Laconic Function Evaluation and ABE for RAMs from (Ring-)LWE

Fangqi Dong[*] IIIS, Tsinghua University
Zihan Hao[†] IIIS, Tsinghua University
Ethan Mook[‡] Northeastern University
Hoeteck Wee[§] NTT Research and ENS, Paris
Daniel Wichs[¶] Northeastern University and NTT Research

June 5, 2024

Abstract

Laconic function evaluation (LFE) allows us to compress a circuit f into a short digest. Anybody can use this digest as a public-key to efficiently encrypt some input x. Decrypting the resulting ciphertext reveals the output $f(x)$, while hiding everything else about x. In this work we consider LFE for *Random-Access Machines* (RAM-LFE) where, instead of a circuit f, we have a RAM program f_{DB} that potentially contains some large hard-coded data DB. The decryption run-time to recover $f_{\mathsf{DB}}(x)$ from the ciphertext should be roughly the same as a plain evaluation of $f_{\mathsf{DB}}(x)$ in the RAM model, which can be sublinear in the size of DB. Prior works constructed LFE for circuits under LWE, and RAM-LFE under indisitinguishability obfuscation (iO) and Ring-LWE. In this work, we construct RAM-LFE with essentially optimal encryption and decryption run-times from just Ring-LWE and a standard circular security assumption, without iO.

RAM-LFE directly yields 1-key succinct functional encryption and reusable garbling for RAMs with similar parameters.

If we only want an *attribute-based* LFE for RAMs (RAM-AB-LFE), then we can replace Ring-LWE with plain LWE in the above. Orthogonally, if we only want *leveled* schemes, where the encryption/decryption efficiency can scale with the depth of the RAM computation, then we can remove the need for a circular-security. Lastly, we also get a leveled many-key *attribute-based encryption for RAMs (RAM-ABE)*, from LWE.

[*]dfq20@mails.tsinghua.edu.cn
[†]haozh20@mails.tsinghua.edu.cn
[‡]mook.e@northeastern.edu
[§]wee@di.ens.fr
[¶]wichs@ccs.neu.edu Research supported by NSF grant CNS-1750795, CNS-2055510 and the JP Morgan Faculty Research Award.

Contents

1 Introduction **1**
- 1.1 Prior Work on LFE for TMs and RAMs 2
- 1.2 Our Results and Applications 3
- 1.3 Our Techniques 5

2 Preliminaries **9**
- 2.1 Learning with Errors and Lattice Tools 10
- 2.2 The GSW FHE Scheme 11
- 2.3 RAM-FHE 12
- 2.4 Garbled Circuits 13

3 RAM Circuits and RAM Programs **14**

4 Homomorphic Operations for RAM Circuits **17**
- 4.1 Database Read Gates 20
- 4.2 Matrix Read Gates 22
- 4.3 Wire Read Gates 24
- 4.4 Bootstrapping 25
- 4.5 Putting it all together 28

5 RAM-LFE **29**
- 5.1 RAM AB-LFE 31
- 5.2 Upgrading to full RAM-LFE 35

6 RAM-ABE **38**

A Laconic Encryption **45**

1 Introduction

Laconic Function Evaluation. Laconic function evaluation (LFE) [QWW18,CDG+17], deterministically generates a short digest $\mathsf{dig} := \mathsf{Hash}(f)$ for a boolean function f. The digest can be used as public key to encrypt some input x resulting in a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$. Encryption should be very efficient, ideally linear in $|x|$, and independent of the complexity of f. Anybody can decrypt the resulting ciphertext ct using only knowledge of the function f to recover the output $f(x) := \mathsf{Dec}(f, \mathsf{ct})$, but the ciphertext should hide all other information about the input x. In short, LFE allows one to only reveal $f(x)$ without computing f. Security is formalized via the simulation paradigm, which dictates that the ciphertext can be simulated given only the function f and the output $f(x)$, but without any additional knowledge of x.[1]

We can think of LFE as a "flipped" version of fully homomorphic encryption (FHE). Consider a powerful server who has some function f and a weak client who holds a private input x, but does not have the computational power to compute f; they want to run a 2-round secure protocol to compute $f(x)$ by having the server do all the work. FHE provides a solution where the client gets the output: the client encrypts x, the server homomorphically computes an encryption of $f(x)$, and the client decrypts. LFE provides a solution where the server learns the output: the server computes the digest of f, the client encrypts x, and the server decrypts $f(x)$. Moreover, the first round is reusable and many different clients can non-interactively encrypt various inputs x_i under the digest to ensure the server only learns the function outputs $f(x_i)$.

As an example application described in [DHMW24], consider a scenario where the FBI has a database of most-wanted suspects. There are many security cameras in public spaces with very limited computational power, and we want to allow the FBI to learn when one of the suspects passes in front of one the cameras, but not to learn anything else about what the cameras are observing. LFE gives a simple non-interactive solution to this problem. The FBI publishes a digest dig for the function f that contains a hard-coded database of suspects and, given an image x, it outputs whether x has a match in the database. The cameras periodically capture images x of their surroundings, encrypt them under the digest dig, and send the ciphertexts to the FBI, which then only learns if a suspect passed in front of the camera but nothing else.

LFE implies succinct 1-key *functional encryption* (FE), which is in turn known to imply reusable garbled circuits [GKP+13b]. It also implies a form of *online-efficient* multiparty computation, where most of the computation can be done offline before the protocol starts and after it ends, but the computation during entire online part of the protocol is independent of the complexity of f.

Prior Work on LFE for Circuits. The work of [QWW18] constructed *leveled LFE for boolean circuits* under the learning-with-errors (LWE) assumption. For a boolean function f represented by a circuit of size $|f|$ depth d and 1-bit output, the server's run-time to generate the digest and to decrypt is $|f| \cdot \mathrm{poly}(d)$, the digest size is $O(1)$, and the client's run-time (to encrypt) is $|x| \cdot \mathrm{poly}(d)$, which also upper bounds the size of the CRS and the ciphertext size.[2] The recent work of [HLL23] shows how to get a fully succinct (non-leveled) LFE scheme that removes all of the above $\mathrm{poly}(d)$

[1]For the above to be possible we also need to have a short *common random string (CRS)* that is given as an input to all the algorithms, but for simplicity we often omit it in the introduction.

[2]The scheme is selectively secure where the input x cannot be chosen depending on the CRS. Selective security will be the default notion throughout the introduction. We also ignore fixed polynomial factors in the security parameter as well as other polylogarithmic factors throughout the introduction.

factors via a "bootstrapping procedure", by relying on the same circular-security assumption as is needed to bootstrap FHE.

1.1 Prior Work on LFE for TMs and RAMs

LFE for TMs. The work of [DGM23] constructed LFE for Turing Machines assuming indistinguishability obfuscation (iO) and somewhere statistically binding hash functions. In particular, for a boolean function f represented as a Turing Machine of description-size $|f|$, the server's run-time to generate a digest is $O(|f|)$, the size of the digest is $O(1)$, the client run time (to encrypt) is $O(|x|)$, and the server run time to decrypt is $O(T)$, where T is the run-time of the computation. The main advantage of LFE for TMs over LFE for circuits is in the smaller run-time to generate the digest, which doesn't depend on the input size or the run-time of the computation.

LFE for RAMs. The recent work of [DHMW24] defined the notion of LFE for RAM programs (RAM-LFE) that we study in this work. Instead of a circuit $f(x)$ we consider a RAM program $f_{\mathsf{DB}}(x)$ with random access to the input x and a potentially large hard-coded dataset DB. The main goal of RAM-LFE is to improve the decryption run-time needed to recover $f_{\mathsf{DB}}(x)$ from the ciphertext to ideally be roughly proportional to the RAM run-time T of the plaintext computation, rather than its potentially much larger circuit size. Most importantly, it allows the decryption run-time to be sublinear in both x and DB. In order for this to be feasible, the deterministic digest generation procedure $(\mathsf{dig}, \mathsf{DS}) := \mathsf{Hash}(f_{\mathsf{DB}})$ also creates an auxiliary data structure DS. The encryption procedure $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$ only gets the small digest dig as before, but the decryption procedure now also uses the data-structure DS to efficiently decrypt the ciphertext ct and compute $f_{\mathsf{DB}}(x)$. It was noted in [DHMW24] that RAM-LFE implies a form of *doubly efficient private information retrieval (DEPIR)* [BIM00, CHR17, BIPW17, LMW23], which was recently constructed from Ring-LWE [LMW23]. The work of [DHMW24] uses DEPIR to achieve their results. It presents two constructions of RAM-LFE. Both constructions rely on a CRS of size $O(1)$, have digest size $O(1)$, and the digest generation run-time of Hash as well as the size of the data structure DS are $O(|f| + |\mathsf{DB}|)^{1+\varepsilon}$ for an arbitrary constant $\varepsilon > 0$, where $|f|$ here denotes the description-size of the RAM program. The first construction achieved "*weak efficiency*" under just Ring-LWE. For a RAM program $f_{\mathsf{DB}}(x)$ with run-time T:

- The encryption run-time to encrypt x under a digest dig is $O(|x| + T)$.
- The decryption run-time to decrypt $f_{\mathsf{DB}}(x)$ from the ciphertexts is $O(T)$.

Here, the encryption run-time scales with the RAM run-time T of the computation, which is antithetical to one of the main goals of LFE, that the encryptor should do less work than computing the function. It is also incomparable with LFE for circuits where the encryption run-time can be much smaller. The second construction achieved essentially optimal "*full efficiency*" by additionally relying on indistinguishability obfuscation (iO).

- The run-time to encrypt x is $O(|x|)$.
- The run-time to decrypt the ciphertext and recover $f_{\mathsf{DB}}(x)$ is $O(T)$.

1.2 Our Results and Applications

RAM-LFE. Our main result is a new construction of RAM-LFE from Ring-LWE and a standard circular-security assumption (i.e., the circular security of the GSW FHE scheme [GSW13]), avoiding the use of iO. Under these assumptions we achieve RAM-LFE with close-to optimal encryption/decryption run-time. For a RAM program $f_{\mathsf{DB}}(x)$ with run-time T:

- The run-time to encrypt x is $O(|x|^{1+o(1)})$.
- The run-time to decrypt the ciphertext and recover $f_{\mathsf{DB}}(x)$ is $O(T^{1+o(1)})$.

We also achieve essentially optimal digest size of $O(1)$. However, compared to both of the RAM-LFE constructions of [DHMW24], we pay a heavier price for the digest generation. Namely, we need to fix the input size $|x|$ upfront, and our CRS size is $O(|x|)$. The run-time of the digest-generation procedure $\mathsf{Hash}(f_{\mathsf{DB}})$ and the size of the produced data structure DS are now $O(s^{1+o(1)})$, where s is the *circuit size* of the function f_{DB} for the given input size. We argue that, in most applications, the digest of a function f_{DB} is computed once and reused repeatedly to evaluate the function on many inputs x, in which case the efficiency of encryption/decryption is much more crucial than that of digest generation. In short, our construction gives a strict improvement over LFE for circuits,[3] by having an improved decryption time that only scales with the RAM run-time of the computation, but it does not improve on the digest generation, which still essentially translates the computation into a circuit.

We can also remove the circular security assumption and get a *leveled* RAM-LFE scheme, where all the parameters scale with the depth of the RAM computation, which can be significantly smaller than the run-time T.

RAM-AB-LFE. Our construction of RAM-LFE follows the high-level approach of [QWW18], by first constructing a weaker *attribute-based* notion of LFE and then upgrading it. In a RAM-AB-LFE scheme, the encryption procedure $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, (x, \mu))$ takes an attribute/message pair (x, μ), and the ciphertetext ct always reveals the attribute x, while the output $f_{\mathsf{DB}}(x)$ determines whether the ciphertext should reveal or hide the message μ. The difference between attribute-based LFE and LFE is the same as that between attribute-based encryption and functional encryption. We construct RAM-AB-LFE under the LWE assumption (no need for Rings) and a circular security assumption. The RAM-AB-LFE scheme also gets a slightly improved encryption run-time $O(|x| + |\mu|)$ and digest generation run-time $O(s)$, removing the $o(1)$ terms in the exponent. Furthermore, we can also remove the circular security assumption and get a leveled RAM-LFE scheme, where all the parameters scale with the depth of the RAM computation.

RAM-ABE. Lastly, as a side result, our techniques also give a leveled many-key *attribute-based encryption* (ABE) for RAMs (RAM-ABE) under the LWE assumption. Each secret key $\mathsf{sk}_{f_{\mathsf{DB}}}$ is associated with a RAM program f_{DB} having some fixed input size and RAM run-time T. The run time of encrypting a message μ under attribute x is $(|x| + |\mu|)\mathrm{poly}(d)$, the run-time of the decryption is $T \cdot \mathrm{poly}(d)$ and the run-time of key generation and the size of the secret key is $s \cdot \mathrm{poly}(d)$, where s is the circuit size of f_{DB} and d is the depth of the RAM-computation. Unfortunately, in

[3] Ignoring the $o(1)$ terms in the exponent, but since $\mathrm{poly}(\lambda)^{o(1)} = o(\lambda)$, all these terms are anyway subsumed by fixed polynomials in the security parameter λ.

the many-key RAM-ABE setting, we do not know how to remove depth dependence via a circular security assumption alone. However, following the approach of [HLL23], it can be done under a significantly stronger *circular evasive LWE assumption* defined there, resulting in a scheme whose efficiency is as above but without the $\mathrm{poly}(d)$ factors. We note that a prior work of [AFS19] gave an alternate construction of RAM-ABE from LWE, albeit one where the encryption run-time is linear in the RAM run-time T. Additionally, [JLL23] gave a construction of a form of RAM-ABE based the assumption of circuit FE, but in their construction decryption runs in time linear in the sum $T + |\mathsf{DB}| + |x|$ of the RAM run-time, database size and attribute size.

Applications. Most of the prior applications of LFE [QWW18] can be directly lifted to RAM-LFE, yielding analogous improvements. We summarize three prominent applications briefly below.

Firstly, LFE implies 1-key succinct *functional encryption* (FE), and the same approach shows that RAM-LFE implies 1-key succinct RAM-FE. In a RAM-FE scheme, a master authority creates a master public key and hands out secret keys $\mathsf{sk}_{f_{\mathsf{DB}}}$ for various RAM programs f_{DB}. Given an encryption of an input x under the master public key and a *single* secret key $\mathsf{sk}_{f_{\mathsf{DB}}}$, the recipient only recovers $f_{\mathsf{DB}}(x)$, but learns nothing else about x. RAM-FE can be constructed from RAM-LFE by using a generic non-succinct FE for circuits (constructed generically from public-key encryption [SS10]) for the small circuit that performs a RAM-LFE encryption of x under the digest of the function f_{DB}. The encryption/decryption run-time of the RAM-FE is linear in that of the RAM-LFE, and the key generation run-time of the RAM-FE is linear in the digest generation run-time of the RAM-LFE.

Secondly, RAM-LFE gives *reusable garbled RAM* [GHRW14], which is a generalization of reusable garbled circuits [GKP+13b], and was previously studied in [GHRW14, CHJV15, CH16, BCG+18]. All prior constructions relied on iO, while we get a new construction without iO. In a reusable garbled RAM scheme a client garbles some RAM program f_{DB} and gives the garbled program $\widetilde{f_{\mathsf{DB}}}$ to a remote server while keeping some short secret key sk. Later the client can use sk to garble some input x and give the garbled input $\widetilde{x}$ to the server. Given the garbled program $\widetilde{f_{\mathsf{DB}}}$ and the garbled input $\widetilde{x}$, the server can recover the output $f_{\mathsf{DB}}(x)$, but should not learn anything else about the values f_{DB}, x (except the code/data/input sizes and the run-time). RAM-LFE almost directly solves the problem by setting the garbled input $\widetilde{x}$ to be an LFE encryption of x. While this hides the input x, the program/data f_{DB} would be given in the clear. To hide them, we follow the same approach as [GKP+13b] and simply encrypt the code f and the database DB under a secret key k and then include k as part of the garbled input. We then apply RAM-LFE on the modified program that decrypts each entry of DB or instruction from the code f as they are accessed, using the key k given as input. In the resulting reusable garbled RAM, the time to garble the RAM program f_{DB} and the size of the garbled program are larger than in the iO based constructions, proportional to the circuit size of the computation. But the run-time to garble each input and to evaluate the garbled RAM on each garbled input are nearly optimal.

Lastly, LFE implies a form of multiparty computation with small online complexity. The parties individually compute an LFE digest for the function f in an offline preprocessing stage. Then they simply run an generic MPC online to compute the LFE encryption of their inputs. Finally, they decrypt the LFE ciphertext in an offline postprocessing stage to recover the output. By plugging in RAM-LFE we get an analogous result in the RAM model, where the run-time of the postprocessing stage scales with the RAM run-time of the computation rather than its circuit size.

Separately from the applications of RAM-LFE, our techniques also yield a new construction of laconic encryption from LWE that is conceptually different from the one in [DKL+23]. We briefly sketch our construction in Appendix A.

1.3 Our Techniques

We start by constructing *attribute-based RAM-LFE (RAM-AB-LFE)*, and then show how to upgrade it to full RAM-LFE. We first review the main techniques of prior works.

Homomorphic Operations. The main tool used to construct many homomorphic cryptosystems including AB-LFE [QWW18] is a *system of homomorphic operations* first developed by [GSW13, BGG+14] and refined in subsequent works [GVW15a, GVW15b, BV15, BTVW17, ...]. This is a pair of polynomial-time algorithms EvalPK and EvalCT with the following syntax. Let $C : \{0,1\}^\ell \to \{0,1\}$ be a circuit, $x \in \{0,1\}^\ell$ be an input, $\mathbf{A}_1, \ldots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m}$ be matrices, and $\mathbf{b}_1, \ldots, \mathbf{b}_\ell \in \mathbb{Z}_q^m$ be vectors. Then:

- $\mathbf{A}_C := \mathsf{EvalPK}(C, \mathbf{A}_1, \ldots, \mathbf{A}_\ell) \in \mathbb{Z}_q^{n \times m}$,
- $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, x, \mathbf{b}_1, \ldots, \mathbf{b}_\ell) \in \mathbb{Z}_q^m$,

such that the following holds for any $\mathbf{s} \in \mathbb{Z}_q^n$:

$$\text{if} \quad \mathbf{b}_i^\top \approx \mathbf{s}^\top(\mathbf{A}_i - x_i\mathbf{G}) \qquad \text{then} \qquad \mathbf{b}_C^\top \approx \mathbf{s}^\top(\mathbf{A}_C - C(x) \cdot \mathbf{G}) \tag{1}$$

where $\mathbf{G}$ is the "gadget matrix" [MP12] and the "$\approx$" hides small error. For convenience we sometimes write $\mathbf{A} = [\mathbf{A}_1 \mid \cdots \mid \mathbf{A}_\ell] \in \mathbb{Z}_q^{n \times m\ell}$ and $\mathbf{b}^\top = [\mathbf{b}_1^\top \mid \cdots \mid \mathbf{b}_\ell^\top] \in \mathbb{Z}_q^{m\ell}$. Furthermore, such systems are *linear*, meaning there is some efficiently computable matrix $\mathbf{H}_{C,x} \in \mathbb{Z}^{(\ell m) \times m}$ that depends on C, x and $\mathbf{A}$, with small norm $||\mathbf{H}_{C,x}||$, such that EvalCT works by outputting $\mathbf{b}_C^\top := \mathbf{b}^\top \mathbf{H}_{C,x}$.

Homomorphic Operations give LFE. The work of [QWW18] showed that a system of homomorphic operations EvalPK, EvalCT for a given circuit class gives an AB-LFE scheme for that circuit class. The CRS consists of random matrices $\mathbf{A}_1 \ldots, \mathbf{A}_\ell$. The EvalPK procedure is used to compress a circuit C into a digest $\mathbf{A}_C$. To encrypt a message μ under an attribute x, the encryption procedure outputs LWE samples $\mathbf{b}_i^\top \approx \mathbf{s}^\top(\mathbf{A}_i - x_i\mathbf{G})$ and uses an LWE sample $\approx \mathbf{s}^\top\mathbf{A}_C$ to derive a one-time-pad for the message μ. The decryption algorithm uses EvalCT computes $\mathbf{b}_C \approx \mathbf{s}^\top(\mathbf{A}_C - C(x) \cdot \mathbf{G})$ and if $C(x) = 0$ then this allows us to remove the one-time-pad and recover the message.

From Gates to Circuits. The homomorphic operations EvalPK, EvalCT compose: if we construct such procedures for some set of gates, then we can compose them to get such procedures for an entire circuit made out of these gates, and the composition preserves linearity. Existing constructions give a linear system of homomorphic operations for (e.g.,) NAND gates, where the error grows by a factor of $O(m)$ with each NAND gate and therefore a factor of $O(m)^d$ for a circuit of depth d made up for NAND gates. This gives a "leveled" scheme where the complexity of all the procedures scales polynomially in d. The recent work of [HLL23] gives a clever new "bootstrapping" procedure that removes this depth-dependence under a circular security assumption.

Random-Access Gates. Our main new technical contribution is to extend the homomorphic operations $\mathsf{EvalPK}, \mathsf{EvalCT}$ to "random-access gates". We consider two types of random-access gates:

- *Data-read* gates $\mathsf{dRead}_{\mathsf{DB}}(x_1, \ldots, x_\ell) = \mathsf{DB}_x$ contain a hard-coded database $\mathsf{DB} \in \{0,1\}^L$ of size $L = 2^\ell$. The input wires $x_1, \ldots, x_\ell \in \{0,1\}$ are interpreted as an index x written in binary, and the gate outputs the x'th bit of the database DB_x.

- *Wire-read* gates $\mathsf{wRead}(x_1, \ldots, x_\ell, y_0, \ldots, y_{L-1}) = y_x$ take as input $L = 2^\ell$ wires y_j along with ℓ additional wires x_j interpreted as an index x in binary, and the gate outputs y_x.

Both of the above gates can be implemented by standard circuits of size $O(L)$, and therefore we can evaluate the homomorphic operations $\mathsf{EvalPK}, \mathsf{EvalCT}$ for them in $\mathcal{O}(L)$ time. However, we show how to do better. We will still evaluate EvalPK for these gates during digest-generation in $\widetilde{O}(L)$ time, but we also build a special data structure DS during this process. Using DS, we can then evaluate EvalCT during decryption in only $\mathrm{poly}(\ell)$ time.

Homomorphic Data-Read. The main idea to construct the data structure DS is to rely on linearity. During the computation $\mathbf{A}_{\mathsf{dRead}} = \mathsf{EvalPK}(\mathsf{dRead}_{\mathsf{DB}}, \mathbf{A})$, we can pre-compute the small matrix $\mathbf{H}_{\mathsf{dRead}_{\mathsf{DB}},x} \in \mathbb{Z}_q^{\ell m \times m}$ for every $x \in [L]$ and store these matrices in the data structure DS. Then, we can compute $\mathbf{b}_{\mathsf{dRead}} = \mathsf{EvalCT}(\mathsf{dRead}_{\mathsf{DB}}, \mathbf{A}, x, \mathbf{b})$ very efficiently in $\mathrm{poly}(\ell)$ time for any $x, \mathbf{b}$, by just setting $\mathbf{b}^\top_{\mathsf{dRead}_{\mathsf{DB}}} := \mathbf{b}^\top \mathbf{H}_{\mathsf{dRead}_{\mathsf{DB}},x}$, where we can look up the correct matrix $\mathbf{H}_{\mathsf{dRead}_{\mathsf{DB}},x}$ for the desired x inside DS. While this already gives us the desired efficiency of EvalCT, computing the data structure DS naively during EvalPK would take $\widetilde{O}(L^2)$ time: each matrix $\mathbf{H}_{\mathsf{dRead}_{\mathsf{DB}},x}$ depends on the entire DB requiring $\widetilde{O}(L)$ time to compute, and we need to compute L such matrices. We show a smarter way to compute the entire data-structure DS in just $\widetilde{O}(L)$ time. The main idea is to implement the $\mathsf{dRead}_{\mathsf{DB}}$ function recursively via:

$$\mathsf{dRead}_{\mathsf{DB}}(x_1, \ldots, x_\ell) = \mathsf{dRead}_{\mathsf{DB}^{\mathsf{L}}}(x_1, \ldots, x_{\ell-1}) \cdot x_\ell + \mathsf{dRead}_{\mathsf{DB}^{\mathsf{R}}}(x_1, \ldots, x_{\ell-1}) \cdot (1 - x_\ell),$$

where $\mathsf{DB}^{\mathsf{L}} = (\mathsf{DB}_0, \ldots, \mathsf{DB}_{L/2-1})$, $\mathsf{DB}^{\mathsf{R}} = (\mathsf{DB}_{L/2}, \ldots, \mathsf{DB}_L)$ are the "left" and "right" halves of the database respectively, and we think of x_ℓ as the most significant bit of x. If we recursively compute the $L/2$-size sub-problems $\mathbf{A}_{\mathsf{dRead}_{\mathsf{DB}^b}} := \mathsf{EvalPK}(\mathsf{dRead}_{\mathsf{DB}^b}, \mathbf{A}_1, \ldots, \mathbf{A}_{\ell-1})$ along with the corresponding data structures DS^b for $b \in \{\mathsf{L}, \mathsf{R}\}$, then we can use these to derive the overall data structure DS in $\widetilde{O}(L)$ time. This is because for each $x \in \{0,1\}^\ell$, the matrix $\mathbf{H}_{\mathsf{dRead}_{\mathsf{DB}},x}$ in DS can be computed very efficiently as a linear combination of the matrices $\mathbf{H}_{\mathsf{dRead}_{\mathsf{DB}^{\mathsf{L}}},x^-}$ and $\mathbf{H}_{\mathsf{dRead}_{\mathsf{DB}^{\mathsf{R}}},x^-}$ stored in $\mathsf{DS}^{\mathsf{L}}, \mathsf{DS}^{\mathsf{R}}$ respectively, where $x^- = (x_1, \ldots, x_{\ell-1})$. Therefore, the run-time $T(L)$ of generating the full data structure DS satisfies the recurrence $T(L) = 2 \cdot T(L/2) + \widetilde{O}(L)$ which implies $T(L) = \widetilde{O}(L)$.

Homomorphic Wire-Read. For the wire-read gate $\mathsf{wRead}(x_1, \ldots, x_\ell, y_1, \ldots, y_L)$, it's not immediately obvious whether an analogous data-structure DS similar to the one above even exists. The issue is that this this gate has $L + \ell$ input wires and hence we would need $2^{L+\ell}$ matrices $\mathbf{H}_{\mathsf{wRead},x,y}$ to handle every possible input, making the data structure exponentially large; moreover, each such matrix would already be of size $\Omega(L)$ so just reading it would already take linear time! We take a different approach.

First, we generalize the data-read gates above to *matrix-read* functions $\mathsf{mRead}_{\mathbf{M}}(x) = \mathbf{M}_x$ where $\mathbf{M} = (\mathbf{M}_0, \ldots, \mathbf{M}_{L-1})$ is a database of $L = 2^\ell$ matrices $\mathbf{M}_i \in \mathbb{Z}_q^{n \times m}$. While mRead is not a a boolean function, we can generalize the homomorphic operations

$$\begin{aligned} \mathbf{A}_{\mathsf{mRead}_{\mathbf{M}}} &:= \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{M}}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell), \\ \mathbf{b}_{\mathsf{mRead}_{\mathbf{M}}} &:= \mathsf{EvalCT}(\mathsf{mRead}_{\mathbf{M}}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, x, \mathbf{b}_1, \ldots \mathbf{b}_\ell), \end{aligned}$$

to satisfy the following:

$$\text{if} \quad \mathbf{b}_i^\top \approx \mathbf{s}^\top(\mathbf{A}_i - x_i\mathbf{G}) \qquad \text{then} \qquad \mathbf{b}_{\mathsf{mRead}_{\mathbf{M}}}^\top \approx \mathbf{s}^\top(\mathbf{A}_{\mathsf{mRead}_{\mathbf{M}}} - \mathbf{M}_x) \tag{2}$$

where EvalCT is computed by some linear function $\mathbf{H}_{\mathsf{mRead}_{\mathbf{M}},x} \in \mathbb{Z}_q^{\ell m \times m}$. The implementation of such $\mathsf{EvalPK}, \mathsf{EvalCT}$ is a small variant of data-read gates above. As before, we can also make EvalPK compute a data structure DS consisting of the matrices $\mathbf{H}_{\mathsf{mRead}_{\mathbf{M}},x}$ for every $x \in \{0,1\}^\ell$, which allows us to later compute EvalCT in only $\mathrm{poly}(\ell)$ time. Furthermore, as before, we can compute the data structure DS in $\widetilde{O}(L)$ time using a similar recursive strategy.

