left(\frac{N}{\varepsilon^2 \cdot \log^2 N}\right) \cdot \text{poly}(\log \lambda, \log \log N) + \sqrt{N} \cdot \text{poly}(\lambda, \log N). \quad (4)$$

Apart from calls of **Merge_{pk}**, **Write_{pk}** appends one record in $\text{poly}(\lambda, \log N)$ time. By Lemma 7.3 with $\ell = M$ and $k = M_{\text{Buf}}$, **Apply** has $M \cdot \text{poly}(\log N)$ gates, so **Merge_{pk}** evaluates it in $N \cdot \text{poly}(\lambda, \log N)$ time and, together with the preprocessing, takes $O(N^{1+\varepsilon}) \cdot \text{poly}(\lambda, \log N)$ time, independently of the inputs. By Lemma 7.3 with $k \leq M_{\text{Buf}}$, **Output_{pk}**($\cdot, \ell$) takes $(\ell + \sqrt{N}) \cdot \text{poly}(\lambda, \log N)$ time.

Each RAM step calls **Read_{pk}** and **Write_{pk}** once and evaluates **Step_t**, which has $\text{poly}(\log N)$ gates by Definition 7.1, in $\text{poly}(\lambda, \log N)$ time. Since **Write_{pk}** calls **Merge_{pk}** only once every $M_{\text{Buf}} \geq \sqrt{N/w}$ calls, the merges add $N^{1/2+\varepsilon} \cdot \text{poly}(\lambda, \log N)$ amortized time per step, so together with (4), the append, and the evaluation of **Step_t**, each step takes at most (3) amortized time. The part $\sqrt{N} \cdot \text{poly}(\lambda, \log N)$ of the running time of **Output_{pk}** is absorbed into the $T \geq 1$ steps. $\square$

7.3 Deamortization and Extensions

Deamortization. Following the incremental rebuilding of [GMOT11], we can deamortize the online runtime by spreading each merge over the writes.

Corollary 7.6. *Theorem 7.5 also holds with worst-case instead of amortized online runtime.*

Proof. We modify the scheme of Figure 4 as follows. Since the running time of **Merge_{pk}** in the proof of Theorem 7.5 does not depend on the inputs, it has a publicly computable bound $\mathcal{B}_\varepsilon(\lambda, N) = O(N^{1+\varepsilon}) \cdot \text{poly}(\lambda, \log N)$. In place of **Buf**, the preprocessed memory holds a frozen buffer **Buf_{old}**, which is being merged into **Snap**, and a current buffer **Buf_{new}**, both empty in the output of **Prep**. Reads and outputs use the records of **Buf_{old}** followed by those of **Buf_{new}** as **Buf**, and each write appends its record to **Buf_{new}** and runs the next $\lceil \mathcal{B}_\varepsilon(\lambda, N)/M_{\text{Buf}} \rceil$ machine steps of **Merge_{pk}**(**Snap**, **Buf_{old}**). When **Buf_{new}** holds M_{Buf} records, the result of

the merge replaces Snap , and Buf_{new} becomes the new Buf_{old} and is replaced by an empty buffer.

The privacy and compactness requirements and the preprocessing runtime follow as in the proof of Theorem 7.5. Correctness follows from the invariant that applying the records of Buf_{old} and then those of Buf_{new} to Snap gives the current memory. Appends preserve it. At a replacement, the merge has run for $M_{\text{Buf}} \lceil \mathcal{B}_\varepsilon(\lambda, N)/M_{\text{Buf}} \rceil \geq \mathcal{B}_\varepsilon(\lambda, N)$ machine steps over the M_{Buf} writes since Prep or the previous replacement, so it is complete. Since Snap and Buf_{old} do not change during these writes, its result is $\text{Merge}_{\text{pk}}(\text{Snap}, \text{Buf}_{\text{old}})$, a snapshot of the memory obtained by applying the records of Buf_{old} to Snap . The new Buf_{old} holds the records of the old Buf_{new} , and the new Buf_{new} is empty, so the replacement also preserves the invariant. For the running times, reads and outputs handle at most $2M_{\text{Buf}}$ records, which the bounds in the proof of Theorem 7.5 still cover, and each write takes $\text{poly}(\lambda, \log N) + \lceil \mathcal{B}_\varepsilon(\lambda, N)/M_{\text{Buf}} \rceil$ time, so each step takes at most (3) time as in that proof. $\square$

