Key-Homomorphic Computations for RAM: Fully Succinct Randomised Encodings and More

Damiano Abram
Bocconi University

Giulio Malavolta
Bocconi University

Lawrence Roy
Aarhus University

Abstract

We propose a new method to construct a public-key encryption scheme, where one can homomorphically transform a ciphertext encrypted under a key $\mathbf{x}$ into a ciphertext under $(P, P(\mathbf{x}))$, for any polynomial-time RAM program $P : \mathbf{x} \mapsto \mathbf{y}$ with runtime T and memory L. Combined with other lattice techniques, this allows us to construct:

- Succinct-randomised encodings from RAM programs with encoder complexity $(|\mathbf{x}| + |\mathbf{y}|) \cdot \mathrm{poly}(\log T, \log L)$ and rate-1 encodings.
- Laconic function evaluation for RAM programs, with encoder runtime bounded by $(|\mathbf{x}| + |\mathbf{y}|) \cdot \mathrm{poly}(\log T, \log L)$ and rate-1 encodings.
- Key-policy attribute-based encryption for RAM programs, with ciphertexts of size $O(T)$. The same scheme can be converted to the *register* setting, obtaining linear CRS size in the number of parties.

All of our schemes rely on the hardness of the *decomposed learning with errors* (LWE) problem, along with other standard computational assumptions on lattices. The decomposed LWE problem can be interpreted as postulating the circular-security of a natural lattice-based public-key encryption scheme. To gain confidence in the assumption, we show that it is implied by the hardness of the succinct LWE problem of Wee (CRYPTO'24).

Contents

1 Introduction

The quest for general-purpose cryptographic objects requires one to select a universal model of computation, in order to encompass all computable functions. By far the most popular choice in the cryptographic literature, is to model the computation as a Boolean/arithmetic circuit. Despite the theoretical appeal, this choice has limitations: Circuits cannot run their own code, they always have to read the entire input, and they always incur in the *worst-case* runtime. As a canonical example, binary search cannot be implemented by a circuit in time better than a linear scan. This concern prompted [GKP+13a] to ask whether one can go beyond the circuit model: Is it possible to compute Turing machines, or even RAM programs, on encrypted/authenticated data? In the last decade, important progress was made on this front, with the development of new techniques for fully-homomorphic encryption [HHWW19, LMW23], laconic-function-evaluation [DHMW24, DHM+24], attribute-based encryption [CW25], functional encryption [ACFQ22], randomised encodings [BGL+15, CHJV15], and program obfuscation [BCG+18, KLW15], to mention a few.

Although the feasibility of this paradigm has been largely established, many important cryptographic primitives, such as attribute-based encryption (ABE) for RAM, rely on techniques from program obfuscation in order to achieve the desired efficiency. Besides the loss in concrete efficiency, the use of very general-purpose cryptographic tools comes at a cost: Known constructions of program obfuscation [JLS21, JLS22] require the hardness of a combination of cryptographic problems, whereas the security of constructions based exclusively on lattices [WW21, GP21, BDGM22] is still not well-understood. To make things worse, the schemes in [JLS21, JLS22] are trivially broken by a quantum algorithm.

To illustrate the contrast, consider the case of ABE for *circuits*: The celebrated work of Boneh et al. [BGG+14] proposed an elegant lattice-based scheme based on *key-homomorphic public-key encryption*. Given a ciphertext encrypted under $\mathbf{x}$, there is a publicly-computable algorithm to transform it into a ciphertext encrypted under $(C, C(\mathbf{x}))$, for any circuit C. Omitting details, one can then issue a functional key for C, by calculating the secret key corresponding to $(C, 0)$,[1] so that decryption is only possible if $C(\mathbf{x}) = 0$. Besides the aforementioned application, this technique for computing on encrypted/authenticated data has been extremely influential, leading to a number of significant results and a better conceptual understanding of the cryptographic capabilities of lattices.

Given the elegance of the scheme from [BGG+14] and the lack of lattice-based schemes from well-understood computational assumptions, we view the problem of computing RAM programs over encrypted/authenticated data as an important gap in our understanding. The objective of this work is to place homomorphic RAM computation on similar footing as circuit-based schemes.

1.1 Our Results

Our main technical contribution is a technique for constructing *key-homomorphic public-key encryption* for RAM programs. We design a public procedure that allows anyone to turn a ciphertext encrypted under $\mathbf{x}$, into a ciphertext encrypted under $(P, P(\mathbf{x}))$. Importantly, the efficiency of the evaluation procedure only depends on the runtime of the RAM computation of P. Our ciphertexts are essentially identical to those of [BGG+14], and we view our work as its natural RAM analogue. We prove the security of our scheme against a new lattice assumption, that we refer to as *decomposed LWE*. This assumption can be interpreted as

[1]In line with the literature on lattice-based cryptography, we adopt the convention that an accepting circuit/program on input $\mathbf{x}$ returns 0.

postulating the circular security of a natural LWE-based public-key encryption scheme, and we show that it is at least as hard as the succinct LWE assumption [Wee24]. We discuss our assumption in more details in Section 1.2.

Our work directly improves upon the recent work of Dong et al. [DHM$^+$24], that achieved similar results in an *amortised sense*, i.e., in a model where a pre-processing of the program (proportional to its *circuit size*) is allowed. Our new technique allows us to make progress on several open problems in the literature, which we summarise below.

Succinct Randomised Encodings. Succinct randomised encodings (SRE) [BGL$^+$15, CHJV15, BCG$^+$18] allow one to encode a program P and an input $\mathbf{x}$ into a different program $\tilde{P}_{\mathbf{x}}$, such that anyone can recover $\mathbf{y} := P(\mathbf{x})$, and nothing more. The crucial property of an SRE scheme is that the encoder's computation should be sublinear, ideally poly-logarithmic, in the time needed to compute P. We show how to use our homomorphic evaluation technique to construct SRE for RAM programs with runtime T and memory L with essentially optimal parameters.

Theorem 1.1 (Informal). *Assuming the hardness of the Ring-LWE and the Decomposed-LWE problem, there exists an SRE scheme for RAM programs with encoder complexity* $(|\mathbf{x}| + |\mathbf{y}|) \cdot \mathrm{poly}(\log T, \log L)$.

We also consider a weakening of the above notion, where the program P is public, and the encoding also takes as input a message μ. Security requires that μ is computationally hidden if $P(\mathbf{x}) \neq 0$. One can think of this primitive as an analogue of witness encryption [GGSW13], but for computations in P. In this context, we obtain the same efficiency, without the need to invoke the hardness of the Ring-LWE problem.

Our construction matches the efficiency of obfuscation-based SRE schemes [BCG$^+$18, KLW15, AL18, GS18], and improves upon recent lattice-based SRE schemes based on a similar template, with encoding complexity $\sqrt{T} \cdot O(L)$ [AMZ24], and $T^{\varepsilon} \cdot O(L)$ [BG25], for any constant $\varepsilon > 0$.

Laconic Function Evaluation. A laconic function evaluation (LFE) [QWW18] allows a server to compress a function $f : X \to Y$ into a small digest h, which can be then used by an encryptor to encode an input $\mathbf{x}$. The server can then recover $\mathbf{y} := f(\mathbf{x})$ from the encoding, without learning anything else about the input. Once again, the crucial property here is that the runtime of the encoder should be sublinear (ideally independent) of f. We present a new LFE scheme for RAM programs P with runtime T and memory L with the following efficiency.

Theorem 1.2 (Informal). *Assuming the hardness of the Ring-LWE and the Decomposed-LWE problem, there exists a selectively secure LFE scheme for RAM programs with encoder complexity* $T \cdot (|\mathbf{x}| + |\mathbf{y}|) \cdot \mathrm{poly}(\log L)$ *and ciphertexts of size* $T \cdot \mathrm{poly}(\log L) + |\mathbf{x}| + |\mathbf{y}|$.

Additionally assuming a circular variant of Decomposed-LWE (where we additionally provide the distinguisher with a fully-homomorphic encryption of its own secret key), we can rely on the bootstrapping technique from [HLL23], and reduce the encoder complexity to $(|\mathbf{x}|+|\mathbf{y}|)\cdot \text{poly}(\log T, \log M)$. On the other hand, for the weaker notion of attribute-based LFE, we can get away without relying on the Ring-LWE assumption. All of our schemes satisfy the standard notion of selective security (in the choice of the input).

This improves upon the recent work of [DHM+24] that achieved similar parameters, but required a program-dependent pre-processing proportional to its *circuit size*, whereas no preprocessing is needed in our scheme. Furthermore, our scheme is rate-1 in the input, and our CRS does not set an a-priori bound on the input size (i.e., our scheme works for unbounded inputs).

Attribute-Based Encryption. Attribute-based encryption (ABE) [SW05, GPSW06] consists of a system, where a key authority publishes a master public key and is in charge of issuing *functional keys* associated with a function f. Anyone can use the master public-key to encrypt a message μ under an attribute $\mathbf{x}$, and the holder of the functional key for f can recover μ only if $f(\mathbf{x}) = 0$. Our techniques imply almost immediately a construction of ABE for RAM programs with the following parameters.

Theorem 1.3 (Informal)**.** *Assuming the hardness of the Decomposed-LWE problem, there exists a selectively-secure ABE scheme for RAM program with where the sizes of the common reference string, the secret keys, and the ciphertexts are bounded by $O(T)$.*

Using a standard transformation [AY20], this implies the existence of a broadcast encryption scheme from the same assumption, with essentially optimal parameters. When restricted to circuits, this matches a recent result of Wee [Wee25], except that we rely on a potentially weaker assumption. By additionally assuming of a variant of the (public-coin) evasive LWE heuristic introduced in [HLL23], we can improve the above bound to $O(1)$ – ignoring polynomial factors in the security parameter. We summarise the comparison with prior works on ABE in Table 1.

Finally, we also consider the problem of *register ABE* [HLWW23], where there is no trusted authority, and users sample their own public/secret key. We show a method to convert from our ABE scheme into a registered one, at the cost of growing the common reference string linearly in the number of users. As the ABE from which it derives, the construction has ciphertext size $O(T)$ (this can be made $O(1)$ by relying on the public-coin evasive LWE heuristic), however, it only achieves very selective security: the attribute and the secret keys of the corrupted parties (along with the relative RAM programs) must all be independent of the CRS. Previous works managed to build register ABE for general policies (in particular, bounded-depth Boolean circuits) either using witness encryption [HLWW23] or under succinct LWE in the *random oracle model* [CHW25] (at the cost of a quadratic dependency of the CRS in the number of users).

1.2 The Decomposed LWE Problem

Let us now state our new assumption explicitly. The decomposed LWE assumption states that the following distributions are computationally indistinguishable:

$$\left\{\mathbf{A}_i, \mathbf{B}_j, \mathbf{s}^\intercal \cdot (\mathbf{A}_i \cdot \mathbf{B}_j + \delta_{i,j} \cdot \mathbf{G}) + \mathbf{e}_{i,j}^\intercal\right\}_{i,j\in[t]} \approx_c \left\{\mathbf{A}_i, \mathbf{B}_j, \mathbf{u}_{i,j}^\intercal\right\}_{i,j\in[t]}$$

where $\mathbf{G}$ is the gadget matrix, $\mathbf{e}_{i,j}$ and $\mathbf{B}_j$ are sampled from a low-norm Gaussian, and the rest of the variables are sampled uniformly. First, we observe that the assumption has indeed a *circularity* characterisation: If we were to omit the terms $\delta_{i,j} \cdot \mathbf{G}$ from the equation, then the assumption would be equivalent to the standard LWE problem, by a simple smudging argument. In this sense, the decomposed LWE assumption can also be interpreted as postulating that an encryption scheme (that encodes copies of the message in the "diagonal terms") remains secure, even when encrypting its own secret key. This is similar in spirit to the assumption needed to prove security of fully-homomorphic encryption schemes [Gen09], although here the base scheme is somewhat different.

As further evidence that this circular variant of LWE is hard, we can show that this assumption is implied by the t-succinct LWE problem [Wee24]. Recall that t-succinct LWE postulates that:

$$\mathbf{T}, \mathbf{M}, \mathbf{A}, \mathbf{s}^\intercal \cdot \mathbf{M} + \mathbf{e}^\intercal \approx_c \mathbf{T}, \mathbf{M}, \mathbf{A}, \mathbf{u}^\intercal$$

is computationally close to uniform, even in the presence of a trapdoor $\mathbf{T}$ sampled from a low-norm distribution conditioned on satisfying:

$$\begin{bmatrix}\mathbf{I}_t \otimes \mathbf{M} & \mathbf{A}\end{bmatrix} \cdot \mathbf{T} = \mathbf{I}_t \otimes \mathbf{G}.$$

Now, parse:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_0 \\ \vdots \\ \mathbf{A}_{t-1} \end{bmatrix} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{T}_{0,0} & \mathbf{T}_{0,1} & \cdots & \mathbf{T}_{0,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{T}_{t-1,0} & \mathbf{T}_{t-1,1} & \cdots & \mathbf{T}_{t-1,t-1} \\ -\mathbf{B}_0 & -\mathbf{B}_1 & \cdots & -\mathbf{B}_{t-1} \end{bmatrix}.$$

To see the implication, it suffices to observe that, on input an LWE sample $\mathbf{s}^\intercal \mathbf{M} + \mathbf{e}^\intercal$, we can derive all of the Decomposed LWE elements by computing:

$$\begin{aligned}(\mathbf{s}^\intercal \mathbf{M} + \mathbf{e}^\intercal) \cdot \mathbf{T}_{i,j} + \tilde{\mathbf{e}}_{i,j}^\intercal &= \mathbf{s}^\intercal \mathbf{G} \cdot \delta_{i,j} - \mathbf{s}^\intercal \mathbf{A}_i \mathbf{B}_j + \mathbf{e} \cdot \mathbf{T}_{i,j} + \tilde{\mathbf{e}}_{i,j}^\intercal \\ &\approx_s \mathbf{s}^\intercal \mathbf{G} \cdot \delta_{i,j} - \mathbf{s}^\intercal \mathbf{A}_i \mathbf{B}_j + \tilde{\mathbf{e}}_{i,j}^\intercal\end{aligned}$$

which is statistically close to the proper distribution. In other words, decomposed LWE is nothing but t-succinct LWE, except that we provide the distinguisher with the sample directly multiplied by each block of the trapdoor. This is something that the distinguisher can also compute on its own, given the full trapdoor $\mathbf{T}$, and therefore decomposed LWE is implied by t-succinct LWE.

On the other hand, the reverse implication seems unlikely, since it appears hard to recover $\mathbf{T}$ given a decomposed LWE sample. In fact, unlike succinct LWE, decomposed LWE has an unstructured CRS. One can therefore conjecture that decomposed LWE is a strictly harder problem than t-succinct LWE, although at present we can only prove one direction. For more discussion on decomposed LWE, we refer to Section 4.

2 Technical Overview

We give an overview of our techniques, assuming familiarity with standard linear algebra and lattice notation. In this discussion, we keep things deliberately informal, and we refer

	Model	CRS	Key	Ciphertext	Assumption
[GVW13]	Circuits	$\lvert\mathbf{x}\rvert \cdot \text{poly}(D)$	$\lvert C\rvert \cdot \text{poly}(D)$	$\lvert\mathbf{x}\rvert \cdot \text{poly}(D)$	LWE
[BGG+14]	Circuits	$\lvert\mathbf{x}\rvert \cdot \text{poly}(D)$	$\text{poly}(D)$	$\lvert\mathbf{x}\rvert \cdot \text{poly}(D)$	LWE
[BV16]	Circuits	$\text{poly}(D)$	$\lvert\mathbf{x}\rvert + \text{poly}(D)$	$\lvert\mathbf{x}\rvert \cdot \text{poly}(D)$	LWE
[CW23]	Circuits	$\lvert\mathbf{x}\rvert \cdot \text{poly}(D)$	$O(1)$	$\lvert\mathbf{x}\rvert \cdot \text{poly}(D)$	LWE
[Wee24]	Circuits	$\lvert\mathbf{x}\rvert^2 \cdot \text{poly}(D)$	$\text{poly}(D)$	$\text{poly}(D)$	SLWE
[Wee24]	Circuits	$\lvert\mathbf{x}\rvert^{\frac{2}{3}} \cdot \text{poly}(D)$	$\text{poly}(D)$	$\lvert\mathbf{x}\rvert^{\frac{2}{3}} \cdot \text{poly}(D)$	SLWE
[Wee25]	Circuits	$\text{poly}(D)$	$\text{poly}(D)$	$\text{poly}(D)$	SLWE
[HLL23]	Circuits	$O(\lvert\mathbf{x}\rvert)$	$O(1)$	$O(\lvert\mathbf{x}\rvert)$	ELWE*
[AKY24b]	Circuits	$O(1)$	$O(1)$	$O(1)$	ELWE†
[AKY24a]	TM	$O(1)$	$O(\lvert M\rvert^2)$	$O(T)^2$	ELWE† + TLWE*
[AKY24b]	TM	$O(1)$	$O(\lvert M\rvert)$	$O(T)$	ELWE†
[CW25]	TM	$O(1)$	$O(1)$	$O(T)$	ELWE*
[DHM+24]	RAM	$\lvert\mathbf{x}\rvert \cdot \text{poly}(T)$	$O(T)$	$\lvert\mathbf{x}\rvert \cdot \text{poly}(T)$	LWE
This work	RAM	$O(T)$	$O(T)$	$O(T)$	DLWE
This work	RAM	$O(1)$	$O(1)$	$O(1)$	ELWE*

Table 1: Comparison among various KP-ABE schemes in terms of CRS size, key size and ciphertext size. We use D to denote the circuit depth, $\mathbf{x}$ to denote the attribute and C for the circuit describing the policy. For RAM programs and Turing machines, we denote the running time by T. We use M to denote the description of the Turing machine associated with the policy. We use SLWE to denote succinct LWE [Wee24], TLWE to denote tensor LWE [Wee22], ELWE to denote public-coin evasive LWE [Wee22], ELWE† to denote private-coin evasive LWE [VWW22, BÜW24] DLWE to denote decomposed LWE (this work). All terms hide $\text{poly}(\lambda)$ multiplicative factors.

the reader to the technical sections for precise statements.

2.1 A Primer on Succinct Oblivious Tensor Evaluation

Our starting point is a recent construction of succinct oblivious tensor evaluation protocol (OTE) [AMR25], which we recall in the following. A succinct OTE is a one-message primitive between Alice and Bob. Alice holds a long input vector $\mathbf{x} \in \mathbb{Z}_q^L$, whereas Bob holds a shorter secret vector $\mathbf{y} \in \mathbb{Z}_q^k$; their goal is to derive an additive secret-sharing of the tensor product $\mathbf{x} \otimes \mathbf{y}$ while preserving the privacy of $\mathbf{y}$ (but not necessarily that of $\mathbf{x}$), using a single round of simultaneous interaction. In [AMR25] it is shown how to construct succinct OTE with communication complexity $k \cdot \text{poly}(\lambda, \log L)$, CRS size is $\text{poly}(\lambda, \log L)$ and, importantly, where the parties are essentially required to perform only linear operations. Given that we are going to use some structural properties of their scheme, we present a somewhat detailed description below. Throughout this discussion, we shall think of the string $\mathbf{x}$ to be low-norm, e.g., it consists of bits.

Succinct OTE: Alice's Digest. Let $n := k \cdot \log q$. The CRS consists of two matrices:

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_0, & \dots & \mathbf{A}_{t-1}\end{bmatrix} \in \mathbb{Z}_q^{k \times (t \cdot m)} \qquad\qquad \mathbf{B} = \begin{bmatrix}\mathbf{B}_0, & \dots & \mathbf{B}_{t-1}\end{bmatrix} \in \mathbb{Z}^{m \times (t \cdot n)}$$

where $\mathbf{A}$ is sampled uniformly, whereas $\mathbf{B}$ is sampled from a low-norm Gaussian distribution. Given this CRS, then we can define a natural hash function $\mathsf{H}_{\mathbf{B}}$ mapping vectors $\mathbf{v} \in \mathbb{Z}_q^t$ intro shorter vectors as:

$$\mathsf{H}_{\mathbf{B}}(\mathbf{v}) = \mathsf{vec}(v_0 \cdot \mathbf{B}_0 + \cdots + v_{t-1} \cdot \mathbf{B}_{t-1}) = \mathsf{vec}(\mathbf{B} \cdot (\mathbf{v} \otimes \mathbf{I}_n))$$

where $\mathsf{vec}(\cdot)$ is the vectorisation of a matrix. For an appropriate choice of parameters, this function is compressing with factor 2. Alice's hash function is then defined to be the binary tree of depth $r := \log L$. That is, for every level $i \in [r]$, let $\mathbf{x}_{i,j} \in \mathbb{Z}_q^t$ be the input to the j-th execution of the basic compressing function $\mathsf{H}_{\mathbf{B}}$ on that level. Define:

$$\mathbf{d}_{i,j} := \mathsf{H}_{\mathbf{B}}(\mathbf{x}_{i,j}) = \mathsf{vec}(\mathbf{B} \cdot (\mathbf{x}_{i,j} \otimes \mathbf{I}_t)) = \mathsf{vec}(\mathbf{X}_{i,j})$$

and $\mathbf{d}_i = (\mathbf{d}_{i,j})_{j \in [2^{r-i-1}]}$. Alice's digest simply consist of the root of the tree. Since $\mathbf{B}$ is a low-norm matrix, we can assume that $\mathbf{x}_{i,j}$ remains low-norm as i increases.

Succinct OTE: Bob's Message. On the other front, Bob samples r random vectors $\mathbf{s}_1, \dots, \mathbf{s}_r \xleftarrow{\$} \mathbb{Z}_q^k$ and, for every $i \in [r]$ and $h \in [t]$, he sends

$$\mathbf{z}_h^{(i)} := \mathbf{s}_{i+1}^\intercal \cdot \mathbf{A}_h \cdot \mathbf{B} + \mathbf{e}_h^{(i)\intercal} + \mathbf{u}_h^\intercal \otimes \mathbf{s}_i^\intercal \otimes \mathbf{g}^\intercal$$

where $\mathbf{g}$ is the one-dimensional gadget, $\mathbf{e}_h^{(i)}$ is sampled from a low-norm Gaussian distribution and $\mathbf{s}_0 := \mathbf{y}$. First of all, let us notice that, under the standard LWE assumption, all of these samples look random, so indeed $\mathbf{y}$ is computationally hidden. Furthermore, the total size of Bob's message is logarithmic in L. So what is left to be shown is how Alice and Bob can obtain an additive secret sharing of $\mathbf{x} \otimes \mathbf{y}$, given the messages that they exchange.

Succinct OT: The Core Idea. Let us start by observing that Alice and Bob hold a trivial additive secret sharing of $\mathbf{d} \otimes \mathbf{s}_r \otimes \mathbf{g}$: Indeed Bob knows both $\mathbf{d}$ and $\mathbf{s}_r$, whereas $\mathbf{g}$ is a fixed vector, so we can set Alice's share to be zero.