Second, we utilize matrix-read functions to efficiently implement the homomorphic operations for wire-read gates. We define the operations:

$$\begin{aligned} \mathbf{A}_{\mathsf{wRead}} &:= \mathsf{EvalPK}(\mathsf{wRead}, \mathbf{A} = (\mathbf{A}_1^X \ldots, \mathbf{A}_\ell^X, \mathbf{A}_0^Y, \ldots, \mathbf{A}_{L-1}^Y)) \\ & \text{Output: } \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{M}}, \mathbf{A}_1^X, \ldots, \mathbf{A}_\ell^X) \text{ where } \mathbf{M} := (\mathbf{A}_0^Y, \ldots, \mathbf{A}_{L-1}^Y) \\ \mathbf{b}_{\mathsf{wRead}} &:= \mathsf{EvalCT}(\mathsf{wRead}, \mathbf{A}, (x,y), \mathbf{b} = (\mathbf{b}_1^X, \ldots, \mathbf{b}_\ell^X, \mathbf{b}_0^Y, \ldots, \mathbf{b}_{L-1}^Y)) \\ & \text{Output: } \mathsf{EvalCT}(\mathsf{mRead}_{\mathbf{M}}, x, \mathbf{A}_1^X, \ldots, \mathbf{A}_\ell^X, \mathbf{b}_1^X, \ldots, \mathbf{b}_\ell^X) + \mathbf{b}_x^Y. \end{aligned}$$

If the preconditions of (1) hold with $\mathbf{b}_i^X \approx \mathbf{s}^\top(\mathbf{A}_i^X - x_i\mathbf{G})$ and $\mathbf{b}_j^Y \approx \mathbf{s}^\top(\mathbf{A}_j^Y - y_j\mathbf{G})$ then:

$$\begin{aligned} \mathbf{b}_{\mathsf{wRead}} &\approx \mathbf{s}^\top(\mathbf{A}_{\mathsf{wRead}} - \mathbf{M}_x) + \mathbf{b}_x^Y \\ &\approx \mathbf{s}^\top(\mathbf{A}_{\mathsf{wRead}} - \mathbf{A}_x^Y) + \mathbf{s}^\top(\mathbf{A}_x^Y - y_x\mathbf{G}) \\ &\approx \mathbf{s}^\top(\mathbf{A}_{\mathsf{wRead}} - y_x\mathbf{G}) \end{aligned}$$

as desired. Furthermore, the efficiency properties of the homomorphic operations for $\mathsf{mRead}_{\mathbf{M}}$ directly translate into those for wRead: the EvalPK procedure constructs a data structure DS in $\widetilde{O}(L)$ time that allows us to later execute EvalCT for any choice of $x, y, \mathbf{b}$ in only $\mathrm{poly}(\ell)$ time.

AB-LFE for RAM Circuits. By composing the above, we get homomorphic operations EvalPK, EvalCT for *RAM-circuits* made up of standard NAND gates as well as arbitrary data-read gates and wire-read gates.[4] The EvalPK procedure now also generates a data structure DS which is used by the EvalCT procedure. The data structure has a separate component DS_g for each data-read/wire-read gate g in the circuit.

For a RAM circuit C, we define its *compressed* size $\mathsf{size}_c(C)$ to be the total number of gates in the circuit, where we count each database-read gate and each wire-read gate as a single gate. The compressed size $\mathsf{size}_c(C)$ can meaningfully be sublinear in the database size or the input size. We define the *expanded* $\mathsf{size}_e(C)$ as the total number of gates one would get if one expanded each

[4] In general, each data-read gate can contain its own different database DB, although we will usually think of the special case where they all have the same database.

database-read gate and each wire-read gate into a standard sub-circuit of just NAND gates. The total run-time of $\mathsf{EvalPK}(C, \ldots)$ and the size of the data-structure DS is $\widetilde{O}(\mathsf{size}_e(C))$, proportional to the expanded size of C. The run-time of $\mathsf{EvalCT}(C, \ldots)$, given random-access to the data structure DS, is then just $\widetilde{O}(\mathsf{size}_c(C))$, proportional to the compressed size of C. Because of the error growth, all parameters also suffer an $\mathrm{poly}(d)$ blow-up where d is the depth of the RAM-circuit. However, by employing the bootstrapping procedure of [HLL23], we can remove this depth dependence at the cost of needing a circular security assumption. Overall, by plugging the corresponding EvalPK, EvalCT procedures for RAM-circuits into the AB-LFE construction of [QWW18], we get a RAM-AB-LFE for RAM-circuits C with input size N, where the digest-generation procedure runs in time $\widetilde{O}(\mathsf{size}_e(C))$, the encryption run-time is $\widetilde{O}(N)$, and the decryption run-time is $\widetilde{O}(\mathsf{size}_c(C))$.

RAM Programs to RAM Circuits. The above gives us AB-LFE for *RAM circuits*, but our goal is to handle *RAM programs*. We show how to generically convert an arbitrary RAM program with some fixed input size into a RAM circuit while preserving efficiency. For a RAM program $f_{\mathsf{DB}}(x)$ having total description size $|f| + |\mathsf{DB}| = L$, and some fixed input size $|x| = N$ and run-time T, we can convert it into a RAM circuit C of compressed-size $\mathsf{size}_c(C) = \widetilde{O}(T)$ and expanded-size $\mathsf{size}_e(C) = \widetilde{O}(T(N + L + T))$. Note that the expanded-size corresponds to the naive conversion of a RAM program into a standard circuit, by implementing each memory access via a linear size sub-circuit, where the memory size is bounded by $N + L + T$.

The main idea for converting a RAM program into a RAM circuit is that the data-read gates allow us to efficiently emulate random-accesses to DB, and wire-read gates allow us to efficiently emulate random-access reads the input x as well as to any additional read/write memory, where the memory contents are represented by a bundle of wires in the RAM circuit. However, there is a mismatch: a RAM program can both read and write to arbitrary locations in memory, while a RAM circuit can only read from arbitrary locations in memory, but each write corresponds to outputting a value on some fixed wire in the circuit. In other words, RAM circuits efficiently emulate *fixed-writes RAM programs*, where each step of the computation can only write to some fixed (input-independent) location in memory, but can read arbitrary locations in memory. We show that fixed-writes RAM programs can efficiently emulate general RAM programs at a small loss. The idea is inspired by hierarchical ORAM [GO96]; and was used in a similar vein in [HOWW19, HHWW19, LMW23]. We emulate general writes to arbitrary locations in memory via fixed writes to a hierarchical data structure. The data structure consists of $t = \log T$ levels, where T upper bounds the total number of writes. Each level $i \in \{0, \ldots, t-1\}$ contains 2^i pairs of the form (location, value). Whenever the program writes a bit b to location j we place the pair (j, b) in the fixed unique slot in level $i = 0$. After every 2^i writes we take all the values in levels $\leq i$ and sort them by location into level $i+1$ using a data-independent sorting circuit. Note that the sequence of writes to the data structure is completely fixed and data independent, and the amortized cost of a write is $O(\log T)$. Whenever we want to read an index j we do a binary search for j at every level of the data-structure and take the "freshest" pair (j, b) that we find from the smallest level. Overall, the above compiler from general RAM to fixed-writes RAM only incurs a polylogarithmic overhead.

From AB-LFE to LFE for RAMs via RAM-FHE. The above gives us AB-LFE for general RAM programs. The work of [QWW18] showed how to upgrade AB-LFE to LFE in the circuit model via FHE and garbled circuits, following the approach of [GKP+13b] for transforming ABE to 1-key

Succinct FE. We show that the same approach allows us to go from AB-LFE to LFE in the RAM model, by relying on RAM-FHE [LMW23] instead of standard FHE.

In a RAM-FHE scheme there is a deterministic procedure to preprocess some public database DB into $\widetilde{\mathsf{DB}} := \mathsf{Prep}(\mathsf{DB})$. There is also a way to encrypt an input x under a secret key sk to gets a ciphertext $\hat{x} \leftarrow \mathsf{Enc}_{\mathsf{sk}}(x)$. Moreover there is an FHE evaluation procedure $\hat{y} := \mathsf{Eval}(f, \hat{x}, \widetilde{\mathsf{DB}})$ that homomorphically evaluates a RAM program $f_{(\cdot)}(\cdot)$ with some fixed run-time T, and outputs an encryption $\hat{y}$ of the output $y = f_{\mathsf{DB}}(x)$. The run-time of the preprocessing and encryption procedures should be nearly linear in x and DB respectively, while the run-time of the Eval procedure should be nearly linear in T. The work of [LMW23] showed how construct RAM-FHE from the Ring-LWE assumption with circular security, by building on top of doubly efficient private information retrieval (DEPIR) from Ring-LWE.

The transformation from AB-LFE to LFE in the RAM model proceeds as follows. To encrypt an input x under the LFE scheme, we first encrypt x under the RAM-FHE scheme with a secret key sk to get a ciphertext $\hat{x} \leftarrow \mathsf{FHE.Enc}_{\mathsf{sk}}(x)$. We then also construct a garbled circuit $\widetilde{C}$ for the FHE decryption circuit $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ with the secret key sk hard-coded. Lastly, we encrypt the labels $\{\mathsf{lab}_{i,b}\}$ of the garbled circuit under a RAM-AB-LFE scheme with $\hat{x}$ in the attribute to ensure that the recipient can only recover the labels corresponding to the bits of $\hat{y}$. In particular, we use a RAM-AB-LFE scheme for the RAM Program $\hat{f}_{\widetilde{\mathsf{DB}}}(\hat{x}, i, b)$ that computes $\hat{y} := \mathsf{Eval}(f, \hat{x}, \widetilde{\mathsf{DB}})$ and, if the i'th bit of $\hat{y}$ is b, it allows the message to be revealed. The encryptor gets the digest $\mathsf{dig}_{\hat{f}}$ and uses it to encrypt the labels $\mathsf{lab}_{i,b}$ under the attributes $(\hat{x}, i, b)$ respectively. The recipient uses the RAM-AB-LFE to recover the labels for $\hat{y}$ and then runs the garbled circuit to recover the output y; security of RAM-AB-LFE, garbled circuits and RAM-FHE ensures that nothing else about x is revealed.

RAM-ABE. Our techniques for constructing RAM-AB-LFE by extending the homomorphic operations to RAM circuits also allow us to get *attribute-based encryption for RAMs* (RAM-ABE). However, there is one major difference. Any system of homomoprhic operations is good enough for AB-LFE [QWW18], but for ABE we further need this system to be linear [BGG+14]. While our extensions for RAM circuits are linear, the bootstrapping technique of [HLL23] is *not* linear. Therefore, we only get a *leveled* RAM-ABE scheme under LWE, where efficiency of all operations scales polynomially with the maximal depth d of the supported RAM circuits. Although we usually think of RAM computation as sequential, and hence the default compiler from a time T RAM program to a RAM circuit results in depth $d = O(T)$, it is often possible to construct much shallower RAM circuits for specific problems (e.g., ones that have small run time on a parallel RAM). Alternatively, the work of [HLL23] also shows that a non-linear system of homomorhpic operations is sufficeint to yield ABE under an additional *evasive circular LWE* assumption, which strengthens the evasive LWE assumption of [Wee22, Tsa22]. The same approach therefore also yields a full RAM-ABE scheme under the evasive circular LWE assumption.

2 Preliminaries

Define $\mathbb{N} = \{0, 1, 2, \ldots\}$ to be the set of natural numbers. For any integer $n \geq 1$, define $[n] = \{1, \ldots, n\}$. For an array $A \in \{0,1\}^n$, we index the array from 1, and $A[i]$ denotes the bit in position $i \in [n]$. By default, all our logarithms are base 2 and $\log n$ stands for $\log_2 n$. A function $\nu : \mathbb{N} \to \mathbb{N}$

is said to be negligible, denoted $\nu(n) = \mathrm{negl}(n)$, if for every positive polynomial $p(\cdot)$ and all sufficiently large n it holds that $\nu(n) < 1/p(n)$. We use the abbreviation PPT for probabilistic polynomial time. For a finite set S, we write $a \leftarrow S$ to mean a is sampled uniformly randomly from S. For a randomized algorithm A, we let $a \leftarrow A(\cdot)$ denote the process of running $A(\cdot)$ and assigning the outcome to a; when A is deterministic, we write $a := A(\cdot)$ instead. We denote the security parameter by λ. For two distributions X, Y parameterized by λ we say that they are computationally indistinguishable, denoted by $X \approx_c Y$ if for every PPT distinguisher D we have $|\Pr[D(X) = 1] - \Pr[D(Y) = 1]| = \mathrm{negl}(\lambda)$. We say X and Y are statistically indistinguishable instead if the same holds for every unbounded distinguisher D.

For any integer $q \geq 1$, let $\mathbb{Z}_q$ be the ring $\mathbb{Z}/q\mathbb{Z}$. We adhere to the convention that vectors $\mathbf{u} \in \mathbb{Z}_q^n$ are column vectors and we define the ℓ_∞-norm over vectors over $\mathbb{Z}_q$ by first lifting each coordinate to its representative in the set $[-q/2, q/2)$. That is, for any $\mathbf{u} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)^\top \in \mathbb{Z}_q^n$ we define $\|\mathbf{u}\|_\infty = \max_i |\mathbf{u}_i|$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we define $\|\mathbf{A}\|$ as the operator norm of $\mathbf{A}$ with respect to the ℓ_∞-norm, that is,

$$\|\mathbf{A}\| = \max_{\|\mathbf{u}\|_\infty = 1} \|\mathbf{A}\mathbf{u}\|_\infty = \max_i \sum_j |\mathbf{A}_{ij}|$$

where $(\mathbf{A}_{ij})$ are the entries of $\mathbf{A}$. We define the function $\mathsf{round}_q : \mathbb{Z}_q \to \{0,1\}$ via $\mathsf{round}_q(x) = \lceil 2x/q \rfloor$ where $x \in \mathbb{Z}_q$ is identified with its representative in the set $[-q/2, q/2)$. Note that for any $\mu \in \{0,1\}$ and $e \in \mathbb{Z}$ with $|e| < q/4$, we have $\mathsf{round}_q(\mu \lceil q/2 \rfloor + e) = \mu$.

2.1 Learning with Errors and Lattice Tools

Learning with Errors. Let $\mathcal{D}_{\mathbb{Z},\sigma}$ denote the discrete Gaussian distribution over $\mathbb{Z}$ with width $\sigma \geq 0$. The truncated discrete Gaussian $\mathcal{D}_{\mathbb{Z},\sigma,\leq B}$ over $\mathbb{Z}$ with width σ and bound $B \geq 0$ is the distribution that samples $x \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}$ and outputs x if $|x| \leq B$ and 0 otherwise. Setting $B = \sigma \cdot \Omega(\sqrt{\lambda})$, we have that $\mathcal{D}_{\mathbb{Z},\sigma} \approx_s \mathcal{D}_{\mathbb{Z},\sigma,\leq B}$. Throughout this paper, we use the shorthand χ_B to denote the truncated discrete Gaussian over $\mathbb{Z}$ with width $\sigma = B/\lambda$ and bound B.

Definition 2.1 ((Decision) LWE). *Let $n, q, B \in \mathbb{N}$ be functions of the security parameter. The* learning with errors (LWE) *assumption* $\mathsf{LWE}_{n,q,B}$ *states that for all* $m = \mathrm{poly}(\lambda)$*,*

$$(\mathbf{A}, \mathbf{s}^\top\mathbf{A} + \mathbf{e}^\top) \approx_c (\mathbf{A}, \mathbf{u}),$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi_B^m$.

We write simply that LWE holds to mean the statement that for any $p = \mathrm{poly}(\lambda)$ there exists some $n = \mathrm{poly}(\lambda)$, $q = 2^{\mathrm{poly}(\lambda)}$ and $B = B(\lambda)$ such that $B \leq q/2^p$ and $\mathsf{LWE}_{n,p,B}$ holds.

The Gadget Matrix. Let $n, q \in \mathbb{N}$ and let $m = n\lceil \log q \rceil$. Define the vector

$$\mathbf{g}^\top = (1, 2, 4, \ldots, 2^{\lceil \log q \rceil - 1}) \in \mathbb{Z}_q^m.$$

The *gadget matrix* is the block diagonal matrix $\mathbf{g}^\top \otimes \mathbf{I} \in \mathbb{Z}_q^{n\times m}$. We also define the gadget matrix for $m \geq n\lceil \log q \rceil$ by appropriately padding with zeros.

There is an efficient algorithm $\mathbf{G}^{-1}(\cdot)$ that, on input a vector $\mathbf{u} \in \mathbb{Z}_q^n$, outputs a binary vector $\mathbf{t} = \mathbf{G}^{-1}(\mathbf{u}) \in \{0,1\}^m$ such that $\mathbf{G}\mathbf{t} = \mathbf{u}$.

Lattice Trapdoors. Let $n, q \in \mathbb{N}$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a *trapdoor for* $\mathbf{A}$ is a matrix $\mathbf{T} \in \mathbb{Z}_q^{m\times m}$ such that $\mathbf{AT} = \mathbf{G}$. There exists some $m_0 = O(n \log q)$ and a PPT algorithm $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TGen}(1^n, 1^m, q)$ that, as long as $m \geq m_0$, samples a matrix along with an associated trapdoor such that $\mathbf{A}$ is statistically close to a uniform matrix over $\mathbb{Z}_q^{n\times m}$ and $\mathbf{T} \in \{0,1\}^{m\times m}$ and hence $\|\mathbf{T}\| \leq m$. Choosing (e.g.) $m_0 = 3n \lceil \log q \rceil$ suffices. Given a trapdoor for $\mathbf{A}$ it is possible to efficiently compute one for $[\mathbf{A} \mid \mathbf{B}]$ or $[\mathbf{B} \mid \mathbf{A}]$ for any matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m'}$.

Moreover there exists a PPT algorithm $\mathbf{t} \leftarrow \mathsf{TSamp}(\mathbf{A}, \mathbf{T}, \mathbf{u}, B)$ that, as long as $B = \Omega(\sqrt{\lambda m \|\mathbf{T}\|})$, samples a vector $\mathbf{t}$ such that the distribution on $\mathbf{t}$ is statistically indistinguishable from the distribution χ_B^m conditioned on $\mathbf{At} = \mathbf{u}$.

Noise Smudging. We use the following fact.

Lemma 2.2 (Noise smudging, (e.g.) [AJL+12]). *Let $B, B' \in \mathbb{Z}$ be functions of the security parameter. Let $e_1 \in [-B, B]$ be arbitrary and let $e_2 \leftarrow [-B', B']$ be chosen uniformly at random. Then the distribution of e_2 is statistically close to $e_1 + e_2$ as long as $B/B' = \mathrm{negl}(\lambda)$.*

2.2 The GSW FHE Scheme

In this section we briefly recall the (leveled) fully homomorphic encryption scheme from [GSW13]. We only describe the details of the scheme that will be relevant for our purposes. Let n, q, B be such that $\mathsf{LWE}_{n,q,B}$ holds. The scheme works as follows:

- **Key Generation:** $\mathsf{GSW.Gen}(1^\lambda, B)$ samples a public key/secret key pair $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{sk} = \mathbf{s} \in \mathbb{Z}_q^n$ is of the form $\mathbf{s} = (\mathbf{r}, -1)^\top$ and $\mathsf{pk} = \mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is of the form $\mathbf{A} = [\overline{\mathbf{A}} \mid \mathbf{r}^\top \overline{\mathbf{A}} + \mathbf{e}^\top]^\top$ where $\overline{\mathbf{A}} \leftarrow \mathbb{Z}_q^{(n-1)\times m}$ and $\mathbf{e} \leftarrow \chi_B^m$.

- **Encryption and Decryption:** For a message $x \in \{0,1\}^\ell$, $\mathsf{GSW.Enc}(\mathbf{A}, x)$ outputs a ciphertext $\mathbf{C}_x \in \mathbb{Z}_q^{n\times m\ell}$ that satisfies the correctness property

$$\mathbf{s}^\top \mathbf{C}_x = \mathbf{s}^\top (x \otimes \mathbf{G}) + \mathbf{e}_x,$$

  where $\mathbf{e}_x$ is a short error vector with $\|\mathbf{e}_x\|_\infty \leq mB$. The message x can then be efficiently recovered from $\mathbf{s}^\top \mathbf{C}_x$.

- **Evaluation:** For any boolean circuit $C : \{0,1\}^\ell \to \{0,1\}$, and any ciphertext $\mathbf{C}$ satisfying $\mathbf{s}^\top \mathbf{C}_x = x\mathbf{s}^\top \mathbf{G} + \mathbf{e}_{\mathsf{ct}}$, $\mathsf{GSW.Eval}(C, \mathbf{C}_x)$ outputs a new ciphertext $\mathbf{C}_{C(x)}$ such that

$$\mathbf{s}^\top \mathbf{C}_{C(x)} = C(x)\mathbf{s}^\top \mathbf{G} + \mathbf{e}_{C(x)},$$

  where $\mathbf{e}_{C(x)}$ satisfies $\left\|\mathbf{e}_{C(x)}\right\|_\infty \leq (m+1)^d \cdot \|\mathbf{e}_x\|_\infty$ where d is the depth of C.

The scheme satisfies standard semantic security. As was observed in [HLL23], GSW evaluation can be extended to handle vector-valued circuits $C : \{0,1\}^\ell \to \mathbb{Z}_q^m$. For such circuits $\mathbf{C}_{C(x)} := \mathsf{GSW.Eval}(C, \mathbf{C}_x)$ satisfies the alternate correctness property that

$$\mathbf{s}^\top \mathbf{C}_{C(x)} = C(x)^\top + \mathbf{e}_{C(x)},$$

where now $\mathbf{e}_{C(x)}$ satisfies $\left\|\mathbf{e}_{C(x)}\right\|_\infty \leq (m+1)^d \lceil \log q \rceil \cdot \|\mathbf{e}_x\|_\infty$.

Circular Security. The circular security assumption for the GSW scheme states that LWE still holds with respect to the GSW secret key in the presence of an encryption of the secret key. That is, for all $m' = \mathrm{poly}(\lambda)$ the following distributions are computationally indistinguishable

$$(\mathbf{A}, \mathbf{S}, \mathbf{B}, \mathbf{s}^\top \mathbf{B} + \mathbf{e}) \approx_c (\mathbf{A}^*, \mathbf{S}^*, \mathbf{B}^*, \mathbf{u})$$

where on the left hand side $(\mathbf{A}, \mathbf{s}) \leftarrow \mathsf{GSW.Gen}(1^\lambda, B)$, $\mathbf{S} \leftarrow \mathsf{GSW.Enc}(\mathbf{A}, \mathsf{bits}(\mathbf{s}))$, $\mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m'}$ and $\mathbf{e} \leftarrow \chi_B^{m'}$, and on the right hand side all the entries are uniform over the appropriate sets.

2.3 RAM-FHE

In this section we define the notion FHE for RAM programs as defined in [LMW23].

Definition 2.3 (RAM-FHE). *A* RAM-FHE *scheme is a tuple of algorithms* (Setup, Gen, Prep, Enc, Eval, Dec) *with the following syntax:*

- $\mathsf{params} := \mathsf{Setup}(1^\lambda, M)$*: On input security parameter λ and a space bound M, deterministically output public parameters* params.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{params})$*: Given public parameters* params*, sample and output a public key* pk *and a secret key* sk.
- $\mathsf{DS} := \mathsf{Prep}(\mathsf{params}, f_{\mathsf{DB}})$*: Given public parameters* params *and a RAM program f_{DB},*[5] *output a preprocessed data structure* DS*. For convenience, we assume* DS *contains a description of f_{DB}.*
- $\mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$ *Given a public key* pk *and a database x, output a preprocessed ciphertext ct_x.*
- $\mathsf{ct}_{\mathsf{out}} := \mathsf{Eval}(\mathsf{pk}, \mathsf{DS}, \mathsf{ct}_x)$*: Given a public key* pk*, a data structure* DS *and a preprocessed ciphertext ct_x, output an output ciphertext $\mathsf{ct}_{\mathsf{out}}$.*
- $\mu := \mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: Given a secret key* sk *and a ciphertext* ct*, it outputs a plaintext μ.*

Correctness. *For any $\lambda, M \in \mathbb{N}$, any RAM program f_{DB} and any input x such that $|x| \le M$, $|\mathsf{DB}| + |f| \le M$, and f_{DB} runs in time $T \le M$, it must hold that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) = P(x, y) \;:\; \begin{array}{rl} \mathsf{params} & := \mathsf{Setup}(1^\lambda, M) \\ (\mathsf{pk}, \mathsf{sk}) & \leftarrow \mathsf{Gen}(\mathsf{params}) \\ \mathsf{DS} & := \mathsf{Prep}(\mathsf{params}, f_{\mathsf{DB}}) \\ \mathsf{ct}_x & \leftarrow \mathsf{Enc}(\mathsf{pk}, x) \\ \mathsf{ct}_{\mathsf{out}} & := \mathsf{Eval}(\mathsf{pk}, \mathsf{DS}, \mathsf{ct}_x) \end{array}\right] = 1.$$

Security. *The algorithms* (Gen, Enc, Dec) *satisfy the standard notion of public-key semantic security.*

The recent work of [LMW23] shows the existence of RAM-FHE assuming the hardness of RingLWE as well as a circular security assumption.

Theorem 2.4 ([LMW23]). *Assume RingLWE holds together with a circular security assumption. Then there is a RAM-FHE scheme* (Setup, Gen, Prep, Enc, Eval, Dec) *with the following efficiency properties: for any $\lambda, M \in \mathbb{N}$,*

[5]See Section 3 for a description of the RAM model we use in this paper.

- Setup *and* Gen *run in time* $M^{o(1)} \cdot \mathrm{poly}(\lambda)$
- Enc *and* Prep *run in time* $|x| \cdot M^{o(1)} \cdot \mathrm{poly}(\lambda)$ *and* $(|\mathsf{DB}| + |f|) \cdot M^{o(1)} \cdot \mathrm{poly}(\lambda)$ *respectively*
- Eval *runs in time* $T \cdot M^{o(1)} \cdot \mathrm{poly}(\lambda)$ *where* T *is an upper bound on the run time of* f_{DB}
- Dec *runs in time* $\ell \cdot M^{o(1)} \cdot \mathrm{poly}(\lambda)$ *where* ℓ *is an upper bound on the output size of* f_{DB}

2.4 Garbled Circuits

In this section we define garbled circuits, originally introduced by Yao ([Yao86]).