Multi-Hop Evaluation. Since the induction in the proof of Lemma 7.4 does not depend on the program, it continues over any number of evaluations, possibly of different programs, before an output. In other words, our scheme supports multi-hop evaluation. Moreover, the ciphertexts returned by $\text{Output}_{\text{pk}}$ lie in the sets $\mathcal{C}_0, \mathcal{C}_1$ used in the proof of Lemma 7.4, and this proof uses only that the input $\mathcal{C}_{\text{mem}}$ of Prep lies in $\mathcal{C}_{\text{mem}}$. Hence outputs of earlier evaluations, together with fresh or trivial encryptions, can also be preprocessed again as a new memory.

Multiple Memories. Several memories, possibly of different sizes, can be preprocessed separately and then used together by a program that fixes, for each step, which of them the step accesses. In other words, our scheme supports multi-input evaluation. It suffices to generate the parameters of Figure 2 and the key once for the size N of the largest memory and to encrypt each memory under this key. The proofs of Lemma 7.4 and Theorem 7.5 then apply to each memory separately, so each memory obeys the bounds of Theorem 7.5 for this N .

Public Databases. A public database $\text{DB} \in (\{0, 1\}^w)^M$ can serve as a memory encrypted with $\text{Enc}_\emptyset$, which the correctness requirement covers. Its preprocessed memory $\text{Prep}(\text{Enc}_\emptyset(\text{DB}))$ depends only on pp and DB , so the evaluator can compute it once and reuse it for clients with different keys. In particular, a working memory initialized to zero is the all-zero public database. Together with multiple memories, a program can thus use an encrypted input, a public database, and a working memory initialized to zero, with the latter two preprocessed in advance, as in the model of [LMW23].

References

- Bat68. Kenneth E. Batchier. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference*

- Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968. doi:10.1145/1468075.1468121. 20
- Ber14. Daniel J. Bernstein. A subfield-logarithm attack against ideal lattices. Blog post, <https://blog.cr.yp.to/20140213-ideal.html>, 2014. 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, 2
- BIPW17. Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 662–693. Springer, Cham, November 2017. doi:10.1007/978-3-319-70503-3_22. 2
- BV14. Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014*, pages 1–12. ACM, January 2014. doi:10.1145/2554797.2554799. 28
- CHR17. Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 694–726. Springer, Cham, November 2017. doi:10.1007/978-3-319-70503-3_23. 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
- Cop82. Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982. doi:10.1137/0211037. 3, 6
- Cop97. Don Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13(1):42–49, 1997. doi:10.1006/JCOM.1997.0438. 3
- 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
- Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. doi:10.1145/1536414.1536440. 1
- GH19. Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 438–464. Springer, Cham, December 2019. doi:10.1007/978-3-030-36033-7_17. 4, 11, 12, 13, 14, 28
- GHV10. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-Hop homomorphic encryption and rerandomizable Yao circuits. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 155–172. Springer, Berlin, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_9. 12
- GMOT11. Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Oblivious RAM simulation with efficient worst-case access overhead. In Christian Cachin and Thomas Ristenpart, editors, *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW*