We then proceed inductively, assuming that Alice and Bob hold an additive secret sharing of $\mathbf{d}_{i,j} \otimes \mathbf{s}_i \otimes \mathbf{g}$, for every $j \in [2^{r-i-1}]$, we describe how they can derive a (noisy) additive secret-sharing of $\mathbf{d}_{i-1,j} \otimes \mathbf{s}_{i-1} \otimes \mathbf{g}$ for every $j \in [2^{r-i}]$. In other words, we show how to "climb one level up" the Merkle tree. A pictorial description of this procedure is shown in Fig. 1. Observe that Alice can compute:

$$\begin{aligned}
\mathbf{c}_{i,j,h}^\intercal &= \mathbf{z}_h^\intercal \cdot (\mathbf{x}_{i,j} \otimes \mathbf{I}_t) \\
&= \mathbf{s}^\intercal \cdot \mathbf{A}_h \cdot \mathbf{B} \cdot (\mathbf{x}_{i,j} \otimes \mathbf{I}_t) + \mathbf{e}_h^\intercal \cdot (\mathbf{x}_{i,j} \otimes \mathbf{I}_t) + x_{i,j,h} \cdot (\mathbf{s}_{i-1}^\intercal \otimes \mathbf{g}^\intercal) \\
&= \mathbf{s}^\intercal \cdot \underbrace{\mathbf{A}_h \cdot \mathbf{X}_{i,j}}_{\text{matrix}} + \underbrace{\mathbf{e}_h^\intercal \cdot (\mathbf{x}_{i,j} \otimes \mathbf{I}_t)}_{\text{noise}} + x_{i,j,h} \cdot (\mathbf{s}_{i-1}^\intercal \otimes \mathbf{g}^\intercal)
\end{aligned}$$

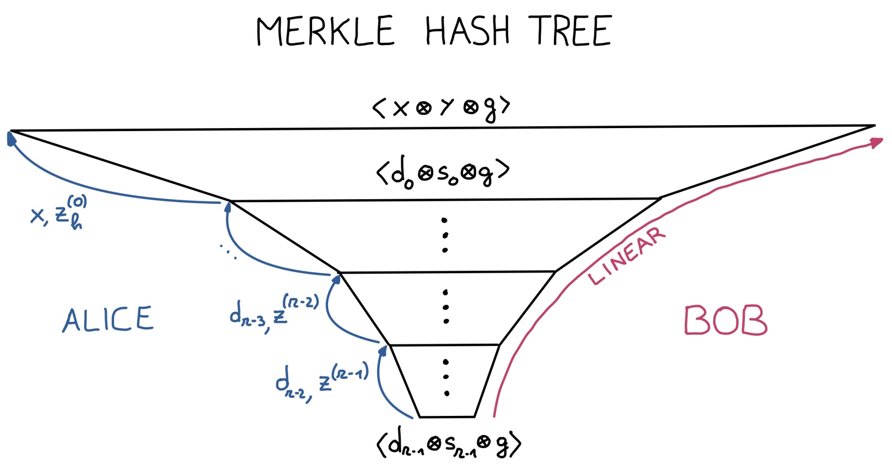

Figure 1: Depiction of the tree climbing procedure. Alice and Bob derive their shares of $\mathbf{x} \otimes \mathbf{y}$ by expanding the initial secret-sharing of $\mathbf{d} \otimes \mathbf{s}_r \otimes \mathbf{g}$. Notice that Bob's computation is linear, whereas Alice relies on $\mathbf{z}^{(i)}$ (denoting the concatenation of $\mathbf{z}_0^{(i)}, \ldots, \mathbf{z}_{t-1}^{(i)}$) to climb from the i-th level to the above one.

where $x_{i,j,h}$ denotes the h-th entry of $\mathbf{x}_{i,j}$. In other words, we can think of the above term as an LWE encryption of $x_{i,j,h} \cdot (\mathbf{s}_{i-1}^\intercal \otimes \mathbf{g}^\intercal)$. Next, since $\mathbf{s}^\intercal \cdot \mathbf{A}_h \cdot \mathbf{X}_{i,j}$ is a bilinear function of $\mathbf{s}$ and $\mathbf{d}_{i,j}$, there exists a *binary* matrix $\mathsf{Lin}(\mathbf{A}_h)$ such that:

$$(\mathbf{d}_{i,j}^\intercal \otimes \mathbf{s}_i^\intercal \otimes \mathbf{g}^\intercal) \cdot \mathsf{Lin}(\mathbf{A}_h) = \mathbf{s}_i^\intercal \cdot \mathbf{A}_h \cdot \mathbf{X}_{i,j}.$$

The rest is linear algebra: By multiplying the (noisy) shares of $\mathbf{d}_{i,j} \otimes \mathbf{s}_i \otimes \mathbf{g}$ by the low-norm matrix $-\mathsf{Lin}(\mathbf{A}_h)$, Alice and Bob derive a (noisy) secret-sharing of $-\mathbf{s}_i^\intercal \cdot \mathbf{A}_h \cdot \mathbf{X}_{i,j}$. Once Alice adds $\mathbf{c}_{i,j,h}$ to her share, the parties obtain a noisy secret-sharing of $x_{i,j,h} \cdot (\mathbf{s}_{i-1} \otimes \mathbf{g})$. By rearranging all of these, the parties obtain a noisy additive secret-sharing of $\mathbf{d}_{i-1,j} \otimes \mathbf{s}_{i-1} \otimes \mathbf{g}$ for every $j \in [2^{r-i}]$. The bottom of the recursion yields a (noisy) additive secret sharing of $\mathbf{x} \otimes \mathbf{y}$, as desired.

2.2 Compressing Lattice Encodings

Our main idea is to connect the above OTE procedure with the lattice encoding scheme of [BGG+14], to create a compressed version of it. A BGG^+ encoding of a vector $\mathbf{x}$ with respect to the public matrix $\mathbf{M}$ and secret $\mathbf{s}$ consists of:

$$\mathbf{c}^\intercal := \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{x}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal$$

where $\mathbf{e}$ is a noise term. Notice that if $\mathbf{x}$ is independent of $\mathbf{M}$, the vector $\mathbf{c}$ looks random under LWE, so $\mathbf{s}$ remains secret. These encodings are extremely useful in cryptography [BGG+14, GVW15, QWW18, LV22] due to their key-homomorphic properties: Given knowledge of $\mathbf{x}$ (but not of $\mathbf{s}$) and any circuit C, we can transform $\mathbf{c}$ into an encoding of $C(\mathbf{x})$ under the same secret $\mathbf{s}$. Specifically, we can derive:

$$\mathbf{c}_C^\intercal := \mathbf{s}^\intercal \cdot (\mathbf{M}_C + C(\mathbf{x})^\intercal \otimes \mathbf{G}) + \mathbf{e}_C^\intercal,$$

here $\mathbf{M}_C$ is a public matrix derived from just $\mathbf{M}$ and C, and $\mathbf{e}_C$ is a noise term. As a first step, we will use the above OTE protocol to *compress* such encodings to constant size, in particular independent of the size of the input $|\mathbf{x}| = L$. The idea is simple: Instead of directly encoding $\mathbf{x}$, we encode its OTE digest $\mathbf{d}$. Thanks to the linear properties of the OTE protocol, given knowledge of $\mathbf{x}$, we will manage to expand the compressed encoding $\mathbf{h}$ into a standard BGG^+ encoding of $\mathbf{x}$. At that point, we can perform all the desired homomorphic operations. Finally, we can re-compress the result to obtain a compressed encoding of the updated state.

Compressed Key-Homomorphic Encodings. Putting the above plan into action, we see that a BGG^+ encoding of $\mathbf{d}$ under the secret $\mathbf{s}$ and key $\mathbf{M}$ consists of:

$$\begin{aligned}\mathbf{h}^\intercal &= \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal \\ &= \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{I}_k \otimes \mathbf{g}^\intercal) + \mathbf{e}^\intercal \\ &= \mathbf{s}^\intercal \cdot \mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{s}^\intercal \otimes \mathbf{g}^\intercal + \mathbf{e}^\intercal.\end{aligned}$$

In other words, $\mathbf{h}$ can be viewed as an encryption of $\mathbf{d}^\intercal \otimes \mathbf{s}^\intercal \otimes \mathbf{g}^\intercal$. If $\mathbf{s}_r = \mathbf{s}$, this coincides with Bob's share at the beginning of the tree climb. We recall that in order to climb the tree, Bob just needs to perform linear operations on his share, and all of these are described by low-norm matrices. Thus, thanks to the linear properties of these ciphertexts, we can easily derive an encryption of Bob's share of the tree top.

Now, imagine that we augment the compressed encodings with Bob's OTE message. Since $\mathbf{x}$ is public, we can compute Alice's share of the tree top in the plain, and add it to the encryption of Bob's share. In this way, we obtain:

$$\mathbf{c}^\intercal = \mathbf{s}^\intercal \cdot \overline{\mathbf{M}} + \mathbf{x}^\intercal \otimes \mathbf{s}_0^\intercal \otimes \mathbf{g}^\intercal + \bar{\mathbf{e}}^\intercal$$

where $\overline{\mathbf{M}}$ depends only on $\mathbf{M}$ and the CRS of the OTE, and $\mathbf{e}$ has low norm. This is almost what we want, except that we have an encoding under a different secret $\mathbf{s}_0$. What we can do however, is simply set $\mathbf{s}_0 := \mathbf{s}$ so that $\mathbf{c}$ becomes a proper BGG^+ encoding of $\mathbf{x}$.

The savvy reader might notice that this introduces a circularity: The vectors $(\mathbf{z}_h^{(i)})_{h\in[t]}$ protect the privacy of $\mathbf{s}_i$ under that of $\mathbf{s}_{i+1}$. That creates a chain of dependencies where the privacy of $\mathbf{s}_0 = \mathbf{s}$ is protected under that of $\mathbf{s}_r = \mathbf{s}$. This is exactly where the *decomposed LWE* assumption (Section 1.2) enters into the picture, allowing us to prove the pseudorandomness of our compressed encodings.

Key-Homomorphic Computations for RAM. Now that we have shown how to expand BGG^+ encodings back to their original form, it is clear that we can do key-homomorphic computation as in the original scheme. It turns out that we can do even better: Note that deriving an encoding for a single entry of $\mathbf{x}$, let's say x_ℓ, does *not* require us to re-expand the whole vector. We just need to perform $\mathrm{poly}(\lambda, \log L)$ computations, climbing up the branch of the tree that leads to the ℓ-th leaf. Even more importantly, for this operation, we do not need to know the whole vector $\mathbf{x}$, but just the co-path towards the ℓ-th leaf.

Imagine we have a BGG^+ encoding of x'_ℓ and we want to substitute x_ℓ with x'_ℓ in our compressed encoding $\mathbf{h}$. Even this computation can be performed in time $\mathrm{poly}(\lambda, \log L)$. This because our hash function is $\mathbb{Z}_q$-linear: The updated hash will be $\mathbf{d}' := \mathbf{d} + \mathsf{Hash}(y \cdot \mathbf{u}_\ell)$ where $y := x'_\ell - x_\ell$ and $\mathbf{u}_\ell$ denotes the ℓ-th element in the standard basis of $\mathbb{Z}_q^L$. It is easy to see that computing $\mathsf{Hash}(y \cdot \mathbf{u}_\ell)$ requires only $\mathrm{poly}(\lambda, \log L)$ computations. Indeed, all the elements in the co-path to the ℓ-th leaf will be zeros. (More generally, we can see that any vector $\mathbf{x}$ can be hashed in time linear in its Hamming weight.) In conclusion, to perform the update, we just need to retrieve a BGG^+ encoding of x_ℓ as we have described above. Then, it is just a matter of applying the regular homomorphic operations (for circuits) to obtain a BGG^+ encoding of $\mathbf{d}'$.

The two procedures we have just described allow us to homomorphically perform reads and writes on the RAM tape $\mathbf{x}$ with $\mathrm{poly}(\lambda, \log L)$ complexity. Together with the regular BGG^+ homomorphic properties, this completes the description of an algorithm to homomorphically evaluate a RAM program P on $\mathbf{x}$ for T steps in time $T \cdot \mathrm{poly}(\lambda, \log L)$, where L denotes the tape size. In other words, we have obtained *compressed, key-homomorphic encodings for RAM.*

We can summarise this fact in the following new *key equation*: There exist low-norm matrices $\mathbf{H}_{P,T,\mathbf{x}}$ and $\mathbf{K}_{P,T}$, computable in time $T \cdot \mathrm{poly}(\lambda, \log L)$ and space $\mathrm{poly}(\lambda, \log L)$, such that:

$$\begin{bmatrix}\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}, & \mathbf{A}_0 \cdot \mathbf{B}, & \dots & \mathbf{A}_{t-1} \cdot \mathbf{B}\end{bmatrix} \cdot \mathbf{H}_{P,T,\mathbf{x}} = \mathbf{M}_{P,T} + \mathbf{d}'^\intercal \otimes \mathbf{G},$$

where $\mathbf{M}_{P,T} := \mathbf{M} \cdot \mathbf{K}_{P,T}$, $\mathbf{d} = \mathsf{Hash}(\mathbf{x})$ and $\mathbf{d}' = \mathsf{Hash}(P^T(\mathbf{x}))$. In particular, $\mathbf{K}_{P,T}$ will be derived from $\mathbf{M}$, the public parameters, the RAM program P and T. On the other hand, $\mathbf{H}_{P,T,\mathbf{x}}$ will also depend on $\mathbf{x}$. Above, $P^T(\mathbf{x})$ denotes the tape after the execution of T steps of P on input $\mathbf{x}$. For more discussion on our key-homomorphic encodings for RAM, we refer to Section 5.

2.3 Applications

We now describe how our newly constructed key-homomorphic encodings are useful to build well-known cryptographic primitives.

Laconic Function Evaluation. The key-homomorphic encodings for RAM directly yield and AB-LFE and LFE for RAM, following the same approach as [QWW18]. Moreover, since the encodings are compressed, the constructions will have rate-1 in the input $\mathbf{x}$. In order to prevent the noise in the encodings from growing proportionally to T (and therefore avoid any cap on the running time of the computation), we can rely on the bootstrapping technique of [HLL23]. This, however, requires us to rely on a circular version of decomposed LWE, where we additionally provide the adversary with a GSW encryption [GSW13] of the secret $\mathbf{s}$ using $\mathbf{s}$ itself as a secret key. Finally, we highlight that we can make our LFE rate-1 in the output, following the approach of [AMR25].

We expand on the properties of our AB-LFE, given that this will be useful for our next application. The scheme allows the server to generate a digest $h_{P,T}$ for T steps of the RAM program P in time $T \cdot \text{poly}(\lambda, \log L)$ and space $\text{poly}(\lambda, \log L)$. Here, L denotes the memory complexity of P. Given $h_{P,T}$, the client can generate an encryption of its message μ using an attribute $\mathbf{x}$. This consists of the pair $(\mathbf{x}, E)$, where the compressed encoding E can be generated by a circuit of size $\text{poly}(\lambda, \log L)$, on input the AB-LFE digest $h_{P,T}$, the message μ and an OTE hash of $\mathbf{x}$. We recall that the OTE hash can be derived in time $|\mathbf{x}| \cdot \text{poly}(\lambda, \log L)$. Upon decryption, the server recovers μ only if $P^T(\mathbf{x})$ satisfies some condition specified by the client (typically, if the k-th element on the tape coincides with some target value α). We can easily modify our AB-LFE to multi-bit messages μ, so that decryption releases the substring of μ corresponding to $P^T(\mathbf{x})$: For every j, the server learns the $(2j)$-th bit of μ or the $(2j+1)$-th one, depending on whether the j-th element in $P^T(\mathbf{x})$ is 0 or 1. For more discussion on RAM AB-LFE and LFE, we refer to Section 6.

Witness Encryption for P. A witness encryption for P is a "privacy-free" version of succinct randomised encodings. In more details, it corresponds to a primitive in which a message μ is encrypted under a RAM program P, a runtime T and an input tape $\mathbf{x}$. For every j, decryption will recover the $(2j)$-th bit in μ or the $(2j+1)$-th one depending on whether the j-th element of $P^T(\mathbf{x})$ is 0 or not. Without any further restrictions, building this primitive would be trivial: The encryptor just evaluates $P^T(\mathbf{x})$ and sends the substring of μ corresponding to the result. This solution would, however, require the encryptor to run in time proportional to T. In the scheme we are going to present, encryption will instead run in time $\text{poly}(\lambda, \log T, \log L)$, where L denotes the space complexity of P.

Our initial observation is that our AB-LFE scheme for RAM allows us to reduce the problem to one in which the description size and the space complexity of P are both upper-bounded by a fixed polynomial in λ: Our ciphertext could consist of a garbled circuit $\tilde{C}$ that, on input the AB-LFE digest h of the universal RAM, outputs an AB-LFE encryption of μ. Specifically, $\tilde{C}$ will generate a compressed encoding using h as AB-LFE digest and $(\mathbf{x}, P)$ as attribute (we hardcode the corresponding OTE hash). The AB-LFE decryption would thus produce the substring of μ corresponding to $P^T(\mathbf{x})$.

Notice that garbling $\tilde{C}$ takes time $\text{poly}(\lambda, \log L, \log T)$, whereas the generation of h could be performed using a RAM program D (with $\text{poly}(\lambda)$ description size) in time $T_0 = T \cdot \text{poly}(\lambda, \log L)$ and space $\text{poly}(\lambda, \log L)$. This suggest an outline to speed-up the encryption time, following the example of [GKP+13b]: Instead of directly providing the labels of $\tilde{C}$ associated with h, we encrypt all input labels using witness encryption for D. This ensures that only the labels for h get released. Given the low space and description complexity of D, we henceforth ignore the overhead dictated by these two quantities. Alas, the runtime of the algorithm is still at least T (in fact has increased), so we need to do something different.

Amortising Computation. To actually speed-up the runtime of the encryptor, we rely on the recursive technique from [AMZ24, BG25]: First, we consider the evaluation of the universal RAM U on D; suppose that this requires $\overline{T} = T \cdot \text{poly}(\lambda)$ steps. Instead of computing an AB-LFE digest for $\overline{T}$ steps of U, we just compute a digest h' for $\overline{T}/p$ steps. This can be generated in time $T/p \cdot \text{poly}(\lambda)$ and can be reused across p chained AB-LFE executions. Then, we use garbled circuits (as in [BG25]) to connect the output of one evaluation to the input of the next one.

Formally, the ciphertext consists of p garbled circuits, $\tilde{C}_0, \ldots, \tilde{C}_{p-1}$, along with the labels of $\tilde{C}_0$ corresponding to the inputs $\mathbf{y}_0 := D$ and h'. The i-th circuit $\tilde{C}_i$ takes as input a tape $\mathbf{y}_i$ and the AB-LFE digest h'. The output includes the labels of $\tilde{C}_{i+1}$ associated with h' and an AB-LFE encoding. This will be an encryption of the labels of the remaining input wires of $\tilde{C}_{i+1}$ (or the input labels of $\tilde{C}$, if $i = p - 1$). We use $\mathbf{y}_i$ as attribute and h' as digest. In other words, by decrypting the AB-LFE encoding produced by $\tilde{C}_i$, the decryptor will obtain the labels of $\tilde{C}_{i+1}$ associated with $\mathbf{y}_{i+1} := U^{\frac{T}{p}}(\mathbf{y}_i)$ and h'. Continuing in this way, for every i, it will recover the labels of $\tilde{C}$ associated with h. This process is visualised in Fig. 2.

Since D has $\mathrm{poly}(\lambda)$-bounded memory and description, all these garbled circuits can be computed in time $p \cdot \mathrm{poly}(\lambda)$. Thus, once again, the only computation that depends on T is the derivation of the AB-LFE digest h' and the corresponding labels for $\tilde{C}_0$. By picking a sufficiently large polynomial $p(\lambda)$, we can ensure that h' is derived by D in at most $T/2$ steps.

Achieving Full Efficiency and Full Security. We achieve the desired efficiency by recursing the above construction: We start by substituting the labels of $\tilde{C}_0$ associated with h' with an encryption of all labels using witness encryption for D. In this way, the encryption time decreases by a factor of 2: The bulk of the computation is still concentrated in the derivation of garbled circuit labels for an AB-LFE digest of U. The latter can be computed by D in less than $T/4$ steps. Thus, we can keep recursing, halving the encryption time at each iteration. We continue in this way until the encryption time becomes $\mathrm{poly}(\lambda, \log T)$. At each recursive step, the decryption time will increase by an ever smaller *additive* factor.

Finally, using our witness encryption for P, we can easily build fully succinct randomised encodings, again following the approach of [GKP+13b]: We just give an FHE encryption of the input, a garbled circuit for the decryption (where the FHE secret key is hardcoded) and a witness encryption (for P) of the input labels under the program that, on input the FHE ciphertext, homomorphically evaluates the desired computation. By using RAM-FHE [LMW23] instead of regular FHE, we can also obtain fully succinct randomised encodings for RAM. Furthermore, we can even make the construction rate-1 in both the input and output size using hybrid encryption and our LFE scheme for RAM (which has rate-1 in the output). For more discussion on succinct randomised encodings and witness encryption for P, we refer to Section 7.

Attribute-Based Encryption for RAM. Using our compressed encodings for RAM and the corresponding key equation, we can easily build optimal bounded-time ABE for RAM following the approach of Boneh et al. [BGG+14]: A ciphertext consists of a compressed encoding of the attribute, along with a Regev encryption of the message μ using the encoding secret $\mathbf{s}$ as randomness. Secret keys are associated with RAM programs P and running times T and correspond to (partial) lattice trapdoors. Decryption succeeds if the first element on the tape, after the execution of T steps of P, is 0. It would also be possible to make our construction unbounded-time by relying on the bootstrapping technique of [HLL23], however,

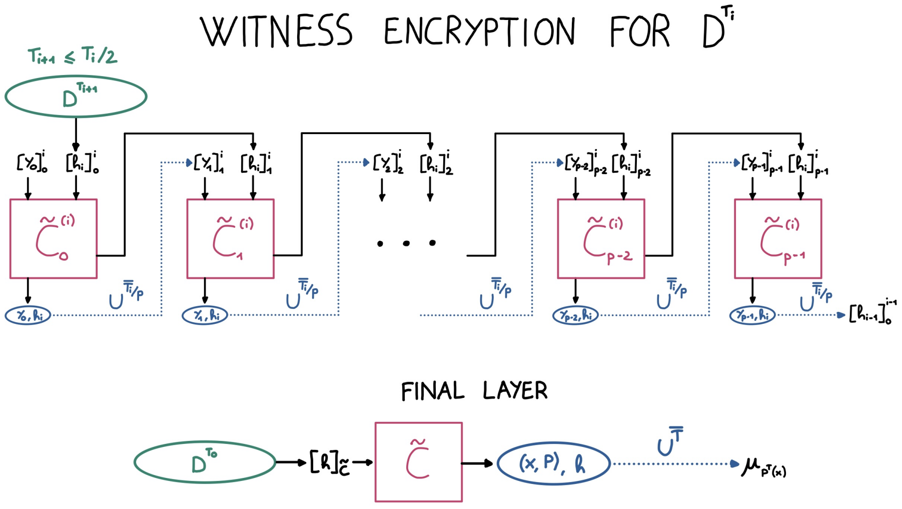

Figure 2: Depiction of our witness encryption for P, from the point of view of the decryptor. The top part represents the recursive construction for T_i steps of D. We use i to denote the index of the recursive iteration. We recall that $T_{i+1} \leq T_i/2$. The bottom part represents the final stage, where the plaintext $\mu_{P^T(\mathbf{x})}$ is finally retrieved. Magenta square boxes represent garbled circuits, all of them can be generated and evaluated in time $\mathrm{poly}(\lambda, \log L, \log T)$. We use square brackets to denote garbled circuit labels. Specifically, $[\mathbf{x}]^i_j$ denotes the labels associated with the input $\mathbf{x}$ for j-th garbled circuit $\tilde{C}^{(i)}_j$ at the i-th recursive iteration. We use $[\mathbf{x}]_{\tilde{C}}$ to denote the labels associated with the input $\mathbf{x}$ for the final layer garbled circuit $\tilde{C}$. Blue ovals represent AB-LFE ciphertexts. Within them, we specify the relative attribute and the underlying AB-LFE digest, but we omit the encrypted messages. We use blue dotted lines to denote the AB-LFE evaluation. Green ovals denote recursive call to witness encryption. Within them is specified the evaluated RAM program and the relative number of steps.

due to a lack of a (linear) key equation, this could be proven secure only by relying on the public-coin evasive LWE heuristic [Wee22].

Before continuing with register ABE, we highlight two essential properties of our ABE scheme. The first is that we do not need to know $\mathbf{x}$ to encrypt μ using $\mathbf{x}$ as an attribute: It is sufficient to know an OTE hash of $\mathbf{x}$. The second observation is that, if we hold a secret key for a program P that is allowed to decrypt, we can retrieve the plaintext even if we do not know the whole attribute $\mathbf{x}$: It is sufficient to know the Merkle hash co-path of all tape positions touched by P. For more discussion on RAM ABE, we refer to Section 8.

Register ABE. We show how to compile these ABE schemes to obtain register ABE. Let N be the number of parties. For every $i \in [N]$, the CRS includes the ABE secret key for the RAM program U_i that, on input $\mathbf{x}$ and

$$(r_j, P_j, t_j, s_j)_{j\in[N]}$$

where P_j is the description of a RAM program, checks whether $\mathsf{Ext}(s_i, r_i) = 0$ and, if that is the case, computes $P_i^{t_i}(\mathbf{x})$. Above, Ext denotes a strong randomness extractor.