Definition 2.5. *A* circuit garbling *scheme is a pair of algorithms* $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$ *with the following syntax:*

- $(\Gamma, \{L_j^0, L_j^1\}_{j=1}^k) \leftarrow \mathsf{GC.Garble}(1^\lambda, C)$*: Given the security parameter* λ *and a circuit* $C : \{0,1\}^k \to \{0,1\}^\ell$*, it outputs a garbled circuit* Γ *and a set of labels* $\{L_j^0, L_j^1\}_{j=1}^k$.
- $z := \mathsf{GC.Eval}(\Gamma, \{L_j\}_{j=1}^k)$*: Given the garbled circuit* Γ *and a subset of labels, it outputs a value* $z \in \{0,1\}^\ell$.

Correctness. *For any circuit* $C : \{0,1\}^k \to \{0,1\}^\ell$ *and any input* $x \in \{0,1\}^k$*, it must hold that*

$$\Pr\left[z = C(x) \ : \ \begin{array}{rl} (\Gamma, \{L_j^0, L_j^1\}_{j=1}^k) & \leftarrow \mathsf{GC.Garble}(1^\lambda, C) \\ z & := \mathsf{GC.Eval}(\Gamma, \{L_j\}_{j=1}^k) \end{array}\right] = 1.$$

Security. *We define the privacy of circuit and input with the following two experiments:*

$\mathsf{Real}_{\mathsf{GC}}^{\mathcal{A}}(1^\lambda)$:

1. $(x, C) \leftarrow \mathcal{A}(1^\lambda)$
2. $(\Gamma, \{L_j^0, L_j^1\}_{j=1}^k) \leftarrow \mathsf{GC.Garble}(1^\lambda, C)$
3. $b \in \{0,1\} \leftarrow \mathcal{A}(\Gamma, \{L_j^{x_j}\}_{j=1}^k)$
4. *Output* b.

$\mathsf{Ideal}_{\mathsf{GC}}^{\mathcal{A}}(1^\lambda)$:

1. $(x, C) \leftarrow \mathcal{A}(1^\lambda)$
2. $(\Gamma, \{L_j^0, L_j^1\}_{j=1}^k) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, |C|, |x|, C(x))$
3. $b \in \{0,1\} \leftarrow \mathcal{A}(\Gamma, \{L_j^{x_j}\}_{j=1}^k)$
4. *Output* b.

We say that the garbling scheme $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$ *is circuit and input private if there exists a PPT simulator* $\mathsf{Sim}_{\mathsf{GC}}$ *such that for any stateful PPT adversary* $\mathcal{A}$*, we have:*

$$\left|\Pr[\mathsf{Real}_{\mathsf{GC}}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}_{\mathsf{GC}}(1^\lambda) = 1]\right| \leq \mathrm{negl}(\lambda).$$

Efficiency. *For any circuit* $C : \{0,1\}^k \to \{0,1\}^\ell$ *and security parameter* λ*, the efficiency holds that:*

- $\mathsf{GC.Garble}(1^\lambda, C)$ *and* $\mathsf{GC.Eval}(\Gamma, \{L_j\}_{j=1}^k)$ *run in time* $|C| \cdot \mathrm{poly}(\lambda)$.
- *The garbled circuit* Γ *is of size* $|C| \cdot \mathrm{poly}(\lambda)$*, and the labels* L_j^b *are of size* $\mathrm{poly}(\lambda)$ *for* $j \in [k]$ *and* $b \in \{0,1\}$.

3 RAM Circuits and RAM Programs

In this section we introduce a "RAM circuit" model of computation that captures the power of RAM programs by augmenting the circuits with special gates for reading from a large database or large bundle of wires. We show how to compile RAM programs with fixed run-time and input size into a RAM circuit with low overhead.

RAM Circuits. We define the class of *RAM circuits* as consisting of standard (fan-in 2) NAND gates along two additional types of gates: dRead and wRead.

For any $\mathsf{DB} = (\mathsf{DB}_0, \ldots, \mathsf{DB}_{L-1}) \in \{0,1\}^{L=2^\ell}$, we define a "data read" gate $\mathsf{dRead}_{\mathsf{DB}} : \{0,1\}^\ell \to \{0,1\}$ via

$$\mathsf{dRead}_{\mathsf{DB}}(x_1, \ldots, x_\ell) = \mathsf{DB}_x \quad \text{where } x = \sum_{i=1}^{\ell} x_i \cdot 2^{i-1}.$$

For any $\ell \in \mathbb{N}$, we define the "wire read" gate $\mathsf{wRead}_\ell : \{0,1\}^{\ell+L} \to \{0,1\}$ with $L = 2^\ell$ via:

$$\mathsf{wRead}_\ell(x_1, \ldots, x_\ell, y_0, \ldots, y_{L-1}) = y_x \quad \text{where } x = \sum_{i=1}^{\ell} x_i \cdot 2^{i-1}.$$

We call the value L above the *load* of a dRead or a wRead gate.

A RAM circuit C is composed of (fan-in-2) NAND gates, along with dRead gates, and wRead gates with arbitrary load. All gates can have arbitrary fan-out. In general, each dRead gate can have its own distinct database DB, although typically we will have the same DB for all gates. For a RAM circuit C, we define its *compressed* size $\mathsf{size}_c(C)$ to be the total number of gates in the circuit, where each NAND gate, each database-read gate and each wire-read gate is counted as a single gate. The compressed size $\mathsf{size}_c(C)$ of a RAM circuit C can meaningfully be sublinear in the size of the database(s) contained in the dRead gates or in the input size of the circuit. We define the *expanded* $\mathsf{size}_e(C)$ as the total number of gates one would get if one expanded each database-read gate and each wire-read gate into a sub-circuit of just NAND gates, thereby converting the overall RAM circuit into a standard boolean circuit of NAND gates. Each database-read gate and wire-read gate with load L has expanded size $O(L)$. Therefore, for a RAM circuit C with compressed size $S = \mathsf{size}_c(C)$ and maximum load L in any dRead, wRead gate, we can bound its expanded size by $\mathsf{size}_e(C) = O(SL)$.

From RAM Program to RAM Circuits. We show how to convert an arbitrary RAM program f_{DB} with a given input size and RAM run-time, into a RAM circuit whose compressed size essentially matches the RAM run-time.

For concreteness, let us briefly describe a RAM model. We think of the RAM program f_{DB} as starting with its description (i.e., code) f and the database DB written in a read-only random-access memory M_R. We also think of the program as having access to additional read/write random-access memory M_{RW} initialized to all 0's. Lastly the program has read-only random-access to its input x. We can think of both memories as arrays of size 2^w with w-bit addresses, and each memory location stores a single bit. The program execution consists of repeated evaluations of a universal "CPU Step" circuit CPU having some fixed $O(w)$ size state (i.e. a constant number of registers) and in each step it updates its state, and specifies which locations i_x, i_R, i_{RW} of the input x and of the memory M_R, M_{RW} (respectively) to read from, and a location/value (j, b) to write to M_{RW}. The read bits b_x, b_R, b_{RW} are then given as inputs to the next step. The execution proceeds by repeatedly evaluating the CPU function, where initially all inputs $\mathsf{state}^0, b^0_x, b^0_R, b^0_{RW}$ are set to 0s, and in each step t we compute

$$(\mathsf{state}^t, i^t_x, i^t_R, i^t_{RW}, j^t, b^t) = \mathsf{CPU}(\mathsf{state}^{t-1}, b^{t-1}_x, b^{t-1}_R, b^{t-1}_{RW}),$$

we update $M_{RW}[j^t] := b^t$, and set $b^t_x := x[i^t_x], b^t_R := M_R[i^t_R]$, $b^t_{RW} := M_{RW}[i^t_{RW}]$. If state^t corresponds to some special state (say all 1s) then the computation terminates and the output is b^t. The size of the CPU step circuit is some fixed $|\mathsf{CPU}| = \mathrm{poly}(w)$.

Lemma 3.1. *Let f_{DB} be a RAM program with some database* DB *and address-size w. For a fixed input size N and run-time T, we can compile f_{DB} into a boolean RAM circuit C of compressed size* $\mathsf{size}_c(C) = \widetilde{O}(T) \cdot \mathrm{poly}(w)$ *with each* $\mathsf{dRead}, \mathsf{wRead}$ *gate having maximum load* $O(\max\{Tw, N, |\mathsf{DB}| + |f|\})$. *The expanded size of C is* $\mathsf{size}_e(C) = \widetilde{O}(T)(T + N + |\mathsf{DB}| + |f|)\mathrm{poly}(w)$. *Furthermore, the run-time of the compiler is* $O(\mathsf{size}_e(C))$.

Proof. The proof proceeds in two steps. First, we compile the RAM program f_{DB} into a *fixed-writes RAM program* f'_{DB}, where each step t writes to some fixed location $j(t)$ of the read/write memory that does not depends on the input of the computation (however, the bit being written can depend on the input); moreover it only writes to the first $O(t \cdot w)$ locations in memory within any t steps. Second, we compile a fixed-writes RAM program into a RAM circuit.

RAM to Fixed-Writes RAM. The compiled RAM program f' works by executing the original program f, but every time f wants to read/write to its "virtual memory" M_{RW}, the compiler translates this into a new sequence of read/write operations to its "physical memory" M'_{RW}. We think of the physical read/write memory M'_{RW} as containing a hierarchical data structure consisting of levels $\ell = 0, 1, \ldots$, where each level ℓ has slots for up to 2^ℓ location/value pairs (j, b) with $j \in \{0,1\}^w, b \in \{0,1\}$. In each time-step t, when f issues a write (j^*, b^*), the compiled program f' does the following:

- Find the largest integer ℓ^* such that 2^{ℓ^*} divides t.
- Take all the pairs (j, b) contained in the levels $\ell = 0, \ldots, \ell^* - 1$ of the data structure together with the new pair (j^*, b^*) and put them in level ℓ^*, sorted by the index j. If there are multiple pairs with the same j, take only the freshest copy from the smallest level and discard the rest. This sorting is done via a data-independent sorting network [Bat68, AKS83, Goo14]. At the end of this process, the levels $\ell = 0, \ldots, \ell^* - 1$ are set to empty.

In each time-step t, whenever f issues a read i_{RW} to read/write memory, do the following:

- For each level $\ell = 0, \ldots, \lfloor \log t \rfloor$, do a binary search for a tuple of the form (i_{RW}, b) in that level, and output the first one found. If none are found, output 0.

To see correctness, we argue that at the end of any step t having binary representation $t = \sum_{\ell=0}^{\lfloor \log t \rfloor} t_\ell 2^\ell$, all the levels ℓ for which $t_\ell = 0$ are empty. This is easy to see by induction: whenever t is incremented, all the bit of t_ℓ that turned from 1 to 0 correspond to levels ℓ that are emptied out. This implies that at the beginning of step t, the level ℓ^* is empty. Therefore the $1 + \sum_{\ell=0}^{\ell^*-1} 2^\ell \le 2^{\ell^*}$ tuples contained in the levels $\ell = 0, \ldots, \ell^* - 1$ of the data structure together with the new pair

(j^*, b^*) can fit in level ℓ^*. It is also easy to see that the compiled program f' has fixed data-independent writes and that it only uses the first $O(tw)$ locations of memory after t steps. Finally, the run-time of any read-operation t is at most $O(\log^2 tw)$ corresponding to $O(\log t)$ copies of binary search with data size $\leq t$. The run-time of any write operation t is $O(2^{\ell^*}\ell^* w)$ corresponding to sorting 2^{ℓ^*} items, where ℓ^* is the largest integer such that 2^{ℓ^*} divides t. To compute the amortized run-time of T write operations, note that each level ℓ acts as ℓ^* in $T/2^\ell$ operations for a total cost of

$$\sum_{\ell=0}^{\lfloor \log T \rfloor} O((T/2^\ell)(2^\ell \cdot \ell w)) = O(T(\log^2 T)w).$$

Therefore, if the run-time of $f_{\mathsf{DB}}(x)$ is T then the run-time of $f'_{\mathsf{DB}}(x)$ is $O(T \log^2 Tw)$.

Fixed-Writes RAM to RAM Circuit. Given a fixed-writes RAM program f'_{DB} along with a time bound T' and an input size N we compile it into a RAM circuit C. Set $\mathsf{DB}' = (f', \mathsf{DB})$ to consist of the code f' and the database DB. Assume that f'_{DB} only writes to the first $L_{RW} = O(T'w)$ locations of read/write memory within time T'. We can assume the size of $|\mathsf{DB}'|$, the input length N and L_{RW} are all powers-of-2 by padding with 0's. Lastly, we assume w.l.o.g. that all the indices i_x, i_R, i_{RW}, j being read/written during the first T' steps of the execution of $f'_{\mathsf{DB}}(x)$ for $x \in \{0,1\}^N$ are in the appropriate ranges $i_x < N, i_R < |\mathsf{DB}'|, i_{RW} < L_{RW}, j < L_{RW}$, and hence we can just consider corresponding first few bits of the index (i.e., can think of $i_x \in \{0,1\}^{\log N}$). The RAM circuit C is constructed as follows.

- *CPU Steps:* We "chain" together T' copies of the CPU circuit (consisting solely of NAND gates) connecting up the wires that correspond to state from the output of t's circuit to the input of the $(t+1)$'st circuit. We fix all the input wires to the first CPU circuit in the chain to 0's.

- *Reads to x:* We create N input wires corresponding to the input x. For each step t, we take the wires corresponding to the index i_x^t coming from the output of the t'th CPU circuit, together with the input wires for x and add a $\mathsf{wRead}(i_x^t, x)$ gate, connecting its output to the input bit b_x^{t+1} in $(t+1)$'st CPU circuit.

- *Reads to M_R:* For each step t, we take the wires corresponding to the index i_R^t coming from the output of the t'th CPU circuit and add a $\mathsf{dRead}_{\mathsf{DB}'}(i_R^t)$ gate, connecting its output to the input bit b_R^{t+1} in $(t+1)$'st CPU circuit.

- *Writes to M_{RW}:* We create an initial bundle of L_{RW} wires corresponding to memory M_{RW} that we initialize to 0. (Concretely, we can create a constant-0 circuit made up of NAND gates and think of it as having fan-out L_{RW} which defines the initial bundle.) We maintain a bundle of wires corresponding to M_{RW} alongside the CPU circuits, but we periodically exchange which wires are in the bundle. In each step t we compute the "write location" $j^t = j(t)$ using the fact that in a fixed-writes RAM program, it does not depend on the input x.[6] We then replace the (j^t)'th wire in the bundle with the output wire b^t of the t'th CPU circuit.

- *Reads to M_{RW}:* For each step t, we take the wires corresponding to the index i_{RW}^t coming from the output of the t'th CPU circuit and the current bundle of wires corresponding to

[6]We can compute all the j^t values efficiently in time $O(T)$ by running $f'_{\mathsf{DB}}(0^N)$ on a dummy input 0^N.

M_{RW} and add a $\mathsf{wRead}(i^t_{RW}, M_{RW})$ gate, connecting its output to the input bit b^{t+1}_{RW} in $(t+1)$'st CPU circuit.

- *Output:* We set the output bit $b^{T'}$ of the T'th CPU circuit as the overall output of the RAM circuit.

The resulting RAM circuit has compressed size $\mathsf{size}_c(C) = O(T'\text{poly}(w))$ and expanded size $\mathsf{size}_e(C) = O(T'(\text{poly}(w) + Tw + N + |\mathsf{DB}| + |f|))$.

Combining the two steps we get the lemma. □

Note that the above gives the worst-case complexity of converting a RAM program into a RAM circuit, especially when it comes to the expanded circuit size $\mathsf{size}_e(C)$. In many special cases, we can do significantly better. For example, if we start with a RAM program that only performs a few accesses to memory and then does some heavy computation without random accesses, we will get a much smaller expanded circuit size.

4 Homomorphic Operations for RAM Circuits

Definition 4.1 (Homomorphic Operations). *Let $n, q \in \mathbb{N}$ and $m \geq n \cdot \lceil \log q \rceil$. A* system of homomorphic operations *with error growth γ and circuit class $\mathcal{C}$ is a pair of deterministic algorithms* (EvalPK, EvalCT) *with the following syntax:*

- $\mathbf{A}_C := \mathsf{EvalPK}(C, \mathbf{A}_1, \dots, \mathbf{A}_\ell)$*: On input a circuit $C : \{0,1\}^\ell \to \{0,1\} \in \mathcal{C}$ and matrices $\mathbf{A}_1, \dots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m} \in \mathcal{C}$, output a matrix $\mathbf{A}_C \in \mathbb{Z}_q^{n \times m}$.*

- $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathbf{A}_1, \dots, \mathbf{A}_\ell, x, \mathbf{b}_1, \dots, \mathbf{b}_\ell)$*: On input a circuit $C \in \mathcal{C}$, matrices $\mathbf{A}_1, \dots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m}$, an input $x \in \{0,1\}^\ell$ and vectors $\mathbf{b}_x \in \mathbb{Z}_q^{\ell m}$, output a vector $\mathbf{b}_C \in \mathbb{Z}_q^m$ that encodes the evaluation $C(x)$.*

The algorithms satisfy the following correctness property: For all circuits $C \in \mathcal{C}$, matrices $\mathbf{A}_1, \dots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m}$, and inputs $x \in \{0,1\}^\ell$, if there exists a vector $\mathbf{s} \in \mathbb{Z}_q^n$ such that each of the vectors $\mathbf{b}_i$ is of the form

$$\mathbf{b}_i^\top = \mathbf{s}^\top(\mathbf{A}_i - x_i\mathbf{G}) + \mathbf{e}_i^\top, \tag{3}$$

for some $\mathbf{e}_i$, then $\mathbf{A}_C := \mathsf{EvalPK}(C, \{\mathbf{A}_i\}_{i \leq \ell})$, $\mathbf{b}_C = \mathsf{EvalCT}(C, \{\mathbf{A}_i\}_{i \leq \ell}, x, \{\mathbf{b}_i\}_{i \leq \ell})$ satisfy

$$\mathbf{b}_C^\top = \mathbf{s}^\top(\mathbf{A}_C - C(x)\mathbf{G}) + \mathbf{e}_C^\top, \tag{4}$$

where $\mathbf{e}_C$ is another short error vector with $\|\mathbf{e}_C\|_\infty \leq \gamma(C) \cdot (\max_i \|\mathbf{e}_i\|_\infty)$.

For the sake of convenience, we sometimes write $\mathbf{A} = [\mathbf{A}_1, | \cdots | \mathbf{A}_\ell] \in \mathbb{Z}_q^{n \times m\ell}$ and similarly $\mathbf{b}^\top = [\mathbf{b}_1^\top | \cdots | \mathbf{b}_\ell^\top] \in \mathbb{Z}_q^{m\ell}$. In this case, equation (3) can be written as $\mathbf{b}^\top = \mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}^\top$. We also define the following notion of homomorphic operations that satisfy an additional linearity property.

Definition 4.2 (Linearity). *Let $n, q \in \mathbb{N}$ and $m \geq n \cdot \lceil \log q \rceil$. A* linear system of homomorphic operations *with error growth γ and circuit class $\mathcal{C}$ is a pair of deterministic algorithms* (EvalPK, $\mathsf{EvalCTCoeffs}$) *where* EvalPK *has the same syntax as in Definition 4.1 and* $\mathsf{EvalCTCoeffs}$ *has syntax:*

- $\mathbf{H}_{C,x} := \mathsf{EvalCTCoeffs}(C, \mathbf{A}, x)$: *On input a circuit* $C : \{0,1\}^\ell \to \{0,1\}$, *matrix* $\mathbf{A} \in \mathbb{Z}_q^{n\times m\ell}$, *an input* $x \in \{0,1\}^\ell$, *output a matrix* $\mathbf{H}_{C,x} \in \mathbb{Z}_q^{m\ell\times m}$.

The algorithms satisfy the following correctness property: For all circuits $C \in \mathcal{C}$, *matrices* $\mathbf{A} \in \mathbb{Z}_q^{n\times m\ell}$ *and inputs* $x \in \{0,1\}^\ell$, *the matrix* $\mathbf{H}_{C,x} = \mathsf{EvalCTCoeffs}(C, \mathbf{A}, x)$ *has* $\left\|\mathbf{H}_{C,x}^\top\right\| \le \gamma(C)$ *and satisfies*

$$(\mathbf{A} - x \otimes \mathbf{G}) \cdot \mathbf{H}_{C,x} = \mathbf{A}_C - C(x)\mathbf{G}, \tag{5}$$

where $\mathbf{A}_C = \mathsf{EvalPK}(C, \mathbf{A})$.

It is easy to see a linear system of homomorphic operations immediately implies a system in the sense of Definition 4.1 because one can simply define $\mathsf{EvalCT}(C, \mathbf{A}, x, \mathbf{b}) = \mathbf{b}^\top \cdot \mathbf{H}_{C,x}$ for $\mathbf{H}_{C,x} := \mathsf{EvalCTCoeffs}(C, \mathbf{A}, x)$ in which case (5) $\Rightarrow$ (4):

$$\mathbf{b}_C^\top = \mathbf{b}^\top \cdot \mathbf{H}_{C,x} = (\mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}^\top)\mathbf{H}_{C,x} = \mathbf{s}^\top(\mathbf{A}_C - C(x)\mathbf{G}) + \mathbf{e}^\top\mathbf{H}_{C,x}$$

with the same error growth $\mathbf{e}_C^\top = \mathbf{e}^\top\mathbf{H}_{C,x}$ having $\|\mathbf{e}_C\|_\infty \le \gamma(C) \cdot \|\mathbf{e}\|_\infty$.

Theorem 4.3 ([BGG+14]). *There is a linear system of homomorphic operations* (EvalPK, $\mathsf{EvalCTCoeffs}$) *for the class of boolean circuits composed of NAND gates. The system has error growth* $\gamma(C, n, q) \le (m+1)^d$ *where d is the depth of the circuit* C. *The algorithms* EvalPK *and* $\mathsf{EvalCTCoeffs}$ *each run in time* $|C| \cdot \mathrm{poly}(n, \log q)$.

We briefly recall the core construction of the BGG+ homomorphic operations below as it will serve as inspiration for our constructions throughout this section. Let $x_1, x_2 \in \{0,1\}$ and let $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{Z}_q^m$ be their vector encodings with respect to matrices $\mathbf{A}_1, \mathbf{A}_2 \in \mathbb{Z}_q^{n\times m}$. For addition gates, the algorithms EvalPK and EvalCT are defined as

$$\begin{aligned}
\mathsf{EvalPK}(+, \mathbf{A}_1, \mathbf{A}_2) &= \mathbf{A}_+ = \mathbf{A}_1 + \mathbf{A}_2 \\
\mathsf{EvalCTCoeffs}(+, \mathbf{A}_1, \mathbf{A}_2, x_1, x_2) &= \mathbf{H}_{+,x} = [\mathbf{I} \mid \mathbf{I}]^\top \\
\mathsf{EvalCT}(+, \mathbf{A}_1, \mathbf{A}_2, x_1, x_2, \mathbf{b}_1, \mathbf{b}_2) &= \mathbf{b}_+ = \mathbf{b}_1 + \mathbf{b}_2.
\end{aligned}$$

And for multiplication gates, the algorithms are defined as

$$\begin{aligned}
\mathsf{EvalPK}(\times, \mathbf{A}_1, \mathbf{A}_2) &= \mathbf{A}_\times = \mathbf{A}_1 \cdot \mathbf{G}^{-1}(\mathbf{A}_2) \\
\mathsf{EvalCTCoeffs}(\times, \mathbf{A}_1, \mathbf{A}_2, x_1, x_2) &= \mathbf{H}_{\times,x} = \begin{pmatrix} \mathbf{G}^{-1}(\mathbf{A}_2) \\ x_1\mathbf{I} \end{pmatrix} \\
\mathsf{EvalCT}(\times, \mathbf{A}_1, \mathbf{A}_2, x_1, x_2, \mathbf{b}_1, \mathbf{b}_2) &= \mathbf{b}_\times = \mathbf{b}_1\mathbf{G}^{-1}(\mathbf{A}_2) + x_1 \cdot \mathbf{b}_2.
\end{aligned}$$

Lastly, for NAND gate, the algorithms are defined as

$$\begin{aligned}
\mathsf{EvalPK}(\mathsf{NAND}, \mathbf{A}_1, \mathbf{A}_2) &= \mathbf{A}_{\mathsf{NAND}} = \mathbf{G} - \mathbf{A}_1 \cdot \mathbf{G}^{-1}(\mathbf{A}_2) \\
\mathsf{EvalCTCoeffs}(\mathsf{NAND}, \mathbf{A}_1, \mathbf{A}_2, x_1, x_2) &= \mathbf{H}_{\mathsf{NAND},x} = \begin{pmatrix} -\mathbf{G}^{-1}(\mathbf{A}_2) \\ -x_1\mathbf{I} \end{pmatrix} \\
\mathsf{EvalCT}(\mathsf{NAND}, \mathbf{A}_1, \mathbf{A}_2, x_1, x_2, \mathbf{b}_1, \mathbf{b}_2) &= \mathbf{b}_{\mathsf{NAND}} = -(\mathbf{b}_1\mathbf{G}^{-1}(\mathbf{A}_2) + x_1 \cdot \mathbf{b}_2).
\end{aligned}$$

Composition. Note that the operations $\mathsf{EvalPK}, \mathsf{EvalCT}, \mathsf{EvalCTCoeffs}$ compose nicely. Assume we have these procedures for some functions $C_1, \ldots, C_k : \{0,1\}^\ell \to \{0,1\}$ and $C : \{0,1\}^k \to \{0,1\}$. Then we can define these procedures for $C^*(x) = C(C_1(x), \ldots, C_k(x))$ by computing:

$$\begin{aligned}
&\mathsf{EvalPK}(C^*, \mathbf{A}_1, \ldots, \mathbf{A}_\ell) = \mathsf{EvalPK}(C, \mathbf{A}_{C_1}, \ldots, \mathbf{A}_{C_k})\\
&\quad \text{where } \mathbf{A}_{C_i} = \mathsf{EvalPK}(C_i, \mathbf{A}_1, \ldots, \mathbf{A}_\ell)\\
&\mathsf{EvalCTCoeffs}(C^*, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, x) = [\mathbf{H}_{C_1,x} \mid \ldots \mid \mathbf{H}_{C_k,x}]\mathbf{H}_{C,y}\\
&\quad \text{where } \mathbf{H}_{C_i,x} = \mathsf{EvalCTCoeffs}(C_i, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, x)\\
&\quad \text{and } \mathbf{H}_{C,y} = \mathsf{EvalCTCoeffs}(C, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, y) \text{ for } y = (C_1(x), \ldots, C_k(x)) \in \{0,1\}^k\\
&\mathsf{EvalCT}(C^*, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, x, \mathbf{b}_1, \ldots, \mathbf{b}_\ell) = \mathsf{EvalCT}(C, \mathbf{A}_{C_1}, \ldots, \mathbf{A}_{C_\ell}, y, \mathbf{b}_{C_1}, \ldots, \mathbf{b}_{C_\ell})\\
&\quad \text{where } \mathbf{b}_{C_i} = \mathsf{EvalCT}(C_i, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, x, \mathbf{b}_1, \ldots, \mathbf{b}_\ell)
\end{aligned}$$

The correctness property in (5) holds since:

$$\begin{aligned}
(\mathbf{A} - x \otimes \mathbf{G}) \cdot \mathbf{H}_{C^*,x} &= (\mathbf{A} - x \otimes \mathbf{G}) \cdot [\mathbf{H}_{C_1,x} \mid \ldots \mid \mathbf{H}_{C_k,x}]\mathbf{H}_{C,y}\\
&= ([\mathbf{A}_{C_1} \mid \ldots \mid \mathbf{A}_{C_k}] - y \oplus \mathbf{G})\mathbf{H}_{C,y}\\
&= \mathbf{A}_{C^*} - C(y)\mathbf{G} = \mathbf{A}_{C^*} - C^*(x)\mathbf{G}
\end{aligned}$$

Moreover the error growth follows $\gamma(C^*) \leq \gamma(C) \max_i \gamma(C_i)$

The above works when the circuits C_i all operate over the entire input. We can also apply it to circuits C_i that operate on different subsets of the input bits. The only difference is that we need to pad the matrices $\mathbf{H}_{C_i,x_i}$ with appropriate 0's for the bits that aren't touched.