- 2011, Chicago, IL, USA, October 21, 2011, pages 95–100. ACM, 2011. doi:10.1145/2046660.2046680. 23
- GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4_5. 4, 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
- HHWW19. Ariel Hamlin, Justin Holmgren, Mor Weiss, and Daniel Wichs. On the plausibility of fully homomorphic encryption for RAMs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 589–619. Springer, Cham, August 2019. doi:10.1007/978-3-030-26948-7_21. 1
- HJO26. Jianhua Hou, Haodong Jiang, and Tabitha Ogilvie. Careful with the ring! concrete hardness gaps between LWE and MLWE. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part III*, volume 16802 of *Lecture Notes in Computer Science*, pages 462–497. Springer, 2026. doi:10.1007/978-3-032-35377-1_15. 2
- 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. 2
- 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. 8
- Lee26. Keewoo Lee. Barely doubly-efficient SimplePIR. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part X*, volume 16809 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2026. doi:10.1007/978-3-032-35428-0_16. 2, 3, 4, 8
- LG12. François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 514–523. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.80. 3
- LGU18. François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1029–1046. SIAM, 2018. doi:10.1137/1.9781611975031.67. 3

- 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. 1, 2, 4, 19, 24
- 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. 1, 2
- LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *DCC*, 75(3):565–599, 2015. doi:10.1007/s10623-014-9938-4. 2
- LW17. Kasper Green Larsen and R. Ryan Williams. Faster online matrix-vector multiplication. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2182–2189. SIAM, 2017. doi:10.1137/1.9781611974782.142. 3
- Mic25. Daniele Micciancio. Fully composable homomorphic encryption. *CiC*, 2(1):1, 2025. doi:10.62056/ak5w186bm. 12
- NIS24. NIST. Module-lattice-based key-encapsulation mechanism standard. National Institute of Standards and Technology, Federal Information Processing Standards Publication 203, 2024. doi:10.6028/NIST.FIPS.203. 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
- RAD78. Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, editors, *Foundations of Secure Computation*, pages 169–179. Academic Press, 1978. 1
- 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, 3, 5
- Str69. Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411. 3, 6
- VXXZ24. Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134. 3
- 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, 6
- Wil14. Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903. 6

A Supplement to Section 5

In this section, we give the evaluation algorithm of the scheme of Section 5.2 (Figure 5) and prove Lemmas 5.2 and 5.3 as well as Proposition 5.5. The bootstrapping ($\text{Refresh}_{\text{pk}}$) exploits the asymmetric noise growth of GSW, following [BV14]. The proofs are standard, and we include them for completeness.

Constant-Rate Compressible FHE (Part 2)

$C_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(f, C_{\text{in}})$:

1. Write f as a circuit of NAND gates. Assign the ciphertexts of C_{in} to the input wires, and the trivial encryptions $\mathbf{0}$ and $\mathbf{G}$ to the constants 0 and 1.
2. For each NAND gate in topological order, assign $\text{Refresh}_{\text{pk}}(\mathbf{G} - C_0 \mathbf{G}^{-1}(C_1))$ to its output wire, where C_0, C_1 are the ciphertexts on its input wires.
3. Output the ciphertexts on the output wires of f , in order.

$C' \leftarrow \text{Refresh}_{\text{pk}}(C)$ for a ciphertext C :

1. Let φ_C be the decryption formula of C from Lemma A.1.
2. Output $\text{Sel}_{\varphi_C}(\mathbf{0}, \mathbf{G})$.

$C' \leftarrow \text{Sel}_{\varphi}(C_0, C_1)$ for a NAND formula φ on ndl variables and ciphertexts C_0, C_1 :

1. If φ is a constant $b \in \{0, 1\}$, output C_b .
2. If φ is a variable x_j , output $C_0 + \mathbf{C}_{\mathbf{S}}[j] \mathbf{G}^{-1}(C_1 - C_0)$.
3. If φ is the NAND of φ_0 and φ_1 , output $\text{Sel}_{\varphi_0}(C_1, \text{Sel}_{\varphi_1}(C_1, C_0))$.

Fig. 5. Constant-rate compressible FHE of [GH19]. The other algorithms are given in Figure 2 of Section 5.2.