The register ABE secret key of the i-th party corresponds to a large random string r_i. The corresponding public key consists of the OTE Merkle hash $\mathbf{d}_i$ of r_i along with the description of the i-th party program P_i, its running time t_i and an extractor seed s_i that is sampled at random conditioned on $\mathsf{Ext}(s_i, r_i) = 0$. The public keys can be aggregated by feeding their concatenation in the OTE Merkle hash. Thus, the master public key mpk has size independent of N. The i-th party's helper key hk_i will consist of the co-path for its public key. (See Fig. 3 for a pictorial description)

To encrypt a message μ under an attribute $\mathbf{x}$, we feed $\mathbf{x}$ and the master public key into the OTE Merkle hash. Notice that, if this is done as in Fig. 3, by the recursive structure of Merkle hash, the result $\mathbf{d}$ can be viewed as an OTE hash of:

$$(r_j, P_j, t_j, s_j)_{j\in[N]}, \mathbf{x}$$

The ciphertext consists of an ABE encryption of μ using $\mathbf{d}$ as compressed attribute. Observe that if the first element on $P_i^{t_i}(\mathbf{x})$ is 0, the i-th party will be able to derive μ using the ABE secret key for U_i. Indeed, it will know all the Merkle hash copaths of the positions touched by U_i. On the other hand, no other party could perform the same operation, as they would not have knowledge of r_i. To formally see this, notice that the i-th party only publishes a hash of r_i, so nobody could tell whether $\mathsf{Ext}(s_i, r_i) = 0$ or $\mathsf{Ext}(s_i, r_i) = 1$. In the latter case, ABE guarantees that μ remains private even given the secret key for U_i. For more discussion on register ABE for RAM, we refer to Section 9.

3 Preliminaries

Notation. For any integer $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{0, 1, \ldots, n-1\}$. We assume that all indices start from 0. We use $\delta_{i,j}$ to denote the Kroenecker delta, i.e. $\delta_{i,j} = 0$ unless

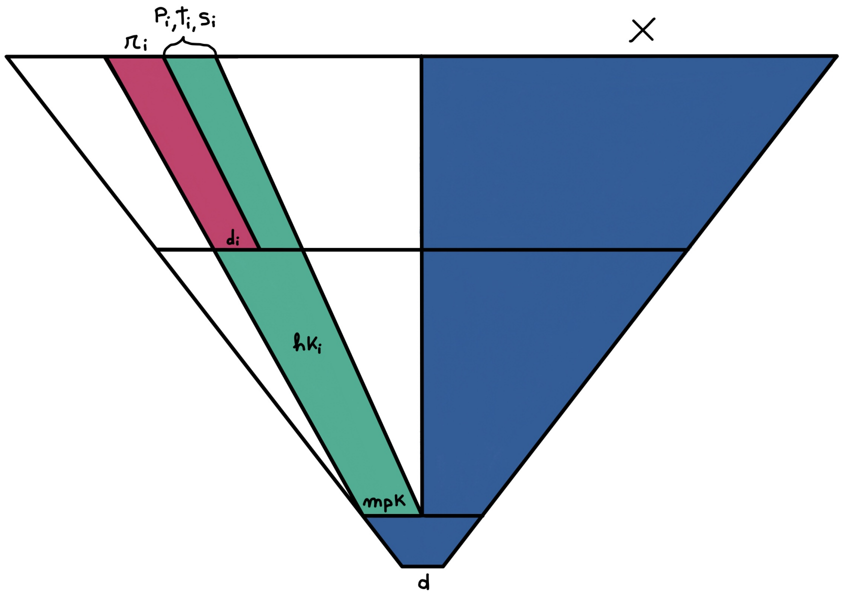

Figure 3: Depiction of the Merkle hash tree corresponding to the OTE digest $\mathbf{d}$. The blue area on the right and at the bottom are public. The green areas correspond on the left, correspond to the i-th party helper key and do not need to be private. The magenta area on the top left corresponds to the i-th party secret key. Notice that the RAM program U_i touches only the colored areas of the tree.

$i = j$; in that case $\delta_{i,j} = 1$. We denote vectors using lowercase bold font, e.g. $\mathbf{v}$. Matrices will be denoted using uppercase bold font, e.g. $\mathbf{A}$. We adopt the standard linear algebra convention in which vectors correspond to columns. We use the transposition symbol $^\intercal$ to denote row vectors. We define the Kroenecker product between two matrices $\mathbf{A}$ and $\mathbf{B}$ as

$$\mathbf{A} \otimes \mathbf{B} := \begin{bmatrix} a_{0,0} \cdot \mathbf{B} & \dots & a_{0,m-1} \cdot \mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n-1,0} \cdot \mathbf{B} & \dots & a_{n-1,m-1} \cdot \mathbf{B} \end{bmatrix}$$

where n and m respectively denote the number of rows and columns of $\mathbf{A}$ and $a_{i,j}$ denotes the entry in the i-th row and j-th column of $\mathbf{A}$. We recall that for any matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ where the number of columns of $\mathbf{A}$ and $\mathbf{B}$ respectively match the number of rows of $\mathbf{C}$ and $\mathbf{D}$, it holds that

$$(\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A} \cdot \mathbf{C}) \otimes (\mathbf{B} \cdot \mathbf{D}).$$

Moreover, $(\mathbf{A} \otimes \mathbf{B})^\intercal = \mathbf{A}^\intercal \otimes \mathbf{B}^\intercal$. We use $\mathsf{vec}(\mathbf{A})$ to denote the vectorisation of the matrix $\mathbf{A}$. Specifically, if $\mathbf{A} \in \mathbb{Z}^{n \times m}$, $\mathsf{vec}(\mathbf{A})$ consists of the $n \cdot m$-dimensional vector obtained by vertically concatenating $\mathbf{a}_0, \dots \mathbf{a}_{n-1}$ where $\mathbf{a}_i^\intercal$ denotes the i-th row of $\mathbf{A}$. We denote the $k \times k$ identity matrix by $\mathbf{I}_k$ and the ℓ_∞-norm by $\|\cdot\|_\infty$.

We denote the security parameter by λ. We say that a function $f : \mathbb{N} \to \mathbb{R}$ is negligible if $|f(\lambda)| = 2^{-\omega(\log \lambda)}$. We say that two ensembles of distributions $\mathcal{X} = (\mathcal{X}_\lambda)_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = (\mathcal{Y}_\lambda)_{\lambda \in \mathbb{N}}$ are $\varepsilon(\lambda)$-statistically indistinguishable (denoted by $\mathcal{X} \approx_\varepsilon \mathcal{Y}$) if the statistical distance

$$\mathsf{SD}(\mathcal{X}_\lambda, \mathcal{Y}_\lambda) := \sum_x |\Pr[\mathcal{X}_\lambda = x] - \Pr[\mathcal{Y}_\lambda = x]| \leq \varepsilon(\lambda)$$

except for finitely many values of $\lambda \in \mathbb{N}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are statistically indistinguishable (denoted by $\mathcal{X} \approx_s \mathcal{Y}$), if they are $\varepsilon(\lambda)$-statistically indistinguishable for a negligible function $\epsilon(\lambda) \leq \mathsf{negl}(\lambda)$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable (denoted by $\mathcal{X} \approx_c \mathcal{Y}$), if, there exists a negligible function $\mathsf{negl}(\lambda)$ such that, for every non-uniform probabilistic polynomial time (PPT) adversary $\mathcal{A}$, the advantage

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) := \left| \Pr\left[\mathcal{A}(1^\lambda, x) = 1 \middle| x \xleftarrow{\$} \mathcal{X}_\lambda \right] - \Pr\left[\mathcal{A}(1^\lambda, x) = 1 \middle| x \xleftarrow{\$} \mathcal{Y}_\lambda \right] \right| \leq \mathsf{negl}(\lambda).$$

All logarithms are in base 2, unless it is explicitly expressed. We use $\mathsf{GL}_n(R)$ to denote the set of $n \times n$ *invertible* matrices over the ring R. We use $\lceil \cdot \rfloor$ to denote rounding to the closest integer. In a similar way, we use $\lceil \cdot \rfloor_2$ to denote the most-significant bit. We denote the length of a string by $|\cdot|$. We use $\mathbb{Z}_q^{\leq \ell}$ to denote $\bigcup_{i=1}^{\ell} \mathbb{Z}_q^i$. We denote the XOR by $\oplus$.

RAM computations. A RAM program P with memory of size L consists of a circuit C that takes as input a state st and a value v. The output consists of of an updated state st', an updated value v' and a next location $\mathsf{next} \in [L]$. We can run the machine on a tape $\mathbf{x}$ containing L values: we start by setting $\mathsf{st} \leftarrow \perp$ and $\mathsf{loc} \leftarrow 0$. At each step, we compute $(\mathsf{st}', v', \mathsf{next}) \leftarrow C(\mathsf{st}, v)$ where v is the loc-th element on the tape. Then, we substitute v with v', we update the state $\mathsf{st} \leftarrow \mathsf{st}'$ and the location $\mathsf{loc} \leftarrow \mathsf{next}$. This procedure is repeated for as many times as needed. We denote the state of the tape after T steps by $P^T(\mathbf{x})$. We use $\mathsf{desc}(P)$ to denote the description of C as a binary string. We use $|P|$ to denote $|\mathsf{desc}(P)|$.

Theorem 3.1 (Universal RAM). *There exists a RAM U such that, for any other RAM program P with memory L, every tape $\mathbf{x}$ and runtime T, there exists $\overline{T} = T \cdot O(|P|)$ such that*

$$U^{\overline{T}}(\mathbf{x}, \mathsf{desc}(P)) = (P^T(\mathbf{x}), \mathsf{aux})$$

where aux *denotes some auxiliary material. Moreover, if P has memory of size L, U requires $L + O(|P|)$ memory.*

3.1 Lattices and Hard Problems

We define the gadget matrix $\mathbf{G} = \mathbf{I} \otimes \mathbf{g}^\intercal$, where $\mathbf{g}^\intercal$ is the row vector $\begin{bmatrix} 1 & 2 & \dots & 2^{\log q} \end{bmatrix}$. As customary, we define the function $\mathbf{G}^{-1}$ as the binary decomposition operator, satisfying the identity $\mathbf{G}\mathbf{G}^{-1}(\mathbf{x}) = \mathbf{x}$ for all $\mathbf{x}$.

Let $\mathcal{D}_\sigma$ denote the discrete Gaussian distribution with parameter σ. We recall the following standard fact about Gaussians.

Lemma 3.2 (Gaussian Flooding). *If X is sampled from D_σ then for any Y such that $|Y| \leq \tau$ and $\sigma/\tau \in \lambda^{\omega(1)}$, then the following distributions are statistically close:*

$$X \approx_s X + Y$$

where $X \xleftarrow{\$} \mathcal{D}_\sigma$.

The Gaussian pre-image sampling procedure [GPV08] allows one to condition a Gaussian sample on a arbitrary linear equation. We recall the main statistical property of pre-image sampling.

Lemma 3.3 (Gaussian Sampling). *Let $k = k(\lambda)$, $m = m(\lambda)$, $\sigma = \sigma(\lambda)$, and $q = q(\lambda)$ be integer parameters with $\sigma = \sqrt{k \cdot \log q} \cdot \omega(\sqrt{\log m})$ and $m \geq 2k \log q$. Then, there exist PPT algorithms* $\mathsf{TrapGen}$ *and* $\mathsf{PreSample}$ *such that the following two distributions are statistically indistinguishable:*

- *Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k \times m}$ and $\mathbf{x} \xleftarrow{\$} \mathcal{D}_\sigma^m$. Return $(\mathbf{A}, \mathbf{x}, \mathbf{A}\mathbf{x})$.*
- *Sample $(\mathbf{A}, \gamma) \xleftarrow{\$} \mathsf{TrapGen}(1^k, 1^m, q)$, $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^k$, and $\mathbf{t} \xleftarrow{\$} \mathsf{PreSample}(\gamma, \mathbf{v}, \sigma)$. Return $(\mathbf{A}, \mathbf{t}, \mathbf{v})$.*

Lemma 3.4 (Partial Trapdoors [CHKP10, ABB10, BGG+14]). *Let $k = k(\lambda)$, $m = m(\lambda)$, $\ell = \ell(\lambda)$, $\sigma = \sigma(\lambda)$, $B = B(\lambda)$ and $q = q(\lambda)$ be integer parameters with $\sigma = \max(\sqrt{k \cdot \log q}, B) \cdot \omega(\sqrt{\log m})$ and $m \geq 2k \log q$. Then, there exist PPT algorithms* $\mathsf{SampleRight}$ *and* $\mathsf{SampleLeft}$ *such that, for every (possibly computationally unbounded) adversary $\mathcal{A}$ that outputs triples $(\mathbf{S}, y, \mathsf{aux})$ where, $\mathbf{S} \in \mathbb{Z}_q^{m \times \ell}$, $\|\mathbf{S}\|_\infty \leq B \cdot (\sqrt{m} \cdot \ell)^{-1}$ and $y \neq 0$, the following two distributions are statistically indistinguishable:*

- *Sample $(\mathbf{A}, \gamma) \xleftarrow{\$} \mathsf{TrapGen}(1^k, 1^m, q)$ and $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^k$. Set $\mathbf{B} \leftarrow \mathbf{A} \cdot \mathbf{S} + y \cdot \mathbf{G}$, where $(\mathbf{S}, y, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathbf{A}, \mathbf{v})$ and $\mathbf{t} \xleftarrow{\$} \mathsf{SampleRight}(\gamma, \mathbf{B}, \mathbf{v}, \sigma)$. Return $(\mathbf{A}, \mathbf{t}, \mathbf{v}, \mathsf{aux})$.*

- *Sample* $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k\times m}$ *and* $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^k$. *Set* $\mathbf{B} \leftarrow \mathbf{A}\cdot\mathbf{S}+y\cdot\mathbf{G}$, *where* $(\mathbf{S}, y, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathbf{A}, \mathbf{v})$ *and* $\mathbf{t} \xleftarrow{\$} \mathsf{SampleLeft}(\mathbf{A}, \mathbf{S}, y, \mathbf{v}, \sigma)$. *Return* $(\mathbf{A}, \mathbf{t}, \mathbf{v}, \mathsf{aux})$.

Moreover, in both cases, it holds that

$$\begin{bmatrix}\mathbf{A}, & \mathbf{B}\end{bmatrix} \cdot \mathbf{t} = \mathbf{v}, \qquad \|\mathbf{t}\|_\infty \leq \sigma \cdot \sqrt{\lambda}.$$

We define the learning with errors (LWE) problem [Reg09] below. The works of [Reg09, AP09] showed that the hardness of the LWE problem follows from the worst-case quantum hardness SIVP and classical hardness of GapSVP.

Definition 3.5 (Learning with Errors)**.** *Let $k = k(\lambda)$, $m = m(\lambda)$, $\sigma = \sigma(\lambda)$, and $q = q(\lambda)$ be integer parameters. The LWE assumption postulates the the following distributions are computationally indistinguishable:*

$$\{\mathbf{A}, \mathbf{s}^\intercal\mathbf{A} + \mathbf{e}^\intercal \pmod q\} \approx_c \{\mathbf{A}, \mathbf{u}^\intercal\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k\times m}$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\mathbf{e} \xleftarrow{\$} \mathcal{D}_\sigma^m$, *and* $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m$.

We also recall the recently introduced t-succinct LWE problem [Wee24]. Loosely speaking, it assumes that the LWE problem remains hard, even when given a trapdoor (in the sense of [GPV08]) for a related matrix.

Definition 3.6 (t-Succinct LWE)**.** *Let $k = k(\lambda)$, $m = m(\lambda)$, $\sigma = \sigma(\lambda)$, and $q = q(\lambda)$ be integer parameters, with $m \geq 2k\log q$. Let $n := k \cdot \log q$. The t-succinct LWE assumption postulates that the following distributions are computationally indistinguishable:*

$$\{\mathbf{A}, \mathbf{s}^\intercal\mathbf{A} + \mathbf{e}^\intercal, \mathbf{W}, \mathbf{T}\} \approx_c \{\mathbf{A}, \mathbf{u}^\intercal, \mathbf{W}, \mathbf{T}\}$$

where $(\mathbf{A}, \gamma) \xleftarrow{\$} \mathsf{TrapGen}(1^k, 1^m, q)$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\mathbf{e} \xleftarrow{\$} \mathcal{D}_\sigma^m$, $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m$, *and*

$$\mathbf{W} = \begin{bmatrix}\mathbf{W}_0 \\ \vdots \\ \mathbf{W}_{t-1}\end{bmatrix} \qquad \mathbf{T} = \begin{bmatrix}\mathbf{T}_{0,0} & \cdots & \mathbf{T}_{0,t-1} \\ \vdots & \ddots & \vdots \\ \mathbf{T}_{t,0} & \cdots & \mathbf{T}_{t,t-1}\end{bmatrix}$$

for $\mathbf{T}_{t,0}, \ldots, \mathbf{T}_{t,t-1} \xleftarrow{\$} \mathcal{D}_\sigma^{m\times n}$, $\mathbf{W}_0, \ldots, \mathbf{W}_{t-1} \xleftarrow{\$} \mathbb{Z}_q^{k\times m}$ *and*

$$\forall i, j \in [t] \qquad \mathbf{T}_{i,j} \xleftarrow{\$} \mathsf{PreSample}(\gamma, \delta_{i,j}\cdot\mathbf{G} - \mathbf{W}_i \cdot \mathbf{T}_{t,j}, \sigma).$$

In particular, it holds that

$$\begin{bmatrix}\mathbf{I}_t \otimes \mathbf{A}, & \mathbf{W}\end{bmatrix} \cdot \mathbf{T} = \mathbf{I}_t \otimes \mathbf{G}.$$

3.2 Key-Homomorphic Encodings à la BGG+ and HLL

In [BGG+14], Boneh *et al.* introduced LWE-based key-homomorphic encodings. Each encoding $\mathbf{c}$ is associated with a string $\mathbf{x} \in \mathbb{Z}_q^\ell$ for some $\ell \in \mathbb{N}$ and is generated using a secret $\mathbf{s} \in \mathbb{Z}_q^k$ and a public matrix $\mathbf{A} \in \mathbb{Z}_q^{k\times(\ell\cdot n)}$ where $n = k \cdot \log q$:

$$\mathbf{c}^\intercal = \mathbf{s}^\intercal \cdot (\mathbf{A} + \mathbf{x}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal.$$

Above $\mathbf{e}$ denotes a low-norm term. When we generate a fresh encoding the latter is usually sample from the discrete Gaussian $\mathcal{D}_{\sigma}^{n\cdot\ell}$. Observe that, for any public vector $\mathbf{v}$, we can view $\mathbf{0}$ as an encoding of $\mathbf{v}$ with relation to the secret $\mathbf{s}$ and the public matrix $-\mathbf{v}^{\intercal} \otimes \mathbf{G}$.

These encodings are homomorphic in the sense that, given an encoding of $\mathbf{x}$ under $\mathbf{s}$ and $\mathbf{A}$ and a function f, we can derive an encoding of $f(\mathbf{x})$ under the same secret $\mathbf{s}$ and another matrix $\mathbf{A}_f$. Crucially, $\mathbf{A}_f$ is publicly derivable and is independent of both $\mathbf{x}$, $\mathbf{s}$ and the noise $\mathbf{e}$. This result is formalised in the following theorem.

Theorem 3.7 (Key Equation [BGG+14]). *There exists deterministic polynomial time algorithms* EvalK *and* EvalH *such that, for every matrix* $\mathbf{A} \in \mathbb{Z}_q^{k\times(\ell\cdot n)}$*, input* $\mathbf{x} \in \mathbb{Z}_q^{\ell}$ *and algebraic circuit* f*, we have*

$$(\mathbf{A} + \mathbf{x}^{\intercal} \otimes \mathbf{G}) \cdot \mathbf{H}_{f,x} = \mathbf{A} \cdot \mathbf{K}_f + f(\mathbf{x})^{\intercal} \otimes \mathbf{G},$$
$$\|\mathbf{H}_{f,\mathbf{x}}\|_\infty \le \mathrm{poly}(\lambda^d, B^d)$$

where d *denotes the multiplicative depth of* f*,* B *denotes an upper bound on the magnitude of the wire values in* $f(\mathbf{x})$ *and*

$$\mathbf{K}_f \leftarrow \mathsf{EvalK}(f, \mathbf{A}), \qquad \mathbf{H}_{f,\mathbf{x}} \leftarrow \mathsf{EvalK}(f, \mathbf{A}, \mathbf{x}).$$

In particular, notice that $\mathbf{c}^{\intercal} \cdot \mathbf{H}_{f,\mathbf{x}} = \mathbf{s}^{\intercal} \cdot (\mathbf{A}_f + f(\mathbf{x})^{\intercal} \otimes \mathbf{G}) + \mathbf{e}_f^{\intercal}$ where $\mathbf{A}_f := \mathbf{A} \cdot \mathbf{K}_f$ and $\|\mathbf{e}_f\|_\infty \le \mathrm{poly}(\lambda^d, B^d) \cdot \|\mathbf{e}\|_\infty$. We therefore obtained an encoding of $f(\mathbf{x})$ under $\mathbf{s}$ and $\mathbf{A}_f$ but with higher noise magnitude. Unfortunately, if the computation has particularly high depth, the noise may end up reaching magnitude proportional to q. This is often particularly problematic as it prevents us from efficiently extract information from the result of the computation. For this reason, Hsieh, Lin and Luo [HLL23] recently introduced a bootstrapping procedure to reduce the noise in the encodings. Their techniques intrinsically rely on a circular version of LWE.

Theorem 3.8 (Encoding Evaluation with Bootstrapping [HLL23]). *Suppose that* $q = q(\lambda)$*,* $k = k(\lambda)$*,* $\sigma = \sigma(\lambda)$*,* $\sigma' = \sigma'(\lambda)$ *and* $\overline{\sigma} = \overline{\sigma}(\lambda)$ *are such that* $\sqrt{\lambda} \cdot \sigma \le B_{\mathsf{max}}$ *and* $B_{\mathsf{bst}} \le B_{\mathsf{max}}$ *where*

$$B_{\mathsf{bst}} = (\sigma' + \overline{\sigma}) \cdot 2^{O(\log^5 \lambda)}, \qquad B_{\mathsf{max}} \le 2^{\frac{1}{3}\log q - \omega(\log \lambda)}.$$

Then, there exist deterministic polynomial time algorithms CEval *and* CEvalK *such that, for all inputs* $\mathbf{x} \in \mathbb{Z}_q^{\ell}$*, algebraic circuit* f *and noise term* $\mathbf{e} \in \mathbb{Z}_q^{\ell\cdot n}$ *such that* $\|\mathbf{e}\|_\infty \le B_{\mathsf{max}}$*, the following holds with probability* $1 - \mathsf{negl}(\lambda)$*,*

$$\mathsf{CEval}(f, \mathbf{c}, \mathbf{A}, \mathbf{x}, \mathsf{c}_{\mathsf{circ}}, \mathbf{A}_{\mathsf{circ}}, \mathbf{S}) = \mathbf{s}^{\intercal}(\mathbf{A}_{\mathsf{bts}} + f(\mathbf{x})^{\intercal} \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{bts}}^{\intercal},$$
$$\|\mathbf{e}_{\mathsf{bts}}\|_\infty \le B_{\mathsf{bst}}$$

where

$$\begin{aligned}\mathbf{A}_{\mathsf{bts}} &= \mathsf{CEvalK}(f, \mathbf{A}, \mathbf{A}_{\mathsf{circ}})\\ \mathbf{c}^{\intercal} &= \mathbf{s}^{\intercal} \cdot (\mathbf{A} + \mathbf{x}^{\intercal} \otimes \mathbf{G}) + \mathbf{e}^{\intercal}\\ \mathbf{S} &= \begin{bmatrix} \mathbf{W} \\ \mathbf{s}^{\intercal} \cdot \mathbf{W} + \overline{\mathbf{e}}^{\intercal} \end{bmatrix} - \mathsf{Bits}(\mathbf{s})^{\intercal} \otimes \mathbf{I}_{k+1} \otimes \mathbf{g}^{\intercal}\end{aligned}$$

$$\mathbf{c}_{\mathsf{circ}}^{\mathsf{T}} = \mathbf{s}^{\mathsf{T}} \cdot (\mathbf{A}_{\mathsf{circ}} + \mathsf{Bits}(\mathbf{S})^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{circ}}{}^{\mathsf{T}}$$

and the probability is taken over the randomness of

$$\begin{array}{ccc} \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k\times(\ell\cdot n)} & \mathbf{A}_{\mathsf{circ}} \xleftarrow{\$} \mathbb{Z}_q^{k\times(n\cdot(n+\log q)^2)} & \mathbf{W} \xleftarrow{\$} \mathbb{Z}_q^{k\times(n^2+n\cdot\log q)} \\ \mathbf{s} \xleftarrow{\$} \mathcal{D}_\sigma^k & \overline{\mathbf{e}} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{n^2+n\cdot\log q} & \mathbf{e}_{\mathsf{circ}} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{n\cdot(n+\log q)^2}. \end{array}$$

Observe that in the above result, $\mathbf{S}$ is essentially a GSW encryption [GSW13] of $\mathbf{s}$ using $\mathbf{s}$ itself as a secret key.