Preprocessing. In the remainder of this section we construct a linear system of homomorphic operations for RAM circuits by extending the BGG+ construction to handle database read and wire read gates. The core idea behind our construction is to preprocess the work of computing the matrices $\mathbf{H}_{C,x}$ in advance during the run time of EvalPK, before we know the value of x. In that pursuit we also define an alternate notion of homomorphic operations where we allow EvalPK to output a data structure for EvalCT to use during it's computation.

Definition 4.4 (Preprocessing Homomorphic Operations)**.** *A* preprocessing system of homomorphic operations *is a pair of algorithms* (EvalPK, EvalCT) *as in Definition 4.1 except with the modified syntax:*

- $(\mathbf{A}_C, \mathsf{DS}) := \mathsf{EvalPK}(C, \mathbf{A})$ *additionally generates a data structure* DS.
- $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b})$*: additionally takes a data structure* DS *as input.*

The algorithms must satisfy the same correctness property as in Definition 4.1 where the data structure DS *generated by* EvalPK *is passed to* EvalCT*. Similarly, a* preprocessing linear system of homomorphic operations *is a pair of algorithms* (EvalPK, EvalCTCoeffs) *as in Definition 4.2 but where* EvalPK *additionally generates* DS *and* EvalCTCoeffs *additionally takes* DS *as input. The algorithms must satisfy the same correctness property as in Definition 4.2, where the data structure* DS *generated by* EvalPK *is passed to is passed to* EvalCTCoeffs.

4.1 Database Read Gates

Theorem 4.5. *There exists a preprocessing linear system of homomorphic operations* $(\mathsf{EvalPK}, \mathsf{EvalCTCoeffs})$ *for the class of database read gates* $\{\mathsf{dRead}_{\mathsf{DB}} : \mathsf{DB} \in \{0,1\}^{2^\ell}, \ell \in \mathbb{N}\}$*. The system has error growth* $\gamma(\mathsf{dRead}_{\mathsf{DB}}) = \ell m$ *for a database* $\mathsf{DB} \in \{0,1\}^{2^\ell}$*, and it has efficiency properties:*

- *The size of the data structure* DS *and run time of* $\mathsf{EvalPK}(\mathsf{dRead}_{\mathsf{DB}}, \mathbf{A})$ *are both bounded by* $O(\ell \cdot 2^\ell) \cdot \mathrm{poly}(n, \log q)$.
- $\mathsf{EvalCTCoeffs}(\mathsf{dRead}_{\mathsf{DB}}, \mathsf{DS}, \mathbf{A}, x)$ *runs in time* $\mathrm{poly}(\ell, n, \log q)$.

Proof. Let $\mathsf{select} : \{0,1\}^3 \to \{0,1\}$ be the function $\mathsf{select}(y_0, y_1, x) = (1-x) \cdot y_0 + x \cdot y_1 = y_x$. We can write $\mathsf{dRead}_{\mathsf{DB}}$ recursively as

$$\mathsf{dRead}_{\mathsf{DB}}(x_1, \dots, x_\ell) = \mathsf{select}(\mathsf{dRead}_{\mathsf{DB}^{\mathsf{L}}}(x_1, \dots, x_{\ell-1}), \mathsf{dRead}_{\mathsf{DB}^{\mathsf{R}}}(x_1, \dots, x_{\ell-1}), x_\ell)$$

where $\mathsf{DB}^{\mathsf{L}} = (\mathsf{DB}_0, \dots, \mathsf{DB}_{L/2-1}), \mathsf{DB}^{\mathsf{R}} = (\mathsf{DB}_{L/2}, \dots, \mathsf{DB}_{L-1}) \in \{0,1\}^{L/2}$ are the "left" and "right" halves of $\mathsf{DB} = (\mathsf{DB}_0, \dots, \mathsf{DB}_{L-1})$ respectively. We abbreviate $\mathsf{dRead}_{\mathsf{DB}}$ as g and $\mathsf{dRead}_{\mathsf{DB}^{\mathsf{L}}}$, $\mathsf{dRead}_{\mathsf{DB}^{\mathsf{R}}}$ as g^{L}, g^{R} respectively so that $g(x) = \mathsf{select}(g^{\mathsf{L}}(x^-), g^{\mathsf{R}}(x^-), x_\ell)$ for $x^- = (x_1, \dots, x_{\ell-1})$. Our construction recursively computes the matrices and data structures for the left and right sub-instances $g^{\mathsf{L}}, g^{\mathsf{R}}$, and then composes them with the constant-size circuit for select. Formally, we define EvalPK and $\mathsf{EvalCTCoeffs}$ as follows:

$\mathsf{EvalPK}(g, \mathbf{A})$**:** Let $L = 2^\ell$ be the length of DB.

1. If $\ell = 1$, parse $\mathsf{DB} = (\mathsf{DB}_0, \mathsf{DB}_1) \in \{0,1\}^2$. Let $\mathsf{select}_{\mathsf{DB}_0,\mathsf{DB}_1} : \{0,1\} \to \{0,1\}$ be the function $\mathsf{select}_{\mathsf{DB}_0,\mathsf{DB}_1}(x) = (1-x)\mathsf{DB}_0 + x \cdot \mathsf{DB}_1 = \mathsf{DB}_x$. Set:

$$\begin{aligned} \mathbf{A}_g &:= \mathsf{EvalPK}(\mathsf{select}_{\mathsf{DB}_0,\mathsf{DB}_1}, \mathbf{A}) \\ &= \mathsf{DB}_0 \cdot \mathbf{G} + \mathbf{A}(\mathbf{G}^{-1}(\mathsf{DB}_1 \cdot \mathbf{G}) - \mathbf{G}^{-1}(\mathsf{DB}_0 \cdot \mathbf{G})) \\ \mathbf{H}_{g,x} &:= \mathsf{EvalCTCoeffs}(\mathsf{select}_{\mathsf{DB}_0,\mathsf{DB}_1}, \mathbf{A}, x) \\ &= \mathbf{G}^{-1}(\mathsf{DB}_1 \cdot \mathbf{G}) - \mathbf{G}^{-1}(\mathsf{DB}_0 \cdot \mathbf{G}) \quad \text{for } x \in \{0,1\} \end{aligned}$$

   and output $(\mathbf{A}_g, \mathsf{DS}_g = \{\mathbf{H}_{g,x}\}_{x \in \{0,1\}})$.[7]

2. Recursively call $(\mathbf{A}_{g^{\mathsf{L}}}, \mathsf{DS}_{g^{\mathsf{L}}}) := \mathsf{EvalPK}(g^{\mathsf{L}}, \mathbf{A}^-)$ and $(\mathbf{A}_{g^{\mathsf{R}}}, \mathsf{DS}_{g^{\mathsf{R}}}) := \mathsf{EvalPK}(g^{\mathsf{R}}, \mathbf{A}^-)$ where $\mathbf{A}^- = [\mathbf{A}_1 \mid \cdots \mid \mathbf{A}_{\ell-1}]$.

3. Set $\mathbf{A}_g := \mathsf{EvalPK}(\mathsf{select}, \mathbf{A}_{g^{\mathsf{L}}}, \mathbf{A}_{g^{\mathsf{R}}}, \mathbf{A}_\ell) = \mathbf{A}_{g^{\mathsf{L}}} + \mathbf{A}_\ell \cdot (\mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{L}}}))$.

4. For all $(y_0, y_1, x_\ell) \in \{0,1\}^3$ compute[8]

$$\begin{aligned} \mathbf{H}_{\mathsf{select},y_0,y_1,x_\ell} &:= \mathsf{EvalCTCoeffs}(\mathsf{select}, \mathbf{A}_{g^{\mathsf{L}}}, \mathbf{A}_{g^{\mathsf{R}}}, \mathbf{A}_\ell, y_0, y_1, x_\ell) \\ &= \begin{pmatrix} (1-x_\ell)\mathbf{I} \\ x_\ell \mathbf{I} \\ \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{L}}}) \end{pmatrix} \end{aligned}$$

[7] In this case it turns out that $\mathbf{H}_{g,0} = \mathbf{H}_{g,1}$, but we do not rely on this fact.

[8] The matrices are independent of y_0, y_1 so it suffices to compute only 2 rather than 8 of them.

For each $x \in \{0,1\}^\ell$, look up $\mathbf{H}_{g^{\mathsf{L}},x^-} \in \mathsf{DS}_{g^{\mathsf{L}}}$ and $\mathbf{H}_{g^{\mathsf{R}},x^-} \in \mathsf{DS}_{g^{\mathsf{R}}}$. Then compute $\mathbf{H}_{g,x}$ by composing $\mathbf{H}_{g^{\mathsf{L}},x^-}, \mathbf{H}_{g^{\mathsf{R}},x^-}, \mathbf{H}_{\mathsf{select},g^{\mathsf{L}}(x^-),g^{\mathsf{R}}(x^-),x_\ell}$, resulting in:

$$\begin{aligned}\mathbf{H}_{g,x} &= \begin{pmatrix}\mathbf{H}_{g^{\mathsf{L}},x^-} & \mathbf{H}_{g^{\mathsf{R}},x^-} & 0 \\ 0 & 0 & \mathbf{I}\end{pmatrix} \cdot \mathbf{H}_{\mathsf{select},g^{\mathsf{L}}(x^-),g^{\mathsf{R}}(x^-),x_\ell} \\ &= \begin{pmatrix}(1-x_\ell)\mathbf{H}_{g^{\mathsf{L}},x^-} + x_\ell \mathbf{H}_{g^{\mathsf{R}},x^-} \\ \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{L}}})\end{pmatrix}\end{aligned}$$

5. Output $(\mathbf{A}_g, \mathsf{DS} = \{\mathbf{H}_{g,x}\}_{x\in\{0,1\}^\ell})$

$\mathsf{EvalCTCoeffs}(\mathsf{dRead}_{\mathsf{DB}}, \mathsf{DS}, \mathbf{A}, x)$**:** Look up $\mathbf{H}_{g,x} \in \mathsf{DS}$ and output it.

In the base case where $\ell = 1$, correctness follows via the correctness of $\mathsf{EvalPK}, \mathsf{EvalCTCoeffs}$ for $\mathsf{select}_{\mathsf{DB}_0,\mathsf{DB}_1}$. In detail, for $x \in \{0,1\}$ and $\mathsf{DB} = (\mathsf{DB}_0, \mathsf{DB}_1) \in \{0,1\}^2$, we have

$$\begin{aligned}(\mathbf{A} - x\mathbf{G})\mathbf{H}_{g,x} &= (\mathbf{A} - x\mathbf{G})(\mathbf{G}^{-1}(\mathsf{DB}_1 \cdot \mathbf{G}) - \mathbf{G}^{-1}(\mathsf{DB}_0 \cdot \mathbf{G})) \\ &= \mathbf{A}(\mathbf{G}^{-1}(\mathsf{DB}_1 \cdot \mathbf{G}) - \mathbf{G}^{-1}(\mathsf{DB}_0 \cdot \mathbf{G})) - x\mathsf{DB}_1 \cdot \mathbf{G} + x\mathsf{DB}_0 \cdot \mathbf{G} \\ &= \mathbf{A}_g - (1-x)\mathsf{DB}_0 \cdot \mathbf{G} - x\mathsf{DB}_1 \cdot \mathbf{G} \\ &= \mathbf{A}_g - \mathsf{DB}_x \cdot \mathbf{G}\end{aligned}$$

We also have that $\mathbf{H}_{g,x}$ is an $m \times m$ matrix with entries in $\{-1,0,1\}$, hence it has $\|\mathbf{H}_{g,x}^\top\| \le m$.

For the inductive case, assume that correctness holds for $\ell - 1$. First we verify the correctness of $\mathsf{EvalPK}, \mathsf{EvalCTCoeffs}$ for the select function. For any y_0, y_1, x_ℓ:

$$\begin{aligned}&[\mathbf{A}_{g^{\mathsf{L}}} - y_0\mathbf{G} \mid \mathbf{A}_{g^{\mathsf{R}}} - y_1\mathbf{G} \mid \mathbf{A}_\ell - x_\ell\mathbf{G}]\mathbf{H}_{\mathsf{select},y_0,y_1,x_\ell} \\ =&[\mathbf{A}_{g^{\mathsf{L}}} - y_0\mathbf{G} \mid \mathbf{A}_{g^{\mathsf{R}}} - y_1\mathbf{G} \mid \mathbf{A}_\ell - x\mathbf{G}] \begin{pmatrix}(1-x_\ell)\mathbf{I} \\ x_\ell\mathbf{I} \\ \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{L}}})\end{pmatrix} \\ =&\underbrace{\mathbf{A}_{g^{\mathsf{L}}} + \mathbf{A}_\ell \cdot (\mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{g^{\mathsf{L}}}))}_{=\mathbf{A}_g} - y_{x_\ell}\mathbf{G}.\end{aligned}$$

Now, by composing select with $g^{\mathsf{L}}, g^{\mathsf{R}}$, we get that for any $x \in \{0,1\}^\ell$ and $\mathsf{DB} \in \{0,1\}^{2^\ell}$:

$$\begin{aligned}(\mathbf{A} - x \otimes \mathbf{G})\mathbf{H}_{g,x} &= (\mathbf{A} - x \otimes \mathbf{G}) \begin{pmatrix}\mathbf{H}_{g^{\mathsf{L}},x^-} & \mathbf{H}_{g^{\mathsf{R}},x^-} & 0 \\ 0 & 0 & \mathbf{I}\end{pmatrix} \cdot \mathbf{H}_{\mathsf{select},g^{\mathsf{L}}(x^-),g^{\mathsf{R}}(x^-),x_\ell} \\ &= [\mathbf{A}_{g^{\mathsf{L}}} - g^{\mathsf{L}}(x^-)\mathbf{G} \mid \mathbf{A}_{g^{\mathsf{R}}} - g^{\mathsf{R}}(x^-)\mathbf{G} \mid \mathbf{A}_\ell - x_\ell\mathbf{G}] \cdot \mathbf{H}_{\mathsf{select},g^{\mathsf{L}}(x^-),g^{\mathsf{R}}(x^-),x_\ell} \\ &= \mathbf{A}_g - \mathsf{select}(g^{\mathsf{L}}(x^-), g^{\mathsf{R}}(x^-), x_\ell) \cdot \mathbf{G} \\ &= \mathbf{A}_g - \mathsf{DB}_x \cdot \mathbf{G}\end{aligned}$$

Additionally we can inductively assume that $\mathbf{H}_{g^{\mathsf{L}},x^-}, \mathbf{H}_{g^{\mathsf{R}},x^-}$ both have entries in $\{-1,0,1\}$, hence $\mathbf{H}_{g,x}$ does too because only one of x_ℓ and $(1-x_\ell)$ are nonzero. Thus we have $\|\mathbf{H}_{g,x}^\top\| \le \ell m$.

It remains to show efficiency. First it is clear that $\mathsf{EvalCTCoeffs}$ runs in the desired time. Let $T(\ell)$ denote the run time of EvalPK when run on a database of size $L = 2^\ell$. After recursively computing $(\mathbf{A}_{g^{\mathsf{L}}}, \mathsf{DS}_{g^{\mathsf{L}}})$ and $(\mathbf{A}_{g^{\mathsf{R}}}, \mathsf{DS}_{g^{\mathsf{R}}})$, the matrix $\mathbf{A}_g$ and each matrix $\mathbf{H}_{g,x}$ can be computed as a linear combination of entries in the outputs of the recursive calls. Since there are 2^ℓ coefficient matrices to compute, we can see that the run time of EvalPK satisfies the recurrence

$$T(\ell) = 2T(\ell-1) + O(2^\ell) \cdot \mathrm{poly}(n, \log q),$$

which implies $T(\ell) = O(\ell 2^\ell) \cdot \mathrm{poly}(n, \log q)$ as desired. $\square$

4.2 Matrix Read Gates

As a stepping stone towards constructing our system of homomorphic operations for wire read gates wRead, we generalize the theorem from the previous section to handle reading from "matrix-valued databases". That is, for a database $\mathbf{M} = (\mathbf{M}_0, \ldots, \mathbf{M}_{L-1}) \in (\mathbb{Z}_q^{n\times m})^L$, we define the matrix read function $\mathsf{mRead}_{\mathbf{M}}(x) = \mathbf{M}_x$. As was observed in [BTVW17], it is possible to extend the notion of homomorphic operations to capture matrix valued functions by slightly modifying the correctness property.

Definition 4.6 (Matrix-valued Homomorphic Operations). *A system of homomorphic operations with error growth γ and a class of matrix-valued circuits $\mathcal{C}$ is a pair of algorithms* (EvalPK, EvalCT) *with syntax as in Definition 4.1 the following correctness property: For all circuits $\mathbf{C} \in \mathcal{C}$, matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m\ell}$, and inputs $x \in \{0,1\}^\ell$, if there exists a vector $\mathbf{s} \in \mathbb{Z}_q^n$ such that $\mathbf{b}^\top = \mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}^\top$ for some short error vector $\mathbf{e}$, then the output $\mathbf{b}_{\mathbf{C}} =$* EvalCT$(\mathbf{C}, \mathbf{A}, x, \mathbf{b})$ *satisfies*

$$\mathbf{b}_{\mathbf{C}}^\top = \mathbf{s}^\top(\mathbf{A}_{\mathbf{C}} - \mathbf{C}(x)) + \mathbf{e}_{\mathbf{C}}^\top,$$

where $\mathbf{e}_C$ is another short error vector with $\|\mathbf{e}_{\mathbf{C}}\|_\infty \le \gamma(\mathbf{C}) \cdot \|\mathbf{e}\|_\infty$.

Similarly we define a linear system of homomorphic operations with γ and $\mathcal{C}$ as above as a pair of algorithms (EvalPK, EvalCTCoeffs) *with syntax as in Definition 4.2 and the following correctness property: For all circuits $\mathbf{C} \in \mathcal{C}$, matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m\ell}$ and inputs $x \in \{0,1\}^\ell$, the matrix $\mathbf{H}_{\mathbf{C},x} =$* EvalCTCoeffs$(\mathbf{C}, \mathbf{A}, x)$ *has $\left\|\mathbf{H}_{\mathbf{C},x}^\top\right\| \le \gamma(\mathbf{C})$ and satisfies*

$$(\mathbf{A} - x \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{C},x} = \mathbf{A}_{\mathbf{C}} - \mathbf{C}(x),$$

where $\mathbf{A}_{\mathbf{C}} =$ EvalPK$(\mathbf{C}, \mathbf{A})$.

We also define the preprocessing variant of matrix-value homomorphic operations with the correctness property as above and syntax as in Definition 4.4.

Theorem 4.7 ([BTVW17]). *There is a linear system of homomorphic operations* (EvalPK, EvalCTCoeffs) *for the class of matrix-valued boolean circuits composed of NAND gates. The system has error growth $\gamma(\mathbf{C}) \le (m+1)^d \cdot \log q$ where d is the depth of the circuit $\mathbf{C}$. The algorithms* EvalPK *and* EvalCTCoeffs *run in time* $\mathrm{poly}(|\mathbf{C}|, n, \log q)$.

We now show how to use the homomorphic operations of [BTVW17] to create a preprocessing system for matrix-read gates mRead in the same manner as we constructed homomorphic operations for data-read gates.

Theorem 4.8. *There exists a preprocessing linear system of homomorphic operations* (EvalPK, EvalCTCoeffs) *for the class of matrix read gates $\{\mathsf{mRead}_{\mathbf{M}} : \mathbf{M} \in (\mathbb{Z}_q^{n\times m})^{2^\ell}, \ell \in \mathbb{N}\}$. The system has error growth $\gamma(\mathsf{mRead}_{\mathbf{M}}) \le \ell m$, and it has efficiency properties:*

- *The size of the data structure* DS *and run time of* $\mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{M}}, \mathbf{A})$ *are both bounded by* $O(\ell \cdot 2^\ell) \cdot \mathrm{poly}(n, \log q)$.
- $\mathsf{EvalCTCoeffs}(\mathsf{mRead}_{\mathbf{M}}, \mathsf{DS}, \mathbf{A}, x)$ *runs in time* $\mathrm{poly}(\ell, n, \log q)$.

Proof. We proceed by an almost identical argument to as in the proof of Theorem 4.5. Define the function $\mathsf{select} : (\mathbb{Z}_q^{n\times m})^2 \times \{0,1\} \to \mathbb{Z}_q^{n\times m}$ similarly to as in Section 4.1 only with matrix-valued inputs/outputs $\mathsf{select}(\mathbf{Y}_0, \mathbf{Y}_1, x) = (1-x)\mathbf{Y}_0 + x\mathbf{Y}_1 = \mathbf{Y}_x$. Recursively write

$$\mathsf{mRead}_{\mathbf{M}}(x) = \mathsf{select}(\mathsf{mRead}_{\mathbf{M}^{\mathsf{L}}}(x^-), \mathsf{mRead}_{\mathbf{M}^{\mathsf{R}}}(x^-), x_\ell),$$

where $x^- = (x_1, \ldots, x_{\ell-1})$ and $\mathbf{M}^{\mathsf{L}}, \mathbf{M}^{\mathsf{R}} \in (\mathbb{Z}_q^{n\times m})^{L/2}$ are respectively the "left" and "right" halves of $\mathbf{M}$ and we think of x_ℓ as the most significant bit in x. We abbreviate $\mathsf{mRead}_{\mathbf{M}}$ as h and $\mathsf{mRead}_{\mathbf{M}^{\mathsf{L}}}, \mathsf{mRead}_{\mathbf{M}^{\mathsf{R}}}$ as $h^{\mathsf{L}}, h^{\mathsf{R}}$ respectively. We define EvalPK and $\mathsf{EvalCTCoeffs}$ as follows:

$\mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{M}}, \mathbf{A})$**:** Let $L = 2^\ell$ be the length of $\mathbf{M}$.

1. If $\ell = 1$, parse $\mathbf{M} = (\mathbf{M}_0, \mathbf{M}_1) \in (\mathbb{Z}_q^{n\times m})^2$. Let $\mathsf{select}_{\mathbf{M}_0,\mathbf{M}_1}(x) = (1-x)\mathbf{M}_0 + x\mathbf{M}_1 = \mathbf{M}_x$. Set:

$$\begin{aligned}\mathbf{A}_h &:= \mathsf{EvalPK}(\mathsf{select}_{\mathbf{M}_0,\mathbf{M}_1}, \mathbf{A}) \\ &= \mathbf{M}_0 + \mathbf{A}(\mathbf{G}^{-1}(\mathbf{M}_1) - \mathbf{G}^{-1}(\mathbf{M}_1)) \\ \mathbf{H}_{h,x} &:= \mathsf{EvalCTCoeffs}(\mathsf{select}_{\mathbf{M}_0,\mathbf{M}_1}, \mathbf{A}, x) \\ &= \mathbf{G}^{-1}(\mathbf{M}_1) - \mathbf{G}^{-1}(\mathbf{M}_0) \quad \text{for } x \in \{0,1\}\end{aligned}$$

   and output $(\mathbf{A}_h, \mathsf{DS}_h = \{\mathbf{H}_{h,x}\}_{x\in\{0,1\}})$.

2. Recursively call $(\mathbf{A}_{h^{\mathsf{L}}}, \mathsf{DS}_{h^{\mathsf{L}}}) := \mathsf{EvalPK}(h^{\mathsf{L}}, \mathbf{A}^-)$ and $(\mathbf{A}_{h^{\mathsf{R}}}, \mathsf{DS}_{h^{\mathsf{R}}}) := \mathsf{EvalPK}(h^{\mathsf{R}}, \mathbf{A}^-)$ where $\mathbf{A}^- = [\mathbf{A}_1 \mid \cdots \mid \mathbf{A}_{\ell-1}]$.

3. Set $\mathbf{A}_h = \mathsf{EvalPK}(\mathsf{select}, \mathbf{A}_h^{\mathsf{L}}, \mathbf{A}_h^{\mathsf{R}}, \mathbf{A}_\ell) = \mathbf{A}_{h^{\mathsf{L}}} + \mathbf{A}_\ell \cdot (\mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{L}}}))$.

4. For all $(\mathbf{Y}_0, \mathbf{Y}_1, x_\ell) \in (\mathbb{Z}_q^{n\times m})^2 \times \{0,1\}$ compute[9]

$$\begin{aligned}\mathbf{H}_{\mathsf{select},\mathbf{Y}_0,\mathbf{Y}_1,x_\ell} &:= \mathsf{EvalCTCoeffs}(\mathsf{select}, \mathbf{A}_h^{\mathsf{L}}, \mathbf{A}_h^{\mathsf{R}}, \mathbf{A}_\ell, \mathbf{Y}_0, \mathbf{Y}_1, x_\ell) \\ &= \begin{pmatrix} (1-x_\ell)\mathbf{I} \\ x_\ell \mathbf{I} \\ \mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{L}}}) \end{pmatrix}.\end{aligned}$$

   For each $x \in \{0,1\}^\ell$, look up $\mathbf{H}_{h^{\mathsf{L}},x^-} \in \mathsf{DS}_{h^{\mathsf{L}}}$ and $\mathbf{H}_{h^{\mathsf{R}},x^-} \in \mathsf{DS}_{h^{\mathsf{R}}}$, then compose with $\mathbf{H}_{\mathsf{select},h^{\mathsf{L}}(x^-),h^{\mathsf{R}}(x^-),x_\ell}$ to compute

$$\mathbf{H}_{h,x} = \begin{pmatrix} (1-x_\ell)\mathbf{H}_{h^{\mathsf{L}},x^-} + x_\ell \mathbf{H}_{h^{\mathsf{R}},x^-} \\ \mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{L}}}) \end{pmatrix}$$

5. Output $(\mathbf{A}_h, \mathsf{DS} = \{\mathbf{H}_{h,x}\}_{x\in\{0,1\}^\ell})$

[9] As in the data-read case, these matrices are independent of $\mathbf{Y}_0, \mathbf{Y}_1$ so it suffices to compute only 2 matrices in total.

$\mathsf{EvalCTCoeffs}(\mathsf{mRead}_{\mathbf{M}}, \mathsf{DS}, \mathbf{A}, x)$**:** Look up $\mathbf{H}_{h,x} \in \mathsf{DS}$ and output it.

In the base case, correctness follows from correctness of EvalPK and EvalCTCoeffs for $\mathsf{select}_{\mathbf{M}_0,\mathbf{M}_1}$. In more detail, for any $x \in \{0,1\}$ and $\mathbf{M} = (\mathbf{M}_0, \mathbf{M}_1) \in (\mathbb{Z}_q^{n\times m})^2$, we have that

$$\begin{aligned}(\mathbf{A} - x\mathbf{G})\mathbf{H}_{h,x} &= (\mathbf{A} - x\mathbf{G})(\mathbf{G}^{-1}(\mathbf{M}_1) - \mathbf{G}^{-1}(\mathbf{M}_0)) \\ &= \mathbf{A}(\mathbf{G}^{-1}(\mathbf{M}_1) - \mathbf{G}^{-1}(\mathbf{M}_{-})) - x\mathbf{M}^{\mathsf{R}} + x\mathbf{M}_0 \\ &= \mathbf{A}_h - (1-x)\mathbf{M}^{\mathsf{L}} - x\mathbf{M}^{\mathsf{R}}.\end{aligned}$$

Additionally noting that $\mathbf{H}_{h,x}$ has entries in $\{-1,0,1\}$, we get that $\left\|\mathbf{H}_{h,x}^{\top}\right\| \le m$.