Lemma A.1. *Consider the scheme of Figure 2. There is $L = \text{poly}(d, \ell)$ such that every ciphertext $C \in \mathbb{Z}_q^{d \times dl}$ has a NAND formula φ_C on ndl variables, called the decryption formula of C , with the following properties.*

- For every secret key $\mathbf{S}$ and every $\mu \in \{0, 1\}$ with $\mathbf{S}C = \mu \mathbf{S}\mathbf{G} + \mathbf{E} \pmod{q}$ and $\|\mathbf{E}\|_{\infty} \leq q/8$, we have $\varphi_C(\mu_{\mathbf{S}}) = \mu$, where $\mu_{\mathbf{S}} \in \{0, 1\}^{ndl}$ is the binary encoding of $\mathbf{S}$.
- φ_C is computable from C in $\text{poly}(d, \ell)$ time and has at most L leaves.

Proof. Let $h = \lfloor \log(q/2) \rfloor \in [\ell]$, let $\Delta = 2^h$, and let $\mathbf{c}$ be column $nl + h$ of C . Fix $\mathbf{S}$, μ , and $\mathbf{E}$ as in the first item, and let $\mathbf{s}$ and $\mathbf{e}$ be the first rows of $\mathbf{S}$ and $\mathbf{E}$. Column $nl + h$ of $\mathbf{G}$ has Δ in row n and zeros elsewhere, and $\mathbf{s}[n] = 1$ since $\mathbf{S} = [\mathbf{S}_0 \mid \mathbf{I}_n]$, so $\mathbf{s}\mathbf{c} = \mu\Delta + \mathbf{e}[nl + h] \pmod{q}$ with $|\mathbf{e}[nl + h]| \leq q/8$. Since

$q/8 < \Delta/2$ and $3\Delta/2 < 7q/8$, the residue $\mathbf{sc} \bmod q$ lies in $(\Delta/2, 3\Delta/2)$ exactly when $\mu = 1$. It therefore suffices to build φ_C that, given the bits of $\mathbf{s}$, outputs 1 exactly when $\mathbf{sc} \bmod q \in (\Delta/2, 3\Delta/2)$.

For $j \in [d]$ and $r \in [\ell]$, let $b_{j,r}$ be the variable for bit r of $\mathbf{s}[j]$, and let $a_{j,r} = (2^r \mathbf{c}[j] \bmod q) \in \{0, 1, \dots, q-1\}$. Then $v = \sum_{j \in [d]} \sum_{r \in [\ell]} a_{j,r} b_{j,r}$ agrees with $\mathbf{sc}$ modulo q and satisfies $0 \leq v < d\ell q$, so φ_C must output 1 exactly when $2uq + \Delta < 2v < 2uq + 3\Delta$ for some $u \in [d\ell]$. We compute this test with a bounded-fan-in circuit of depth $O(\log(d\ell))$ whose constants are computable from $\mathbf{c}$ in $\text{poly}(d, \ell)$ time. Each term $a_{j,r} b_{j,r}$ is the bitwise AND of $a_{j,r}$ with $b_{j,r}$. Carry-save addition reduces the $d\ell$ terms to two integers whose sum is v , in depth $O(\log(d\ell))$, and a carry-lookahead adder adds these two in depth $O(\log(\ell + \log(d\ell)))$. The $d\ell$ interval tests compare $2v$ with constants in parallel in depth $O(\log(\ell + \log(d\ell)))$, and a balanced OR combines their outcomes in depth $O(\log(d\ell))$. Writing the gates of this circuit with NAND gates and unfolding it gives a formula, computable from C in $\text{poly}(d, \ell)$ time, with at most L leaves, where $L = 2^{O(\log(d\ell))} = \text{poly}(d, \ell)$ depends only on d and ℓ . $\square$