3.3 Information Theory

Recall the definition of the min-entropy of a random variable X as:

$$H_\infty(X) = -\log\left(\mathsf{max}_x \Pr[X = x]\right).$$

We recall the definition of average conditional min-entropy in the following.

Definition 3.9 (Average Conditional Min-Entropy)**.** *Let X be a random-variable supported on a finite set $\mathcal{X}$ and let Z be a (possibly correlated) random variable supported on a finite set $\mathcal{Z}$. The average-conditional min-entropy $\tilde{H}_\infty(X|Z)$ is defined as:*

$$\tilde{H}_\infty(X|Z) = -\log\left(\mathbb{E}_z\left[\mathsf{max}_{x\in\mathcal{X}} \Pr[X = x|Z = z]\right]\right).$$

We also recall the chain rule of average min-entropy.

Theorem 3.10 (Chain Rule [DORS08])**.** *Let X, Y and Z be random variables. Then,*

$$\tilde{H}_\infty(X|Y, Z) \le \tilde{H}_\infty(X, Y|Z) - |Y|$$

where $|Y|$ denotes the bit-length of the value of Y.

Next, we recall the definition of a seeded randomness extractor.

Definition 3.11 (Extractor)**.** *A function $\mathsf{Ext} : \{0,1\}^d \times \mathcal{X} \to \{0,1\}^\ell$ is called a seeded strong average-case (k, ε)-extractor, if it holds for all random variables X with support $\mathcal{X}$ and Z defined on some finite support that if $\tilde{H}_\infty(X|Z) \ge k$, then it holds that the statistical distance of the following distributions is a most ε:*

$$\{\mathsf{seed}, \mathsf{Ext}(\mathsf{seed}, X), Z\} \approx_\varepsilon \{\mathsf{seed}, U, Z\}$$

where $\mathsf{seed} \xleftarrow{\$} \{0,1\}^d$ and $U \xleftarrow{\$} \{0,1\}^\ell$.

Recall that a hash function $\mathsf{Hash} : \mathcal{X} \to \mathcal{Y}$ is a universal hash if for all $x \neq x' \in \mathcal{X}$ it holds that:

$$\Pr[\mathsf{Hash}(x) = \mathsf{Hash}(x')] \le \frac{1}{|\mathcal{Y}|}$$

where the probability is taken over the choice of the hash function. It is shown [DRS04, DORS08] that any universal hash function is an average-case randomness extractor.

Lemma 3.12 (Leftover Hash Lemma)**.** *Let X be a random-variable supported on a finite set $\mathcal{X}$ and let Z be a (possibly correlated) random variable supported on a finite set $\mathcal{Z}$ such that $\tilde{H}_\infty(X|Z) \geq k$. Let $\mathsf{Hash} : \mathcal{X} \to \{0,1\}^\ell$, where $\ell \leq k - 2\log\left(\frac{1}{\varepsilon}\right)$, be a family of universal hash functions. Then Hash is a seeded strong average-case (k, ε)-extractor.*

3.4 Garbled Circuits

We recall the definition of garbled circuits [Yao86, BHR12].

Definition 3.13 (Garbling Schemes)**.** *A garbling scheme consists of two PPT algorithms $(\mathsf{Garble}, \mathsf{Eval})$ with the following syntax.*

- $\mathsf{Garble}(1^\lambda, C)$*: On input the security parameter 1^λ and a circuit $C : \{0,1\}^n \to \{0,1\}^m$, the garbling algorithm returns a garbled circuit $\tilde{C}$ and a set of $2n$ labels $(\mathsf{lab}_{i,b})_{i\in[n],b\in\{0,1\}}$.*
- $\mathsf{Eval}(\tilde{C}, (\mathsf{lab}_i)_{i\in[n]})$*: On input a garbled circuit $\tilde{C}$ and a set of n labels $(\mathsf{lab}_i)_{i\in[n]}$, the evaluation algorithm returns an output $y \in \{0,1\}^m$.*

For correctness, we require that for all λ, all circuits C, and all inputs x, we have that:

$$\mathsf{Eval}(\tilde{C}, (\mathsf{lab}_{i,x_i})_{i\in[n]}) = C(x)$$

where $(\tilde{C}, (\mathsf{lab}_{i,b})_{i\in[n],b\in\{0,1\}}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C)$. We recall the security definition of garbling schemes.

Definition 3.14 (Simulation Security)**.** *A garbling scheme $(\mathsf{Garble}, \mathsf{Eval})$ is simulation secure if there exists a PPT simulator Sim such that for every circuit C and every input x, the following distributions are computationally indistinguishable:*

$$\left\{\tilde{C}, (\mathsf{lab}_{i,x_i})_{i\in[n]}\right\} \approx_c \mathsf{Sim}(1^\lambda, C(x))$$

where $(\tilde{C}, (\mathsf{lab}_{i,b})_{i\in[n],b\in\{0,1\}}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C)$.

4 The Decomposed LWE Problem

In the following we introduce the hardness assumption that we will use in this work, that we refer to as the *decomposed LWE* assumption. We formally define the problem below.

Definition 4.1 (Decomposed LWE)**.** *Let $k = k(\lambda)$, $m = m(\lambda)$, $\sigma = \sigma(\lambda)$, $\tilde{\sigma} = \tilde{\sigma}(\lambda)$, $t = t(\lambda)$, and $q = q(\lambda)$ be integer parameters. Let $n := k \cdot \log q$. The decomposed LWE assumption postulates the the following distributions are computationally indistinguishable:*

$$\left\{\mathbf{A}_i, \mathbf{B}_j, \mathbf{s}^\intercal\mathbf{A}_i\mathbf{B}_j + \mathbf{s}^\intercal\mathbf{G}\cdot\delta_{i,j} + \mathbf{e}_{i,j}^\intercal\right\}_{i,j\in[t]} \approx_c \left\{\mathbf{A}_i, \mathbf{B}_j, \mathbf{u}_{i,j}^\intercal\right\}_{i,j\in[t]}$$

where $\mathbf{A}_i \xleftarrow{\$} \mathbb{Z}_q^{k\times m}$, $\mathbf{B}_j \xleftarrow{\$} \mathcal{D}_\sigma^{m\times n}$, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\mathbf{e}_{i,j} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^n$, and $\mathbf{u}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^n$.

The following theorem shows that the decomposed LWE problem is at least as hard as the t-succinct LWE problem. In our proof, we only consider the settings where the modulus to noise ratio is super-polynomial, i.e., the so-called flooding regime. Although we do not explicitly prove this in our work, we expect that the implication holds also in the polynomial regime, with a more sophisticated argument.

Theorem 4.2. *Suppose that $m \geq 2k \cdot \log q$ and $\sigma = \sqrt{k \cdot \log q} \cdot \omega(\sqrt{\log m})$. If the t-succinct LWE problem is hard, then the decomposed LWE assumption holds with a prime q and $\tilde{\sigma}/\sigma \in \lambda^{\omega(1)}$.*

Proof. We prove the theorem with a Karp reduction, i.e., we are going to show how to map any instance of the t-succinct LWE into a valid instance of decomposed LWE, with the same solution. The statement follows because if one can break the decomposed LWE then we can use this algorithm to break the t-succinct LWE problem as well.

On input a tuple $(\mathbf{A}, \mathbf{u}^\intercal, \mathbf{W}, \mathbf{T})$, where $\mathbf{u}$ is either uniform or an LWE sample, we proceed as follows. Let us split the input matrices into blocks:

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_0 \\ \vdots \\ \mathbf{W}_{t-1} \end{bmatrix} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{T}_{0,0} & \mathbf{T}_{0,1} & \cdots & \mathbf{T}_{0,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{T}_{t-1,0} & \mathbf{T}_{t-1,1} & \cdots & \mathbf{T}_{t-1,t-1} \\ \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_{t-1} \end{bmatrix}$$

The decomposed LWE instance consists of the matrices $\{-\mathbf{W}_i, \mathbf{B}_i\}_{i \in [t]}$ and the vectors $\mathbf{u}_{i,j}$ computed as:

$$\mathbf{u}_{i,j}^\intercal \leftarrow \mathbf{u}^\intercal \cdot \mathbf{T}_{i,j} + \tilde{\mathbf{e}}_{i,j}^\intercal$$

where $\tilde{\mathbf{e}}_{i,j} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^n$. We analyze the distribution of the instance that we constructed.

- (Real) Consider the case where $\mathbf{u} = \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal$ is an LWE sample. Note that the matrices $-\mathbf{W}_i$ are uniformly distributed in $\mathbb{Z}_q^{k \times m}$, whereas the matrices $\mathbf{B}_i$ are (statistically close to) sampled according to $\mathcal{D}_\sigma^{m \times n}$. By assumption, we have that $\mathbf{T}$ is Gaussian, subject to:
$$\mathbf{A}\mathbf{T}_{i,j} = \mathbf{G} \cdot \delta_{i,j} - \mathbf{W}_i \mathbf{B}_j$$
for all $i, j \in [t]$. Thus, by linearity, we have that:
$$\begin{aligned} \mathbf{u}^\intercal \cdot \mathbf{T}_{i,j} + \tilde{\mathbf{e}}_{i,j}^\intercal &= (\mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal) \cdot \mathbf{T}_{i,j} + \tilde{\mathbf{e}}_{i,j}^\intercal \\ &= \mathbf{s}^\intercal \mathbf{G} \cdot \delta_{i,j} - \mathbf{s}^\intercal \mathbf{W}_i \mathbf{B}_j + \mathbf{e} \cdot \mathbf{T}_{i,j} + \tilde{\mathbf{e}}_{i,j}^\intercal \\ &\approx_s \mathbf{s}^\intercal \mathbf{G} \cdot \delta_{i,j} - \mathbf{s}^\intercal \mathbf{W}_i \mathbf{B}_j + \tilde{\mathbf{e}}_{i,j}^\intercal \end{aligned}$$
where the last implication follows by Lemma 3.2. This corresponds precisely to an instance of the decomposed LWE problem.

- (Random) Consider the case where $\mathbf{u}$ is uniform in $\mathbb{Z}_q^m$. The correct distribution of the matrices follows with the same argument as above, so all is left to be shown is that the vectors $\mathbf{u}_{i,j}$ are also uniformly distributed. We show this with a hybrid argument, where we progressively substitute each $\mathbf{u}_{i,j}$ with a uniformly sampled vector in $\mathbb{Z}_q^n$. In the (i, j)-th hybrid, we modify the view of the distinguisher as follows:

– Hybrid $(i,j,0)$: This is the original distribution, and we define $\mathbf{Z}_{i,j} := -\mathbf{W}_i\mathbf{B}_j$.

– Hybrid $(i,j,1)$: We sample $(\mathbf{W}_i, \gamma_i) \xleftarrow{\$} \mathsf{TrapGen}(1^k, 1^m, q)$, $\mathbf{Z}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^{k\times n}$, $\mathbf{B}_j \xleftarrow{\$} \mathsf{PreSample}(\gamma_i, -\mathbf{Z}_{i,j}, \sigma)$. In other words, $\mathbf{B}_j$ is distributed according to $\mathcal{D}_\sigma^{m\times n}$, but conditioned on $\mathbf{Z}_{i,j} = -\mathbf{W}_i\mathbf{B}_j$. All other variables are unchanged. Statistical indistinguishability follows by Lemma 3.3.

– Hybrid $(i,j,2)$: We sample $\mathbf{T}_{i,j} \xleftarrow{\$} \mathcal{D}_\sigma^{m\times n}$, and compute $\mathbf{Z}_{i,j} \leftarrow \mathbf{A}\mathbf{T}_{i,j} - \mathbf{G}\cdot\delta_{i,j}$. Statistical indistinguishability follows by another invocation of Lemma 3.3.

Notice that the variable $\mathbf{T}_{i,j}$ is statistically close to a discrete Gaussian (see Lemma 3.3), so the min entropy of each of its columns is at least m and the only variable that depend on $\mathbf{T}_{i,j}$ (besides $\mathbf{u}_{i,j}$) is $\mathbf{A}\mathbf{T}_{i,j}$. Thus, by the chain rule, the conditional min entropy of each column of $\mathbf{T}_{i,j}$ is at least $m - n\log q \geq n\log q$, and we can conclude that $\mathbf{u}^\intercal \cdot \mathbf{T}_{i,j}$ is statistically close to uniform, by Lemma 3.12. The series of hybrids is concluded by substituting $\mathbf{u}_{i,j}$ with uniform, then undoing the previous hybrid changes.

This concludes our proof. □

The above theorem shows that the decomposed LWE problem is at least as hard as the t-succinct LWE problem. However, we believe that the new problem is qualitatively weaker, since it does not involve any lattice trapdoor, and thus a reduction in the reverse direction is unlikely. Decomposed LWE seems to fall into the category of *circularity assumptions*. For instance, if the function $\delta_{i,j}$ was substituted with the constant 0, then the problem is not easier than LWE: Given LWE samples $\{\mathbf{s}^\intercal\mathbf{A}_i + \mathbf{e}_i^\intercal\}_i$, one can generate a valid instance by computing:

$$\mathbf{u}_{i,j}^\intercal = (\mathbf{s}^\intercal\mathbf{A}_i + \mathbf{e}_i^\intercal)\,\mathbf{B}_j + \mathbf{e}_{i,j}^\intercal \approx_s \mathbf{s}^\intercal\mathbf{A}_i\mathbf{B}_j + \mathbf{e}_{i,j}^\intercal$$

where $\mathbf{e}_{i,j}$ is sampled with a Gaussian parameter super-polynomially larger than that of $\mathbf{e}_i$. More broadly, our problem can be thought of as a special case of a generalised variant of the circular-LWE problem, akin to [MV24]. Given a distribution of matrices $\mathbf{M}$ for which the LWE problem is hard, the generalised circular LWE assumption postulates that:

$$\mathbf{s}^\intercal\mathbf{M} + f(\mathbf{s}) + \mathbf{e}^\intercal \approx_c \mathbf{u}^\intercal$$

for some (fixed) function f. This kind of assumptions are at the heart of known constructions of fully-homomorphic encryption from lattices [Gen09], although the function f in that case is somewhat different from ours. We leave a precise characterization of our assumption in the context of circular security as ground for future work.

4.1 Variants of Decomposed LWE

We introduce a few variants of the decomposed LWE problems.

Definition 4.3 (Small-Secret Extended Decomposed LWE). *Let $k = k(\lambda)$, $m = m(\lambda)$, $\sigma = \sigma(\lambda)$, $\tilde{\sigma} = \tilde{\sigma}(\lambda)$, $\sigma' = \sigma'(\lambda)$, $t = t(\lambda)$, $\ell = \ell(\lambda)$, and $q = q(\lambda)$ be integer parameters.*

Define $n := k \cdot \log q$. The extended decomposed LWE assumption postulates the the following distributions are computationally indistinguishable:

$$\left\{\begin{matrix} \mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{M} \\ \mathbf{s}^\intercal \cdot \mathbf{M} + \mathbf{e}^\intercal \\ (\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot \big(\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})\big) + \mathbf{E} \end{matrix}\right\} \approx_c \{\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{M}, \mathbf{v}^\intercal, \mathbf{U}\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{(t\cdot k)\times m}$, $\mathbf{B} \xleftarrow{\$} \mathcal{D}_\sigma^{m\times(n\cdot t)}$*,* $\mathbf{Q} \xleftarrow{\$} \mathsf{GL}_k(\mathbb{Z}_q)$*,* $\mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{k\times\ell}$*,* $\mathbf{s} \xleftarrow{\$} \mathcal{D}_{\sigma'}^k$*,* $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\sigma'}^\ell$*,* $\mathbf{E} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^{t\times(t\cdot n)}$*,* $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{t\times(t\cdot n)}$*, and* $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^\ell$*.*

Theorem 4.4. *If $\sigma'/(\sigma \cdot \widetilde{\sigma}) = \lambda^{\omega(1)}$ and $\ell = \mathrm{poly}(\lambda)$, decomposed LWE implies small-secret, extended decomposed LWE with the same choice of parameters.*

Proof. We start by proving that decomposed LWE implies its extended version if $\mathbf{s}$ is sample uniformly at random over $\mathbb{Z}_q^k$ and $\mathbf{Q} = \mathbf{I}_k$. We proceed by means of a series of hybrids.

- Hybrid 0: This corresponding to the distribution of an extended decomposed LWE sample but with $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$ and $\mathbf{Q} = \mathbf{I}_k$.

- Hybrid 1: We change the distribution of $\mathbf{M}$ sampling: we set $\mathbf{M}$ to $\mathbf{A}_0 \cdot \mathbf{B}_1 \cdot \mathbf{R}$, where $\mathbf{R} \xleftarrow{\$} \mathcal{D}_\sigma^{n\times\ell}$ and $\mathbf{A}_0$ denotes the 0-th $k \times m$ block in $\mathbf{A}$ and $\mathbf{B}_1$ denotes the 1-th $m \times n$ block in $\mathbf{B}$. This hybrid is statistically indistinguishable from Hybrid 0 by the leftover hash lemma. Indeed, $\mathbf{R}$ is independent of $(\mathbf{A}_0, \mathbf{B}_1)$ and the function that $f_{\mathbf{A}_0,\mathbf{B}_1}(\mathbf{x}) := \mathbf{A}_0 \cdot \mathbf{B}_1 \cdot \mathbf{x}$ is a 2-universal hash function (notice that $\mathbf{A}_0 \cdot \mathbf{B}_1$ is statistically close to random, again, by the leftover hash lemma).

- Hybrid 2: We change the distribution of $(\mathbf{U}, \mathbf{v})$ sampling them at random. This hybrid is indistinguishable from Hybrid 1 under decomposed LWE.

  We sketch the reduction: suppose we are given a tuple $(\mathbf{A}_i, \mathbf{B}_j, \mathbf{u}_{i,j}^\intercal)_{i,j\in[t]}$ where $\mathbf{u}_{i,j}^\intercal$ is either $\mathbf{s}^\intercal \mathbf{A}_i \mathbf{B}_j + \delta_{i,j} \cdot \mathbf{s}^\intercal \mathbf{G} + \mathbf{e}_{i,j}^\intercal$ or uniformly random.

  We define $\mathbf{A}$ as the vertical concatenation of $\mathbf{A}_0, \dots, \mathbf{A}_{t-1}$. We define $\mathbf{B}$ as the horizontal concatenation of $\mathbf{B}_0, \dots, \mathbf{B}_{t-1}$. Let $\mathbf{U}$ be the matrix where the i-th row consists of the horizontal concatenation of $(\mathbf{u}_{i,j}^\intercal)_{i\in[t]}$. Finally, we generate $\mathbf{v}$ by setting $\mathbf{u}_{0,1}^\intercal \cdot \mathbf{R} + \tilde{\mathbf{e}}^\intercal$ for $\tilde{\mathbf{e}} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^\ell$.

  We observe that, with this choice of $\mathbf{A}$ and $\mathbf{B}$, we have the i-th row of $(\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot (\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes \mathbf{G}) + \mathbf{E}$ consists of the horizontal concatenation of

  $$\mathbf{s}^\intercal \cdot \mathbf{A}_i \cdot \mathbf{B}_j + \delta_{i,j} \cdot \mathbf{s}^\intercal \cdot \mathbf{G} + \mathbf{e}_{i,j}^\intercal$$

  where $\mathbf{e}_{i,j}^\intercal$ denotes the j-th n-dimensional block in the i-th row of $\mathbf{E}$. This is identical to the distribution of $\mathbf{u}_{i,j}^\intercal$ in a decomposed LWE sample. Notice also that if $(\mathbf{u}_{i,j})_{i,j\in[t]}$ were random, the distribution of $\mathbf{U}$ in the reduction is also random. We also observe that

  $$\mathbf{s}^\intercal \cdot \mathbf{M} + \mathbf{e}^\intercal = \mathbf{s}^\intercal \cdot \mathbf{A}_0 \cdot \mathbf{B}_1 \cdot \mathbf{R} + \mathbf{e}^\intercal.$$

Notice that if this is identical to the distribution of $\mathbf{u}_{0,1}^{\mathsf{T}} \cdot \mathbf{R} - \mathbf{e}_{0,1}^{\mathsf{T}} \cdot \mathbf{R} + \mathbf{e}^{\mathsf{T}}$, if the reduction received a real decomposed LWE sample. Notice also that this is statistically indistinguishable from $\mathbf{u}_{0,1}^{\mathsf{T}} \cdot \mathbf{R} + \mathbf{e}^{\mathsf{T}}$ given that $\mathbf{e}$ is sampled from a discrete Gaussian where the parameter is $\lambda^{\omega(1)}$ times greater than $\|\mathbf{e}_{0,1}^{\mathsf{T}} \cdot \mathbf{R}\|_\infty$. Finally, observe that if we obtained a random $\mathbf{u}_{0,1}$, the distribution of $\mathbf{v}^{\mathsf{T}} = \mathbf{u}_{0,1}^{\mathsf{T}} \cdot \mathbf{R} + \mathbf{e}^{\mathsf{T}}$ is statistically close to random by the leftover hash lemma.

Next, we prove that if extended decomposed LWE is hard for $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$ and $\mathbf{Q} \leftarrow \mathbf{I}_k$, the problem is hard even when $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$ and $\mathbf{Q} \xleftarrow{\$} \mathsf{GL}_k(\mathbb{Z}_q)$. The reduction is pretty straightforward: given a tuple $(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{M}, \mathbf{v}^{\mathsf{T}}, \mathbf{U})$, we simply output $((\mathbf{I}_t \otimes \mathbf{Q}^{-1}) \cdot \mathbf{A}, \mathbf{B}, \mathbf{I}_k, \mathbf{Q}^{-1} \cdot \mathbf{M}, \mathbf{v}^{\mathsf{T}}, \mathbf{U})$. Notice that, if the reduction obtained a real instance, the output is identical to an extended decomposed LWE sample with $\mathbf{Q} \leftarrow \mathbf{I}_k$ (if the secret in the input tuple is $\mathbf{s}$, the secret in the output tuple is $\mathbf{Q}^{\mathsf{T}} \cdot \mathbf{s}$), otherwise, the distribution consists just of a random tuple.