In the inductive case, for any $x \in \{0,1\}^{\ell}$, $\mathbf{M} \in (\mathbb{Z}_q^{n\times m})^{2^{\ell}}$, we inductively assume $\mathsf{EvalPK}(h^{\mathsf{L}}, \mathbf{A}^{-})$ and $\mathsf{EvalPK}(h^{\mathsf{R}}, \mathbf{A}^{-})$ are correct. Choosing $\mathbf{H}_{h^{\mathsf{L}},x^{-}} \in \mathsf{DS}_{h^{\mathsf{L}}}$, $\mathbf{H}_{h^{\mathsf{L}},x^{-}} \in \mathsf{DS}_{h^{\mathsf{L}}}$ yields

$$\begin{aligned}(\mathbf{A} - x\otimes\mathbf{G})\mathbf{H}_{h,x} &= [\mathbf{A}^{-} - x^{-}\otimes\mathbf{G} \mid \mathbf{A}_{\ell} - x_{\ell}\mathbf{G}] \begin{pmatrix}(1-x_{\ell})\mathbf{H}_{h^{\mathsf{L}},x^{-}} + x_{\ell}\mathbf{H}_{h^{\mathsf{R}},x^{-}} \\ \mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{h^{\mathsf{L}}})\end{pmatrix} \\ &= \mathbf{A}_h - \mathbf{M}_x.\end{aligned}$$

As before, we inductively assume that $\mathbf{H}_{h^{\mathsf{L}},x^{-}}$, $\mathbf{H}_{h^{\mathsf{L}},x^{-}}$ have entries in $\{-1,0,1\}$, hence so does $\mathbf{H}_{h,x}$, and thus $\left\|\mathbf{H}_{h,x}^{\top}\right\| \le \ell m$.

The proof of efficiency is the same as that in the proof of Theorem 4.5. The run time of EvalPK on a matrix database $\mathbf{M}$ of length 2^{ℓ} satisfies the recurrence $T(\ell) = 2T(\ell-1) + O(2^{\ell})\cdot \mathrm{poly}(n, \log q)$, thus $T(\ell) = O(\ell 2^{\ell}) \cdot \mathrm{poly}(n, \log q)$. □

4.3 Wire Read Gates

Theorem 4.9. *There exists a preprocessing linear system of homomorphic operations* (EvalPK, EvalCTCoeffs) *for the class of database read gates* $\{\mathsf{wRead}_{\ell} : \ell \in \mathbb{N}\}$*. The system has error growth* $\gamma(\mathsf{wRead}_{\ell}) \le \ell m + 1$*, and it has efficiency properties:*

- *The size of the data structure* DS *and run time of* $\mathsf{EvalPK}(\mathsf{wRead}_{\ell}, \mathbf{A})$ *are both bounded by* $O(\ell \cdot 2^{\ell}) \cdot \mathrm{poly}(n, \log q)$.
- *The implied homomorphic operation* EvalCT *that computes the function*

$$\mathsf{EvalCT}(\mathsf{wRead}_{\ell}, \mathsf{DS}, \mathbf{A}, (x,y), \mathbf{b}) = \mathbf{b}^{\top} \cdot \mathbf{H}_{\mathsf{wRead}_{\ell},(x,y)}$$

for $\mathbf{H}_{\mathsf{wRead}_{\ell},(x,y)} := \mathsf{EvalCTCoeffs}(\mathsf{wRead}_{\ell}, \mathsf{DS}, \mathbf{A}, (x,y))$ *runs in time* $\mathrm{poly}(\ell, n, \log q)$.

Proof. We use the system of homomorphic operations for matrix read functions mRead from Theorem 4.8 to construct EvalPK and EvalCTCoeffs as follows:

$\mathsf{EvalPK}(\mathsf{wRead}_{\ell}, \mathbf{A})$**:** Parse $\mathbf{A} = [\mathbf{A}^X \mid \mathbf{A}^Y]$ for $\mathbf{A}^X = [\mathbf{A}_1^X \mid \ldots \mid \mathbf{A}_{\ell}^X] \in \mathbb{Z}_q^{n\times m\ell}$ and $\mathbf{A}^Y = [\mathbf{A}_0^Y \mid \ldots \mid \mathbf{A}_{L-1}^Y] \in \mathbb{Z}_q^{n\times mL}$ where $L = 2^{\ell}$. Then compute and output $(\mathbf{A}_{\mathsf{wRead}_{\ell}} = \mathbf{A}_{\mathsf{mRead}_{\mathbf{A}^Y}}, \mathsf{DS}) = \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{A}^Y}, \mathbf{A}^X)$.

$\mathsf{EvalCTCoeffs}(\mathsf{wRead}_\ell, \mathsf{DS}, \mathbf{A}, (x,y))$**:** Parse $\mathbf{A} = [\mathbf{A}^X \mid \mathbf{A}^Y]$ as above. Compute $\mathbf{H}_{\mathsf{mRead}_{\mathbf{A}^Y},x} = \mathsf{EvalCTCoeffs}(\mathsf{mRead}_{\mathbf{A}^Y}, \mathsf{DS}, \mathbf{A}^X, x)$ and let $\mathbf{E}_x = [\mathbf{0} \mid \cdots \mid \mathbf{I} \mid \cdots \mid \mathbf{0}]^\top \in \mathbb{Z}_q^{m \times mL}$ denote the matrix with the identity in the x-th block and zeros elsewhere. Output $\mathbf{H}_{\mathsf{wRead}_\ell,x} = [\mathbf{H}^\top_{\mathsf{mRead}_{\mathbf{A}^Y},x} \mid \mathbf{E}]^\top$.

We also construct the algorithm for the function $\mathsf{EvalCT}(\mathsf{wRead}_\ell, \mathsf{DS}, \mathbf{A}, (x,y), \mathbf{b}) = \mathbf{b}^\top \mathbf{H}_{\mathsf{wRead}_\ell,x}$ as follows:

$\mathsf{EvalCT}(\mathsf{wRead}, \mathsf{DS}, \mathbf{A}, (x,y), \mathbf{b})$**:** Parse $\mathbf{A} = [\mathbf{A}^X \mid \mathbf{A}^Y]$ as above and $\mathbf{b} = [\mathbf{b}^X \mid \mathbf{b}^Y]^\top$ with $\mathbf{b}^X = [\mathbf{b}_1^X \mid \ldots \mid \mathbf{b}_\ell^X] \in \mathbb{Z}_q^{m\ell}$ and $\mathbf{b}^Y = [\mathbf{b}_0^Y \mid \ldots \mid \mathbf{b}_{L-1}^Y] \in \mathbb{Z}_q^{mL}$. Compute $\mathbf{H}_{\mathsf{mRead}_{\mathbf{A}^Y},x} = \mathsf{EvalCTCoeffs}(\mathsf{mRead}_{\mathbf{A}^Y}, \mathsf{DS}, \mathbf{A}^X, x)$ and output $\mathbf{b}_{\mathsf{wRead}_\ell} = (\mathbf{b}^X)^\top \cdot \mathbf{H}_{\mathsf{mRead}_{\mathbf{A}^Y},x} + \mathbf{b}_x^Y$.

Let $x \in \{0,1\}^\ell$ and $y \in \{0,1\}^{L=2^\ell}$. By correctness of the homomorphic operations for mRead, we get

$$\begin{aligned}[\mathbf{A}^X - x \otimes \mathbf{G} \mid \mathbf{A}^Y - y \otimes \mathbf{G}]\mathbf{H}_{\mathsf{wRead}_\ell,x} &= (\mathbf{A}_{\mathsf{wRead}_\ell} - \mathbf{A}_x^Y) + (\mathbf{A}_x^Y - y_x\mathbf{G}) \\ &= \mathbf{A}_{\mathsf{wRead}_\ell} - y_x\mathbf{G}.\end{aligned}$$

Additionally, the matrix $\mathbf{H}_{\mathsf{mRead}_{\mathbf{A}^Y},x}$ has $\left\|\mathbf{H}^\top_{\mathsf{mRead}_{\mathbf{A}^Y},x}\right\| \le \ell m$ and thus $\mathbf{H}_{\mathsf{wRead}_\ell,x} = [\mathbf{H}_{\mathsf{mRead}_{\mathbf{A}^Y},x} \mid \mathbf{E}]^\top$ has $\left\|\mathbf{H}^\top_{\mathsf{wRead}_\ell,x}\right\| \le \ell m + 1$ because each column of $\mathbf{E}$ is a standard basis vector. The stated efficiency properties follow immediately from those in Theorem 4.8. $\square$

4.4 Bootstrapping

In this section we recall the bootstrapping algorithms of [HLL23]. These algorithms take in a vector encoding $\mathbf{b}_x$ of an input x and output a new vector encoding of x with a smaller error vector. In order to do so, the algorithms require a circular GSW ciphertext encrypting the secret vector s as well as an vector encoding of that ciphertext.

Theorem 4.10 ([HLL23]). *Let $\lambda, n, q, m, B \in \mathbb{N}$, be such that q is a power of 2, $m \ge n\lceil \log q \rceil$ and $m = O(n \log q)$, $(2^{3\lambda} mB)^2 < q$ and $\lambda \gg (\log n + \log\log q)^3$. There is a pair of deterministic algorithms* $(\mathsf{BootStrapPK}, \mathsf{BootStrapCT})$*, with syntax:*

- $\mathbf{A}_{\mathsf{out}} := \mathsf{BootStrapPK}(\mathbf{A}, \mathbf{B}, \mathbf{w})$*: On input matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{B} \in \mathbb{Z}_q^{n \times mK}$ along with a random vector $\mathbf{w} \in \mathbb{Z}_q^m$, output a matrix $\mathbf{A}_{\mathsf{out}} \in \mathbb{Z}_q^{n\times m}$.*
- $\mathbf{b}_{\mathsf{out}} := \mathsf{BootStrapCT}(\mathbf{A}, \mathbf{B}, x, \mathbf{S}, \mathbf{b}_x, \mathbf{b}_\mathbf{S}, \mathbf{w})$*: On input matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{B} \in \mathbb{Z}_q^{n\times mK}$ along with an input $x \in \{0,1\}$ and ciphertext matrix $\mathbf{S} \in \mathbb{Z}_q^{n\times mk}$ and their vector encodings $\mathbf{b}_x \in \mathbb{Z}_q^m$, $\mathbf{b}_\mathbf{S} \in \mathbb{Z}_q^{mK}$ and also a random vector $\mathbf{w} \in \mathbb{Z}_q^m$, output a new vector encoding of x, $\mathbf{b}_{\mathsf{out}} \in \mathbb{Z}_q^m$.*

Where $k = n \log q$ is the number bits in the description of a vector in $\mathbb{Z}_q^n$ and $K = m(n \log q)^2$ is the number of bits in the description of $\mathbf{S} \in \mathbb{Z}_q^{n\times mk}$. The algorithms BootStrapPK *and* BootStrapCT *satisfy the following correctness property: Let $x \in \{0,1\}$, $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{B} \in \mathbb{Z}_q^{n\times mK}$. Suppose there exists a* *vector* $\mathbf{s} \in \mathbb{Z}_q^n$ *such that* $\mathbf{S}, \mathbf{b}_x, \mathbf{b}_{\mathbf{S}}$ *satisfy*

$$\mathbf{b}_x^\top = \mathbf{s}^\top(\mathbf{A} - x\mathbf{G}) + \mathbf{e}_x^\top$$
$$\mathbf{s}^\top \mathbf{S} = \mathbf{s}^\top(\mathsf{bits}(\mathbf{s}) \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{ct}}^\top \tag{6}$$
$$\mathbf{b}_{\mathbf{S}}^\top = \mathbf{s}^\top(\mathbf{B} - \mathsf{bits}(\mathbf{S}) \otimes \mathbf{G}) + \mathbf{e}_{\mathbf{S}}^\top, \tag{7}$$

for error vectors $\mathbf{e}_x, \mathbf{e}_{\mathsf{ct}}, \mathbf{e}_{\mathbf{S}}$ *with* $\|\mathbf{e}_x\|_\infty \le 2^{2\lambda}B$ *and* $\|\mathbf{e}_{\mathsf{ct}}\|_\infty, \|\mathbf{e}_{\mathbf{S}}\|_\infty \le B$*. Then*

$$\Pr_{\mathbf{w} \leftarrow \mathbb{Z}_q^m}\left[\|\mathbf{e}_{\mathsf{out}}\|_\infty \le 2^\lambda B \;\middle|\; \begin{array}{l} \mathbf{A}_{\mathsf{out}} := \mathsf{BootStrapPK}(\mathbf{A}, \mathbf{B}, \mathbf{w}) \\ \mathbf{b}_{\mathsf{out}} := \mathsf{BootStrapCT}(\mathbf{A}, \mathbf{B}, x, \mathbf{S}, \mathbf{b}_x, \mathbf{b}_{\mathbf{S}}, \mathbf{w}) \\ \mathbf{e}_{\mathsf{out}}^\top := \mathbf{b}_{\mathsf{out}}^\top - \mathbf{s}^\top(\mathbf{A}_{\mathsf{out}} - x\mathbf{G}) \end{array} \right] \ge 1 - \mathrm{negl}(\lambda).$$

The algorithms $\mathsf{BootStrapPK}$ *and* $\mathsf{BootStrapCT}$ *run in time* $\mathrm{poly}(n, \log q)$.

This theorem is adapted from Theorems 11 and 12 in [HLL23] with the "stronger correctness" variants (see Section 3.4 in [HLL23]). The core idea of the construction is to first round the encoded vector $\mathbf{b}_x$ according to some rounding resolution M and then use a circular FHE ciphertext of the secret s to compute an additive term that is used to return the rounded vector to the proper form. We sketch the argument below

Proof sketch. Let $M = \Theta(\sqrt{q})$ be a power of two dividing q such that $M^2 < q$. Such an M satisfies $2^{2\lambda+1}mB/M = O(2^{-\lambda}) = \mathrm{negl}(\lambda)$. Let $\mathbf{G}_{\mathsf{L}}, \mathbf{G}_{\mathsf{R}}$ denote a rearrangement of the columns of $\mathbf{G}$ such that $\mathbf{G}_{\mathsf{R}}$ contains all of the columns of $\mathbf{G}$ that are divisible by M and $\mathbf{G}_{\mathsf{L}}$ contains the rest. Then let $\mathbf{Q}$ be the permutation matrix such that $\mathbf{G} = (\mathbf{G}_{\mathsf{L}}, \mathbf{G}_{\mathsf{R}})\mathbf{Q}$. Now define the matrices

$$\mathbf{\Gamma} = \mathbf{G}^{-1}(M\mathbf{G}_{\mathsf{L}}, \mathbf{G}_{\mathsf{R}}) \qquad \mathbf{U} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & M\mathbf{I} \end{pmatrix} \mathbf{Q}.$$

The rounding procedure $\mathsf{RemoveNoise}(\mathbf{u}, \mathbf{w})$ is as follows. It takes as input two vectors $\mathbf{u}, \mathbf{w} \in \mathbb{Z}_q^m$ and outputs

$$\mathsf{RemoveNoise}(\mathbf{u}, \mathbf{w}) = M \left\lceil \frac{\mathbf{u}^\top \mathbf{\Gamma} + \mathbf{w}^\top}{M} \right\rfloor \mathbf{U}.$$

The matrices $\mathbf{\Gamma}, \mathbf{U}$ satisfy the properties that M divides $\mathbf{G\Gamma} = (M\mathbf{G}_{\mathsf{L}}, \mathbf{G}_{\mathsf{R}})$ and $M\lceil \mathbf{G\Gamma}/M \rfloor \mathbf{U} = \mathbf{G}$. This means that, for $\mathbf{b}_x^\top$ as in the hypothesis and a randomly sampled $\mathbf{w} \leftarrow \mathbb{Z}_q^m$, we have

$$\mathsf{RemoveNoise}(\mathbf{b}, \mathbf{w}) = \mathsf{RemoveNoise}(\mathbf{s}^\top \mathbf{A} + \mathbf{e}_x, \mathbf{w}) - x\mathbf{s}^\top \mathbf{G}.$$

Additionally note that $\|\mathbf{\Gamma}^\top\| = 1$, so as long as each entry of the uniformly random vector $\mathbf{s}^\top \mathbf{A\Gamma} + \mathbf{w}^\top$ is distance at least $2^{2\lambda}B$ away from a rounding boundary we have $\mathsf{RemoveNoise}(\mathbf{s}^\top \mathbf{A} + \mathbf{e}_x, \mathbf{w}) = \mathsf{RemoveNoise}(\mathbf{s}^\top \mathbf{A}, \mathbf{w})$. By a union bound, this occurs with probability at least $1 - 2^{2\lambda+1}Bm/M$. Thus with overwhelming probability

$$\mathsf{RemoveNoise}(\mathbf{b}, \mathbf{w}) = \mathsf{RemoveNoise}(\mathbf{s}^\top \mathbf{A}, \mathbf{w}) - x\mathbf{s}^\top \mathbf{G}.$$

Now define $\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}(\mathbf{s}) = \mathsf{RemoveNoise}(\mathbf{s}^\top \mathbf{A}, \mathbf{w})$. It's easy to see that the circuit computing $\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}$ has depth $d = O(\log n + \log\log q)$. Let $\mathbf{S}$ be as in the hypothesis and let $\mathsf{GSW.Eval}$

be the (vector-valued) GSW FHE evaluation procedure from Section 2.2. By FHE correctness the evaluated ciphertext $\mathbf{C}_{\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}(\mathbf{s})} = \mathsf{GSW.Eval}(\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}, \mathbf{S})$ satisfies

$$\mathbf{s}^\top \mathbf{C}_{\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}(\mathbf{s})} = \mathsf{RndPad}_{\mathbf{A},\mathbf{w}}(\mathbf{s}) + \mathbf{e}_{\mathsf{FHE}}^\top,$$

where $\|\mathbf{e}_{\mathsf{FHE}}\|_\infty \leq O(m^d \cdot \log q)\,\|\mathbf{e}_{\mathsf{ct}}\|_\infty$. Additionally the circuit $C := \mathsf{GSW.Eval}(\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}, \cdot)$ has depth $d' = d \cdot O(\log n + \log\log q)$. Let $(\mathsf{EvalPK}, \mathsf{EvalCT})$ be the matrix-value system of homomorphic operations from Theorem 4.7. Then given a vector encoding $\mathbf{b}_{\mathbf{S}}$ of $\mathbf{S}$ as in the hypothesis, homomorphically executing C using EvalCT yields $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathbf{B}, \mathbf{S}, \mathbf{b}_{\mathbf{S}})$ satisfying

$$\mathbf{b}_C^\top = \mathbf{s}^\top(\mathbf{A}_{\mathsf{out}} - \mathbf{C}_{\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}(\mathbf{s})}) + \mathbf{e}_{\mathsf{EvalCT}},$$

where $\mathbf{A}_{\mathsf{out}} = \mathsf{EvalPK}(C, \mathbf{B})$ and $\|\mathbf{e}_{\mathsf{EvalCT}}\|_\infty \leq O(m^{d'} \cdot \log q)\,\|\mathbf{e}_{\mathbf{S}}\|_\infty$. Combining the above, define $(\mathsf{BootStrapPK}, \mathsf{BootStrapCT})$ as follows:

- $\mathsf{BootStrapPK}(\mathbf{A}, \mathbf{B}, \mathbf{w})$: Compute and output $\mathbf{A}_{\mathsf{out}} = \mathsf{EvalPK}(\mathsf{GSW.Eval}(\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}, \cdot), \mathbf{B})$.
- $\mathsf{BootStrapCT}(\mathbf{A}, \mathbf{B}, x, \mathbf{S}, \mathbf{b}_x, \mathbf{b}_{\mathbf{S}}, \mathbf{w})$: Compute

$$\begin{aligned}\mathbf{b}_1 &:= \mathsf{EvalCT}(\mathsf{GSW.Eval}(\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}, \cdot), \mathbf{B}, \mathbf{S}, \mathbf{b}_{\mathbf{S}})\\ \mathbf{b}_2 &:= \mathsf{RemoveNoise}(\mathbf{b}_x, \mathbf{w})\end{aligned}$$

  Output $\mathbf{b}_{\mathsf{out}} = \mathbf{b}_1 - \mathbf{b}_2$.

It is clear by inspection that BootStrapPK and BootStrapCT satisfy the stated efficiency properties. By correctness of the system of homomorphic operations and GSW FHE evaluation,

$$\begin{aligned}\mathbf{b}_1 &= \mathbf{s}^\top(\mathbf{A}_{\mathsf{out}} - \mathbf{C}_{\mathsf{RndPad}_{\mathbf{A},\mathbf{w}}(\mathbf{s})}) + \mathbf{e}_{\mathsf{EvalCT}}^\top\\ &= \mathbf{s}^\top\mathbf{A}_{\mathsf{out}} - \mathsf{RndPad}_{\mathbf{A},\mathbf{w}}(\mathbf{s}) + \underbrace{\mathbf{e}_{\mathsf{EvalCT}}^\top + \mathbf{e}_{\mathsf{FHE}}^\top}_{:=\mathbf{e}_{\mathsf{out}}^\top},\end{aligned}$$

where the error satisfies

$$\begin{aligned}\|\mathbf{e}_{\mathsf{out}}\|_\infty &\leq \|\mathbf{e}_{\mathsf{EvalCT}}\|_\infty + \|\mathbf{e}_{\mathsf{FHE}}\|_\infty\\ &\leq O(m^{d'} \cdot \log q)B + O(m^d \cdot \log q)B \leq 2^{O(\log n, \log\log q)^3} \cdot B < 2^\lambda \cdot B.\end{aligned}$$

Finally, with high probability over the sampling $\mathbf{w} \leftarrow \mathbb{Z}_q^m$, subtracting $\mathbf{b}_2$ yields

$$\mathbf{b}_{\mathsf{out}}^\top = \mathbf{s}^\top(\mathbf{A}_{\mathsf{out}} - x\mathbf{G}) + \mathbf{e}_{\mathsf{out}}^\top.$$

□

Using their bootstrapping algorithms, [HLL23] define a notion of a "bootstrapped" system of homomorphic operations, where the error in the output encoding vector does not grow as a function of the circuits that is evaluated. We state their definition here, making the minor modification to allow preprocessing.

Definition 4.11 (Bootstrapped Homomorphic Operations, [HLL23]). *A* bootstrapped *preprocessing system of homomorphic operations with error bound B and circuit class $\mathcal{C}$ is a tuple* $(\mathsf{EvalPK}, \mathsf{EvalCT}, \mathcal{K})$ *where $\mathcal{K}$ is a set and* $\mathsf{EvalPK}, \mathsf{EvalCT}$ *are a pair of deterministic algorithms as in Definition 4.4 except with the modified syntax:*

- $(\mathbf{A}_C, \mathsf{DS}) := \mathsf{EvalPK}(C, \mathbf{A}, \mathbf{B}, R)$ *additionally takes a matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times mK}$ and a random value $R \in \mathcal{K}$ from some appropriate set $\mathcal{K}$.*
- $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x, \mathbf{B}, \mathbf{S}, \mathbf{b}_\mathbf{S}, R)$ *additionally takes $\mathbf{B} \in \mathbb{Z}_q^{n\times mK}$ along with a matrix $\mathbf{S} \in \mathbb{Z}_q^{n\times mk}$ and vector encoding $\mathbf{b}_\mathbf{S} \in \mathbb{Z}_q^m K$ as well as the random value $R \in \mathcal{K}$.*

Where $k = n \log q$ is the number bits in the description of a vector in $\mathbb{Z}_q^n$ and $K = m(n\log q)^2$ is the number of bits in the description of $\mathbf{S} \in \mathbb{Z}_q^{n\times mk}$. The algorithms satisfy the following correctness property: Let $C \in \mathcal{C}$ with $C : \{0,1\}^\ell \to \{0,1\}$, $x \in \{0,1\}^\ell$, $\mathbf{A} \in \mathbb{Z}_q^{n\times m\ell}$ and $\mathbf{B} \in \mathbb{Z}_q^{n\times mK}$. Suppose there is a vector $\mathbf{s} \in \mathbb{Z}_q^n$ such that $\mathbf{b}_x^\top = \mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}_x^\top$ and $\mathbf{S}, \mathbf{b}_\mathbf{S}$ satisfy equations (6) and (7) as in Theorem 4.10 where the error vectors have $\|\mathbf{e}_x\|_\infty, \|\mathbf{e}_{\mathsf{ct}}\|_\infty, \|\mathbf{e}_\mathbf{S}\|_\infty \le B$. Then with all but negligible probability over the sampling of $R \leftarrow \mathcal{K}$, $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x, \mathbf{B}, \mathbf{S}, \mathbf{b}_\mathbf{S}, R)$ *satisfies $\mathbf{b}_C^\top = \mathbf{s}^\top(\mathbf{A}_C - C(x)\mathbf{G}) + \mathbf{e}_C^\top$ where* $(\mathbf{A}_C, \mathsf{DS}) := \mathsf{EvalPK}(C, \mathbf{A}, \mathbf{B}, R)$ *and the error vector $\mathbf{e}_C$ satisfies $\|\mathbf{e}_C\|_\infty \le 2^\lambda B$.*

4.5 Putting it all together

Now that we have shown how to define systems of homomorphic operations for data-read and wire-read gates, we can combine our results with the BGG+ system of homomorphic operations for NAND gates to get homomorphic operations for all RAM circuits.

Theorem 4.12. *Let $n, q, m \in \mathbb{N}$ be such that $m \ge n \lceil \log q \rceil$ and $m = O(n \log q)$. There is a preprocessing linear system of homomorphic operations* $(\mathsf{EvalPK}, \mathsf{EvalCTCoeffs})$ *for the class of RAM circuits. The system has error growth $\gamma(C) = (m \log L + 1)^d$ where d is the depth of C and L is the maximum load of C.*[10] *The system has the following efficiency properties:*

- $\mathsf{EvalPK}(C, \mathbf{A})$ *runs in time* $\mathsf{size}_e(C) \cdot \mathrm{poly}(n, \log q, \log L)$.
- *The algorithm* EvalCT *that computes the function* $\mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x) = \mathbf{b}_x^\top \mathsf{EvalCTCoeffs}(C, \mathbf{A}, x)$ *runs in time* $\mathsf{size}_c(C) \cdot \mathrm{poly}(n, \log q, \log L)$.

Proof. The theorem immediately follows from Theorems 4.3, 4.5 and 4.9 together with the composition procedure described earlier in this section. Recall that composition works "gate-by-gate", so in composing the operations we call EvalPK and EvalCTCoeffs/EvalCT once on each gate of the overall circuit C. The final data structure DS is the concatenation of the data structures for each gate. By Theorems 4.5 and 4.9, executing EvalPK for the gates wRead_ℓ and $\mathsf{dRead}_{\mathsf{DB}}$ for $\mathsf{DB} \in \{0,1\}^{2^\ell}$ runs in time $O(\ell 2^\ell) \cdot \mathrm{poly}(n, \log q)$ which is proportional to the size of the canonical expanded circuits for wRead_ℓ and $\mathsf{dRead}_{\mathsf{DB}}$. On the other hand, running EvalCT for those gates only takes time $\mathrm{poly}(n, \log q, \ell) = \mathrm{poly}(n, \log q, \log L)$. ☐

[10]Recall the load of a database-read or wire-read gate is size of the database DB or wire-bundle y being read in that gate. The maximum load of a RAM circuit C is the maximum over the load of all database-read and wire-read gates in C.

To make use of the above, we need to set the modulus $q > \gamma(C)$ for correctness, meaning that efficiency scales polynomially with the depth d. We can get rid of this depth-dependence via bootstrapping. Combining our system of homomorphic operations for RAM circuits with the bootstrapping algorithms of [HLL23], we extend the bootstrapped homomorphic operations to RAM circuits with dRead and wRead gates. It is important to note, however, that this transformation comes at the cost of losing the linearity property as well as requiring a GSW circular encryption of the secret vector $\mathbf{s}$.