Lemma A.2. *Consider the scheme of Figures 2 and 5. Let B_χ be the bound from Definition 2.3, let $B = Ld\ell k B_\chi$, where L is the leaf bound of Lemma A.1, and condition on $\|\mathbf{E}_0\|_\infty \leq B_\chi$, which holds except with negligible probability for $N = \text{poly}(\lambda)$. Then for every $\mu \in \{0, 1\}$ and every $C \in \mathbb{Z}_q^{d \times d\ell}$ with*

$$\mathbf{SC} = \mu \mathbf{SG} + \mathbf{E} \pmod{q}, \quad \|\mathbf{E}\|_\infty \leq q/8,$$

the ciphertext $C' \leftarrow \text{Refresh}_{\text{pk}}(C)$ satisfies

$$\mathbf{SC}' = \mu \mathbf{SG} + \mathbf{E}' \pmod{q}, \quad \|\mathbf{E}'\|_\infty \leq B.$$

Moreover, $\text{Refresh}_{\text{pk}}$ runs in $\text{poly}(d, \ell)$ time.

Proof. For a NAND formula φ on $nd\ell$ variables, let $|\varphi|$ be its number of leaves. We show by induction on φ that, for all $C_0, C_1 \in \mathbb{Z}_q^{d \times d\ell}$, there is $\mathbf{E}_\varphi$ with

$$\mathbf{S} \text{Sel}_\varphi(C_0, C_1) = \mathbf{SC}_{\varphi(\mu_{\mathbf{S}})} + \mathbf{E}_\varphi \pmod{q}, \quad \|\mathbf{E}_\varphi\|_\infty \leq |\varphi| d\ell k B_\chi.$$

Constants give $\mathbf{E}_\varphi = \mathbf{0}$. For $\varphi = x_j$, the ciphertext $\mathbf{C}_{\mathbf{S}}[j] = \mathbf{A}\mathbf{R}_j + \mu_{\mathbf{S}}[j]\mathbf{G}$ satisfies $\mathbf{SC}_{\mathbf{S}}[j] = \mu_{\mathbf{S}}[j]\mathbf{SG} + \mathbf{E}_0\mathbf{R}_j \pmod{q}$ with $\|\mathbf{E}_0\mathbf{R}_j\|_\infty \leq k B_\chi$, so $\mathbf{G}\mathbf{G}^{-1}(\mathbf{X}) = \mathbf{X}$ gives

$$\mathbf{S}(C_0 + \mathbf{C}_{\mathbf{S}}[j]\mathbf{G}^{-1}(C_1 - C_0)) = \mathbf{SC}_{\mu_{\mathbf{S}}[j]} + \mathbf{E}_0\mathbf{R}_j\mathbf{G}^{-1}(C_1 - C_0) \pmod{q},$$

where the last term has norm at most $d\ell k B_\chi$. For φ the NAND of φ_0 and φ_1 , the induction hypothesis for φ_1 with endpoints (C_1, C_0) gives $\mathbf{S} \text{Sel}_{\varphi_1}(C_1, C_0) = \mathbf{SC}_{1-\varphi_1(\mu_{\mathbf{S}})} + \mathbf{E}_{\varphi_1} \pmod{q}$. The induction hypothesis for φ_0 with endpoints $(C_1, \text{Sel}_{\varphi_1}(C_1, C_0))$ then shows that, modulo q , $\mathbf{S} \text{Sel}_\varphi(C_0, C_1)$ is $\mathbf{SC}_1 + \mathbf{E}_{\varphi_0}$ if $\varphi_0(\mu_{\mathbf{S}}) = 0$, and $\mathbf{SC}_{1-\varphi_1(\mu_{\mathbf{S}})} + \mathbf{E}_{\varphi_1} + \mathbf{E}_{\varphi_0}$ if $\varphi_0(\mu_{\mathbf{S}}) = 1$. Since $\varphi = 1 - \varphi_0\varphi_1$, both cases give the claim with $\mathbf{E}_\varphi = \mathbf{E}_{\varphi_0} + \varphi_0(\mu_{\mathbf{S}})\mathbf{E}_{\varphi_1}$, of norm at most $(|\varphi_0| + |\varphi_1|)d\ell k B_\chi = |\varphi|d\ell k B_\chi$.