Finally, we prove that extended decomposed LWE with $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$ and $\mathbf{Q} \xleftarrow{\$} \mathsf{GL}_k(\mathbb{Z}_q)$ implies the hardness of the version in which $\mathbf{s} \xleftarrow{\$} \mathcal{D}_\sigma^k$. We show a reduction: suppose we are given a tuple $(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{M}, \mathbf{v}^{\mathsf{T}}, \mathbf{U})$ which is either a real extended decomposed LWE sample (with uniformly sampled $\mathbf{s}$), or uniformly random terms. Let $\mathbf{W}$ be a $\mathbb{Z}_q$-invertible $k \times k$ matrix obtained by horizontally stacking columns of $\mathbf{M}$ among the first L. Notice that $\mathbf{W}$ exists with overwhelming probability, assuming that L is sufficiently bigger than k and (we can assume without loss of generality that the reduction receives $L + \ell$ LWE samples as input). Let $\mathbf{w}$ be the subvector of $\mathbf{v}$ containing all the entries corresponding with the columns of $\mathbf{M}$ used to build $\mathbf{W}$. In other words, if the reduction received an extended decomposed LWE sample, it holds that $\mathbf{w}^{\mathsf{T}} = \mathbf{s}^{\mathsf{T}} \cdot \mathbf{W} - \overline{\mathbf{s}}^{\mathsf{T}}$ where $\overline{\mathbf{s}} \xleftarrow{\$} \mathcal{D}_{\sigma'}^k$. Let $\overline{\mathbf{M}}$ consist of the last ℓ columns of $\mathbf{M}$ and let $\overline{\mathbf{v}}$ denote the last ℓ entries of $\mathbf{v}$. Finally, let $\overline{\mathbf{e}}$ denote the noise term used in $\overline{\mathbf{v}}$ when this is an LWE sample. The reduction computes

$$\begin{aligned}
\mathbf{A}' &\leftarrow (\mathbf{I}_t \otimes \mathbf{W}^{-1}) \cdot \mathbf{A} \\
\mathbf{Q}' &\leftarrow \mathbf{W}^{-1} \cdot \mathbf{Q} \\
\mathbf{U}' &\leftarrow \mathbf{U} - (\mathbf{I}_t \otimes (\mathbf{w}^{\mathsf{T}} \cdot \mathbf{W}^{-1})) \cdot \big(\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})\big) \\
\mathbf{M}' &\leftarrow \mathbf{W}^{-1} \cdot \overline{\mathbf{M}} \\
\mathbf{v}'^{\mathsf{T}} &\leftarrow \overline{\mathbf{v}}^{\mathsf{T}} - \mathbf{w}^{\mathsf{T}} \cdot \mathbf{W}^{-1} \cdot \overline{\mathbf{M}}.
\end{aligned}$$

Then, it outputs $(\mathbf{A}', \mathbf{B}, \mathbf{Q}', \mathbf{M}', \mathbf{v}'^{\mathsf{T}}, \mathbf{U}')$. We observe that, if the reduction received a real extended decomposed LWE sample (with uniformly random $\mathbf{s}$), we have

$$\begin{aligned}
\mathbf{U}' &= \mathbf{U} - (\mathbf{I}_t \otimes (\mathbf{w}^{\mathsf{T}} \cdot \mathbf{W}^{-1})) \cdot \big(\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})\big) \\
&= \mathbf{U} - (\mathbf{I}_t \otimes (\mathbf{s}^{\mathsf{T}} - \overline{\mathbf{s}}^{\mathsf{T}} \cdot \mathbf{W}^{-1})) \cdot \big(\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})\big) \\
&= \mathbf{U} - (\mathbf{I}_t \otimes \mathbf{s}^{\mathsf{T}}) \cdot \big(\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})\big) + (\mathbf{I}_t \otimes (\overline{\mathbf{s}}^{\mathsf{T}} \cdot \mathbf{W}^{-1})) \cdot \big(\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})\big) \\
&= (\mathbf{I}_t \otimes \overline{\mathbf{s}}^{\mathsf{T}}) \cdot \big((\mathbf{I}_t \otimes \mathbf{W}^{-1}) \cdot \mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{W}^{-1} \cdot \mathbf{Q} \cdot \mathbf{G})\big) + \mathbf{E} \\
&= (\mathbf{I}_t \otimes \overline{\mathbf{s}}^{\mathsf{T}}) \cdot \big(\mathbf{A}' \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q}' \cdot \mathbf{G})\big) + \mathbf{E}.
\end{aligned}$$

Moreover,

$$\mathbf{v}'^{\mathsf{T}} = \overline{\mathbf{v}}^{\mathsf{T}} - \mathbf{w}^{\mathsf{T}} \cdot \mathbf{W}^{-1} \cdot \overline{\mathbf{M}}$$

$$
\begin{aligned}
&= \overline{\mathbf{v}}^\intercal - (\mathbf{s}^\intercal \mathbf{W} - \overline{\mathbf{s}}^\intercal) \cdot \mathbf{W}^{-1} \cdot \overline{\mathbf{M}} \\
&= \overline{\mathbf{v}}^\intercal - \mathbf{s}^\intercal \cdot \overline{\mathbf{M}} + \overline{\mathbf{s}}^\intercal \cdot \mathbf{M}' \\
&= \overline{\mathbf{s}}^\intercal \cdot \mathbf{M}' + \overline{\mathbf{e}}^\intercal.
\end{aligned}
$$

In other words, the output of the reduction consists of a small-secret extended decomposed LWE sample. If instead the input was a random tuple, the output is too. □

The following is a circular variant of the small-secret decomposed LWE problem, which roughly corresponds to additionally providing the distinguisher with a GSW encryption of its own secret key.

Definition 4.5 (Small-Secret Circular Decomposed LWE)**.** *Let $k = k(\lambda)$, $m = m(\lambda)$, $\sigma = \sigma(\lambda)$, $\tilde{\sigma} = \tilde{\sigma}(\lambda)$, $\sigma' = \sigma'(\lambda)$, $\overline{\sigma} = \overline{\sigma}(\lambda)$, $t = t(\lambda)$, $\ell = \ell(\lambda)$, and $q = q(\lambda)$ be integer parameters. Define $n := k \cdot \log q$. The small-secret, circular decomposed LWE assumption postulates the the following distributions are computationally indistinguishable:*

$$
\left\{ \begin{matrix} \mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{M}, \\ \mathbf{s}^\intercal \cdot \mathbf{M} + \mathbf{e}^\intercal \\ (\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot \big(\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})\big) + \mathbf{E} \\ \begin{bmatrix} \mathbf{K} \\ \mathbf{s}^\intercal \cdot \mathbf{K} + \overline{\mathbf{e}}^\intercal \end{bmatrix} - \mathsf{Bits}(\mathbf{s})^\intercal \otimes \mathbf{I}_{k+1} \otimes \mathbf{g}_q \end{matrix} \right\} \approx_c \{\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{M}, \mathbf{v}^\intercal, \mathbf{U}, \mathbf{S}\}
$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{(t \cdot k) \times m}$, $\mathbf{B} \xleftarrow{\$} \mathcal{D}_\sigma^{m \times (n \cdot t)}$, $\mathbf{Q} \xleftarrow{\$} \mathsf{GL}_k(\mathbb{Z}_q)$, $\mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{k \times \ell}$, $\mathbf{K} \xleftarrow{\$} \mathbb{Z}_q^{k \times n \cdot (n + \log q)}$, $\mathbf{s} \xleftarrow{\$} \mathcal{D}_{\sigma'}^k$, $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\sigma'}^\ell$, $\mathbf{E} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^{t \times (t \cdot n)}$, $\overline{\mathbf{e}} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{n \cdot (n + \log q)}$, $\mathbf{S} \xleftarrow{\$} \mathbb{Z}_q^{(k+1) \times n \cdot (n + \log q)}$, and $\mathbf{v} \xleftarrow{\$} \mathbb{Z}_q^\ell$.

5 Rate-1 Lattice Encodings for RAM Programs

In this section we describe the main technical contribution of our work, namely the lattice encodings for RAM program. Let $n := k \cdot \log q$ where $k = \Omega(\lambda)$ and let $t := \beta \cdot n \cdot m$ and $L := \beta^r \cdot n^2$. Our construction is given in Fig. 4 and we defer the description of the reading and procedure to Section 5.1 and Section 5.2, respectively. We state the following claim, which can be easily verified.

Lemma 5.1. *Define $\mathbf{\Pi} := \mathsf{vec}(\mathbf{I}_t)^\intercal \otimes \mathbf{I}_k$. Then, for every matrix $\mathbf{M} \in \mathbb{Z}_q^{(t \cdot k) \times (t \cdot n)}$ and vector $\mathbf{s} \in \mathbb{Z}_q^k$, we have*

$$\mathbf{s}^\intercal \cdot \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes \mathbf{M}) = \mathsf{vec}((\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot \mathbf{Z})^\intercal.$$

The following lemma bounds the complexity of computing the hash depending on the Hamming weight of the input string.

Lemma 5.2. *If $\mathbf{x} \in \mathbb{Z}_q^L$ has Hamming weight w, we can compute $\mathsf{Hash}(\mathsf{pk}, \mathbf{x})$ in time $w \cdot \log L \cdot \mathrm{poly}(\lambda)$ on a RAM.*

Proof. Suppose that $\mathbf{x} = \sum_{i=0}^{w-1} x_i \cdot \mathbf{u}_{\ell_i}$ where $\mathbf{u}_{\ell_i}$ denotes the ℓ_i-th vector of the standard basis of $\mathbb{Z}_q^L$. Since Hash is $\mathbb{Z}_q$-linear, we have that $\mathsf{Hash}(\mathsf{pk}, \mathbf{x}) = \sum_{i=0}^{w-1} x_i \cdot \mathsf{Hash}(\mathsf{pk}, \mathbf{u}_{\ell_i})$. Moreover, for every $i \in [w]$, $\mathsf{Hash}(\mathsf{pk}, \mathbf{u}_{\ell_i})$ can be computed in time $\log L \cdot \mathrm{poly}(\lambda)$ (indeed, if $\mathbf{x}_{i,j} = \mathbf{0}$, it holds that $\mathbf{X}_{i,j} = \mathbf{0}$). □

Succinct Lattice Encodings for RAM Programs

$\mathsf{Setup}(1^\lambda)$**:** Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{(t\cdot k)\times m}, \overline{\mathbf{B}} \xleftarrow{\$} \mathcal{D}_\sigma^{m\times(t\cdot n)}$ and $\mathbf{Q} \xleftarrow{\$} \mathsf{GL}_k(\mathbb{Z}_q)$. Then, compute

$$\mathbf{B} \leftarrow \overline{\mathbf{B}} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})) \qquad \mathbf{Z} \leftarrow \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{B}) + \mathbf{I}_t \otimes \mathbf{I}_t \otimes \mathbf{G}).$$

and return $\mathsf{pk} := (\mathbf{A}, \overline{\mathbf{B}}, \mathbf{B}, \mathbf{Q}, \mathbf{Z})$.

$\mathsf{Hash}(\mathsf{pk}, \mathbf{x})$**:** If $|\mathbf{x}| \le L$, pad it with zeros. Set $\mathbf{x}_0 \leftarrow \mathbf{x}$, then perform the following operations for $i = 0, \dots, r-1$:

1. Split $\mathbf{x}_i$ into blocks in $\mathbb{Z}_q^t$. Let them be $\mathbf{x}_{i,0}, \dots, \mathbf{x}_{i,\beta^{r-1-i}-1}$.
2. For every $j \in [\beta^{r-1-i}]$, set $\mathbf{X}_{i,j} \leftarrow \mathbf{B} \cdot (\mathbf{x}_{i,j} \otimes \mathbf{I}_n)$.
3. Define $\mathbf{x}_{i+1}$ to be the vertical concatenation of $\mathsf{vec}(\mathbf{X}_{i,j})$ for $j \in [\beta^{r-1-i}]$.

Output $\mathbf{d} := \mathbf{x}_r$ and $\tau := (\mathbf{x}_{i,j})_{i\in[r], j\in[\beta^{r-1-i}]}$.

$\mathsf{Enc}(\mathsf{pk}, \mathbf{M}, \mathbf{x})$**:** Sample $\mathbf{s} \xleftarrow{\$} \mathcal{D}_{\sigma'}^k$, compute $(\mathbf{d}, \tau) \leftarrow \mathsf{Hash}(\mathsf{pk}, \mathbf{x})$,

$$\mathbf{h}^\intercal \leftarrow \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal$$
$$\mathbf{z}^\intercal \leftarrow \mathbf{s}^\intercal \cdot \mathbf{Z} + \tilde{\mathbf{e}}^\intercal$$

where $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{n^2\cdot m}$, $\mathbf{E} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^{t\times(t\cdot n)}$ and $\tilde{\mathbf{e}} \leftarrow \mathsf{vec}(\mathbf{E} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))$. Finally, output $\mathbf{d}$, $\mathbf{h}, \mathbf{z}$ and τ.

$\mathsf{Path}(\tau, \ell)$**:** Define $\ell_0 \in [t]$, $\ell_i \in [\beta]$ and $\overline{\ell_j}$ so that

$$\ell = \ell_0 + \sum_{i=1}^{r-1} \ell_i \cdot t \cdot \beta^{i-1}, \qquad \overline{\ell_j} = \sum_{i=j+1}^{r-1} \ell_i \cdot \beta^{i-j-1}.$$

Rewrite τ as $(\mathbf{x}_{i,j})_{i\in[r], j\in[\beta^{r-1-i}]}$ and output $\tau_\ell := (\mathbf{x}_{i,\overline{\ell_i}})_{i\in[r]}$

Figure 4: Succinct Lattice Encodings for RAM Programs

5.1 Reading Procedure

For every $i \in [r]$, let π_i be the function that takes as input a vector $\mathbf{x} \in \mathbb{Z}_q^t$ and $\ell \in [L]$, and performs the following operations: First of all, it rewrites ℓ as

$$\ell = \ell_0 + \sum_{j=1}^{r-1} \ell_j \cdot t \cdot \beta^{j-1}$$

where $\ell_0 \in [t]$ and $\ell_j \in [\beta]$ for every $j \geq 1$. If $i \geq 1$, π_i splits $\mathbf{x}$ into β blocks $\mathbf{x}_0, \ldots, \mathbf{x}_{\beta-1} \in \mathbb{Z}_q^{n \cdot m}$, then it outputs $\mathbf{x}_{\ell_i}$. If instead $i = 0$, π_i splits $\mathbf{x}$ into elements $x_0, \ldots, x_{t-1} \in \mathbb{Z}_q$, then it outputs x_{ℓ_0}. The reading procedure is described in Fig. 5.

Lemma 5.3. *There exists a deterministic polynomial time algorithms such that, for any matrices* $\mathbf{X} \in \mathbb{Z}_q^{m \times n}$ *and* $\mathbf{A} \in \mathbb{Z}_q^{(t \cdot k) \times m}$*, we have*

$$(\mathsf{vec}(\mathbf{X})^\intercal \otimes \mathbf{G}) \cdot \mathsf{Lin}(\mathbf{A}) = \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{X}))$$

Moreover, $\|\mathsf{Lin}(\mathbf{A})\|_\infty \leq 1$ *and each of its columns has at most* $m \cdot n$ *non-zero entries.*

Proof. Split $\mathbf{A}$ into $k \times m$ blocks $\mathbf{A}_0, \ldots, \mathbf{A}_{t-1}$. Similarly, split $\mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{X}))$ into blocks of size $k \times n$. We observe that, for any $i \in [t]$ and $j \in [n]$, the i-th block of $\mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{X}))$ consists of $\mathbf{A}_i \cdot \mathbf{X}$. In other words, if we denote the j-th column in $\mathbf{X}$ by $\mathbf{x}_j$, the $(i \cdot n + j)$-th column of $\mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{X}))$ is $\mathbf{A}_i \cdot \mathbf{x}_j$. Now, observe that, if we denote the ℓ-th columns of $\mathbf{A}_i$ by $\mathbf{a}_{i,\ell}$ and the ℓ-th entry of $\mathbf{x}_j$ by $x_{j,\ell}$, we have that

$$\mathbf{A}_i \cdot \mathbf{x}_j = \sum_{\ell=0}^{m-1} x_{j,\ell} \cdot \mathbf{a}_{i,\ell}.$$

Moreover, for any $\mathbf{v} \in \mathbb{Z}_q^{n^2 \cdot m}$, we have

$$(\mathsf{vec}(\mathbf{X})^\intercal \otimes \mathbf{G}) \cdot \mathbf{v} = \sum_{j \in [n], \ell \in [m]} x_{j,\ell} \cdot \mathbf{G} \cdot \mathbf{v}_{j,\ell}$$

where $(\mathbf{v}_{j,\ell})_{j,\ell}$ are obtained by splitting $\mathbf{v}$ into blocks in $\mathbb{Z}_q^n$. So, to satisfy the relation in our claim, it is sufficient to set the $(i \cdot n + j)$-th column of $\mathsf{Lin}(\mathbf{A})$ to the vector $\mathbf{v}$ where $\mathbf{v}_{j,\ell} = \mathbf{G}^{-1}(\mathbf{a}_{i,\ell})$ for every $\ell \in [m]$, and all other blocks are instead entirely made of zeros. $\square$

The following lemma establishes the correctness of the reading procedure.

Lemma 5.4. *Consider Fig. 4 and Fig. 5 and suppose that* $\mathcal{D}_\sigma$ *is* B*-bounded. Then, there exist polynomial-time deterministic algorithms* ReadK*,* ReadH *such that, for any*

$$\begin{aligned}(\mathbf{d}, \tau) &= \mathsf{Hash}(\mathsf{pk}, \mathbf{x})\\ \tau_\ell &= \mathsf{Path}(\tau, \ell)\\ \mathbf{h}^\intercal &= \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}) + \mathbf{e}_\mathbf{h}^\intercal\\ \mathbf{z}^\intercal &= \mathbf{s}^\intercal \cdot \mathbf{Z} + \tilde{\mathbf{e}}^\intercal\end{aligned}$$

Succinct Lattice Encodings for RAM Programs – Reading Procedure

$\mathsf{Read}(\mathsf{pk}, \mathbf{z}, \mathbf{h}, \mathbf{M}, \tau_\ell, \mathbf{p}, \mathbf{P}, \ell)$**:** Define $\ell_0 \in [t]$, $\ell_i \in [\beta]$ and $\overline{\ell_j}$ so that

$$\ell = \ell_0 + \sum_{i=1}^{r-1} \ell_i \cdot t \cdot \beta^{i-1}, \qquad \overline{\ell_j} = \sum_{i=j+1}^{r-1} \ell_i \cdot \beta^{i-j-1}.$$

Rewrite τ_ℓ as $(\mathbf{x}_{i,\overline{\ell_i}})_{i\in[r]}$ and set $\mathbf{h}_r \leftarrow \mathbf{h}$ and $\mathbf{M}_r \leftarrow \mathbf{M}$. Then, for $i = r-1, \ldots, 0$, compute

```
1. $\mathbf{h}_i'^{\mathsf{T}} \leftarrow \mathbf{z}^{\mathsf{T}} \cdot (\mathbf{I}_t \otimes \mathbf{x}_{i,\overline{\ell_i}} \otimes \mathbf{I}_n) + \mathbf{h}_{i+1}^{\mathsf{T}} \cdot \mathsf{Lin}(-\mathbf{A})$
2. $\mathbf{M}_i' \leftarrow \mathbf{M}_{i+1} \cdot \mathsf{Lin}(-\mathbf{A})$
3. $\mathbf{H}_{\pi_i, \mathbf{x}_{i,\overline{\ell_i}}, \ell} \leftarrow \mathsf{EvalH}(\pi_i, \mathbf{M}_i', \mathbf{P}, \mathbf{x}_{i,\overline{\ell_i}}, \ell)$
4. $\mathbf{K}_{\pi_i} \leftarrow \mathsf{EvalK}(\pi_i, \mathbf{M}_i', \mathbf{P})$
5. $\mathbf{h}_i^{\mathsf{T}} \leftarrow \begin{bmatrix} \mathbf{h}_i'^{\mathsf{T}}, & \mathbf{p}^{\mathsf{T}} \end{bmatrix} \cdot \mathbf{H}_{\pi_i, \mathbf{x}_{i,\overline{\ell_i}}, \ell}$
6. $\mathbf{M}_i \leftarrow \begin{bmatrix} \mathbf{M}_i', & \mathbf{P} \end{bmatrix} \cdot \mathbf{K}_{\pi_i}$
```

Let y be the ℓ_0-th entry of $\mathbf{x}_{0,\overline{\ell_1}}$. Output y, $\mathbf{c} := \mathbf{h}_0$ and $\mathbf{C} := \mathbf{M}_0$.

Figure 5: Succinct Lattice Encodings for RAM Programs – Reading Procedure

$$\mathbf{p}^\intercal = \mathbf{s}^\intercal \cdot (\mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}) + \mathbf{e}_\mathbf{p}^\intercal$$
$$(y, \mathbf{c}, \mathbf{C}) = \mathsf{Read}(\mathsf{pk}, \mathbf{z}, \mathbf{h}, \mathbf{M}, \tau_\ell, \mathbf{p}, \mathbf{P}, \ell)$$

where $\ell \in [L]$, $\|\mathbf{x}\|_\infty \leq T$, it holds that

$$\begin{aligned}\mathbf{c}^\intercal &= \begin{bmatrix}\mathbf{h}^\intercal, & \mathbf{p}^\intercal, & \mathbf{z}^\intercal\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x},\ell} \\ &= \mathbf{s}^\intercal \cdot (\mathbf{C} + y \cdot \mathbf{G}) + \begin{bmatrix}\mathbf{e}_\mathbf{h}^\intercal, & \mathbf{e}_\mathbf{p}^\intercal, & \tilde{\mathbf{e}}^\intercal\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x},\ell} \\ \mathbf{C} &= \begin{bmatrix}\mathbf{M}, & \mathbf{P}\end{bmatrix} \cdot \mathbf{K}\end{aligned}$$
$$\|\begin{bmatrix}\mathbf{e}_\mathbf{h}^\intercal, & \mathbf{e}_\mathbf{p}^\intercal, & \tilde{\mathbf{e}}^\intercal\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x},\ell}\|_\infty \leq (n^3 \cdot m^2 \cdot B \cdot \beta)^r \cdot O(T) \cdot \|\tilde{\mathbf{e}}\|_\infty + (\log \beta \cdot n \cdot m)^r \cdot O\left(\|\mathbf{e}_\mathbf{h}\|_\infty + \|\mathbf{e}_\mathbf{p}\|_\infty\right)$$

where y is the ℓ-th entry of $\mathbf{x}$, $\mathbf{K} := \mathsf{ReadK}(\mathbf{A}, \mathbf{M}, \mathbf{P})$, $\mathbf{H}_{\mathbf{x},\ell} := \mathsf{ReadH}(\mathbf{A}, \mathbf{M}, \mathbf{P}, \mathbf{d}, \tau_\ell, \ell)$. In particular, we have

$$\mathbf{C} + y \cdot \mathbf{G} = \begin{bmatrix}\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}, & \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}, & \mathbf{Z}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x},\ell}.$$

Finally, ReadK *and* ReadH *can be evaluated in time* $\mathrm{poly}(\lambda, \log L)$ *on a RAM.*

Proof. We observe that $\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G} = \mathbf{M}_r + \mathsf{vec}(\mathbf{X}_{r-1,\overline{\ell_{r-1}}})^\intercal \otimes \mathbf{G}$. We define

$$\mathbf{H}_{\mathbf{x},\ell} := \left(\prod_{i=1}^{r}\left(\begin{bmatrix}\mathbf{H}_\mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{\log L \cdot n} & \mathbf{0} \\ \mathbf{H}_{r-i} & \mathbf{0} & \mathbf{I}_{t^2 \cdot n}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{H}^0_{\pi_{r-i},\mathbf{x}_{r-i,\overline{\ell_{r-i}}},\ell} & \mathbf{0} & \mathbf{0} \\ \mathbf{H}^1_{\pi_{r-i},\mathbf{x}_{r-i,\overline{\ell_{r-i}}},\ell} & \mathbf{I}_{\log L \cdot n} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{t^2 \cdot n}\end{bmatrix}\right)\right) \cdot \begin{bmatrix}\mathbf{I}_n \\ \mathbf{0} \\ \mathbf{0}\end{bmatrix}$$
$$\mathbf{K} := \left(\prod_{i=1}^{r}\left(\begin{bmatrix}\mathbf{H}_\mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{\log L \cdot n}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{K}^0_{\pi_{r-i}} & \mathbf{0} \\ \mathbf{K}^1_{\pi_{r-i}} & \mathbf{I}_{\log L \cdot n}\end{bmatrix}\right)\right) \cdot \begin{bmatrix}\mathbf{I}_n \\ \mathbf{0}\end{bmatrix}$$

where

$$\mathbf{H}_{\pi_i,\mathbf{x}_{i,\overline{\ell_i}},\ell} = \begin{bmatrix}\mathbf{H}^0_{\pi_i,\mathbf{x}_{i,\overline{\ell_i}},\ell} \\ \mathbf{H}^1_{\pi_i,\mathbf{x}_{i,\overline{\ell_i}},\ell}\end{bmatrix} \qquad \mathbf{K}_{\pi_i} = \begin{bmatrix}\mathbf{K}^0_{\pi_i} \\ \mathbf{K}^1_{\pi_i}\end{bmatrix}.$$

Then claim that

$$\begin{aligned}\begin{bmatrix}\mathbf{M}, & \mathbf{P}\end{bmatrix} \cdot \mathbf{K} + y \cdot \mathbf{G} &= \begin{bmatrix}\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}, & \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}, & \mathbf{Z}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x},\ell} \\ \mathbf{C} &= \begin{bmatrix}\mathbf{M}, & \mathbf{P}\end{bmatrix} \cdot \mathbf{K}.\end{aligned}$$

and we defer the proof to Claim 5.5. We also observe that $\mathbf{c}^\intercal = \begin{bmatrix}\mathbf{h}^\intercal, & \mathbf{p}^\intercal, & \mathbf{z}^\intercal\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x},\ell}$. All remains to prove is a bound on the norm of $\mathbf{H}_{\mathbf{x},\ell}$. We notice that $\|\mathbf{H}_\mathbf{A}\|_\infty \leq 1$, moreover, each of its columns has at most $n \cdot m$ non-zero entries. We also know that $\|\mathbf{H}_i\|_\infty = \|\mathbf{x}_{i,\overline{\ell_i}}\|_\infty \leq (t \cdot n \cdot B)^i \cdot T$ and each of its columns has at most t non-zero entries. Finally, we have $\|\mathbf{H}^0_{\pi_i,\mathbf{x}_{i,\overline{\ell_i}},\ell}\|_\infty, \|\mathbf{H}^1_{\pi_i,\mathbf{x}_{i,\overline{\ell_i}},\ell}\|_\infty \leq 1$. If $i \geq 1$, each of column of $\mathbf{H}^0_{\pi_i,\mathbf{x}_{i,\overline{\ell_i}},\ell}$ has at most $\log \beta$ non-zero entries, whereas each of column of $\mathbf{H}^1_{\pi_i,\mathbf{x}_{i,\overline{\ell_i}},\ell}$ has at most $n \cdot \log \beta$ non-zero entries. The columns of $\mathbf{H}^0_{\pi_0,\mathbf{x}_{0,\overline{\ell_0}},\ell}$ have instead at most $\log t$ non-zero entries, whereas, the columns of $\mathbf{H}^1_{\pi_0,\mathbf{x}_{0,\overline{\ell_0}},\ell}$ have at most $n \cdot \log t$ non-zero entries. We conclude that

$$\left\|\begin{bmatrix}\mathbf{e}_\mathbf{h}^\intercal, & \mathbf{e}_\mathbf{p}^\intercal, & \tilde{\mathbf{e}}^\intercal\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x},\ell}\right\|_\infty = (n^3 \cdot m^2 \cdot B \cdot \beta)^r \cdot O(T) \cdot \|\tilde{\mathbf{e}}\|_\infty + (\log \beta \cdot n \cdot m)^r \cdot O\left(\|\mathbf{e}_\mathbf{h}\|_\infty + \|\mathbf{e}_\mathbf{p}\|_\infty\right).$$

□

We complete the above proof with the following claim.