Theorem 4.13. *Let $\lambda, n, q, m, B \in \mathbb{N}$ satisfy the conditions of Theorem 4.10 and let $L \in \mathbb{N}$ be such that $\log(m \log L + 1) \leq \lambda$. Assuming the existence of one-way functions, there is a bootstrapped preprocessing system of homomorphic operations $(\mathsf{EvalPK}, \mathsf{EvalCT}, \mathcal{K})$ with error bound B for the class of RAM circuits with maximum load L. The system satisfies the following efficiency properties:*

- *Elements $R \in \mathcal{K}$ have size $|R| = \mathrm{poly}(\lambda)$.*
- $\mathsf{EvalPK}(C, \mathbf{A}, \mathbf{B}, R)$ *runs in time* $\mathsf{size}_e(C) \cdot \mathrm{poly}(\lambda, n, \log q, \log L)$
- $\mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x, \mathbf{B}, \mathbf{S}, \mathbf{b}_{\mathbf{S}}, R)$ *runs in time* $\mathsf{size}_c(C) \cdot \mathrm{poly}(\lambda, n, \log q, \log L)$

Proof. We work gate by gate, where for each gate we compose one step of the homomorphic evaluation using the system from Theorem 4.12 on that gate with an application of the bootstrapping algorithms from Theorem 4.10. The set $\mathcal{K}$ is the set of keys for a PRF family $\mathcal{F} = \{\mathcal{F}_\lambda = \{F : \{0,1\}^* \to \mathbb{Z}_q^m\}\}$. The randomness passed to the bootstrapping algorithms will be a PRF evaluated on the index of the gate within the circuit C.

We argue correctness inductively, layer by layer. Beginning with vector encodings of the input wires with error bounded by $B < 2^\lambda B$, correctness of the system of homomorphic operations from Theorem 4.12 gives that evaluating one gate yields a vector encoding with error at most $2^\lambda B(m \log L + 1) < 2^{2\lambda} B$. Then correctness of the bootstrapping algorithms implies that, if we bootstrap with truly random $\mathbf{w} \leftarrow \mathbb{Z}_q^m$, then with overwhelming probability, resultant bootstrapped vector encoding has error bounded by $2^\lambda B$. Applying PRF security means that the error bound on the encoding vector only negligibly changes when we replace $\mathbf{w} \leftarrow \mathbb{Z}_q^m$ with $F(i)$ where i is the index of the gate being evaluated.

The stated efficiency properties follow from those of Theorems 4.10 and 4.12 together with the fact that this composition procedure only evaluates the homomorphic and bootstrapping algorithms once per gate. □

5 RAM-LFE

In this section we show how to use our system of homomorphic operations for RAM circuits to construct laconic function evaluation schemes for RAM circuits. We give two constructions, one for bounded depth circuits based on the homomorphic operations from Theorem 4.12 and one unbounded circuits where we use the bootstrapped homomorphic operations from Theorem 4.13. Our constructions closely follow those of [QWW18] and [HLL23]. We begin by defining (RAM-) LFE.

Definition 5.1 (RAM-LFE)**.** *A* laconic function evaluation scheme *(LFE) for a class of functions $\mathcal{F} = \mathcal{F}(\lambda)$ is a tuple of algorithms* (Gen, Hash, Enc, Dec) *that have the following syntax:*

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^N)$: *Given the security parameter* 1^λ *and an input length bound* 1^N*, the generation algorithm returns a common random string* crs*. For the sake of convenience we assume that the data of the* crs *contains the input length bound* N*.*
- $(\mathsf{dig}_f, \mathsf{DS}) := \mathsf{Hash}(\mathsf{crs}, f)$: *Given the common random string* crs *and a function* $f \in \mathcal{F}$*, the compression algorithm deterministically outputs a short digest* dig *and a preprocessed data structure* DS*. For convenience, we assume that* DS *contains a description of* f*.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{dig}_f, x)$: *Given the common random string* crs*, a digest* dig_f *and a secret input* $x \in \{0,1\}^N$*, the encoding algorithm returns a ciphertext* ct*.*
- $z := \mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$: *Given the common random string* crs*, the preprocessed database* DS*, and a ciphertext* ct*, the decoding algorithm returns a RAM program output* $z \in \{0,1\}$*.*

We require the algorithms to satisfy the following correctness and security properties.

Correctness: *We require that for all* $\lambda, N \in \mathbb{N}$*, functions* $f \in \mathcal{F}$*, and inputs* $x \in \{0,1\}^N$ *it holds that*

$$\Pr\left[z = f(x) \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^N) \\ (\mathsf{dig}_f, \mathsf{DS}) := \mathsf{Hash}(\mathsf{crs}, f) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{dig}_f, x) \\ z := \mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct}) \end{array} \right] = 1 - \mathrm{negl}(\lambda).$$

We say an LFE has perfect correctness if the above occurs with probability 1.

(Selective) Security: *There exists a PPT algorithm* Sim *such that for any stateful PPT adversary* $\mathcal{A}$*, we have*

$$\left| \Pr\left[\mathsf{Real}_{\mathsf{LFE}}(1^\lambda) = 1 \right] - \Pr\left[\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda) = 1 \right] \right| = \mathrm{negl}(\lambda),$$

where the experiments $\mathsf{Real}_{\mathsf{LFE}}$ *and* $\mathsf{Ideal}_{\mathsf{LFE}}$ *are defined as follows:*

$\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$:

1. $x \leftarrow \mathcal{A}(1^\lambda)$
2. *Sample* $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^{|x|})$
3. $f \leftarrow \mathcal{A}(\mathsf{crs})$ *where* $f \in \mathcal{F}$
4. $(\mathsf{dig}_f, \mathsf{DS}) := \mathsf{Hash}(\mathsf{crs}, f)$;
   $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{dig}_f, x)$
5. *Output* $\mathcal{A}(\mathsf{ct})$

$\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$:

1. $x \leftarrow \mathcal{A}(1^\lambda)$
2. *Sample* $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^{|x|})$
3. $f \leftarrow \mathcal{A}(\mathsf{crs})$ *where* $f \in \mathcal{F}$
4. $\mathsf{ct} \leftarrow \mathsf{Sim}(\mathsf{crs}, f, f(x))$
5. *Output* $\mathcal{A}(\mathsf{ct})$

We refer to the above as (input-)selective *security. We also define a weaker version of* input/function-selective *security where the above experiments are modified so that* $\mathcal{A}$ *has to choose* f *along with* x *in step 1.*

We will consider LFE schemes for classes $\mathcal{F}$ *consisting of RAM circuits or RAM programs, and refer to these as RAM-LFE.*

Remark 5.1. While in this work we will only consider function classes that consist of RAM circuits C or RAM programs f_{DB} as described in Section 3, we note that this definition is a generalization of the definition of LFE in [QWW18]. While we generalize the syntax to allow the Hash algorithm to output a preprocessed data structure, this is only for the sake of efficiency and can be removed without loss of generality at the cost of increasing the decryption run-time. Indeed, because we require Hash to be deterministic, a RAM-LFE for a function class $\mathcal{F}$ implies a (standard) LFE for $\mathcal{F}$ wherein the decoding algorithm Dec simply recomputes the data structure itself before proceeding as in the RAM-LFE scheme.

5.1 RAM AB-LFE

Following the construction of [QWW18], we first define and construct *attribute-based* RAM-LFE where the ciphertext contains a public attribute x and a hidden message μ. Decryption recovers the message if and only if $C(x) = 0$. More formally, we define RAM-AB-LFE as follows.

Definition 5.2. *For any function $f : \{0,1\}^N \to \{0,1\}$, the* conditional disclosure functionality CDF *for f is defined as*

$$\mathsf{CDF}[f](x,\mu) = \begin{cases} (x,\mu) & f(x) = 0 \\ (x,\perp) & \textit{otherwise.} \end{cases}$$

An attribute based LFE *(AB-LFE) for a function class $\mathcal{F}$ is an LFE for the function class $\mathsf{CDF}[\mathcal{F}] = \{\mathsf{CDF}[f] : f \in \mathcal{F}\}$. We refer to such a scheme as a RAM-AB-LFE if $\mathcal{F}$ consists of RAM circuits or RAM programs.*

Theorem 5.3. *Assume LWE holds. For any $\lambda, d \in \mathbb{N}$, there exists an AB-LFE scheme* (Gen, Hash, Enc, Dec) *with perfect correctness and input-selective security for the class $\mathcal{C}_{\lambda,d}$ of RAM circuits C with binary output, depth at most d and maximum load $L < 2^\lambda$. The scheme satisfies the following efficiency properties:*

- *The running time of* $\mathsf{Gen}(1^\lambda, 1^N)$ *and the size of the* crs *are bounded by* $N \cdot \mathrm{poly}(\lambda, d)$.
- *The running time of* $\mathsf{Hash}(\mathsf{crs}, \mathsf{CDF}[C])$ *and the size of* DS *are bounded by* $\mathsf{size}_e(C) \cdot \mathrm{poly}(\lambda, d)$.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{dig}, (x,\mu))$ *runs in time* $N \cdot \mathrm{poly}(\lambda, d)$.
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$ *runs in time* $\mathsf{size}_c(C) \cdot \mathrm{poly}(\lambda, d)$.

Proof. Let $n = \mathrm{poly}(\lambda, d)$, $q = 2^{\mathrm{poly}(\lambda,d)}$, $m = n \lceil \log q \rceil$ and let $B, B' \in \mathbb{Z}$ be such that $\mathsf{LWE}_{n,q,B}$ holds and $(\lambda m + 1)^{d+1} \cdot 2^\lambda B < B' < q/8$. Let (EvalPK, EvalCT) be the system of homomorphic operations for RAM circuits from Theorem 5.6. We construct our RAM-AB-LFE as follows:

$\mathsf{Gen}(1^\lambda, 1^N)$**:** Sample and output $\mathsf{crs} = \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times mN}$.

$\mathsf{Hash}(\mathsf{crs}, \mathsf{CDF}[C])$**:** Parse $\mathsf{crs} = \mathbf{A}$. Then compute and output $(\mathsf{dig}_C, \mathsf{DS}) = (\mathbf{A}_C, \mathsf{DS}) := \mathsf{EvalPK}(C, \mathbf{A})$.

$\mathsf{Enc}(\mathsf{crs}, \mathsf{dig}_C, (x,\mu))$**:** Parse $\mathsf{crs} = \mathbf{A}$ and $\mathsf{dig}_C = \mathbf{A}_C$. Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_x \leftarrow \chi_B^{Nm}$ and $\widetilde{e} \leftarrow [-B', B']$. Sample $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ and put $\mathbf{t} = \mathbf{G}^{-1}(\mathbf{u})$. Set:

$$\mathbf{b}_x^\top := \mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}_x^\top$$
$$\beta = \mathbf{s}^\top \mathbf{A}_C \mathbf{t} + \widetilde{e} + \mu \cdot \lceil q/2 \rfloor .$$

Output $\mathsf{ct} = (\mathbf{b}_x, \beta, \mathbf{t}, x)$.

$\mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$**:** Parse $\mathsf{crs} = \mathbf{A}$ and $\mathsf{ct} = (\mathbf{b}_x, \beta, \mathbf{t}, x)$. If $C(x) = 1$ output $(x, \perp)$. Otherwise, compute $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}', \mathbf{A}, x, \mathbf{b}_x)$, $\mu := \mathsf{round}_q(\beta - \mathbf{b}_C^\top \mathbf{t})$. Output (x, μ).

Correctness. Let C have maximum load L and let $\ell = \log L < \lambda$. Assume $C(x) = 0$. By correctness of $(\mathsf{EvalPK}, \mathsf{EvalCT})$, $\mathbf{b}_C^\top = \mathbf{s}^\top \mathbf{A}_C + \mathbf{e}_C^\top$, where $\|\mathbf{e}_C\|_\infty \le B(\ell m + 1)^d$. Therefore

$$\begin{aligned}\beta - \mathbf{b}_C^\top \mathbf{t} &= (\mathbf{s}^\top \mathbf{A}_C \mathbf{t} + \widetilde{e} + \mu \cdot \lceil q/2 \rfloor) - (\mathbf{s}^\top \mathbf{A}_C \mathbf{t} + \mathbf{e}_C^\top \mathbf{t}) \\ &= \mu \cdot \lceil q/2 \rfloor + (\widetilde{e} - \mathbf{e}_C^\top \mathbf{t}).\end{aligned}$$

And $|\widetilde{e} - \mathbf{e}_C^\top \mathbf{t}| \le B' + B(\ell m + 1)^{d+1} < 2B' < q/4$.

Security. It suffices to consider the case where $C(x) = 1$ because otherwise (x, μ) is revealed in the output and the simulator can just generate an honest ciphertext for (x, μ). Define a simulator as follows:

$\mathsf{Sim}(\mathsf{crs}, \mathsf{CDF}[C], (x, \perp))$**:** Sample $\mathbf{b}_x \leftarrow \mathbb{Z}_q^{Nm}$, $\beta \leftarrow \mathbb{Z}_q$, $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ and set $\mathbf{t} = \mathbf{G}^{-1}(\mathbf{u})$. Output $\mathsf{ct} = (\mathbf{b}_x, \beta, \mathbf{t}, x)$.

We now show indistinguishability of the $\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$ and $\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$ experiments by defining the following hybrid experiment:

Hybrid: This is the same as $\mathsf{Real}_{\mathsf{LFE}}$ except we change how β is computed during the encryption. After computing $\mathbf{b}_x^\top = \mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}_x$ and $\mathbf{t} \in \mathbb{Z}_q^m$, compute $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x)$. Then set $\alpha = \mathbf{s}^\top \mathbf{G} \mathbf{t} + e_0$ for $e_0 \leftarrow \chi_B$ and put

$$\beta = \mathbf{b}_C^\top \mathbf{t} + \alpha + \mu \cdot \lceil q/2 \rfloor .$$

In the hybrid experiment, correctness of $(\mathsf{EvalPK}, \mathsf{EvalCT})$ together with the fact that $P_C(x) = 1$ gives

$$\mathbf{b}_C^\top \mathbf{t} + \alpha = \mathbf{s}^\top (\mathbf{A}_C - \mathbf{G}) \mathbf{t} + \mathbf{e}_C^\top \mathbf{t} + (\mathbf{s}^\top \mathbf{G} \mathbf{t} + e_0) = \mathbf{s}^\top \mathbf{A}_C \mathbf{t} + \mathbf{e}_C^\top \mathbf{t} + e_0.$$

Thus we have

$$\beta = \mathbf{s}^\top \mathbf{A}_C \mathbf{t} + (\widetilde{e} + \mathbf{e}_C^\top \mathbf{t} + e_0) + \mu \cdot \lceil q/2 \rfloor ,$$

which is statistically indistinguishable from the value of β in $\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$ by noise smudging (Lemma 2.2).

Next we show that the hybrid experiment is computationally indistinguishable from $\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$ under the LWE assumption. Consider the reduction that when given LWE challenges $(\mathbf{u}, \alpha)$ and $(\mathbf{M}, \mathbf{b}_x)$ for $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ and $\mathbf{M} \leftarrow \mathbb{Z}_q^{n \times Nm}$, it gets x from the adversary and sets $\mathsf{crs} = \mathbf{A} := \mathbf{M} - x \otimes \mathbf{G}$, $\mathbf{t} = \mathbf{G}^{-1}(\mathbf{u})$ and β is computed as in the hybrid. In the case that the $(\alpha, \mathbf{b}_x)$ are LWE samples, the view of the adversary is exactly as in the hybrid experiment. On the other hand if $(\alpha, \mathbf{b}_x)$ are uniform then the view of the adversary is as in $\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$.

Efficiency. The efficiency properties of Hash and Dec follow respectively from the efficiency properties of EvalPK and EvalCT in Theorem 4.12. The properties of Gen and Enc follow by inspection. □

Similarly we can modify the above construction to use the bootstrapped system of homomorphic operations from Theorem 4.13 yielding a RAM-AB-LFE for RAM circuits of unbounded depth.

Theorem 5.4. *Assume LWE and circular security of the GSW encryption scheme (see Section 2.2). For all* $\lambda \in \mathbb{N}$*, there exists a RAM-AB-LFE scheme* $(\mathsf{Gen}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{Dec})$ *with input/function-selective security for the class* $\mathcal{C}_\lambda$ *of RAM circuits* C *with binary output and maximum load* $L < 2^\lambda$*. The scheme satisfies the following efficiency properties:*

- *The running time of* $\mathsf{Gen}(1^\lambda, 1^N)$ *and the size of the* crs *are bounded by* $N \cdot \mathrm{poly}(\lambda)$*.*
- *The running time of* $\mathsf{Hash}(\mathsf{crs}, \mathsf{CDF}[C])$ *and the size of* DS *are bounded by* $\widetilde{O}(\mathsf{size}_e(C)) \cdot \mathrm{poly}(\lambda)$*.*
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{dig}, (x, \mu))$ *runs in time* $N \cdot \mathrm{poly}(\lambda)$*.*
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$ *runs in time* $\mathsf{size}_c(C) \cdot \mathrm{poly}(\lambda, \log L)$*.*

Proof. The construction and proof are very similar to as in Theorem 5.3. Let $n = \mathrm{poly}(\lambda)$, $q = 2^{\mathrm{poly}(\lambda)}$, $m = n \log q$ and let $B, B' \in \mathbb{N}$ be as chosen to satisfy the conditions of Theorem 4.13, such that $\mathsf{LWE}_{n,q,B}$ holds and such that $2^{2\lambda} B m < B' < q/8$. Let $(\mathsf{EvalPK}, \mathsf{EvalCT}, \mathcal{K})$ be the bootstrapped system of homomorphic for RAM circuits from Theorem 4.13 and let GSW be the GSW FHE scheme. We modify the construction from Theorem 5.3 as follows:

$\mathsf{Gen}(1^\lambda, 1^N)$**:** In addition to $\mathbf{A}$, sample $\mathbf{B} \leftarrow \mathbb{Z}_q^{n \times mK}$ and $R \leftarrow \mathcal{K}$ and output $\mathsf{crs} = (\mathbf{A}, \mathbf{B}, R)$.

$\mathsf{Hash}(\mathsf{crs}, \mathsf{CDF}[C])$**:** This is the same as in the previous construction except $(\mathbf{A}_C, \mathsf{DS}) := \mathsf{EvalPK}(C, \mathbf{A}, \mathbf{B}, R)$ is computed according to the bootstrapped system of homomorphic operations.

$\mathsf{Enc}(\mathsf{crs}, \mathsf{dig}_C, (x, \mu))$**:** Instead of sampling the secret $\mathbf{s}$ uniformly, sample a GSW secret key $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GSW.Gen}(1^\lambda, \chi_B)$ and set $\mathbf{s} = \mathsf{sk}$. Then sample $\mathbf{e}_\mathbf{S} \leftarrow \chi_B^{mK}$ and set:

$$\begin{aligned} \mathbf{S} &\leftarrow \mathsf{GSW.Enc}(\mathsf{pk}, \mathsf{bits}(\mathbf{s})) \\ \mathbf{b}_\mathbf{S}^\top &:= \mathbf{s}^\top(\mathbf{B} - \mathsf{bits}(\mathbf{S}) \otimes \mathbf{G}) + \mathbf{e}_\mathbf{S}. \end{aligned}$$

Then output $\mathsf{ct} = (\mathbf{b}_x, \beta, \mathbf{t}, x, \mathbf{S}, \mathbf{b}_\mathbf{S})$, where $\mathbf{b}_x, \beta, \mathbf{t}$ are all computed as in the previous construction.

$\mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$**:** This is the same as in the previous construction except the output encoding $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x, \mathbf{B}, \mathbf{S}, \mathbf{b}_\mathbf{S}, R)$ is computed according to the bootstrapped system of homomorphic operations.

Correctness. The correctness argument is almost identical to that in Theorem 5.3. Assume $C(x) = 0$, then, by correctness of the bootstrapped system of homomorphic operations, with overwhelming probability over the choice of $R \leftarrow \mathcal{K}$ in the crs, $\mathbf{b}_C^\top = \mathbf{s}^\top \mathbf{A}_C + \mathbf{e}_C^\top$ where $\|\mathbf{e}_C\|_\infty \leq B2^\lambda$. Therefore

$$\beta - \mathbf{b}_C^\top \mathbf{t} = \mu \cdot \lceil q/2 \rfloor + (\widetilde{e} - \mathbf{e}_C^\top \mathbf{t}).$$

And $|\widetilde{e} - \mathbf{e}_C^\top \mathbf{t}| \leq B' + Bm2^\lambda \leq 2B' < q/4$.

Security. As before, it suffices to consider the case $C(x) = 1$. Define the simulator as follows:

$\mathsf{Sim}(\mathsf{crs}, \mathsf{CDF}[C], (x, \bot))$**:** As in the previous simulator, uniformly and independently sample $\mathbf{b}_x \leftarrow \mathbb{Z}_q^{Nm}, \beta \leftarrow \mathbb{Z}_q, \mathbf{u} \leftarrow \mathbb{Z}_q^n$, setting $\mathbf{t} = \mathbf{G}^{-1}(\mathbf{u})$. Similarly pick $\mathbf{S} \leftarrow \mathbb{Z}_q^{n \times mk}$ and $\mathbf{b_S} \leftarrow \mathbb{Z}_q^{n \times mK}$. Output $\mathsf{ct} = (\mathbf{b}_x, \beta, \mathbf{t}, x, \mathbf{S}, \mathbf{b_S})$.

We show that, instantiated with this simulator, the input/function-selective experiments $\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$ and $\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$ are computationally indistinguishable. We define hybrid experiments as follows:

Hybrid 1: This is analogous to the hybrid experiment in the proof of Theorem 5.3. The experiment is the same as $\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$ except we change how β is computed. After computing $\mathbf{b}_x, \mathbf{S}, \mathbf{b_S}$, compute the encoding of the output $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x, \mathbf{B}, \mathbf{S}, \mathbf{b_S}, R)$. Then set $\alpha = \mathbf{s}^\top \mathbf{G} \mathbf{t} + e_0$ for $e_0 \leftarrow \chi_B$ and put

$$\beta = \mathbf{b}_C^\top \mathbf{t} + \alpha + \mu \cdot \lceil q/2 \rfloor .$$

The experiments $\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$ and hybrid 1 are statistically indistinguishable by noise smudging using the same argument as in the proof of Theorem 5.3. Indistinguishability of hybrid 1 and $\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$ follows from circular security of the GSW encryption scheme. We define the reduction that takes in a matrix $\mathbf{S}$ and LWE challenges $(\mathbf{u}, \alpha)$, $(\mathbf{M}_x, \mathbf{b}_x)$ and $(\mathbf{M_S}, \mathbf{b_S})$ for $\mathbf{u} \leftarrow \mathbb{Z}_q^m, \mathbf{M}_x \leftarrow \mathbb{Z}_q^{n \times Nm}$ and $\mathbf{M_S} \leftarrow \mathbb{Z}_q^{n \times mK}$, then it gets C, x from the adversary and sets $\mathsf{crs} = (\mathbf{A} = \mathbf{M}_x - x \otimes \mathbf{G}, \mathbf{B} = \mathbf{M_S} - \mathsf{bits}(\mathbf{S}) \otimes \mathbf{G}, R)$ for $R \leftarrow \mathcal{K}$. If the reduction is given a GSW circular ciphertext $\mathbf{S}$ encrypting $\mathbf{s}$ and LWE samples with respect to $\mathbf{s}$, then it matches the view of the adversary in hybrid 2, and if the reduction is given uniform $\mathbf{S}$ and uniform samples, then it matches the ideal experiment. Note that it is necessary for the reduction to receive C, x before sampling $R \leftarrow \mathcal{K}$ because correctness of the bootstrapped $(\mathsf{EvalPK}, \mathsf{EvalCT})$ requires that C, x are chosen independently from R.

Efficiency. As in Theorem 5.3, efficiency follows directly from the efficiency properties of the homomorphic operations as stated in Theorem 4.13 with the key difference here being that here the noise growth and hence also the parameters n, q, m do not depend on the depth of the RAM circuits. □

We can combine the above construction with Lemma 3.1, to get RAM-ABE-LFE for the function class consisting of RAM *programs*. To do this we modify the definition of Hash to first run the compiler that, given a fixed input size N and time bound T, converts a RAM program f_{DB} into a RAM circuit C before running the hash function of the above construction. This modification yields the following corollary.

Corollary 5.5. *Assume LWE and circular security of the GSW encryption scheme (see Section 2.2). For any $\lambda \in N$, there exists a RAM-AB-LFE scheme* (Gen, Hash, Enc, Dec) *with input/function-selective security for the function class $\mathcal{F}_\lambda = \{(f_{\mathsf{DB}}, T)\}$ consisting of all RAM programs f_{DB} with binary output, run time at most T, and address size $w \leq \lambda$. The scheme satisfies the following efficiency properties:*

- *The running time of* $\mathsf{Gen}(1^\lambda, 1^N)$ *and the size of the* crs *are bounded by* $N \cdot \mathrm{poly}(\lambda)$.
- *The running time of* $\mathsf{Hash}(\mathsf{crs}, \mathsf{CDF}[f_{\mathsf{DB}}])$ *and the size of* DS *are bounded by* $TM \cdot \mathrm{poly}(\lambda, \log T)$, *where* $M = \max\{T, |\mathsf{DB}| + |f|, N\}$.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{dig}, (x, \mu))$ *runs in time* $N \cdot \mathrm{poly}(\lambda)$.
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$ *runs in time* $T \cdot \mathrm{poly}(\lambda, \log T)$.

Remark 5.2 (RAM circuits of low depth)**.** Naively applying the above transformation to Theorem 5.3 would yield a RAM-AB-LFE where the encryption time scales polynomially in the run-time T of the RAM program being evaluated. This is because, in Lemma 3.1 a RAM program with run time T is transformed into a RAM circuit with depth at least T. However, we note that there are indeed many RAM programs that can be represented as a low-depth RAM circuit. In particular, highly parallel RAM computations need not have large depth.

Remark 5.3 (Multi-bit output)**.** While the above constructions build RAM-AB-LFE for function classes with single-bit output, it is possible to generically build RAM-AB-LFE for larger output functions by merely running many copies in parallel. This transformation incurs a multiplicative cost in the output length to the ciphertext size and decryption run time. However it is possible to reduce this cost to an additive overhead in the output length using the techniques of [QWW18]. We omit the details here.

5.2 Upgrading to full RAM-LFE

Theorem 5.6. *Assuming the existence of RAM-FHE and a RAM-AB-LFE with input/function-selective security for the function class $\mathcal{F}_\lambda = \{(f_{\mathsf{DB}}, T)\}$ consisting of all RAM programs f_{DB} with binary output, run time at most T, and address size $w \leq \lambda$, there exists a RAM-LFE scheme with input/function-selective security for $\mathcal{F}_\lambda$. In particular, assuming the RingLWE assumption holds as well as the circularity assumptions in Section 2.2 and Theorem 2.4, there is such a scheme with the following efficiency properties:*

- *The running time of* $\mathsf{Gen}(1^\lambda, 1^N)$ *and the size of the* crs *are bounded by* $N \cdot \mathrm{poly}(\lambda)$.
- *The running time of* $\mathsf{Hash}(\mathsf{crs}, f_{\mathsf{DB}})$ *and the size of* DS *are bounded by* $TM^{1+o(1)} \cdot \mathrm{poly}(\lambda)$, *where* $M = \max\{T, |\mathsf{DB}| + |f|, N\}$.
- $\mathsf{Enc}(\mathsf{crs}, \mathsf{dig}, (x, \mu))$ *runs in time* $NM^{o(1)} \cdot \mathrm{poly}(\lambda)$.
- $\mathsf{Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$ *runs in time* $TM^{o(1)} \cdot \mathrm{poly}(\lambda)$.