By Lemma A.1, $\varphi_C(\mu_S) = \mu$ and $|\varphi_C| \leq L$, so the claim above with $C_0 = \mathbf{0}$ and $C_1 = \mathbf{G}$ gives $\mathbf{SC}' = \mu\mathbf{SG} + \mathbf{E}_{\varphi_C} \pmod{q}$ with $\|\mathbf{E}_{\varphi_C}\|_\infty \leq Ld\ell k B_\chi = B$. The formula φ_C is computable in $\text{poly}(d, \ell)$ time by Lemma A.1, and Sel_{φ_C} spends $\text{poly}(d, \ell)$ time on each of its at most L leaves, so $\text{Refresh}_{\text{pk}}$ runs in $\text{poly}(d, \ell)$ time. $\square$

Lemma 5.2. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), there is a constant c such that, for every $N \geq 2$, the parameters of Figure 2 admit a noise bound $B \geq 1$ with $q/B > 2d^7\ell^4 N^3$. That is, for $N = \text{poly}(\lambda)$, except with negligible probability over $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\text{pp})$, the sets C_0 and C_1 satisfy Corr_1 , Corr_2 , and Corr_3 of Definition 5.1, where C_μ is the set of matrices $C \in \mathbb{Z}_q^{d \times d\ell}$ with*

$$\mathbf{SC} = \mu\mathbf{SG} + \mathbf{E} \pmod{q}, \quad \|\mathbf{E}\|_\infty \leq B. \quad (2)$$

Proof. Let $B = Ld\ell k B_\chi \geq 1$ be the bound of Lemma A.2, which holds for every choice of c . We fix c so that, for every $N \geq 2$,

$$\frac{q}{B} > 2d^7\ell^4(\lambda + N)^3,$$

which implies the bound $q/B > 2d^7\ell^4 N^3$ of the statement. This is possible because $q/B_\chi > (\lambda + N)^c$ by Definition 2.3, while B/B_χ and $2d^7\ell^4(\lambda + N)^3$ are bounded by polynomials in c and $\lambda + N$ whose degrees do not depend on c . Indeed, by Definition 2.3 and Lemma A.1, d , ℓ , and $B/B_\chi = Ld\ell k$ are bounded by fixed polynomials in λ and $\log(1/\alpha) = c \log(\lambda + N)$.

Let $N = \text{poly}(\lambda)$, and condition on $\|\mathbf{E}_0\|_\infty \leq B_\chi$, which holds except with negligible probability.

Corr₁. Since $\mathbf{SA} = \mathbf{E}_0$, for every $\mu \in \{0, 1\}$ and $\mathbf{R} \in \{0, 1\}^{k \times d\ell}$, the ciphertext $C = \mathbf{AR} + \mu\mathbf{G}$ of Enc_{pk} satisfies

$$\mathbf{SC} = \mu\mathbf{SG} + \mathbf{E}_0\mathbf{R} \pmod{q}, \quad \|\mathbf{E}_0\mathbf{R}\|_\infty \leq k B_\chi \leq B,$$

and the trivial encryption $\mu\mathbf{G}$ satisfies (2) with zero error.

Corr₂. We show by induction over the gates that Eval_{pk} assigns to every wire a ciphertext satisfying (2) for the bit on that wire. The input ciphertexts satisfy (2) by assumption, and the trivial encryptions $\mathbf{0}$ and $\mathbf{G}$ of the constants have zero error. For a NAND gate whose input wires carry ciphertexts C_0, C_1 satisfying (2) for bits μ_0, μ_1 with errors $\mathbf{E}_{C_0}, \mathbf{E}_{C_1}$, respectively, we have

$$\mathbf{S}(\mathbf{G} - C_0\mathbf{G}^{-1}(C_1)) = (1 - \mu_0\mu_1)\mathbf{SG} - \mu_0\mathbf{E}_{C_1} - \mathbf{E}_{C_0}\mathbf{G}^{-1}(C_1) \pmod{q},$$

with error of norm at most $(d\ell+1)B < q/8$, since $q/B > 2d^7\ell^4 N^3$. By Lemma A.2, the ciphertext $\text{Refresh}_{\text{pk}}(\mathbf{G} - C_0\mathbf{G}^{-1}(C_1))$ assigned to its output wire satisfies (2) for the bit $1 - \mu_0\mu_1$, the NAND of μ_0 and μ_1 .