Claim 5.5. *Define* $\overline{\ell_{r-1}} := 0$, $\mathbf{H}_{\mathbf{A}} := \mathsf{Lin}(-\mathbf{A})$ *and, for every* $i \in [r]$, $\mathbf{H}_i := (\mathbf{I}_t \otimes \mathbf{x}_{i,\overline{\ell_i}} \otimes \mathbf{I}_n)$. *Then, for every* $1 \leq i < r$, *we have*

$$\begin{aligned}&\mathbf{M}_i + \mathsf{vec}(\mathbf{X}_{i-1,\overline{\ell_i - 1}})^\intercal \otimes \mathbf{G}\\ &\qquad = \left[\mathbf{M}_{i+1} + \mathsf{vec}(\mathbf{X}_{i,\overline{\ell_i}})^\intercal \otimes \mathbf{G}, \quad \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}, \quad \mathbf{Z}\right] \cdot \begin{bmatrix}\mathbf{H}_{\mathbf{A}} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_{\log L \cdot n}\\ \mathbf{H}_i & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\pi_i, \mathbf{x}_{i,\overline{\ell_i}}, \ell}.\end{aligned}$$

Moreover, denoting the ℓ*-th entry of* $\mathbf{x}$ *by* y*, we have*

$$\mathbf{M}_0 + y \cdot \mathbf{G} = \left[\mathbf{M}_1 + \mathsf{vec}(\mathbf{X}_{0,\overline{\ell_0}})^\intercal \otimes \mathbf{G}, \quad \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}, \quad \mathbf{Z}\right] \cdot \begin{bmatrix}\mathbf{H}_{\mathbf{A}} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_{\log L \cdot n}\\ \mathbf{H}_0 & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\pi_0, \mathbf{x}_{0,\overline{\ell_0}}, \ell}.$$

Proof. We proceed by induction over i, starting from $i = r - 1$ and ending with $i = 0$. We observe that

$$\begin{aligned}\mathbf{Z} \cdot \mathbf{H}_i &= \mathbf{Z} \cdot (\mathbf{I}_t \otimes \mathbf{x}_{i,\overline{\ell_i}} \otimes \mathbf{I}_n)\\ &= \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{B}) + \mathbf{I}_t \otimes \mathbf{I}_t \otimes \mathbf{G}) \cdot (\mathbf{I}_t \otimes \mathbf{x}_{i,\overline{\ell_i}} \otimes \mathbf{I}_n)\\ &= \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{B} \cdot (\mathbf{x}_{i,\overline{\ell_i}} \otimes \mathbf{I}_n))) + \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes \mathbf{x}_{i,\overline{\ell_i}} \otimes \mathbf{G})\\ &= \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{X}_{i,\overline{\ell_i}})) + \mathbf{x}^\intercal_{i,\overline{\ell_i}} \otimes \mathbf{G}.\end{aligned}$$

We also notice that, by inductive hypothesis,

$$\begin{aligned}(\mathbf{M}_{i+1} + \mathsf{vec}(\mathbf{X}_{i,\overline{\ell_i}})^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{A}} =&(\mathbf{M}_{i+1} + \mathsf{vec}(\mathbf{X}_{i,\overline{\ell_i}})^\intercal \otimes \mathbf{G}) \cdot \mathsf{Lin}(-\mathbf{A})\\ =&\mathbf{M}_{i+1} \cdot \mathsf{Lin}(-\mathbf{A}) + (\mathsf{vec}(\mathbf{X}_{i,\overline{\ell_i}})^\intercal \otimes \mathbf{G}) \cdot \mathsf{Lin}(-\mathbf{A})\\ =&\mathbf{M}'_i - \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{X}_{i,\overline{\ell_i}})).\end{aligned}$$

By putting our observations together, we obtain that

$$\mathbf{Z} \cdot \mathbf{H}_i + (\mathbf{M}_{i+1} + \mathsf{vec}(\mathbf{X}_{i,\overline{\ell_i}})^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{A}} = \mathbf{M}'_i + \mathbf{x}^\intercal_{i,\overline{\ell_i}} \otimes \mathbf{G}.$$

To complete the proof of the claim, we just observe that

$$\begin{aligned}&\left[\mathbf{M}_{i+1} + \mathsf{vec}(\mathbf{X}_{i,\overline{\ell_i}})^\intercal \otimes \mathbf{G}, \quad \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}, \quad \mathbf{Z}\right] \cdot \begin{bmatrix}\mathbf{H}_{\mathbf{A}} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_{\log L \cdot n}\\ \mathbf{H}_i & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\pi_i, \mathbf{x}_{i,\overline{\ell_i}}, \ell}\\ =&\left[\mathbf{M}'_i + \mathbf{x}^\intercal_{i,\overline{\ell_i}} \otimes \mathbf{G}, \quad \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}\right] \cdot \mathbf{H}_{\pi_i, \mathbf{x}_{i,\overline{\ell_i}}, \ell}\\ =&\mathbf{M}_i + \pi_i(\mathbf{x}_{i,\overline{\ell_i}}, \ell)^\intercal \otimes \mathbf{G}.\end{aligned}$$

If $i \geq 1$, this coincides with $\mathbf{M}_i + \mathsf{vec}(\mathbf{X}_{i-1,\overline{\ell_{i-1}}})^\intercal \otimes \mathbf{G}$. If $i = 0$, we obtain $\mathbf{M}_0 + y \cdot \mathbf{G}$. □

Combining Lemma 5.4 with [HLL23] (see Theorem 3.8) we obtain the following lemma.

Lemma 5.6. *Consider Fig. 4 and Fig. 5 and suppose that $\mathcal{D}_\sigma$ is B-bounded and $\mathcal{D}_{\widetilde{\sigma}}$ is $\widetilde{B}$-bounded. Suppose that*

$$m \cdot n \cdot B_{\mathsf{bst}} + (t \cdot n \cdot B)^{\log L} \cdot \tilde{B} \le B_{\mathsf{max}},$$
$$B_{\mathsf{bst}} = (\sigma' + \overline{\sigma}) \cdot 2^{O(\log^5 \lambda)}, \qquad B_{\mathsf{max}} \le 2^{\frac{1}{3}\log q - \omega(\log \lambda)}.$$

Then, there exist deterministic polynomial time algorithms $\mathsf{CReadPH}$ *and* CReadK *such that, for any $\mathbf{x} \in \mathbb{Z}_q^L$, position $\ell \in [L]$ and noise terms $\mathbf{e_h} \in \mathbb{Z}_q^{n^2 \cdot m}$ and $\mathbf{e_p} \in \mathbb{Z}_q^{n \cdot \log L}$ such that $\|\mathbf{e_h}\|_\infty, \|\mathbf{e_p}\|_\infty \le B_{\mathsf{max}}$, the following holds with $1 - \mathsf{negl}(\lambda)$ probability:*

$$\begin{aligned} y &= \mathbf{x}_\ell \\ \mathbf{c}^\intercal &= \mathbf{s}^\intercal \cdot (\mathbf{C} + y \cdot \mathbf{G}) + \mathbf{e}_\mathbf{c}^\intercal \\ \|\mathbf{e_c}\|_\infty &\le B_{\mathsf{bts}} \end{aligned}$$

where

$$\begin{aligned} (\mathbf{d}, \tau) &= \mathsf{Hash}(\mathsf{pk}, \mathbf{x}) \\ \tau_\ell &= \mathsf{Path}(\tau, \ell) \\ \tilde{\mathbf{e}} &= \mathsf{vec}(\mathbf{E} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G}))) \\ \mathbf{h}^\intercal &= \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}) + \mathbf{e_h}^\intercal \\ \mathbf{p}^\intercal &= \mathbf{s}^\intercal \cdot (\mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}) + \mathbf{e_p}^\intercal \\ \mathbf{z}^\intercal &= \mathbf{s}^\intercal \cdot \mathbf{Z} + \tilde{\mathbf{e}}^\intercal \\ \mathbf{S} &= \begin{bmatrix} \mathbf{W} \\ \mathbf{s}^\intercal \cdot \mathbf{W} + \overline{\mathbf{e}}^\intercal \end{bmatrix} - \mathsf{Bits}(\mathbf{s})^\intercal \otimes \mathbf{I}_{k+1} \otimes \mathbf{g}_q \\ \mathbf{c}_{\mathsf{circ}}^\intercal &= \mathbf{s}^\intercal \cdot (\mathbf{A}_{\mathsf{circ}} + \mathsf{Bits}(\mathbf{S})^\intercal \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{circ}}{}^\intercal \\ (y, \mathbf{c}) &= \mathsf{CReadPH}(\mathsf{pk}, \mathbf{z}, \mathbf{h}, \mathbf{M}, \mathbf{d}, \tau_\ell, \mathbf{p}, \mathbf{P}, \ell, \mathbf{c}_{\mathsf{circ}}, \mathbf{A}_{\mathsf{circ}}, \mathbf{S}) \\ \mathbf{C} &= \mathsf{CReadK}(\mathsf{pk}, \mathbf{M}, \mathbf{P}, \mathbf{A}_{\mathsf{circ}}) \end{aligned}$$

and the probability is taken over the randomness of $\mathsf{pk} := (\mathbf{A}, \overline{\mathbf{B}}, \mathbf{B}, \mathbf{Q}, \mathbf{Z}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *and*

$$\begin{array}{llll} \mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{k \times (m \cdot n^2)} & \mathbf{P} \xleftarrow{\$} \mathbb{Z}_q^{k \times (n \cdot \log L)} & \mathbf{A}_{\mathsf{circ}} \xleftarrow{\$} \mathbb{Z}_q^{k \times (n \cdot (n + \log q)^2)} & \mathbf{W} \xleftarrow{\$} \mathbb{Z}_q^{k \times n \cdot (n + \log q)} \\ \mathbf{s} \xleftarrow{\$} \mathcal{D}_{\sigma'}^k & \overline{\mathbf{e}} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{n^2 + n \cdot \log q} & \mathbf{e}_{\mathsf{circ}} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{n \cdot (n + \log q)^2} & \mathbf{E} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^{t \times (n \cdot t)} \end{array}$$

Finally, $\mathsf{CReadPH}$ *and* CReadK *can be evaluated in time* $\mathrm{poly}(\lambda, \log q)$ *on a RAM.*

5.2 Writing Procedure

For every $i \in [r]$, we define the function $\mu_{\mathbf{B},i}$ that takes as input a vector $\mathbf{x}$ (which length depends on i) and $\ell \in [L]$, and performs the following operations: first of all, it rewrites ℓ as

$$\ell = \ell_0 + \sum_{j=1}^{r-1} \ell_j \cdot t \cdot \beta^{j-1}$$

Succinct Lattice Encodings for RAM Programs – Writing Procedure

$\mathsf{Write}(\mathsf{pk}, \mathbf{h}, \mathbf{M}, \mathbf{d}, \tau_\ell, \mathbf{p}, \mathbf{P}, \ell, \mathbf{c}, \mathbf{C}, y, \mathbf{c}', \mathbf{C}', y')$**:** Define $\ell_0 \in [t]$, $\ell_i \in [\beta]$ and $\overline{\ell_j}$ so that

$$\ell = \ell_0 + \sum_{i=1}^{r-1} \ell_i \cdot t \cdot \beta^{i-1}, \qquad \overline{\ell_j} = \sum_{i=j+1}^{r-1} \ell_i \cdot \beta^{i-j-1}.$$

Rewrite τ_ℓ as $(\mathbf{x}_{i,\overline{\ell_i}})_{i\in[r]}$ and let $\mathbf{x}'_0 \leftarrow y' - y$, $\mathbf{M}'_0 \leftarrow \mathbf{C}' - \mathbf{C}$ and $\mathbf{h}'_0 \leftarrow \mathbf{c}' - \mathbf{c}$. Then, for $i = 0, \ldots, r-1$, compute

1. $\mathbf{x}'_{i+1} \leftarrow \mathsf{vec}(\mathbf{B} \cdot (\mathbf{u}_{\ell_i} \otimes \mathbf{x}'_i \otimes \mathbf{I}_n))$
2. $\mathbf{H}_{\mu_{\mathbf{B},i}, \mathbf{x}'_i, \ell} \leftarrow \mathsf{EvalH}(\mu_{\mathbf{B},i}, \mathbf{M}'_i, \mathbf{P}, \mathbf{x}'_i, \ell)$
3. $\mathbf{K}_{\mu_{\mathbf{B},i}} \leftarrow \mathsf{EvalK}(\mu_{\mathbf{B},i}, \mathbf{M}'_i, \mathbf{P})$
4. ${\mathbf{h}'_{i+1}}^{\intercal} \leftarrow \begin{bmatrix}\mathbf{h}'^{\intercal}_i, & \mathbf{p}^{\intercal}\end{bmatrix} \cdot \mathbf{H}_{\mu_{\mathbf{B},i}, \mathbf{x}'_i, \ell}$
5. $\mathbf{M}'_{i+1} \leftarrow \begin{bmatrix}\mathbf{M}'_i, & \mathbf{P}\end{bmatrix} \cdot \mathbf{K}_{\mu_{\mathbf{B},i}}$

Output $\mathbf{d}' \leftarrow \mathbf{d} + \mathbf{x}'_r$, $\tau'_\ell \leftarrow (\mathbf{x}_{i,\overline{\ell_i}} + \mathbf{x}'_i)_{i\in[r]}$, $\mathbf{h}' \leftarrow \mathbf{h} + \mathbf{h}'_r$ and $\mathbf{M}' \leftarrow \mathbf{M} + \mathbf{M}'_r$.

Figure 6: Succinct Lattice Encodings for RAM Programs – Writing Procedure

where $\ell_0 \in [t]$ and $\ell_j \in [\beta]$ for every $j \geq 1$. If $i \geq 1$, the vector $\mathbf{x}$ provided as input belongs to $\mathbb{Z}_q^{n\cdot m}$. In such case, the function $\mu_{\mathbf{B},i}$ outputs $\mathsf{vec}(\mathbf{B} \cdot (\mathbf{u}_{\ell_i} \otimes \mathbf{x} \otimes \mathbf{I}_n))$ where $\mathbf{u}_{\ell_i}$ denotes the ℓ_i-th element in the standard basis of $\mathbb{Z}_q^{\beta}$. If instead $i = 0$, the element $\mathbf{x}$ received by $\mu_{\mathbf{B},i}$ belongs top $\mathbb{Z}_q$. In this other case, the function outputs $\mathsf{vec}(\mathbf{B} \cdot (\mathbf{u}_{\ell_0} \otimes \mathbf{x} \otimes \mathbf{I}_n))$ where $\mathbf{u}_{\ell_0}$ denotes the ℓ_0-th element in the standard basis of $\mathbb{Z}_q^{t}$. The writing procedure is described in Fig. 6.

The following lemma establishes the correctness of the writing procedure.

Lemma 5.7. *Consider Fig. 4 and Fig. 6 and suppose that $\mathcal{D}_\sigma$ is B-bounded. Then, there exist deterministic polynomial-time algorithms* WriteP, WriteK *and* WriteH *such that, for any*

$$\begin{aligned}
(\mathbf{d}, \tau) &= \mathsf{Hash}(\mathsf{pk}, \mathbf{x}) \\
\tau_\ell &= \mathsf{Path}(\tau, \ell) \\
\mathbf{h}^{\intercal} &= \mathbf{s}^{\intercal} \cdot (\mathbf{M} + \mathbf{d}^{\intercal} \otimes \mathbf{G}) + {\mathbf{e_h}}^{\intercal} \\
\mathbf{z}^{\intercal} &= \mathbf{s}^{\intercal} \cdot \mathbf{Z} + \tilde{\mathbf{e}}^{\intercal} \\
\mathbf{p}^{\intercal} &= \mathbf{s}^{\intercal} \cdot (\mathbf{P} + \mathsf{Bits}(\ell)^{\intercal} \otimes \mathbf{G}) + {\mathbf{e_p}}^{\intercal} \\
\mathbf{c}^{\intercal} &= \mathbf{s}^{\intercal} \cdot (\mathbf{C} + y \cdot \mathbf{G}) + \mathbf{e}_{\mathbf{c}}^{\intercal} \\
\mathbf{c}'^{\intercal} &= \mathbf{s}^{\intercal} \cdot (\mathbf{C}' + y' \cdot \mathbf{G}) + \mathbf{e}_{\mathbf{c}'}^{\intercal} \\
(\mathbf{d}', \tau'_\ell, \mathbf{h}', \mathbf{M}') &= \mathsf{Write}(\mathsf{pk}, \mathbf{h}, \mathbf{M}, \mathbf{d}, \tau_\ell, \mathbf{p}, \mathbf{P}, \ell, \mathbf{c}, \mathbf{C}, y, \mathbf{c}', \mathbf{C}', y')
\end{aligned}$$

where y is the ℓ-th entry in $\mathbf{x}$, it holds that

$$(\mathbf{d}', \tau') = \mathsf{Hash}(\mathsf{pk}, \mathbf{x} + (y' - y) \cdot \mathbf{u}_\ell)$$

$$\begin{aligned}
\tau'_\ell &= \mathsf{Path}(\tau', \ell) \\
\mathbf{h}'^\intercal &= \mathbf{h}^\intercal + \begin{bmatrix}\mathbf{c}'^\intercal - \mathbf{c}^\intercal, & \mathbf{p}^\intercal\end{bmatrix} \cdot \mathbf{H}_{y',y,\ell} \\
&= \mathbf{s}^\intercal \cdot (\mathbf{M}' + \mathbf{d}'^\intercal \otimes \mathbf{G}) + (\mathbf{e}_\mathbf{h}^\intercal + \begin{bmatrix}\mathbf{e}_{\mathbf{c}'}^\intercal - \mathbf{e}_\mathbf{c}^\intercal, & \mathbf{e}_\mathbf{p}^\intercal\end{bmatrix} \cdot \mathbf{H}_{y',y,\ell}) \\
\mathbf{M}' &= \mathbf{M} + \begin{bmatrix}\mathbf{C}' - \mathbf{C}, & \mathbf{P}\end{bmatrix} \cdot \mathbf{K} \\
\|\begin{bmatrix}\mathbf{e}_{\mathbf{c}'}^\intercal - \mathbf{e}_\mathbf{c}^\intercal, & \mathbf{e}_\mathbf{p}^\intercal\end{bmatrix} \cdot \mathbf{H}_{y',y,\ell}\|_\infty &\le (\log(n \cdot B) \cdot n \cdot m)^r \cdot O(\|\mathbf{e}_{\mathbf{c}'} - \mathbf{e}_\mathbf{c}\|_\infty + \beta \cdot n \cdot \|\mathbf{e}_\mathbf{p}\|_\infty).
\end{aligned}$$

where

$$(\mathbf{d}', \tau'_\ell) := \mathsf{WriteP}(\mathsf{pk}, \mathbf{d}, \tau_\ell, y', \ell) \qquad \mathbf{K} := \mathsf{WriteK}(\mathbf{B}, \mathbf{C}', \mathbf{C}, \mathbf{P})$$
$$\mathbf{H}_{y',y,\ell} := \mathsf{WriteH}(\mathbf{B}, \mathbf{C}', \mathbf{C}, \mathbf{P}, y', y, \ell).$$

In particular, it holds that

$$\mathbf{M}' + \mathbf{d}'^\intercal \otimes \mathbf{G} = \mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G} + \begin{bmatrix}\mathbf{C}' - \mathbf{C} + (y' - y) \cdot \mathbf{G}, & \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}\end{bmatrix} \cdot \mathbf{H}_{y',y,\ell}.$$

Finally, WriteP*,* WriteK *and* WriteH *can be evaluated in time* $\mathrm{poly}(\lambda, \log L)$ *on a RAM.*

Proof. We start by observing that Hash is $\mathbb{Z}_q$-linear: All the operations we perform are multiplications by $\mathbf{B}$ and rearranging of the terms. We also observe that

$$(\mathbf{x}'_r, (\delta_{j,\overline{\ell_i}} \cdot \mathbf{x}'_i)_{i\in[r],j\in[\beta^{r-i-1}]}) = \mathsf{Hash}(\mathsf{pk}, (y' - y) \cdot \mathbf{u}_\ell)$$

where $\mathbf{u}_\ell$ the ℓ-th vector of the standard basis of $\mathbb{Z}_q^L$ (indeed, if any $\mathbf{x}_{i,j} = \mathbf{0}$, also $\mathbf{X}_{i,j} = \mathbf{0}$). By relying on the linearity of the hash, we therefore conclude that

$$(\mathbf{d}', \tau') = \mathsf{Hash}(\mathsf{pk}, \mathbf{x} + (y' - y) \cdot \mathbf{u}_\ell).$$

Continuing with the proof, by the homomorphic properties of the BGG^+ evaluation [BGG+14], we easily derive that

$$\mathbf{M}'_{i+1} + {\mathbf{x}'_{i+1}}^\intercal \otimes \mathbf{G} = \begin{bmatrix}\mathbf{M}'_i + \mathbf{x}'^\intercal_i \otimes \mathbf{G}, & \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}\end{bmatrix} \cdot \mathbf{H}_{\mu_{\mathbf{B},i},\mathbf{x}'_i,\ell}.$$

Now, define

$$\mathbf{H}_{y',y,\ell} := \left(\prod_{i=0}^{r-1} \begin{bmatrix}\mathbf{H}^0_{\mu_{\mathbf{B},i},\mathbf{x}'_i,\ell} & \mathbf{0} \\ \mathbf{H}^1_{\mu_{\mathbf{B},i},\mathbf{x}'_i,\ell} & \mathbf{I}_{\log L \cdot n}\end{bmatrix}\right) \cdot \begin{bmatrix}\mathbf{I}_{n^2 \cdot m} \\ \mathbf{0}\end{bmatrix}$$
$$\mathbf{K} := \left(\prod_{i=0}^{r-1} \begin{bmatrix}\mathbf{K}^0_{\mu_{\mathbf{B},i}} & \mathbf{0} \\ \mathbf{K}^1_{\mu_{\mathbf{B},i}} & \mathbf{I}_{\log L \cdot n}\end{bmatrix}\right) \cdot \begin{bmatrix}\mathbf{I}_{n^2 \cdot m} \\ \mathbf{0}\end{bmatrix}$$

where $\mathbf{H}_{\mu_{\mathbf{B},i},\mathbf{x}'_i,\ell} = \begin{bmatrix}\mathbf{H}^0_{\mu_{\mathbf{B},i},\mathbf{x}'_i,\ell} \\ \mathbf{H}^1_{\mu_{\mathbf{B},i},\mathbf{x}'_i,\ell}\end{bmatrix}$ and $\mathbf{K}_{\mu_{\mathbf{B},i}} = \begin{bmatrix}\mathbf{K}^0_{\mu_{\mathbf{B},i},} \\ \mathbf{K}^1_{\mu_{\mathbf{B},i}}\end{bmatrix}$. It is easy to see that

$$\mathbf{M}' - \mathbf{M} + (\mathbf{d}' - \mathbf{d})^\intercal \otimes \mathbf{G} = \begin{bmatrix}\mathbf{M}'_0 + (y' - y) \cdot \mathbf{G}, & \mathbf{P} + \mathsf{Bits}(\ell)^\intercal \otimes \mathbf{G}\end{bmatrix} \cdot \mathbf{H}_{y',y,\ell},$$
$$\mathbf{M}' - \mathbf{M} = \begin{bmatrix}\mathbf{M}'_0, & \mathbf{P}\end{bmatrix} \cdot \mathbf{K}$$

$$\mathbf{h}_r'^{\intercal} = \begin{bmatrix}\mathbf{c}'^{\intercal} - \mathbf{c}^{\intercal}, & \mathbf{p}^{\intercal}\end{bmatrix} \cdot \mathbf{H}_{y',y,\ell}.$$

Finally, we observe that, for any vectors $\mathbf{e}_0$ and $\mathbf{e}_1$ of suitable length, we have

$$\|\mathbf{e}_0^{\intercal} \cdot \mathbf{H}^0_{\mu_{\mathbf{B},i,\mathbf{x}_i',\ell}}\|_\infty \leq \log(n \cdot B) \cdot n \cdot m \cdot \|\mathbf{e}_0\|_\infty$$
$$\|\mathbf{e}_1^{\intercal} \cdot \mathbf{H}^1_{\mu_{\mathbf{B},i,\mathbf{x}_i',\ell}}\|_\infty \leq t \cdot n \cdot \|\mathbf{e}_1\|_\infty.$$

We conclude that

$$\|\begin{bmatrix}\mathbf{e}_{\mathbf{c}'}^{\intercal} - \mathbf{e}_{\mathbf{c}}^{\intercal}, & \mathbf{e}_{\mathbf{p}}^{\intercal}\end{bmatrix} \cdot \mathbf{H}_{y',y,\ell}\|_\infty \leq (\log(n \cdot B) \cdot n \cdot m)^r \cdot O(\|\mathbf{e}_{\mathbf{c}'} - \mathbf{e}_{\mathbf{c}}\|_\infty + \beta \cdot n \cdot \|\mathbf{e}_{\mathbf{p}}\|_\infty).$$

□

5.3 Homomorphic Evaluation of RAM

In this section, we state our main theorems regarding the homomorphic evaluation of RAM programs. In a slight abuse the notation, we write $\mathsf{Path}(\tau, P^T)$ to mean

$$\bigcup_{\ell \in \Omega_{P,T}} \mathsf{Path}(\tau, \ell)$$

where $\Omega_{P,T}$ is the set of all positions $\ell \in [L]$ that may be touched (read or written) during the execution of P^T. The following theorem follows by combining Lemma 5.4 and Lemma 5.7 with [BGG+14], and can be seen as stating a key equation for RAM programs.