Notation for Two-Outcome RAM AB-LFE. Before we prove the theorem, we first define the following convenient notation. Given a RAM-AB-LFE for the class of all RAM programs with multi-bit output, we consider the task of encrypting a pair of messages (μ^0, μ^1) under attribute x such that the decoding algorithm for a RAM program f_{DB} will recover μ^b when $f_{\mathsf{DB}}(x) = b$. This can be accomplished using RAM-AB-LFE by using the Hash algorithm to produce a digest for the RAM program that outputs the concatenation of $f_{\mathsf{DB}}(x)$ with its bitwise complement.

For a RAM AB-LFE scheme (AB-LFE.Gen, AB-LFE.Hash, AB-LFE.Enc, AB-LFE.Dec), we define the following "two-outcome" RAM-AB-LFE syntax (AB-LFE.Gen′, AB-LFE.Hash′, AB-LFE.Enc′, AB-LFE.Dec′):

- $\mathsf{AB\text{-}LFE.Gen}'(1^\lambda, 1^N)$: Output $\mathsf{crs} \leftarrow \mathsf{AB\text{-}LFE.Gen}(1^\lambda, 1^N)$.
- $\mathsf{AB\text{-}LFE.Hash}'(\mathsf{crs}, \mathsf{CDF}[f_{\mathsf{DB}}])$: Given f_{DB} with ℓ-bit output, define $\tilde{f}$ such that $\tilde{f}_{\mathsf{DB}}(x) = (f_{\mathsf{DB}}(x)||\bar{f}_{\mathsf{DB}}(x))$, where $\bar{f}_{\mathsf{DB}}(x)$ is the bitwise complement of $f_{\mathsf{DB}}(x)$. Output $(\mathsf{dig}, \mathsf{DS}) := \mathsf{AB\text{-}LFE.Hash}(\mathsf{crs}, \mathsf{CDF}[\tilde{f}_{\mathsf{DB}}])$.
- $\mathsf{AB\text{-}LFE.Enc}'(\mathsf{crs}, \mathsf{dig}, (x, \{\mu_j^0, \mu_j^1\}_{j=1}^\ell))$: Output $\mathsf{ct} \leftarrow \mathsf{AB\text{-}LFE.Enc}(\mathsf{crs}, \mathsf{dig}, (x, \{\mu_j^0\}_{j=1}^\ell, \{\mu_j^1\}_{j=1}^\ell))$.
- $\mathsf{AB\text{-}LFE.Dec}'(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$: Run $(\{\tilde{\mu}_j^0, \tilde{\mu}_j^1\}_{j=1}^\ell) := \mathsf{AB\text{-}LFE.Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$. Output $\{\mu_j\}_{j=1}^\ell$ such that, for each j, μ_j is the one of $\tilde{\mu}_j^0, \tilde{\mu}_j^1$ which is not $\perp$.

Proof of Theorem 5.6. The proof is similar to the ones in [GKP+13a] and [QWW18]. We use the following building blocks for the construction of RAM LFE.

- A RAM-FHE scheme $\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Gen}, \mathsf{FHE.Prep}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$.
- A RAM-AB-LFE scheme $\mathsf{AB\text{-}LFE} = (\mathsf{AB\text{-}LFE.Gen}, \mathsf{AB\text{-}LFE.Hash}, \mathsf{AB\text{-}LFE.Enc}, \mathsf{AB\text{-}LFE.Dec})$ for the class of all RAM programs, using the two-outcome syntax as above.
- A circuit garbling scheme $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$.

We construct our RAM LFE scheme $\mathsf{LFE} = (\mathsf{LFE.Gen}, \mathsf{LFE.Hash}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ as follows:

$\mathsf{LFE.Gen}(1^\lambda, 1^N)$**:** Sample and output $\mathsf{crs} \leftarrow \mathsf{AB\text{-}LFE.Gen}(1^\lambda, 1^N)$.

$\mathsf{LFE.Hash}(\mathsf{crs}, (f_{\mathsf{DB}}, T))$**:** Let $M = \max\{N, |\mathsf{DB}| + |f|, T\}$. Compute $\mathsf{params} := \mathsf{FHE.Setup}(1^\lambda, M)$ and $\mathsf{DS}_f := \mathsf{FHE.Prep}(\mathsf{params}, f_{\mathsf{DB}})$. Then let $\tilde{f}_{\mathsf{DS}_f}$ be the RAM program computing the function $\tilde{f}_{\mathsf{DS}_f}(\mathsf{pk}, \mathsf{ct}_x) = \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{DS}_f, \mathsf{ct}_x)$. Compute $(\mathsf{dig}_{\tilde{f}}, \mathsf{DS}) := \mathsf{AB\text{-}LFE.Hash}(\mathsf{crs}, \tilde{f}_{\mathsf{DS}_f})$. And output $(\mathsf{dig}_f = (\mathsf{dig}_{\tilde{f}}, \mathsf{params}), \mathsf{DS})$.

$\mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{dig}_f, x)$**:** $(\mathsf{dig}_f = (\mathsf{dig}_{\tilde{f}}, \mathsf{params}), \mathsf{DS})$ and then run the following:

1. Generate keys for RAM-FHE $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Gen}(\mathsf{params})$ and compute $\mathsf{ct}_x \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$.
2. Garble the RAM-FHE decryption circuit to get $(\Gamma, \{L_j^0, L_j^1\}_{j=1}^\ell) \leftarrow \mathsf{GC.Garble}(\mathsf{FHE.Dec}(\mathsf{sk}, \cdot))$.
3. Use the RAM-AB-LFE to encrypt the labels with respect to the attribute $(\mathsf{pk}, \mathsf{ct}_x)$: $\mathsf{ct}' \leftarrow \mathsf{AB\text{-}LFE.Enc}(\mathsf{crs}, \mathsf{dig}_{\tilde{f}}, (\mathsf{pk}, \mathsf{ct}_x), \{L_j^0, L_j^1\}_{j=1}^\ell)$.
4. Output $\mathsf{ct} = (\Gamma, \mathsf{ct}')$.

$\mathsf{LFE.Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct})$**:** Parse $\mathsf{ct} = (\Gamma, \mathsf{ct}')$. Run AB-LFE decryption to recover $\{L_j\}_{j=1}^\ell := \mathsf{AB\text{-}LFE.Dec}(\mathsf{crs}, \mathsf{DS}, \mathsf{ct}')$. Then output the result of the garbling evaluation $z := \mathsf{GC.Eval}(\Gamma, \{L_j\}_{j=1}^\ell)$.

Correctness. In the decryption step, following the correctness of RAM AB-LFE, we recover $L_j = L_j^{\mathsf{ct}_j^*}$ for all $j \in [\ell]$, where ct_j^* is the j'th bit in $\mathsf{ct}^* = \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{DS}_f, \mathsf{ct}_x)$. Then by the correctness of the circuit garbling scheme and the RAM-FHE scheme, the result $z = \mathsf{GC.Eval}(\Gamma, \{L_j^{d_j}\}_{j=1}^{\ell}) = \mathsf{FHE.Dec}(\mathsf{sk}, \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{DS}_f, \mathsf{ct}_x)) = f_{\mathsf{DB}}(x)$.

Security. We define the following simulator:

$\mathsf{Sim}(\mathsf{crs}, (f_{\mathsf{DB}}, T), f_{\mathsf{DB}}(x))$:

1. Let $M = \max\{N, |\mathsf{DB}| + |f|, T\}$. Compute $\mathsf{params} := \mathsf{FHE.Setup}(1^\lambda, M)$ and $\mathsf{DS}_f := \mathsf{FHE.Prep}(\mathsf{params}, f_{\mathsf{DB}})$.
2. Sample RAM-FHE keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Keygen}(\mathsf{params})$ and a dummy encryption $\mathsf{ct}_0 \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, \mathbf{0})$.
3. Run the simulator for the garbling scheme: $(\hat{\Gamma}, \{\hat{L}_j\}_{j=1}^{\ell}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, |\mathsf{FHE.Dec}(\mathsf{sk}, \cdot)|, |\mathsf{ct}_0|, f_{\mathsf{DB}}(x))$.
4. Run the simulator for the RAM-AB-LFE scheme:
$$\hat{\mathsf{ct}}' \leftarrow \mathsf{Sim}_{\mathsf{AB\text{-}LFE}}(\mathsf{crs}, \mathsf{CDF}[\tilde{f}_{\mathsf{DS}_f}], (\mathsf{ct}_0, \{\hat{L}_j\}_{j=1}^{\ell})),$$
where $\tilde{f}_{\mathsf{DS}_f}(\mathsf{pk}, \mathsf{ct}) = \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{DS}_f, \mathsf{ct})$.
5. Output $\mathsf{ct} = (\hat{\Gamma}, \hat{\mathsf{ct}}')$.

We show that, instantiated with this simulator, the input/function-selective experiments $\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$ and $\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$ are computationally indistinguishable. We define hybrid experiments as follows:

Hybrid 0: This is just the experiment $\mathsf{Real}_{\mathsf{LFE}}(1^\lambda)$.

Hybrid 1: We use the simulator for RAM-AB-LFE to generate $\hat{\mathsf{ct}}'$. Let $d = \mathsf{FHE.Eval}(\mathsf{pk}, \mathsf{DS}_f, \mathsf{ct}_x)$ and d_j be its j-th bit, compute
$$\hat{\mathsf{ct}}' \leftarrow \mathsf{Sim}_{\mathsf{AB\text{-}LFE}}(\mathsf{crs}, \mathsf{CDF}[\tilde{f}_{\mathsf{DS}_f}], (\mathsf{ct}_x, \{L_j^{d_j}\}_{j=1}^{\ell})).$$

By the input/function-selective security of the RAM-AB-LFE scheme, hybrid 0 is computationally indistinguishable from hybrid 1.

Hybrid 2: We use the simulator for the circuit garbling scheme to generate Γ. That is, we first use $\mathsf{Sim}_{\mathsf{GC}}$ to compute
$$(\hat{\Gamma}, \{\hat{L}_j\}_{j=1}^{\ell}) \leftarrow \mathsf{Sim}_{\mathsf{GC}}(1^\lambda, |\mathsf{FHE.Dec}(\mathsf{sk}, \cdot)|, |\mathsf{ct}_x|, f_{\mathsf{DB}}(x)),$$
and then generate $\hat{\mathsf{ct}}'$ with $\{\hat{L}_j\}_{j=1}^{\ell}$,
$$\hat{\mathsf{ct}}' \leftarrow \mathsf{Sim}_{\mathsf{AB\text{-}LFE}}(\mathsf{crs}, \mathsf{CDF}[\tilde{f}_{\mathsf{DS}_f}], (\mathsf{ct}_x, \{\hat{L}_j\}_{j=1}^{\ell})).$$

By the security of the circuit garbling scheme, hybrid 1 and 2 are computationally indistinguishable.

Hybrid 3: This is the ideal experiment $\mathsf{Ideal}_{\mathsf{LFE}}(1^\lambda)$. That is, we replace ct_x by $\mathsf{ct}_0 = \mathsf{FHE.Enc}(\mathsf{pk}, 0)$, and then generate $\hat{\mathsf{ct}}'$ as

$$\hat{\mathsf{ct}}' \leftarrow \mathsf{Sim}_{\mathsf{AB\text{-}LFE}}(\mathsf{crs}, \mathsf{CDF}[\tilde{f}_{\mathsf{DS}_f}], (\mathsf{ct}_0, \{\hat{L}_j\}_{j=1}^{\ell})).$$

Hybrid 2 and hybrid 3 are computationally indistinguishable by the security of the RAM-FHE scheme.

□

Efficiency. Efficiency follows in a straightforward manner from the efficiency of the underlying primitives. In particular, the run time of LFE.Hash is dominated by the time it takes to produce an AB-LFE digest for the RAM program $\tilde{f}_{\mathsf{DS}_f}$. The efficiency of RAM-FHE implies that $\tilde{f}_{\mathsf{DS}_f}$ has run time $TM^{o(1)}$, thus the RAM-AB-LFE digest for $\tilde{f}_{\mathsf{DS}_f}$ can be computed in time $TM^{1+o(1)}$. The run time of LFE.Enc is dominated by the time it takes to compute a RAM-FHE ciphertext for x which takes time $NM^{o(1)}$. Finally, the run time of LFE.Dec is dominated by running the AB-LFE decryption procedure for $\tilde{f}_{\mathsf{DS}_f}$ which runs in time $TM^{o(1)}$.

Remark 5.4 (RAM-LFE for bounded depth)**.** It is possible to avoid the circular security assumptions of the above construction at the cost of only being able to evaluate RAM programs that can be represented by RAM circuits of bounded depth. The construction of such a scheme is very similar to the one above where instead we use the bounded RAM-AB-LFE from Theorem 5.3 and a leveled RAM-FHE. We omit the details here for the sake of brevity.

6 RAM-ABE

In this section we show how to use our linear system of homomorphic operations for RAM circuits (Theorem 4.12) to construct attribute based encryption for bounded-depth RAM circuits under the LWE assumption. Our construction closely follows the (circuit-)ABE construction from [BGG+14]. We first recall the definition of ABE below.

Definition 6.1 (ABE)**.** *An* attribute-based encryption *(ABE) scheme for a class of circuits $\mathcal{C}$ is a tuple of algorithms* (Setup, KeyGen, Enc, Dec)*, with the following syntax*

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$*: On input the security parameter λ and an attribute size bound N, output a master public key* mpk *and master secret key* msk.
- $\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C)$*: On input the master secret key and a circuit $C \in \mathcal{C}$, output a function specific secret key* sk_C.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mu)$*: On input the master public key* mpk *an attribute $x \in \{0,1\}^N$ and a message $\mu \in \{0,1\}$, output a ciphertext* ct *encrypting μ.*
- $\mu^* := \mathsf{Dec}(\mathsf{sk}_C, x, \mathsf{ct})$*: On input a function key* sk_C*, attribute $x \in \{0,1\}^N$ and ciphertext* ct*, output a bit $\mu^* \in \{0,1\}$.*

The algorithms satisfy the following correctness and security properties.

Correctness: *For any $\lambda, N \in \mathbb{N}$, $C \in \mathcal{C}$ and $x \in \{0,1\}^N$ such that $C(x) = 0$, and any $\mu \in \{0,1\}$,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_C, x, \mathsf{ct}) = \mu \,\middle|\, \begin{array}{l} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ \mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mu) \end{array}\right] = 1,$$

where the probability is over the random coins of Setup, KeyGen *and* Enc.

(Selective) Security: *For any stateful PPT adversary $\mathcal{A}$,*

$$\left|\Pr[\mathsf{Expt}^{\mathcal{A}}_{\mathsf{ABE}}(1^\lambda, 0) = 1 - \Pr[\mathsf{Expt}^{\mathcal{A}}_{\mathsf{ABE}}(1^\lambda, 1) = 1]\right| = \mathrm{negl}(\lambda),$$

where, for $b \in \{0,1\}$, the experiment $\mathsf{Expt}^{\mathcal{A}}_{\mathsf{ABE}}(1^\lambda, b)$ is defined as follows:

1. $x \leftarrow \mathcal{A}(1^\lambda)$ *where* $x \in \{0,1\}^N$.
2. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$.
3. $(\mu_0, \mu_1) \leftarrow \mathcal{A}^{\mathsf{KG}_{\mathsf{msk},x}(\cdot)}(\mathsf{mpk})$.
4. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, \mu)$
5. $b^* \leftarrow \mathcal{A}^{\mathsf{KG}_{\mathsf{msk},x}(\cdot)}(\mathsf{ct})$. *The output of the experiment is b^*.*

Where in the above $\mathsf{KG}_{\mathsf{mpk},x}(C)$ is an oracle that outputs $\mathsf{KeyGen}(\mathsf{msk}, C)$ if $C(x) = 1$ and $\perp$ otherwise.

We will consider ABE schemes for classes $\mathcal{F}$ consisting of RAM circuits or RAM programs, and refer to these as RAM-ABE.

Theorem 6.2. *Assume LWE holds. For any $d, L \in \mathbb{N}$, there exists an ABE scheme* (Setup, KeyGen, Enc, Dec) *for the class of RAM circuits that have depth at most d and maximum load L. The scheme has the following efficiency properties:*

- $\mathsf{Setup}(1^\lambda, 1^N)$ *runs in time* $N \cdot \mathrm{poly}(\lambda, d)$.
- *The running time of* $\mathsf{KeyGen}(\mathsf{msk}, C)$ *and the size of* sk_C *are bounded by* $\mathsf{size}_e(C) \cdot \mathrm{poly}(\lambda, d, \log L)$.
- $\mathsf{Enc}(\mathsf{mpk}, x, \mu)$ *runs in time* $N \cdot \mathrm{poly}(\lambda, d)$.
- $\mathsf{Dec}(\mathsf{sk}_C, x, \mathsf{ct})$ *runs in time* $\mathsf{size}_c(C) \cdot \mathrm{poly}(\lambda, d, \log L)$.

Proof. Let $n = \mathrm{poly}(\lambda, d)$, $q = 2^{\mathrm{poly}(\lambda,d)}$, $m \geq 3n\lceil \log q \rceil$ and let $B, B' \in \mathbb{N}$ be such that $\mathsf{LWE}_{n,q,B}$ holds and the following inequalities hold: $B = \Omega(m\sqrt{\lambda N(\ell m + 1)^d})$, $B' > 2^\lambda m B$ and $B^2 B'(\ell m + 1)^{d+1} \leq q/4$. Let (EvalPK, EvalCTCoeffs) be the linear system of homomorphic operations for RAM circuits from Theorem 5.6, and let EvalCT be the algorithm given by the efficiency statement in that theorem. We define our ABE for RAM circuits as follows:

$\mathsf{Setup}(1^\lambda, 1^N)$**:** Sample a matrix with an associate trapdoor $(\mathbf{P}, \mathbf{T}) \leftarrow \mathsf{TGen}(1^n, q)$. Then uniformly and independently sample $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$. Output $(\mathsf{mpk} = (\mathbf{P}, \mathbf{A}, \mathbf{u}), \mathsf{msk} = (\mathsf{mpk}, \tau))$.

$\mathsf{KeyGen}(\mathsf{msk}, C)$**:** Parse $\mathsf{msk} = (\mathsf{mpk}, \tau)$ and $\mathsf{mpk} = (\mathbf{P}, \mathbf{A}, \mathbf{u})$. Compute $(\mathbf{A}_C, \mathsf{DS}) := \mathsf{EvalPK}(C, \mathbf{A})$. Sample $\mathbf{t} \leftarrow \mathsf{TSamp}([\mathbf{P} \mid \mathbf{A}_C], \mathbf{T}, \mathbf{u}, B)$. Output $\mathsf{sk}_C = (\mathbf{t}, \mathsf{DS})$.

$\mathsf{Enc}(\mathsf{mpk}, x, \mu)$**:** Parse $\mathsf{mpk} = (\mathbf{P}, \mathbf{A}, \mathbf{u})$. Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, $\mathbf{e}_x \leftarrow [-B', B']^{Nm}$, $\mathbf{e}_0 \leftarrow \chi_B^m$ and $\widetilde{e} \leftarrow \chi_B$ and set:

$$\begin{aligned}
\mathbf{b}_x^\top &= \mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}_x^\top \\
\mathbf{b}_0^\top &= \mathbf{s}^\top \mathbf{P} + \mathbf{e}_0^\top \\
\beta &= \mathbf{s}^\top \mathbf{u} + \widetilde{e} + \mu \lceil q/2 \rfloor .
\end{aligned}$$

Output $\mathsf{ct} = (\mathbf{b}_x, \mathbf{b}_0, \beta)$.

$\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}, x)$**:** Parse $\mathsf{sk}_C = (\mathbf{t}, \mathsf{DS})$ and $\mathsf{ct} = (\mathbf{b}_x, \mathbf{b}_0, \beta)$. If $C(x) = 1$, output $\perp$. Otherwise, compute $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x)$ and output $\mathsf{round}_q(\beta - [\mathbf{b}_0^\top | \mathbf{b}_x^\top]\mathbf{t})$.

Correctness. Let $C \in \mathcal{C}$ and $x \in \{0,1\}^N$ such that $C(x) = 0$. By correctness of $(\mathsf{EvalPK}, \mathsf{EvalCTCoeffs})$ and the fact that $C(x) = 0$, the vector $\mathbf{b}_C := \mathsf{EvalCT}(C, \mathsf{DS}, \mathbf{A}, x, \mathbf{b}_x)$ computed in Dec satisfies $\mathbf{b}_C^\top = \mathbf{s}^\top \mathbf{A}_C + \mathbf{e}_C^\top$ where $\|\mathbf{e}_C\|_\infty \leq (\ell m + 1)^d \cdot B$. Thus by the fact that $[\mathbf{P} \mid \mathbf{A}]\mathbf{t} = \mathbf{u}$ we have

$$\begin{aligned}
\beta - \left[\mathbf{b}_0^\top \mid \mathbf{b}_x^\top\right] \mathbf{t} &= \mathbf{s}^\top \mathbf{u} + \widetilde{e} + \mu \lceil q/2 \rfloor - \mathbf{s}^\top \left[\mathbf{P} \mid \mathbf{A}_C\right] \mathbf{t} - \left[\mathbf{e}_0^\top \mid \mathbf{e}_C^\top\right] \\
&= \mu \lceil q/2 \rfloor + \underbrace{\widetilde{e} - \left[\mathbf{e}_0^\top \mid \mathbf{e}_C^\top\right] \mathbf{t}}_{:=e^*} .
\end{aligned}$$

Then correctness of the scheme follows from the fact that

$$|e^*| \leq |\widetilde{e}| + \left\|\mathbf{t}^\top\right\|_\infty (\|\mathbf{e}_0\|_\infty + \|\mathbf{e}_C\|_\infty) \leq B + mB(B + B' \cdot (\ell m + 1)^{d+1}) \leq B^2 B' (\ell m + 1)^{d+1} < q/4.$$

Security. We proceed by defining the following sequence of hybrid experiments:

Hybrid 0: This is the experiment $\mathsf{Expt}_{\mathsf{ABE}}^{\mathcal{A}}(1^\lambda, 0)$.

Hybrid 1: This is the same as hybrid 0 except we change the way $\mathbf{A}$ is sampled in the master public key. Given the adversary's chosen attribute $x \in \{0,1\}^N$, we sample $\mathbf{R} \leftarrow \{-1, 1\}^{m \times Nm}$ and put $\mathbf{A} = \mathbf{P}\mathbf{R} + x \otimes \mathbf{G}$.

Hybrid 2: This is the same as hybrid 1 except we now sample $\mathbf{P} \leftarrow \mathbb{Z}_q^{n \times m}$ in the master public key uniformly instead of via TGen; then we still set $\mathbf{A} = \mathbf{P}\mathbf{R} + x \otimes \mathbf{G}$ as in hybrid 1. Then to instantiate the KeyGen oracle, since we don't have a trapdoor for $\mathbf{P}$ anymore, we compute $\mathbf{H}_{C,x} := \mathsf{EvalCTCoeffs}(C, \mathsf{DS}, \mathbf{A}, x)$ and sample $\mathbf{t} \leftarrow \mathsf{TSamp}([\mathbf{P} \mid \mathbf{A}_C], \mathbf{T}', \mathbf{u}, B)$ where $\mathbf{T}' = [-\mathbf{R}\mathbf{H}_{C,x} \mid \mathbf{I}]^\top$.

Hybrid 3: This is the same as hybrid 2 except we change how $\mathbf{b}_x$ is sampled in the challenge ciphertext. Sample $\mathbf{b}_0$ and $\mathbf{e}_x$ as normal and then set $\mathbf{b}_x^\top = \mathbf{b}_0 \mathbf{R} + \mathbf{e}_x^\top$.

Hybrid 4: This is the same as hybrid 3 except we now sample $\mathbf{b}_0 \leftarrow \mathbb{Z}_q^m$ and $\beta \leftarrow \mathbb{Z}_q$ uniformly at random.

We can see hybrid 0 is statistically indistinguishable from hybrid 1 by a straightforward application of the leftover hash lemma. Next we show that hybrid 1 is statistically indistinguishable from hybrid 2. In hybrid 2, the matrix $\mathbf{T}'$ is a trapdoor for $[\mathbf{P} \mid \mathbf{A}_C]$ because, by correctness of $(\mathsf{EvalPK}, \mathsf{EvalCTCoeffs})$ together with the fact that $C(x) = 1$,

$$\mathbf{A}_C = (\mathbf{A} - x \otimes \mathbf{G})\mathbf{H}_{C,x} + \mathbf{G} = \mathbf{P}\mathbf{R}\mathbf{H}_{C,x} + \mathbf{G}$$

and hence $[\mathbf{P} \mid \mathbf{A}_C]\mathbf{T}' = \mathbf{G}$. Moreover correctness of the homomorphic operations also implies that $\|\mathbf{T}'\| \leq \|\mathbf{R}\| \cdot \left\|\mathbf{H}_{C,x}^\top\right\| \leq Nm(\ell m + 1)^d$. Thus hybrid 1 is statistically indistinguishable from hybrid 2 by the fact that matrices sampled via TGen are statistically close to uniform and the distributions on the vector $\mathbf{t}$ sampled in each hybrid are both statistically close to the distribution that samples from χ_B^m conditioned on $[\mathbf{P} \mid \mathbf{A}_C]\mathbf{t} = \mathbf{u}$.

Next observe that, since $\mathbf{A} = \mathbf{P}\mathbf{R} + x \otimes \mathbf{G}$, we have that in hybrid 3

$$\begin{aligned}\mathbf{b}_x^\top &= \mathbf{b}_0^\top \mathbf{R} + \mathbf{e}_x^\top \\ &= \mathbf{s}^\top \mathbf{P}\mathbf{R} + \mathbf{e}_0^\top \mathbf{R} + \mathbf{e}_x^\top \\ &= \mathbf{s}^\top(\mathbf{A} - x \otimes \mathbf{G}) + \mathbf{e}_x^\top + \mathbf{e}_0^\top \mathbf{R}.\end{aligned}$$

Thus hybrids 2 and 3 are statistically indistinguishable because the distributions of $\mathbf{e}_x^\top$ and $\mathbf{e}_x^\top + \mathbf{e}_0^\top \mathbf{R}$ are indistinguishable by noise smudging (see Lemma 2.2) since $B' > 2^\lambda mB \geq 2^\lambda \|\mathbf{e}_0\|_\infty \cdot \|\mathbf{R}^\top\|$.

Finally we argue that hybrids 3 and 4 are computationally indistinguishable under the LWE assumption. Consider the reduction that takes as input LWE challenges $(\mathbf{P}, \mathbf{b}_0)$ and $(\mathbf{u}, \alpha)$ and then gets x from the adversary and generates the master public key as $\mathsf{mpk} = (\mathbf{P}, \mathbf{A} = \mathbf{P}\mathbf{R} + x \otimes \mathbf{G}, \mathbf{u})$ for $\mathbf{R} \leftarrow \{-1, 1\}^{m \times Nm}$. It answers KeyGen queries as in hybrid 3, and constructs that challenge ciphertext by using $\mathbf{b}_0$ from the LWE challenge, setting $\beta = \alpha + \mu_0 \lceil q/2 \rfloor$ and setting $\mathbf{b}_x$ as in hybrid 3. In the case that $(\mathbf{P}, \mathbf{b}_0)$ and $(\mathbf{u}, \alpha)$ are LWE samples, then the reduction exactly matches the view of the adversary in hybrid 3. On the other hand if $\mathbf{b}_0$ and α are uniform, then the reduction matches the view of the adversary in hybrid 4.