Corr₃. The padding $\text{Enc}_\emptyset(0) = \mathbf{0}$ satisfies (2) for bit 0 with zero error, so every block of Comp consists of $b_{\text{pt}} = n^2 \log p$ ciphertexts $C_{i,j,r}$ with $(i, j, r) \in [n] \times$

$[n] \times [\log p]$, satisfying (2) for bits $\mu_{i,j,r}$. Let $\mathbf{X}$ be the matrix that **Comp** computes from such a block. Since $\mathbf{S}(\mathbf{U}_{i,j}^0) = \mathbf{U}_{i,j}$ and $\mathbf{G}\mathbf{G}^{-1}(\cdot)$ is the identity, we have

$$\mathbf{S}\mathbf{X} = \lfloor q/p \rfloor \mathbf{M} + \mathbf{E} \pmod{q}, \quad \mathbf{M}[i,j] = \sum_{r \in [\log p]} 2^r \mu_{i,j,r},$$

where $\mathbf{E}$ is a sum of b_{pt} errors of (2), each multiplied on the right by a binary matrix with $d\ell$ rows, so $\|\mathbf{E}\|_\infty \leq b_{\text{pt}}d\ell B$. Write $\lfloor q/p \rfloor = q/p - \delta$ with $0 \leq \delta < 1$. The integer lift $\hat{\mathbf{Y}}$ of $\mathbf{S}\mathbf{X}$ in Dec_{sk} satisfies, for some $\mathbf{Q} \in \mathbb{Z}^{n \times n}$,

$$\frac{p}{q} \hat{\mathbf{Y}} = \mathbf{M} + p\mathbf{Q} + \frac{p}{q}(\mathbf{E} - \delta\mathbf{M}).$$

Since every entry of $\mathbf{M}$ lies in $\{0, 1, \dots, p-1\}$, the last term has norm at most $(p/q)(b_{\text{pt}}d\ell B + p) \leq p^2 b_{\text{pt}}d\ell B/q$, where the second step uses $b_{\text{pt}}d\ell B \geq 2$ and $p \geq 2$. Since $b_{\text{pt}} = n^2 \log p \leq d^2 p/4$ and $p = 2^{\lceil \log(\lambda+N) \rceil} < 2(\lambda+N)$, this is less than $2(\lambda+N)^3 d^3 \ell B/q < 1/2$ by $q/B > 2d^7 \ell^4 (\lambda+N)^3$. Rounding therefore gives the matrix $\mathbf{Z} = \mathbf{M} + p\mathbf{Q}$ of Dec_{sk} , and since p is a power of two, the lowest $\log p$ bits of its entries are those of $\mathbf{M}$, which are the bits $\mu_{i,j,r}$. Hence Dec_{sk} recovers every block, and returning the first t bits gives exactly the original t -bit plaintext. $\square$

Lemma 5.3. *Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), for every $N \geq 2$, the parameters of Figure 2 satisfy*

$$d = \text{poly}(\lambda, \log N), \quad \log q = O(\log(\lambda + N)), \quad b_{\text{ct}}/b_{\text{pt}} = O(1),$$

where **Comp** compresses each block of $b_{\text{pt}} = n^2 \log p$ ciphertexts into a matrix of $b_{\text{ct}} = dn\ell$ bits. Moreover, keys and bit ciphertexts have length $\text{poly}(\lambda, \log N)$, and every algorithm of Figures 2 and 5 runs in $\text{poly}(\lambda, \log N)$ time, counted per gate for Eval_{pk} and per block for **Comp** and **Dec**.