Theorem 5.8. *Consider Fig. 4, Fig. 5 and Fig. 6 and suppose that $\mathcal{D}_\sigma$ is B-bounded. There exist deterministic polynomial-time algorithms* $\mathsf{RAMEvalP}$, $\mathsf{RAMEvalH}$ *and* $\mathsf{RAMEvalK}$ *such that, for any $\mathbf{x} \in \mathbb{Z}_2^L$, RAM program P, time $T \in \mathbb{N}$ and $\mathsf{pk} \in \mathsf{Supp}(\mathsf{Setup}(1^\lambda))$, it holds that*

$$\begin{aligned}
(\mathbf{d}', \tau') &= \mathsf{Hash}(\mathsf{pk}, P^T(\mathbf{x}))\\
\tau'_{P,T} &= \mathsf{Path}(\tau', P^T)\\
\begin{bmatrix}\mathbf{h}^{\intercal} & \mathbf{z}^{\intercal}\end{bmatrix} \cdot \mathbf{H}_{P,T,\mathbf{x}} &= \mathbf{s}^{\intercal} \cdot (\mathbf{M} \cdot \mathbf{K}_{P,T} + \mathbf{d}'^{\intercal} \otimes \mathbf{G}) + \begin{bmatrix}\mathbf{e}_{\mathbf{h}}^{\intercal} & \tilde{\mathbf{e}}^{\intercal}\end{bmatrix} \cdot \mathbf{H}_{P,T,\mathbf{x}}\\
\left\|\begin{bmatrix}\mathbf{e}_{\mathbf{h}}^{\intercal} & \tilde{\mathbf{e}}^{\intercal}\end{bmatrix} \cdot \mathbf{H}_{P,T,\mathbf{x}}\right\|_\infty &\leq \mathrm{poly}(\lambda^{T \cdot \log_\beta L}, B^{T \cdot \log_\beta L}) \cdot (\|\mathbf{e}_{\mathbf{h}}\|_\infty + \|\tilde{\mathbf{e}}\|_\infty)
\end{aligned}$$

where

$$\begin{aligned}
(\mathbf{d}, \tau) &= \mathsf{Hash}(\mathsf{pk}, \mathbf{x})\\
\tau_{P,T} &= \mathsf{Path}(\tau, P^T)\\
\mathbf{h}^{\intercal} &= \mathbf{s}^{\intercal} \cdot (\mathbf{M} + \mathbf{d}^{\intercal} \otimes \mathbf{G}) + \mathbf{e_h}^{\intercal}\\
\mathbf{z}^{\intercal} &= \mathbf{s}^{\intercal} \cdot \mathbf{Z} + \tilde{\mathbf{e}}^{\intercal}\\
(\mathbf{d}', \tau'_{P,T}) &= \mathsf{RAMEvalP}(\mathsf{pk}, 1^T, P, \mathbf{d}, \tau_{P,T})\\
\mathbf{H}_{P,T,\mathbf{x}} &= \mathsf{RAMEvalH}(\mathsf{pk}, 1^T, P, \mathbf{M}, \mathbf{d}, \tau_{P,T})\\
\mathbf{K}_{P,T} &= \mathsf{RAMEvalK}(\mathsf{pk}, 1^T, P, \mathbf{M}).
\end{aligned}$$

In particular, we have

$$\mathbf{M} \cdot \mathbf{K}_{P,T} + \mathbf{d}'^{\mathsf{T}} \otimes \mathbf{G} = \begin{bmatrix}\mathbf{M} + \mathbf{d}^{\mathsf{T}} \otimes \mathbf{G}, & \mathbf{Z}\end{bmatrix} \cdot \mathbf{H}_{P,T,\mathbf{x}}.$$

Finally, RAMEvalP, RAMEvalH *and* RAMEvalK *can be evaluated in time* $T \cdot \mathrm{poly}(\lambda, \log L)$ *on a RAM.*

Additionally combining Lemma 5.4 and Lemma 5.7 with [BGG+14] and the bootstrapping procedure of [HLL23], we obtain the following theorem for *unbounded* evaluation of RAM programs.

Theorem 5.9. *Consider Fig. 4, Fig. 5 and Fig. 6 and suppose that* $\mathcal{D}_\sigma$ *is* B*-bounded and* $\mathcal{D}_{\widetilde{\sigma}}$ *is* $\widetilde{B}$*-bounded. Suppose that*

$$m \cdot n \cdot B_{\mathsf{bst}} + (t \cdot n \cdot B)^{\log L} \cdot \tilde{B} \leq B_{\mathsf{max}},$$
$$B_{\mathsf{bst}} = (\sigma' + \overline{\sigma}) \cdot 2^{O(\log^5 \lambda)}, \qquad B_{\mathsf{max}} \leq 2^{\frac{1}{3}\log q - \omega(\log \lambda)}.$$

Then, there exist deterministic polynomial time algorithms RAMCEvalPH *and* RAMCEvalK *such that, for any* $\mathbf{x} \in \mathbb{Z}_2^L$*, RAM program* P*, time* $T \in \mathbb{N}$ *and noise term* $\mathbf{e_h} \in \mathbb{Z}_q^{n^2 \cdot m}$ *such that* $\|\mathbf{e_h}\|_\infty \leq B_{\mathsf{max}}$*, the following holds with* $1 - \mathsf{negl}(\lambda)$ *probability:*

$$\begin{aligned}
(\mathbf{d}', \tau') &= \mathsf{Hash}(\mathsf{pk}, P^T(\mathbf{x})) \\
\tau'_{P,T} &= \mathsf{Path}(\tau', P^T) \\
\mathbf{h}'^{\mathsf{T}} &= \mathbf{s}^{\mathsf{T}} \cdot (\mathbf{M}' + \mathbf{d}'^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}_{\mathbf{h}'}^{\mathsf{T}} \\
\|\mathbf{e}_{\mathbf{h}'}\|_\infty &\leq B_{\mathsf{bts}}
\end{aligned}$$

where

$$\begin{aligned}
(\mathbf{d}, \tau) &= \mathsf{Hash}(\mathsf{pk}, \mathbf{x}) \\
\tau_{P,T} &= \mathsf{Path}(\tau, P^T) \\
\tilde{\mathbf{e}} &= \mathsf{vec}(\mathbf{E} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G}))) \\
\mathbf{h}^{\mathsf{T}} &= \mathbf{s}^{\mathsf{T}} \cdot (\mathbf{M} + \mathbf{d}^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e_h}^{\mathsf{T}} \\
\mathbf{z}^{\mathsf{T}} &= \mathbf{s}^{\mathsf{T}} \cdot \mathbf{Z} + \tilde{\mathbf{e}}^{\mathsf{T}} \\
\mathbf{S} &= \begin{bmatrix} \mathbf{W} \\ \mathbf{s}^{\mathsf{T}} \cdot \mathbf{W} + \overline{\mathbf{e}}^{\mathsf{T}} \end{bmatrix} - \mathsf{Bits}(\mathbf{s})^{\mathsf{T}} \otimes \mathbf{I}_{k+1} \otimes \mathbf{g}_q \\
\mathbf{c}_{\mathsf{circ}}^{\mathsf{T}} &= \mathbf{s}^{\mathsf{T}} \cdot (\mathbf{A}_{\mathsf{circ}} + \mathsf{Bits}(\mathbf{S})^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{circ}}{}^{\mathsf{T}} \\
(\mathbf{d}', \tau'_{P,T}, \mathbf{h}') &= \mathsf{RAMCEvalPH}(\mathsf{pk}, 1^T, P, \mathbf{z}, \mathbf{h}, \mathbf{M}, \mathbf{d}, \tau_{P,T}, \mathbf{c}_{\mathsf{circ}}, \mathbf{A}_{\mathsf{circ}}, \mathbf{S}) \\
\mathbf{M}' &= \mathsf{RAMCEvalK}(\mathsf{pk}, 1^T, P, \mathbf{M}, \mathbf{A}_{\mathsf{circ}})
\end{aligned}$$

and the probability is taken over the randomness of $\mathsf{pk} := (\mathbf{A}, \overline{\mathbf{B}}, \mathbf{B}, \mathbf{Q}, \mathbf{Z}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *and*

$$\mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{k \times (m \cdot n^2)} \qquad \mathbf{A}_{\mathsf{circ}} \xleftarrow{\$} \mathbb{Z}_q^{k \times (n \cdot (n + \log q)^2)} \qquad \mathbf{W} \xleftarrow{\$} \mathbb{Z}_q^{k \times (n^2 + n \cdot \log q)}$$
$$\mathbf{s} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{k} \qquad \overline{\mathbf{e}} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{n^2 + n \cdot \log q} \qquad \mathbf{e}_{\mathsf{circ}} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{n \cdot (n + \log q)^2} \qquad \mathbf{E} \xleftarrow{\$} \mathcal{D}_{\tilde{\sigma}}^{t \times (n \cdot t)}$$

Finally, RAMCEvalPH *and* RAMCEvalK *can be evaluated in time* $T \cdot \mathrm{poly}(\lambda, \log q)$ *on a RAM.*

6 Rate-1 AB-LFE for RAM Programs

In this section, we construct a rate-1 AB-LFE for RAM.

6.1 Definition

We recall the definition of attribute-based laconic function evaluation (AB-LFE) [QWW18] for RAM programs. For convenience, we define only the version for single-bit messages, but it is straightforward to generalize it to strings of arbitrary length.

Definition 6.1 (Attribute-Based Laconic Function Evaluation for RAM). *Let $\ell := \ell(\lambda)$ and $L := L(\lambda)$ be positive integers. An AB-LFE scheme for RAM consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax.*

$\mathsf{Setup}(1^\lambda)$: *The setup algorithm is probabilistic, it takes as input the security parameter 1^λ and outputs a public key* pk.

$\mathsf{Compress}(\mathsf{pk}, 1^T, P, \mathbf{y})$: *The compression algorithm is deterministic, it takes as input a public key, a running time 1^T, the description of a RAM program P and an initial state for the RAM $\mathbf{y} \in \mathbb{Z}_2^{\leq L-\ell}$. The output is a digest h.*

$\mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, \mu)$: *The encryption algorithm is probabilistic, it takes as input a public key* pk*, a digest h, an attribute $\mathbf{x} \in \mathbb{Z}_2^{\leq \ell}$, an index $j \in [L]$, a value α, and a message $\mu \in \{0, 1\}$. The output is an encoding E.*

$\mathsf{Dec}(\mathsf{pk}, E, 1^T, P, \mathbf{y})$: *The decoding procedure is deterministic, it takes as input a public key* pk*, an encoding E, a running time 1^T, a RAM program P and its initial state $\mathbf{y}$. The output is a message $\mu \in \{0, 1, \perp\}$.*

For correctness, we require that there must exist a negligible function $\mathsf{negl}(\lambda)$ such that, for every sufficiently large λ, every RAM program P, every running time $T \in \mathbb{N}$, every $\mathbf{x} \in \mathbb{Z}_2^{\leq \ell}$ and $\mathbf{y} \in \mathbb{Z}_2^{\leq L-\ell}$, index $j \in [L]$ and value α such that the j-th element in the state of $P^T(\mathbf{x}, \mathbf{y})$ is α, and every message $\mu \in \{0, 1\}$, it holds that:

$$\Pr\left[\mathsf{Dec}(\mathsf{pk}, E, 1^T, P, \mathbf{y}) \neq \mu\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the randomness of the procedures $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $h \leftarrow \mathsf{Compress}(\mathsf{pk}, 1^T, P, \mathbf{y})$ and $E \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, \mu)$. We recall the definition of selective security.

Definition 6.2 (Selective Security). *An AB-LFE scheme* $(\mathsf{Setup}, \mathsf{Compress}, \mathsf{Enc}, \mathsf{Dec})$ *is selectively secure if for every $\mathbf{x} \in \mathbb{Z}_2^{\leq \ell}$, $\mathbf{y} \in \mathbb{Z}_2^{\leq L-\ell}$ and PPT adversary $\mathcal{A}$ that outputs a tuple $(j, \alpha, P, 1^T, \mathsf{aux})$ where the j-th entry in the state of $P^T(\mathbf{x}, \mathbf{y})$ is not α, the following distributions are computationally indistinguishable:*

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, 0), \mathsf{aux}) \approx_c (\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, 1), \mathsf{aux})$$

where $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $h \leftarrow \mathsf{Compress}(\mathsf{pk}, 1^T, P, \mathbf{y})$ and $(j, \alpha, P, 1^T, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathsf{pk})$.

For a security parameter λ and a RAM program with running time T and memory L, we require that the algorithms of AB-LFE satisfy the following efficiency requirements:

- The runtime of the Setup algorithm is bounded by $\mathrm{poly}(\lambda, \log L)$.
- The runtime of the Enc algorithm is bounded by $|\mathbf{x}| \cdot \mathrm{poly}(\lambda, \log L)$. Furthermore, the Enc algorithm is split into two subroutines:

$$\mathbf{d} \leftarrow \mathsf{Hash}(\mathsf{pk}, \mathbf{x}) \quad \text{and} \quad \tilde{E} \xleftarrow{\$} \mathsf{Complete}(\mathsf{pk}, h, \mathbf{d}, j, \alpha, \mu)$$

  where the runtime of Hash is bounded by $\mathrm{poly}(\lambda, \log L)$, and the final encoding consists of $E := (\mathbf{x}, \tilde{E})$.
- The runtime of the $\mathsf{Compress}$ algorithm is bounded by $(T + |\mathbf{y}|) \cdot \mathrm{poly}(\lambda, \log L)$ and its memory is bounded by $\mathrm{poly}(\lambda, \log L, |\mathbf{y}|)$.
- The runtime of the Dec algorithms is bounded by $(T + |\mathbf{x}|) \cdot \mathrm{poly}(\lambda, \log L)$.

6.2 Construction

Our construction is described in Fig. 7. We only present the construction for unbounded RAM programs, which requires assuming the hardness of a circular variant of the decomposed LWE assumption (Definition 4.5). It is straightforward to modify the scheme so that the runtime of the RAM program is a-priori bounded (and the ciphertext grows with such runtime) and this variant can be proven secure only invoking the hardness of the plain decomposed LWE.

Let $q \geq 3\sqrt{\lambda} \cdot \sigma' \cdot 2^{\lambda}$. To see that the construction in Fig. 7 is correct, we observe that, by the linearity of the hash function, it holds that $(\mathbf{d}, \tau) = \mathsf{E.Hash}(\mathsf{pk}', (\mathbf{x}, \mathbf{y}))$. In other words, $\mathbf{h}$ is a succinct encoding for the concatenation of $\mathbf{x}$ and $\mathbf{y}$. As a consequence, due to Theorem 5.9 and Lemma 5.6, we have that

$$\mathbf{c}^{\intercal} = \mathbf{s}^{\intercal} \cdot (\mathbf{A}_{P,T}^{j,\alpha} + (v_j - \alpha) \cdot \mathbf{G}) + \mathbf{e}_{P,T}^{\intercal}$$

where v_j the j-th entry of $P^T(\mathbf{x}, \mathbf{y})$ and $\|\mathbf{e}_{P,T}\|_\infty \leq B_{\mathsf{bst}}$. In other words, if $v_j = \alpha$, we have that

$$\mathbf{c}^{\intercal} = \mathbf{s}^{\intercal} \cdot \mathbf{A}_{P,T}^{j,\alpha} + \mathbf{e}_{P,T}^{\intercal}.$$

Continuing with the analysis, we obtain that

$$\begin{aligned} c' - \mathbf{c}^{\intercal} \cdot \mathbf{G}^{-1}(\mathbf{u}) &= \mathbf{s}^{\intercal} \cdot \mathbf{A}_{P,T}^{j,\alpha} \cdot \mathbf{G}^{-1}(\mathbf{u}) + e' + \lceil q/2 \rfloor \cdot \mu - \mathbf{s}^{\intercal} \cdot \mathbf{A}_{P,T}^{j,\alpha} \cdot \mathbf{G}^{-1}(\mathbf{u}) - \mathbf{e}_{P,T}^{\intercal} \cdot \mathbf{G}^{-1}(\mathbf{u}) \\ &= \lceil q/2 \rfloor \cdot \mu + e' - \mathbf{e}_{P,T}^{\intercal} \cdot \mathbf{G}^{-1}(\mathbf{u}). \end{aligned}$$

We conclude by observing that

$$|e' - \mathbf{e}_{P,T}^{\intercal} \cdot \mathbf{G}^{-1}(\mathbf{u})| \leq \sqrt{\lambda} \cdot \sigma' \cdot 2^{\lambda} + n \cdot B_{\mathsf{bst}} < q/4.$$

In other words, $\lceil (c' - \mathbf{c}^{\intercal} \cdot \mathbf{G}^{-1}(\mathbf{u}) \rfloor_2 = \mu$.

Next, we show that the scheme satisfies the standard definition of selective security.

Rate-1 AB-LFE for RAM Programs

$\mathsf{Setup}(1^\lambda)$**:** Compute $\mathsf{pk}' := (\mathbf{A}, \overline{\mathbf{B}}, \mathbf{B}, \mathbf{Q}, \mathbf{Z}) \xleftarrow{\$} \mathsf{E.Setup}(1^\lambda)$, $\mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{k\times(n\cdot m)}$ and $\mathbf{A}_{\mathsf{circ}} \xleftarrow{\$} \mathbb{Z}_q^{k\times n\cdot(n+\log q)^2}$. Output the public key $\mathsf{pk} := (\mathbf{A}, \overline{\mathbf{B}}, \mathbf{B}, \mathbf{Q}, \mathbf{Z}, \mathbf{M}, \mathbf{A}_{\mathsf{circ}})$

$\mathsf{Compress}(\mathsf{pk}, 1^T, P, \mathbf{y})$**:** Compute $(\mathbf{d}_0, \tau_0) \leftarrow \mathsf{E.Hash}(\mathsf{pk}', \mathbf{y}')$ where $\mathbf{y}'$ is obtained by padding $\mathbf{y}$ with ℓ zeros at the beginning. Compute

$$\mathbf{A}_{P,T} \leftarrow \mathsf{RAMCEvalK}(\mathsf{pk}', 1^T, P, \mathbf{M}, \mathbf{A}_{\mathsf{circ}})$$

Output $h := (\mathbf{A}_{P,T}, \mathbf{d}_0)$.

$\mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, \mu)$**:** Compute $(\mathbf{d}_1, \tau_1) \leftarrow \mathsf{E.Hash}(\mathsf{pk}', \mathbf{x})$ and set $\mathbf{d} \leftarrow \mathbf{d}_0 + \mathbf{d}_1$. Then, sample $\mathbf{s} \xleftarrow{\$} \mathcal{D}_{\sigma'}^k$ and $\mathbf{W} \xleftarrow{\$} \mathbb{Z}_2^{k\times n\cdot(n^2+n\cdot\log q)}$ and compute

$$\begin{aligned}
\mathbf{S} &\leftarrow \begin{bmatrix} \mathbf{W} \\ \mathbf{s}^\intercal \cdot \mathbf{W} + \overline{\mathbf{e}}^\intercal \end{bmatrix} - \mathsf{Bits}(\mathbf{s})^\intercal \otimes \mathbf{I}_{k+1} \otimes \mathbf{g}^\intercal \\
\mathbf{h}^\intercal &\leftarrow \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal \\
\mathbf{z}^\intercal &\leftarrow \mathbf{s}^\intercal \cdot \mathbf{Z} + \tilde{\mathbf{e}}^\intercal \\
\mathbf{c}_{\mathsf{circ}}^\intercal &\leftarrow \mathbf{s}^\intercal \cdot (\mathbf{A}_{\mathsf{circ}} + \mathsf{Bits}(\mathbf{S})^\intercal \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{circ}}^\intercal
\end{aligned}$$

where $\mathbf{e} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{n^2\cdot m}$, $\mathbf{e}_{\mathsf{circ}} \xleftarrow{\$} \mathcal{D}_{\sigma'}^{n\cdot(n+\log q)^2}$, $\overline{\mathbf{e}} \xleftarrow{\$} \mathcal{D}_{\overline{\sigma}}^{n\cdot(n+\log q)}$, $\mathbf{E} \xleftarrow{\$} \mathcal{D}_{\widetilde{\sigma}}^{t\times(t\cdot n)}$ and

$$\tilde{\mathbf{e}} \leftarrow \mathsf{vec}(\mathbf{E} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G}))).$$

Finally, let R_j be the RAM program that just outputs the j-th entry in its state and let T' be its running time. Set

$$\begin{aligned}
\mathbf{A}_{P,T}^{j,\alpha} &\leftarrow \mathsf{CReadK}(\mathsf{pk}', \mathbf{A}_{P,T}, -\mathsf{Bits}(j)^\intercal \otimes \mathbf{G}, \mathbf{A}_{\mathsf{circ}}) + \alpha \cdot \mathbf{G} \\
c' &\leftarrow \mathbf{s}^\intercal \cdot \mathbf{A}_{P,T}^{j,\alpha} \cdot \mathbf{G}^{-1}(\mathbf{u}) + e' + \lceil q/2 \rfloor \cdot \mu
\end{aligned}$$

where $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^k$ and $e' \xleftarrow{\$} \mathcal{D}_{2^\lambda\cdot\sigma'}$. Output $E := (\mathbf{x}, \mathbf{h}, \mathbf{z}, \mathbf{S}, \mathbf{c}_{\mathsf{circ}}, \mathbf{u}, c', j, \alpha)$.