A symmetrical argument to the one above also shows that $\mathsf{Expt}_{\mathsf{ABE}}^{\mathcal{A}}(1^\lambda, 1)$ is indistinguishable from hybrid 4, and thus we have $\mathsf{Expt}_{\mathsf{ABE}}^{\mathcal{A}}(1^\lambda, 0) \approx_c \mathsf{Expt}_{\mathsf{ABE}}^{\mathcal{A}}(1^\lambda, 1)$ which completes the proof of security.

Efficiency. The efficiency properties of KeyGen and Dec follow respectively from the efficiency properties of EvalPK and EvalCT in Theorem 4.12. The properties of Setup and Enc follow by inspection. □

Remark 6.1 (RAM-ABE for unbounded depth circuits from evasive LWE)**.** The security argument for the above construction crucially relied on the linearity property of the system of homomorphic operations in order for the reduction to be able to instantiate the KeyGen oracle for the adversary when it no longer had a trapdoor for $\mathbf{P}$. This made it impossible to use the bootstrapped system of homomorphic operations we constructed in Theorem 4.13 because it lacked linearity. However, [HLL23] show how to fix the security argument for non-linear systems under the evasive LWE assumption. Their techniques also apply in our setting.

References

[AFS19] Prabhanjan Ananth, Xiong Fan, and Elaine Shi. Towards attribute-based encryption for RAMs from LWE: Sub-linear decryption, and more. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 112–141. Springer, Heidelberg, December 2019. 4

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, Heidelberg, April 2012. 11

[AKS83] Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n \log n)$ sorting network. In *15th Annual ACM Symposium on Theory of Computing*, pages 1–9. ACM Press, April 1983. 15

[Bat68] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 307–314. Association for Computing Machinery, 1968. 15

[BCG$^+$18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for ram programs and succinct randomized encodings. *SIAM Journal on Computing*, 47(3):1123–1210, 2018. 4

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, Heidelberg, May 2014. 5, 9, 18, 38

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 55–73. Springer, Heidelberg, August 2000. 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: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 662–693. Springer, Heidelberg, November 2017. 2

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 264–302. Springer, Heidelberg, November 2017. 5, 22

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 1–30. Springer, Heidelberg, March 2015. 5

[CDG$^+$17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65. Springer, Heidelberg, August 2017. 1

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 169–178. Association for Computing Machinery, January 2016. 4

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 429–437. ACM Press, June 2015. 4

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 694–726. Springer, Heidelberg, November 2017. 2

[DGM23] Nico Döttling, Phillip Gajland, and Giulio Malavolta. Laconic function evaluation for turing machines. In *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*, Lecture Notes in Computer Science, pages 606–634. Springer, Heidelberg, May 2023. 2

[DHMW24] Fangqi Dong, Zihan Hao, Ethan Mook, and Daniel Wichs. Laconic function evaluation, functional encryption and obfuscation for rams with sublinear computation. To appear in EUROCRYPT, 2024. `https://eprint.iacr.org/2024/068`. 1, 2, 3

[DKL$^+$23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In *Advances in Cryptology – EUROCRYPT 2023, Part III*, Lecture Notes in Computer Science, pages 417–446. Springer, Heidelberg, June 2023. 5, 45, 48

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private RAM computation. In *55th Annual Symposium on Foundations of Computer Science*, pages 404–413. IEEE Computer Society Press, October 2014. 4

[GKP$^+$13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer, Heidelberg, August 2013. 36

[GKP$^+$13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564. ACM Press, June 2013. 1, 4, 8

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, may 1996. 8

[Goo14] Michael T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 684–693. ACM Press, May / June 2014. 15

[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, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, Heidelberg, August 2013. 3, 5, 11

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, Heidelberg, August 2015. 5

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 469–477. ACM Press, June 2015. 5

[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, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 589–619. Springer, Heidelberg, August 2019. 8

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 415–434. IEEE, 2023. 1, 4, 5, 8, 9, 11, 25, 26, 27, 28, 29, 41

[HOWW19] Ariel Hamlin, Rafail Ostrovsky, Mor Weiss, and Daniel Wichs. Private anonymous data access. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 244–273. Springer, Heidelberg, May 2019. 8

[JLL23] Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In *Advances in Cryptology – EUROCRYPT 2023, Part III*, Lecture Notes in Computer Science, pages 479–510. Springer, Heidelberg, June 2023. 4

[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, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 595–608. ACM, 2023. 2, 8, 9, 12

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, Heidelberg, April 2012. 5

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 859–870. IEEE Computer Society Press, October 2018. 1, 3, 4, 5, 8, 9, 29, 31, 35, 36

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 463–472. ACM Press, October 2010. 4

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In *Advances in Cryptology – CRYPTO 2022, Part I*, Lecture Notes in Computer Science, pages 535–559. Springer, Heidelberg, August 2022. 9

[Wee22] Hoeteck Wee. Optimal broadcast encryption and CP-ABE from evasive lattice assumptions. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 217–241. Springer, Heidelberg, May / June 2022. 9

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, October 1986. 13

A Laconic Encryption

In this section, we sketch how to use our techniques to derive a laconic encryption scheme, as defined in [DKL$^+$23]. While [DKL$^+$23] gave a construction using LWE, our techniques yield an alternate construction that is conceptually different and achieves a smaller ciphertext size (ℓ vs. 2ℓ LWE matrices). We start with a simpler definition that does not allow updates, but then discuss how to also enable efficient updates as well. Our construction essentially uses homomorphic computation for matrix read gates from Section 4.2 to get a simple laconic encryption.

Syntax. A laconic encryption scheme allows a group of users to individually select public/secret key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ derived using some public parameters pp. A server Alice gets all the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ and can deterministically construct a short digest $\mathsf{dig} = \mathsf{Hash}(\mathsf{pp}, \mathsf{pk}_1, \ldots, \mathsf{pk}_L)$. In addition, Alice deterministically computes decryption hints $\mathsf{wit}_1, \ldots, \mathsf{wit}_L$

for each of the users respectively. Given dig along with an index $\mathsf{ind} \in [L]$ and a message μ, anyone can compute a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, \mathsf{ind}, \mu)$ encrypting the message μ to user ind. Correctness requires that the designated user can decrypt given the corresponding secret key and decryption hint, namely:

$$\mathsf{Dec}(\mathsf{sk}_{\mathsf{ind}}, \mathsf{wit}_{\mathsf{ind}}, \mathsf{Enc}(\mathsf{dig}, \mathsf{ind}, \mu)) = \mu$$

Security stipulates that μ is hidden even given all other secret keys except for $\mathsf{sk}_{\mathsf{ind}}$. The efficiency requirement is that the digest and the hints have size $\mathrm{poly}(\lambda, \log L)$ and that encryption and decryption run in time $\mathrm{poly}(\lambda, \log L)$.

Construction. We write $L = 2^\ell$ and identify $[L]$ with $\{0,1\}^\ell$. Let n, m, q, B, β be parameters and let χ_β be the truncated discrete Gaussian distribution. We use $\mathsf{EvalPK}, \mathsf{EvalCTCoeffs}$ from Section 4.2.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$: Sample $\mathbf{A}_0 \leftarrow \mathbb{Z}_q^{n\times m}$, $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times \ell m}$ and set $\mathsf{pp} = (\mathbf{A}_0, \mathbf{A})$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$: Sample $\mathbf{v}_i \leftarrow \{0,1\}^m$, compute $\mathbf{p}_i := \mathbf{A}_0 \mathbf{v}_i$ and output $(\mathsf{pk}_i, \mathsf{sk}_i) := (\mathbf{p}_i, \mathbf{v}_i)$.
- $(\mathsf{dig}, \{\mathsf{wit}_i\}_{i\in[L]}) := \mathsf{Hash}(\mathsf{pp}, \mathsf{pk}_1, \dots, \mathsf{pk}_L)$: For $i \in [L]$, parse $\mathsf{pk}_i = \mathbf{p}_i$, and set $\mathbf{P}_i := [\mathbf{p}_i \mid \mathbf{0} \mid \cdots \mid \mathbf{0}] \in \mathbb{Z}_q^{n\times m}$. Set $\mathbf{P} = (\mathbf{P}_1, \dots, \mathbf{P}_L)$, and compute $(\mathbf{A}_{\mathsf{mr}}, \{\mathbf{H}_{\mathsf{mr},i}\}_{i\in[L]}) := \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}}, \mathbf{A})$. Set $\mathsf{dig} := \mathbf{A}_{\mathsf{mr}}, \mathsf{wit}_i = (i, \mathbf{h}_{\mathsf{mr},i})$ where $\mathbf{h}_{\mathsf{mr},i}$ is the first column of $\mathbf{H}_{\mathsf{mr},i}$.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{dig}, \mathsf{ind}, \mu)$: Sample $\mathbf{s}^\top \leftarrow \mathbb{Z}_q^n, \mathbf{e}_0 \leftarrow \chi_\beta^m, \mathbf{e}_1 \leftarrow \chi_\beta^{m\ell}, e_2 \leftarrow [-B, B]$ and output

  $$\mathsf{ct} := (\mathbf{s}^\top \mathbf{A}_0 + \mathbf{e}_0^\top, \mathbf{s}^\top(\mathbf{A} - \overbrace{\mathsf{ind}}^{\in\{0,1\}^\ell} \otimes \mathbf{G}) + \mathbf{e}_1^\top, \mathbf{s}^\top \mathbf{a}_{\mathsf{mr}} + e_2 + \mu \cdot \lceil q/2 \rfloor)$$

  where $\mathbf{a}_{\mathsf{mr}} \in \mathbb{Z}_q^n$ denotes the first column of $\mathbf{A}_{\mathsf{mr}}$.
- $\mu := \mathsf{Dec}(\mathsf{sk}, \mathsf{wit}_{\mathsf{ind}}, \mathsf{ct})$: Parse $\mathsf{ct} = (\mathbf{c}_0, \mathbf{c}_1, c_2)$, $\mathsf{sk} = \mathbf{v}_{\mathsf{ind}}, \mathsf{wit}_{\mathsf{ind}} = (\mathsf{ind}, \mathbf{h}_{\mathsf{mr},\mathsf{ind}})$ and output

  $$\mathsf{round}_q(c_2 - \mathbf{c}_1^\top \mathbf{h}_{\mathsf{mr},\mathsf{ind}} - \mathbf{c}_0^\top \mathbf{v}_{\mathsf{ind}}).$$

Lemma A.1 (Correctness). *Let the parameters $n, q, B, m \geq \lceil n \log q \rceil, \beta$ in the construction be such that $B + \beta m(\ell + 1) < q/4$. Then the construction achieves correctness with probability 1. Namely, for all* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $\mathsf{ind} \in [L = 2^\ell]$, $(\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, *all* $\{\mathsf{pk}_i\}_{i\in[L]\setminus\{\mathsf{ind}\}}$, $\mu \in \{0,1\}$, *and for* $(\mathsf{dig}, \{\mathsf{wit}_i\}_{i\in[L]}) := \mathsf{Hash}(\mathsf{pp}, \mathsf{pk}_1, \dots, \mathsf{pk}_L)$, *if* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{dig}, \mathsf{ind}, \mu)$, *then* $\mathsf{Dec}(\mathsf{sk}, \mathsf{wit}_{\mathsf{ind}}, \mathsf{ct}) = \mu$ *with probability 1.*

Proof. Let $\mathsf{ct} = (\mathbf{c}_0, \mathbf{c}_1, c_2)$, $\mathsf{sk} = \mathbf{v}_{\mathsf{ind}}, \mathsf{wit}_{\mathsf{ind}} = (\mathsf{ind}, \mathbf{h}_{\mathsf{mr},\mathsf{ind}})$ where $\mathbf{h}_{\mathsf{mr},i\mathsf{ind}}$ is the first column of $\mathbf{H}_{\mathsf{mr},\mathsf{ind}}$ be generated as in the lemma with:

$$\mathbf{c}_0^\top = \mathbf{s}^\top \mathbf{A}_0 + \mathbf{e}_0^\top, \mathbf{c}_1^\top = \mathbf{s}^\top(\mathbf{A} - \mathsf{ind} \otimes \mathbf{G}) + \mathbf{e}_1^\top, c_2 = \mathbf{s}^\top \mathbf{a}_{\mathsf{mr}} + e_2 + \mu \cdot \lceil q/2 \rfloor)$$

By the correctness of the homomorphic operation for matrix-read gates (Theorem 4.8) we have:

$$\begin{aligned} & \mathbf{c}_1^\top \mathbf{H}_{\mathsf{mr},\mathsf{ind}} = \mathbf{s}^\top(\mathbf{A}_{\mathsf{mr}} - \mathbf{P}_{\mathsf{ind}}) + \mathbf{e}_{\mathsf{mr}}^\top \\ \Rightarrow\ & \mathbf{c}_1^\top \mathbf{h}_{\mathsf{mr},\mathsf{ind}} = \mathbf{s}^\top(\mathbf{a}_{\mathsf{mr}} - \mathbf{p}_{\mathsf{ind}}) + e_{\mathsf{mr}} \end{aligned}$$

with $|e_{\mathsf{mr}}| \leq \|\mathbf{e}_{\mathsf{mr}}\|_\infty \leq \gamma(\mathsf{mRead}) \cdot \|\mathbf{e}_1\|_\infty \leq \ell m \beta$. Therefore:

$$\begin{aligned} d &:= c_2 - \mathbf{c}_1^\top \mathbf{h}_{\mathsf{mr,ind}} - \mathbf{c}_0^\top \mathbf{v}_{\mathsf{ind}} \\ &= \mathbf{s}^\top \mathbf{a}_{\mathsf{mr}} + e_2 + \mu \cdot \lceil q/2 \rfloor - \mathbf{s}^\top (\mathbf{a}_{\mathsf{mr}} - \mathbf{p}_{\mathsf{ind}}) - e_{\mathsf{mr}} - \mathbf{s}^\top \overbrace{\mathbf{A}_0 \cdot \mathbf{v}_{\mathsf{ind}}}^{\mathbf{p}_{\mathsf{ind}}} - \mathbf{e}_0^\top \mathbf{v}_{\mathsf{ind}} \\ &= \mu \cdot \lceil q/2 \rfloor + \underbrace{(e_2 - e_{\mathsf{mr}} - \mathbf{e}_0^\top \mathbf{v}_{\mathsf{ind}})}_{e^*} \end{aligned}$$

with $\|e^*\|_\infty \leq B + \ell m \beta + m\beta < q/4$ and therefore $\mathsf{round}_q(d) = \mu$. □

Lemma A.2 (Security). *Consider the following security game* $\mathsf{LESec}_{\mathcal{A},b}(1^\lambda)$ *with a stateful adversary* $\mathcal{A}$*:*

1. *The adversary chooses* 1^ℓ *and* $\mathsf{ind} \in [L = 2^\ell]$.
2. *The challenger chooses* $(\mathsf{pk}_{\mathsf{ind}}, \mathsf{sk}_{\mathsf{ind}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and gives* $\mathsf{pk}_{\mathsf{ind}}$ *to the adversary.*
3. *The adversary chooses* $\{\mathsf{pk}_i\}_{i \in [L] \setminus \{\mathsf{ind}\}}$ *and sends these to the challenger.*
4. *The challenger computes* $(\mathsf{dig}, \{\mathsf{wit}_i\}_{i \in [L]}) := \mathsf{Hash}(\mathsf{pp}, \mathsf{pk}_1, \ldots, \mathsf{pk}_L)$.[11] *The challenger set* $\mu := b$, *samples* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{dig}, \mathsf{ind}, \mu)$ *and sends* ct *to the adversary.*
5. *The adversary outputs a bit which is the output of the game.*

We define a laconic encryption scheme to be secure if for all PPT $\mathcal{A}$ *we have* $|\Pr[\mathsf{LESec}_{\mathcal{A},0}(1^\lambda) = 1] - \Pr[\mathsf{LESec}_{\mathcal{A},1}(1^\lambda) = 1]| = \mathrm{negl}(\lambda)$. *Then the above construction is secure under the* $\mathsf{LWE}_{n,q,\beta}$ *assumption, as long as* $B \geq (\ell m + 1)\beta\lambda^{\omega(1)}$.

Proof. The proof is similar to that of the AB-LFE scheme.

First, we consider a modified game LESec' where the last component of the ciphertext $\mathsf{ct} = (\mathbf{c}_0, \mathbf{c}_1, c_2)$ is modified by replacing c_2 with $c_2' = \mathbf{s}^\top \mathbf{p}_{\mathsf{ind}} + e_2 + e_2' + \mu \lceil q/2 \rfloor$ where $e_2' \leftarrow \chi_\beta$. Note that in the modified game the ciphertext does not depend on the digest at all. First we claim that if a PPT adversary $\mathcal{A}$ can break the security of the original game LESec with non-negligible probability then we can construct a PPT adversary $\mathcal{A}'$ that breaks the security or the modified game LESec' with non-negligible probability. The adversary $\mathcal{A}'$ simply runs $\mathcal{A}$ until the last step of the game. In the last step it receives a ciphertext $\mathsf{ct}' = (\mathbf{c}_0, \mathbf{c}_1, c_2')$. It computes

$$\begin{aligned} c_2 &:= c_2' + \mathbf{c}_1^\top \mathbf{h}_{\mathsf{mr,ind}} \\ &= \mathbf{s}^\top \mathbf{p}_{\mathsf{ind}} + e_2 + e_2' + \mu \lceil q/2 \rfloor + \mathbf{s}^\top (\mathbf{a}_{\mathsf{mr}} - \mathbf{p}_{\mathsf{ind}}) + e_{\mathsf{mr}} \\ &= \mathbf{s}^\top \mathbf{a}_{\mathsf{mr}} + (e_2 + e_2' + e_{\mathsf{mr}}) \end{aligned}$$

with $|e_2' + e_{\mathsf{mr}}| \leq \beta + \ell m \beta$ by the same argument used to analyze correctness. It sends $\mathsf{ct} = (\mathbf{c}_0, \mathbf{c}_1, c_2)$ to $\mathcal{A}$ and outputs what it outputs. The resulting view of $\mathcal{A}$ is essentially the same as in the original game LESec' except that the error c_2 is $(e_2 + e_2' + e_{\mathsf{mr}})$ instead of just e_2, but this is statistically close by noise smudging (Lemma 2.2). Therefore the advantage of $\mathcal{A}'$ in the game LESec' is negligibly close to that of $\mathcal{A}$ in the game LESec.

[11]The adversary can compute these values on its own as well.

Secondly, we argue that any PPT adversary $\mathcal{A}'$ has at most a negligible advantage in the modified game LESec'. This is because (a) we can replace the vector $\mathbf{p}_{\mathsf{ind}} = \mathbf{A}_0 \mathbf{v}_{\mathsf{ind}}$ by a uniformly random vector by the leftover hash lemma, and (b) the ciphertext

$$\mathsf{ct} = (\mathbf{c}_0, \mathbf{c}_1, c_2') = \mathbf{s}^\top \underbrace{[\mathbf{A}_0 \mid \mathbf{A} - \mathsf{ind} \otimes \mathbf{G} \mid \mathbf{p}_{\mathsf{ind}}]}_{=\mathbf{A}'} + \underbrace{[\mathbf{e}_0^\top, \mathbf{e}_1^\top, e_2']}_{=\mathbf{e}'} + [\mathbf{0}, \mathbf{0}, e_2 + \mu \lceil q/2 \rfloor]$$

is then computationally indistinguishable from uniform by the LWE assumption with the uniformly random coefficient matrix $\mathbf{A}' \in \mathbb{Z}_q^{n \times (m + \ell m + 1)}$ and error $\mathbf{e}' \leftarrow \chi^{m+\ell m+1}$. □

Efficiency Optimization. Instead of choosing $e_2 \leftarrow [-B, B]$ for a super-polynomial $B = \lambda^{\omega(1)}$ we can choose e_2 from a smaller polynomially-bounded discrete Gaussian distribution by utilizing the same "LWE with error-leakage" analysis from [DKL+23] to argue security. This allows us to use a smaller polynomial-size modulus q, which improves efficiency.

Exponential Index Space. We can extend the above construction to a scenario where the bit-length ℓ of an index ind is an arbitrary polynomial in the security parameter, meaning that the index-space $L = 2^\ell$ is exponential. In this case, only some polynomial set of public keys $\{\mathsf{pk}_{\mathsf{ind}}\}_{\mathsf{ind} \in S}$ is defined for some subset $S \subseteq \{0,1\}^\ell$ and we can think of $\mathsf{pk}_{\mathsf{ind}} = 0$ for all $\mathsf{ind} \notin S$. Alternately, we can think of this as corresponding to an exponentially large vector of $L = 2^\ell$ public keys, which is sparse with only polynomially many non-zero entries. We show that it is possible to compute $(\mathsf{dig}, \{\mathsf{wit}_{\mathsf{ind}}\}_{\mathsf{ind} \in S}) := \mathsf{Hash}(\mathsf{pp}, \{\mathsf{pk}_{\mathsf{ind}}\}_{\mathsf{ind} \in S})$ efficiently in time $|S| \mathrm{poly}(\ell, \lambda)$.

The construction is the same but now we need to compute:

$$(\mathbf{A}_{\mathsf{mr}}, \{\mathbf{H}_{\mathsf{mr},\mathsf{ind}}\}_{\mathsf{ind} \in S}) := \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}}, \mathbf{A})$$

where the database of matrices $\mathbf{P} = (\mathbf{P}_1, \ldots, \mathbf{P}_L)$ is an exponentially large with only $|S|$ non-zero entries $\{\mathbf{P}_{\mathsf{ind}}\}_{\mathsf{ind} \in S}$. We can do this efficiently by going under the hood of the recursive construction of EvalPK form mRead gates in the proof of Theorem 4.8. Recall that we compute $(\mathbf{A}_{\mathsf{mr}}, \{\mathbf{H}_{\mathsf{mr},i}\}) = \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}}, \mathbf{A})$ by recursively computing $(\mathbf{A}_{\mathsf{mr}^{\mathsf{L}}}, \{\mathbf{H}_{\mathsf{mr}^{\mathsf{L}},i}\}) = \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}^{\mathsf{L}}}, \mathbf{A}^-)$ and $(\mathbf{A}_{\mathsf{mr}^{\mathsf{R}}}, \{\mathbf{H}_{\mathsf{mr}^{\mathsf{R}},i}\}) = \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}^{\mathsf{R}}}, \mathbf{A}^-)$ where $\mathbf{P}^{\mathsf{L}}, \mathbf{P}^{\mathsf{R}}$ denote the first/last $L/2$ matrices in $\mathbf{P}$ respectively. We observe the following:

- Whenever $\mathbf{P}$ only has $\mathbf{0}$ matrices, then $\mathbf{A}_{\mathsf{mr}} = \mathbf{0}$ and $\{\mathbf{H}_{\mathsf{mr},i} = \mathbf{0}\}$ for all $i \in [L]$.
- Therefore, we can compute $\mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}}, \mathbf{A})$ recursively as before, but we stop the recursion whenever the value $\mathbf{P}^{\mathsf{L}}$ or $\mathbf{P}^{\mathsf{R}}$ in one of the recursive calls is empty, in which case we return $\mathbf{A}_{\mathsf{mr}^{\mathsf{L}}} = \mathbf{0}$ or $\mathbf{A}_{\mathsf{mr}^{\mathsf{R}}} = \mathbf{0}$ respectively. This results in a recursion tree with only $O(|S|)$ leafs and a total of $|S| \cdot \mathrm{poly}(\ell, \lambda)$ total run-time.

The security of the scheme is the same as before, but now in the security game described in Lemma A.2, we modify step 3 to let the adversary chooses some set S containing ind and all the public keys $\{\mathsf{pk}_i\}_{i \in S \setminus \{\mathsf{ind}\}}$ except for $\mathsf{pk}_{\mathsf{ind}}$ which is chosen by the challenger. In step 4 the challenger computes $(\mathsf{dig}, \{\mathsf{wit}_i\}_{i \in [L]}) := \mathsf{Hash}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S})$. The proof of security is identical.

Updates. We can further extend the above construction to allow for updates, where we can add or replace a public key $\mathsf{pk}_{\mathsf{ind}}$ at index ind and efficiently update the digest dig and the decryption hints wit_i accordingly. Note that all the decryption hints wit_i change when a new public is $\mathsf{pk}_{\mathsf{ind}}$ is added at any index ind. Therefore, our model allows the server Alice to keep an auxiliary data structure aux. Every time the set of public keys $\{\mathsf{pk}_{\mathsf{ind}}\}_{\mathsf{ind} \in S}$ is updated, Alice can update the digest dig as well as the data structure aux in $\mathrm{poly}(\ell, \lambda)$ time via some procedure $\mathsf{dig}' := \mathsf{Update}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{ind}, \mathsf{pk}_{\mathsf{ind}})$ that read/writes to the data structure aux, independent of the current number of keys $|S|$. Alice can use the data structure to compute a decryption hint $\mathsf{wit}_{\mathsf{ind}} := \mathsf{WGen}^{\mathsf{aux}}(\mathsf{ind})$ for any user in $\mathrm{poly}(\ell, \lambda)$ time.

The data structure aux consists of an incomplete binary tree with all the matrices $\mathbf{A}_{\mathsf{mr}} := \mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}}, \mathbf{A})$ computed during the recursive evaluation, where the recursion is cut off when it reaches a node that has no public keys in its sub-tree. To update the tree by adding/modifying a public key $\mathsf{pk}_{\mathsf{ind}}$ at index ind, the procedure $\mathsf{dig}' := \mathsf{Update}^{\mathsf{aux}}(\mathsf{pp}, \mathsf{ind}, \mathsf{pk}_{\mathsf{ind}})$ just needs to update the matrices along one path in the recursion tree going to the leaf ind. In other words, it redoes the recursive computation of $\mathsf{EvalPK}(\mathsf{mRead}_{\mathbf{P}}, \mathbf{A})$ with the updated set $\mathbf{P}$, but it uses the cached results stored in aux for all the recursive calls that are not on the path to ind. We do not keep track of the matrices $\mathbf{H}_{\mathsf{mr},\mathsf{ind}}$ during this process. Instead, looking at step 3 in the proof of Theorem 4.8, we notice that these matrices are defined recursively via:

$$\mathbf{H}_{\mathsf{mr},\mathsf{ind}} = \begin{pmatrix} (1 - \mathsf{ind}_\ell)\mathbf{H}_{\mathsf{mr}^{\mathsf{L}},\mathsf{ind}^-} + \mathsf{ind}_\ell \mathbf{H}_{\mathsf{mr}^{\mathsf{R}},\mathsf{ind}^-} \\ \mathbf{G}^{-1}(\mathbf{A}_{\mathsf{mr}^{\mathsf{R}}}) - \mathbf{G}^{-1}(\mathbf{A}_{\mathsf{mr}^{\mathsf{L}}}) \end{pmatrix}$$

where ind_ℓ is the last bit of the index ind. Note that only one of $\mathsf{ind}_\ell, 1 - \mathsf{ind}_\ell$ is non-zero. Since all the matrices $\mathbf{A}_{\mathsf{mr}^{\mathsf{L}}}, \mathbf{A}_{\mathsf{mr}^{\mathsf{R}}}$ in the tree are stored in aux, the procedure $\mathbf{H}_{\mathsf{mr},\mathsf{ind}} := \mathsf{WGen}^{\mathsf{aux}}(\mathsf{ind})$ can compute $\mathbf{H}_{\mathsf{mr},\mathsf{ind}}$ in $\mathrm{poly}(\ell, \lambda)$ time by recursively computing only one of $\mathbf{H}_{\mathsf{mr}^{\mathsf{L}},\mathsf{ind}^-}$ (if $\mathsf{ind}_\ell = 0$) or $\mathbf{H}_{\mathsf{mr}^{\mathsf{R}},\mathsf{ind}^-}$ (if $\mathsf{ind}_\ell = 1$).