Proof. Since $\log(1/\alpha) = c \log(\lambda + N)$ for the constant c of Lemma 5.2, Definition 2.3 gives $n = \text{poly}(\lambda, \log N)$ and $\log q = O(\log(\lambda + N))$, so $d = 2n = \text{poly}(\lambda, \log N)$ and $\ell = O(\log(\lambda + N))$. Since $b_{\text{ct}} = dn\ell = 2n^2\ell$, $b_{\text{pt}} = n^2 \log p$, and $\log p = \lceil \log(\lambda + N) \rceil$, we get $b_{\text{ct}}/b_{\text{pt}} = 2\ell/\log p = O(1)$. The algorithm of Definition 2.3 and the matrix dimensions in Figure 2 give the stated lengths and the running times of setup, key generation, and encryption. Each NAND gate computes $\mathbf{G} - C_0 \mathbf{G}^{-1}(C_1)$ in $\text{poly}(d, \ell)$ time and calls $\text{Refresh}_{\text{pk}}$ once, which takes $\text{poly}(d, \ell)$ time by Lemma A.2, and $\text{poly}(d, \ell) = \text{poly}(\lambda, \log N)$. Compression and decryption perform $\text{poly}(d, \ell, \log p) = \text{poly}(\lambda, \log N)$ arithmetic operations on ℓ -bit integers per block, so they run in $\text{poly}(\lambda, \log N)$ time per block. $\square$

Proposition 5.5. *Let $N \geq 2$ be a function of λ with $N = \text{poly}(\lambda)$. Under the LWE with polynomial modulus-to-noise ratio assumption (Definition 2.3), if GSW encryption is single-key circular secure (Definition 5.4), then the scheme of Figures 2 and 5 with $\text{Setup}(1^\lambda, N)$ as the setup algorithm is a constant-rate compressible FHE scheme (Definition 5.1).*

Proof. By Lemma 5.3 and $N = \text{poly}(\lambda)$, all algorithms run in polynomial time.

CORRECTNESS. By Lemma 5.2, with overwhelming probability, the sets $\mathcal{C}_0, \mathcal{C}_1$ given there satisfy Corr_1 , Corr_2 , and Corr_3 .

PRIVACY. The algorithm Enc_{pk} is GSW encryption, and the public key consists of the GSW public key $\mathbf{A}$ together with GSW encryptions $\mathbf{C}_{\mathbf{S}}$ of the binary encoding of $\text{sk} = \mathbf{S}$ under independent randomness. Hence IND-CPA security follows from the single-key circular security of GSW encryption (Definition 5.4).

COMPACTNESS. By Lemma 5.3, every ciphertext output by Enc_{pk} , $\text{Enc}_{\emptyset}$, or Eval_{pk} has length $\text{poly}(\lambda, \log N) = \text{poly}(\lambda)$.

CONSTANT-RATE. For every $t \geq 1$ and all ciphertexts $C_0, \dots, C_{t-1}$, Comp outputs one matrix of b_{ct} bits per block of b_{pt} ciphertexts after padding, so

$$|\text{Comp}(C_0, \dots, C_{t-1})| = b_{\text{ct}} \lceil t/b_{\text{pt}} \rceil \leq \frac{b_{\text{ct}}}{b_{\text{pt}}} t + b_{\text{ct}}.$$

By Lemma 5.3, $b_{\text{ct}}/b_{\text{pt}}$ is at most a constant $\rho \geq 1$, and $b_{\text{ct}} = dn\ell = \text{poly}(\lambda, \log N) = \text{poly}(\lambda)$ since $N = \text{poly}(\lambda)$. Hence $|\text{Comp}(C_0, \dots, C_{t-1})| \leq \rho \cdot t + \text{poly}(\lambda)$. $\square$