$\mathsf{Dec}(\mathsf{pk}, E, 1^T, P, \mathbf{y})$**:** If the j-th entry of $P^T(\mathbf{x}, \mathbf{y})$ is not α, output $\perp$. Otherwise, compute $(\mathbf{d}, \tau) \leftarrow \mathsf{E.Hash}(\mathsf{pk}', (\mathbf{x}, \mathbf{y}))$ and

$$\begin{aligned}
\mathbf{A}_{P,T} &\leftarrow \mathsf{RAMCEvalK}(\mathsf{pk}', 1^T, P, \mathbf{M}, \mathbf{A}_{\mathsf{circ}}) \\
(\mathbf{d}', \tau', \mathbf{h}') &\leftarrow \mathsf{RAMCEvalPH}(\mathsf{pk}', 1^T, P, \mathbf{z}, \mathbf{h}, \mathbf{M}, \mathbf{d}, \tau, \mathbf{c}_{\mathsf{circ}}, \mathbf{A}_{\mathsf{circ}}, \mathbf{S}) \\
(v_j, \mathbf{c}) &\leftarrow \mathsf{CReadPH}(\mathsf{pk}', \mathbf{z}, \mathbf{h}', \mathbf{A}_{P,T}, \mathbf{d}', \tau', \mathbf{0}, -\mathsf{Bits}(j)^\intercal \otimes \mathbf{G}, j, \mathbf{c}_{\mathsf{circ}}, \mathbf{A}_{\mathsf{circ}}, \mathbf{S}).
\end{aligned}$$

Output $\lceil c' - \mathbf{c}^\intercal \cdot \mathbf{G}^{-1}(\mathbf{u}) \rfloor_2$.

Figure 7: Rate-1 AB-LFE for RAM Programs

Theorem 6.3. *Assuming the hardness of the small-secret circular decomposed LWE problem, the construction in Fig. 7 is a selectively secure AB-LFE scheme for RAM programs.*

Proof. Suppose that the j-th entry of $P^T(\mathbf{x}, \mathbf{y})$ (let it be v_j) is not equal to α. We proceed by means of an hybrid argument, showing that the distributions

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, 0), \mathsf{aux}) \qquad\qquad (\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, 1), \mathsf{aux})$$

are both computationally indistinguishable from the distribution in Hybrid 4.

- Hybrid 0: This corresponds to $(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}, j, \alpha, b), \mathsf{aux})$.
- Hybrid 1: We modify the distribution of c'. Specifically, we set

$$c' \leftarrow \lceil q/2 \rfloor \cdot \mu + e' + \mathbf{c}^\intercal \cdot \mathbf{G}^{-1}(\mathbf{u}) - \mathbf{s}^\intercal \cdot \mathbf{u} - \mathbf{e}_{P,T}^\intercal \cdot \mathbf{G}^{-1}(\mathbf{u}).$$

  This hybrid is identical to the previous one. Indeed,

$$\mathbf{c}^\intercal \cdot \mathbf{G}^{-1}(\mathbf{u}) - (v_j - \alpha) \cdot \mathbf{s}^\intercal \cdot \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{u}) - \mathbf{e}_{P,T}^\intercal \cdot \mathbf{G}^{-1}(\mathbf{u}) = \mathbf{s}^\intercal \cdot \mathbf{A}_{P,T}^{j,\alpha} \cdot \mathbf{G}^{-1}(\mathbf{u}).$$

- Hybrid 2: We modify the distribution of c'. Specifically, we set

$$c' \leftarrow \lceil q/2 \rfloor \cdot \mu + e' - e'' + \mathbf{c}^\intercal \cdot \mathbf{G}^{-1}(\mathbf{u}) - \mathbf{s}^\intercal \cdot \mathbf{u}$$

  where $e'' \xleftarrow{\$} \mathcal{D}_{\sigma'}$. This hybrid is statistically indistinguishable from the previous one as e' is sampled from a discrete Gaussian distribution with parameter 2^λ times bigger than σ'.

- Hybrid 3: we modify the distribution of $\mathbf{M}$ and $\mathbf{A}_{\mathsf{circ}}$: we sample $\mathbf{M}' \xleftarrow{\$} \mathbb{Z}_q^{k\times(n^2\cdot m)}$, $\mathbf{A}'_{\mathsf{circ}} \xleftarrow{\$} \mathbb{Z}_q^{k\times n\cdot(n+\log q)^2}$ and we set $\mathbf{M} \leftarrow \mathbf{M}' - \mathbf{d}^\intercal \otimes \mathbf{G}$ and $\mathbf{A}_{\mathsf{circ}} \leftarrow \mathbf{A}'_{\mathsf{circ}} - \mathsf{Bits}(\mathbf{S})^\intercal \otimes \mathbf{G}$. This hybrid is perfectly indistinguishable from Hybrid 2 (notice that this works only because we are proving selective security).

- Hybrid 4: We modify the distribution of $(\mathbf{h}, \mathbf{z}, \mathbf{S}, \mathbf{c}_{\mathsf{circ}}, c')$: we sample all of the at random except for $\mathbf{z}$ which we set to $\mathsf{vec}(\mathbf{U} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))$ where $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{t\times(n\cdot t)}$. This hybrid is indistinguishable from Hybrid 3 under small-secret circular decomposed LWE. Notice that c' looks random because it is masked by $\mathbf{s}^\intercal \cdot \mathbf{u} + e''$. The reduction works as follows: given a tuple $(\mathbf{A}, \overline{\mathbf{B}}, \mathbf{Q}, \mathbf{M}, \mathbf{v}^\intercal, \mathbf{U}, \mathbf{S})$, we set $\mathbf{B} \leftarrow \overline{\mathbf{B}} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G}))$ and $\mathbf{z} \leftarrow \mathsf{vec}(\mathbf{U} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))$. Then, we split $\mathbf{M}$ into three blocks $\mathbf{A}'_{\mathsf{circ}} \in \mathbb{Z}_q^{k\times n\cdot(n+\log q)^2}$, $\mathbf{M}' \in \mathbb{Z}_q^{k\times(n^2\cdot m)}$ and $\mathbf{u} \in \mathbb{Z}_q^k$. We split $\mathbf{v}$ accordingly: we derive $\mathbf{c}_{\mathsf{circ}} \in \mathbb{Z}_q^{n\cdot(n+\log q)^2}$, $\mathbf{h} \in \mathbb{Z}_q^{n^2\cdot m}$ and $c'' \in \mathbb{Z}_q$. The rest is computed as in Hybrid 3. The only exception is c' which is computed as $c' \leftarrow \lceil q/2 \rfloor \cdot \mu + e' + \mathbf{c}^\intercal \cdot \mathbf{G}^{-1}(\mathbf{u}) - c''$. Observe that if we received a real circular decomposed LWE sample, the view of the adversary is as in Hybrid 3. Otherwise, the view of the adversary is as in Hybrid 4. Indeed, notice that

$$\mathbf{z}^\intercal = \mathbf{s}^\intercal \cdot \mathbf{Z} + \tilde{\mathbf{e}}^\intercal$$

$$
\begin{aligned}
&= \mathbf{s}^\intercal \cdot \mathbf{\Pi} \cdot (\mathbf{I}_t \otimes (\mathbf{A} \cdot \mathbf{B}) + \mathbf{I}_t \otimes \mathbf{I}_t \otimes \mathbf{G}) + \mathsf{vec}(\mathbf{E} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))^\intercal \\
&= \mathsf{vec}((\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot (\mathbf{A} \cdot \mathbf{B} + \mathbf{I}_t \otimes \mathbf{G}))^\intercal + \mathsf{vec}(\mathbf{E} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))^\intercal \\
&= \mathsf{vec}(((\mathbf{I}_t \otimes \mathbf{s}^\intercal) \cdot (\mathbf{A} \cdot \overline{\mathbf{B}} + \mathbf{I}_t \otimes (\mathbf{Q} \cdot \mathbf{G})) + \mathbf{E}) \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))^\intercal \\
&\approx_c \mathsf{vec}(\mathbf{U} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))^\intercal.
\end{aligned}
$$

Given that the view of the adversary in Hybrid 4 is independent of the message, the proof is concluded. □

6.3 Fully-Secure LFE for RAM

Our AB-LFE scheme for RAM can be easily turned into a full-blown LFE [QWW18], i.e., the decoder will now receive $P^T(\mathbf{x}, \mathbf{y})$ without learning any additional information about $\mathbf{x}$. We can obtain this using two approaches, both of them summarised in [QWW18], that we sketch below.

The Garbled Circuit Approach. The first approach is due Goldwasser et al. [GKP+13b]: We encrypt $\mathbf{x}$ using RAM-FHE [LMW23] and we garble the decryption circuit with the secret-key hardcoded. Then, we use AB-LFE to encrypt the wire labels and ensure that we only reveal those associated with the encryption of $P^T(\mathbf{x}, \mathbf{y})$.

This construction can be made rate-1 in $\mathbf{x}$ by using hybrid encryption. However, both the size of the LFE hash and that of the encoder message will suffer from a $\text{poly}(\lambda)$ multiplicative overhead in the output size.

The Dual Use Approach. The other solution is to make a non-black-box use of our AB-LFE scheme and rely on the trick of [BTVW17]. Specifically, referring to Fig. 7, we now drop μ and its encryption c'. The encoder will generate a RAM-FHE encryption [LMW23] of $\mathbf{x}$ (we make it rate-1 in $|\mathbf{x}|$ using hybrid encryption) and will produce an AB-LFE encoding for such ciphertext. Instead of homomorphically evaluating P, we will homomorphically evaluate the RAM-FHE evaluation of P on the ciphertext. In other words, the decoder will derive a BGG^+ encoding [BGG+14] of the RAM-FHE encryption of $P^T(\mathbf{x}, \mathbf{y})$ in time proportional to $T \cdot \text{poly}(\lambda, \log L)$. To ensure decryption, we make the encoder produce also a GSW encryption [GSW13] of the RAM-FHE secret key $\mathsf{sk}_{\mathsf{ram\text{-}fhe}}$ using $\mathbf{s}$ as decryption key, along with a BGG^+ encoding of such ciphertext using $\mathbf{s}$ as a secret. Using this material, the decoder is therefore able to derive a BGG^+ encoding of the GSW encryption of $P^T(\mathbf{x}, \mathbf{y})$ (we are performing the RAM-FHE decryption inside GSW, inside BGG^+ encodings). We recall that the decryption key for such GSW ciphertext is the vector $\mathbf{s}$ under which the BGG^+ encoding is generated. Therefore, using the trick of [BTVW17], we can obliviously perform the decryption inside the encoding, obtaining

$$\tilde{\mathbf{c}}^\intercal = \mathbf{s}^\intercal \cdot \mathbf{A}''_{P,T} - P^T(\mathbf{x}, \mathbf{y})^\intercal \cdot \lceil q/2 \rfloor + \tilde{\mathbf{e}}^\intercal$$

where $\mathbf{A}''_{P,T}$ is a matrix known to the decoder and computable in time $T \cdot \text{poly}(\lambda, \log L)$ given P and T, and $\tilde{\mathbf{e}}$ is a noise vector with ℓ_∞-norm smaller than $q/4$. To decrypt, we need to provide the decoder with information about $\mathbf{s}^\intercal \cdot \mathbf{A}''_{P,T}$. Notice that this vector has size $|P^T(\mathbf{x},\mathbf{y})| \cdot \log q$.

To make the scheme rate-1 in the output size, we therefore rely on a succinct MOLE[2] as in [AMR25, BJSS25, ARS24]: The decoder will compute a MOLE hash $h_{\mathsf{MOLE}} \leftarrow \mathsf{MOLE.Hash}(\mathbf{A}''_{P,T})$ at hashing time, sending it to the encoder. The encoder will augment its message with a MOLE encoding of $\mathbf{s}$ and a *rounded* MOLE encoder share $\mathbf{c}_0 \leftarrow \lceil \mathsf{MOLE.EncEval}(h_{\mathsf{MOLE}}, \mathbf{s}) \rfloor_2$. The decoder will then retrieve $\mathbf{c}_1 \leftarrow \left\lceil \mathsf{MOLE.HashEval}(E_{\mathsf{MOLE}}, \mathbf{A}''_{P,T}) \right\rfloor_2$ which satisfies

$$\mathbf{c}_0 \oplus \mathbf{c}_1 = \left\lceil \mathbf{s}^\intercal \cdot \mathbf{A}''_{P,T} \right\rfloor_2 .$$

Finally, it will obtain $P^T(\mathbf{x},\mathbf{y}) = \mathbf{c}_0 \oplus \mathbf{c}_1 \oplus \lceil \tilde{\mathbf{c}} \rfloor_2$ (we are relying on the "distributed rounding" trick of [DHRW16]). We have obtained an LFE scheme for RAM programs where the hash size is $\mathrm{poly}(\lambda, \log L)$ and the encoding size is $|\mathbf{x}| + |P^T(\mathbf{x},\mathbf{y})| + \mathrm{poly}(\lambda, \log L)$. Moreover, the decoder running time is $(T + |\mathbf{x}| + |\mathbf{y}|) \cdot \mathrm{poly}(\lambda, \log L)$.

7 Succinct Randomised Encodings

In the following we present our construction of succinct randomised encodings (SRE).

7.1 Witness Encryption for P

As a first step, we define the notion of *witness encryption* [GGSW13] for RAM programs. In more details, a witness encryption scheme consists of two algorithms $\mathsf{WE.Enc}$ and $\mathsf{WE.Dec}$ where $\mathsf{WE.Enc}(1^\lambda, T, P, \mathbf{x}, \alpha, \mu)$ takes as input the security parameter 1^λ, a running time T, a RAM program P, its initial state $\mathbf{x}$, a target vector α and a message μ and returns a ciphertext c. We require that:

$$\mathsf{WE.Dec}(1^T, P, \mathbf{x}, i, c) = \mu_i$$

if the i-th bit of $P^T(\mathbf{x})$ is α_i. We define security next.

Definition 7.1 (Semantic Security)**.** *We say that a witness encryption scheme* ($\mathsf{WE.Enc}$, $\mathsf{WE.Dec}$) *is secure, if for all messages* $(\mu^{(0)}, \mu^{(1)})$*, all programs* P*, all running time* T*, all initial states* $\mathbf{x}$ *and all targets* α *such that* $\mu_i^{(0)} = \mu_i^{(1)}$ *whenever the* i*-th bit of* $P^T(\mathbf{x})$ *coincides with* α_i*, it holds that the following distributions are computationally indistinguishable:*

$$\mathsf{WE.Enc}(1^\lambda, T, P, \mathbf{x}, \alpha, \mu^{(0)}) \approx_c \mathsf{WE.Enc}(1^\lambda, T, P, \mathbf{x}, \alpha, \mu^{(1)}).$$

Note that the notion is trivial to achieve if one does not place any further constraints: The encryption algorithm can simply run P and check if the i-th output entry is α_i, returning μ_i if that is the case and $\perp$ otherwise. In this work, we are interested in the settings where the runtime of $\mathsf{WE.Enc}$ is much smaller (ideally poly-logarithmic) than that of P.

[2]A succinct MOLE (matrix oblivious linear evaluation) consists of a primitive that allows two parties, a hasher holding a matrix $\mathbf{A} \in \mathbb{Z}_q^{k \times t}$ and an encoder holding a vector $\mathbf{s} \in \mathbb{Z}_q^k$, to derive an additive secret-sharing of $\mathbf{s}^\intercal \cdot \mathbf{A}$ using one round of simultaneous interaction and total communication $k \cdot \mathrm{poly}(\lambda, \log t)$. The primitive can be built from LWE.

7.2 Construction of Witness Encryption for P

Suppose that μ has size m. Our compilation process starts from the following ingredients:

- A garbling scheme for circuits $(\mathsf{Garble}, \mathsf{Eval})$.
- The ABE-LFE for RAM programs $(\mathsf{Setup}, \mathsf{Compress}, \mathsf{Hash}, \mathsf{Complete}, \mathsf{Dec})$ constructed in Section 6. Given that we consider inputless programs, we omit the parameter $\mathbf{y}$ from the inputs of Hash. Let $\ell := |h|$, where h is the output of the $\mathsf{Compress}$ algorithm. Notice that $\ell \leq \mathrm{poly}(\lambda)$.

As the first step, we reduce the memory of the encryption algorithm to a polynomial independent of the memory bound L. The new witness encryption ciphertext consists of $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ and $\tilde{C}, (\mathsf{lab}_{j,h_j})_{j\in[\ell]}$, where

$$h \leftarrow \mathsf{Compress}(\mathsf{pk}, 1^{T'}, U), \qquad \left(\tilde{C}, (\mathsf{lab}_{j,b})_{\substack{j\in[\ell] \\ b\in\{0,1\}}}\right) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C)$$

and C is the circuit that, on input h, returns

$$\left(E_{i,\alpha_i} \xleftarrow{\$} \mathsf{Complete}(\mathsf{pk}, h, \mathbf{d}, i, \alpha_i, \mu_i)\right)_{i\in[m]}$$

for $\mathbf{d} := \mathsf{Hash}(\mathsf{pk}, (\mathbf{x}, P))$. Above, U denotes the universal RAM and T' denotes the time required for U to evaluate $P^T(\mathbf{x})$. Notice that $T' \leq T \cdot O(|P|)$. Moreover, the operation will have a $O(|P|)$ additive overhead in memory. The decryption algorithm simply evaluates the garbled circuit to recover the encoding E, which is decoded to μ_i if the i-th bit of $P^T(\mathbf{x})$ is α_i. Security follows immediately from the security of AB-LFE and the garbling scheme.

We can also see that, the encryption algorithm requires memory

$$(|x| + |P|) \cdot \mathrm{poly}(\lambda) + m \cdot \mathrm{poly}(\lambda),$$

by the efficiency of the AB-LFE. Furthermore, except for $\mathsf{Compress}(\mathsf{pk}, 1^{T'}, U)$, which runs in $T \cdot \mathrm{poly}(\lambda) \cdot |P|$ steps, the runtime of all other operations is at most

$$(|x| + |P|) \cdot \mathrm{poly}(\lambda) + m \cdot \mathrm{poly}(\lambda).$$

Finally, the ciphertext size is just $m \cdot \mathrm{poly}(\lambda)$.

Now, imagine that instead of directly providing $(\mathsf{lab}_{j,h_j})_j$, we encrypt $(\mathsf{lab}_{j,b})_{j,b}$ using another witness encryption scheme that computes $\mathsf{Compress}(\mathsf{pk}, 1^{T'}, U)$ and reveals the labels based on the result of such computation. This would immediately lead to another witness encryption scheme for P with potentially better efficiency: we just need the encryption of $(\mathsf{lab}_{j,b})_{j,b}$ to run in time $\mathrm{poly}(\lambda, \log T)$. The good news is that achieving this for such RAM program is easier than achieving this for the original program P. This is because there exists a RAM program D that, on input pk, computes $\mathsf{Compress}(\mathsf{pk}, 1^{T'}, U)$ in time $T' \cdot \mathrm{poly}(\lambda)$ and space $\mathrm{poly}(\lambda, \log L, \log|P|) = \mathrm{poly}(\lambda)$. Moreover, the description of D is upper-bounded by a fixed polynomial in λ. We can henceforth consider without loss of generality programs whose memory and description size is bounded by a fixed polynomial $\mathrm{poly}(\lambda)$.

Basic Compiler. Next, we describe our basic compiler. We consider a program P with tape of size L, for some $L = \mathrm{poly}(\lambda)$, which determines whether to output a tuple of L-many messages $(\mu_{0,0}, \mu_{0,1} \ldots, \mu_{L-1,0}, \mu_{L-1,1})$. Specifically, decryption should reveal $\mu_{j,b}$ only if the j-th bit output by P is b. Suppose that the description size of P is upper-bounded by a fixed polynomial in λ and that P has runtime T. We then describe how to construct a witness encryption for P with reduced runtime. Let $p := p(\lambda)$ be a polynomial to be determined later, we call this the compression parameter. The compiler is specified in Fig. 8.

Correctness. We show that during decryption,

$$\mathsf{lab}_j^{(k)} = \mathsf{lab}_{j,\mathbf{y}_j^{(k)}}^{(k)} \qquad \text{and} \qquad \overline{\mathsf{lab}}_w^{(k)} = \overline{\mathsf{lab}}_{w,h_w}^{(k)}$$

where $\mathbf{y}^{(k)}$ is obtained by running U on $(\mathbf{x}, P)$ for $k \cdot \overline{T}/p$ steps.

We proceed by induction over k starting from $k = 0$ until $k = p$. The claim is clearly true for $k = 0$. We show that if it is true for k, it is also true for $k + 1$. By the correctness of garbled circuits, at the $(k + 1)$-th step, for every $j \in [\overline{L}]$, $w \in [\ell]$ and $b \in \{0, 1\}$, it holds that

$$\overline{\mathsf{lab}}_w^{(k+1)} = \overline{\mathsf{lab}}_{w,h_w}^{(k+1)}, \qquad \text{and} \qquad E_{k,j,b} = \mathsf{Enc}(\mathsf{pk}, h, \mathbf{x}^{(k)}, j, b, \mathsf{lab}_{j,b}^{(k+1)}; r_{j,b}^{(k)}).$$

Therefore, by the correctness of the AB-LFE, we have that, for every $k < p - 1$

$$\mathsf{lab}_j^{(k+1)} = \mathsf{lab}_{j,\mathbf{y}_j^{(k+1)}}^{(k+1)}.$$

To conclude, by the correctness of AB-LFE, we obtain that, if the j-th bit output by P is 0, $\mathsf{lab}_j^{(p)} = \mu_{j,0}$. Otherwise, $\mathsf{lab}_j^{(p)} = \mu_{j,1}$.

Efficiency and Parameters. We derive a bound on the runtime of the encryption algorithm. Note that $\overline{L} = L + \mathrm{poly}(\lambda)$ whereas $\overline{T} = T \cdot \mathrm{poly}(\lambda)$. The garbling algorithm to compute $\{\tilde{C}_k\}_k$ is bounded by $p \cdot \mathrm{poly}(\lambda, L)$, since the runtime of the underlying circuit is a fixed polynomial in the security parameter, and it is repeated p-many times (we recall that L is bounded by a fixed polynomial $\mathrm{poly}(\lambda)$). The only subroutine of the computation that depends on T is the derivation of the AB-LFE digests h. This computation can be performed by D on input pk in time

$$\frac{T \cdot \mathrm{poly}(\lambda)}{p}$$

and memory $\mathrm{poly}(\lambda, \log L)$ by the efficiency property of the AB-LFE. By choosing a sufficiently large polynomial $p(\lambda)$, we can ensure that the running time remains smaller than $T/2$. Overall, we obtained a witness encryption algorithm with:

- Runtime bounded by $\frac{T}{2}$.
- Memory bounded by some fixed $\mathrm{poly}(\lambda)$.
- Output (i.e., ciphertext) size bounded by some fixed $\mathrm{poly}(\lambda, L, p) = \mathrm{poly}(\lambda)$.

Basic Compiler – Witness Encryption for P

$\mathsf{Enc}(1^\lambda, T, P, \mathbf{x}, \mu)$: Sample $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ and let $\mathbf{y} \leftarrow (\mathbf{x}, P)$. Let $\overline{T}$ and $\overline{L}$ be the running time and the memory required by the universal RAM U to compute $P^T(\mathbf{x})$ on input $\mathbf{y}$. Then compute:

$$h \leftarrow \mathsf{Compress}(\mathsf{pk}, 1^{\frac{\overline{T}}{p}}, U)$$

Define the variables $(\mathsf{lab}_{j,b}^{(p)} := \mu_{j,b})_{j\in[L],b\in\{0,1\}}$, $(\mathsf{lab}_{j,b}^{(p)} := \perp)_{j\in[\overline{L}]\setminus[L],b\in\{0,1\}}$ and $(\overline{\mathsf{lab}}_{w,b}^{(p)} := \perp)_{w\in[\ell],b\in\{0,1\}}$. Then, for $k = p-1, \dots, 0$:

1. Define C_k as the circuit that, on input a value $\overline{\mathbf{y}}$ and an AB-LFE digest $\overline{h}$, outputs

$$\left(\overline{\mathsf{lab}}_{w,\overline{h}_w}^{(k+1)}\right)_{w\in[\ell]} \qquad \text{and} \qquad \left(\mathsf{Enc}(\mathsf{pk}, \overline{h}, \overline{\mathbf{y}}, j, b, \mathsf{lab}_{j,b}^{(k+1)}; r_{j,b}^{(k)})\right)_{j\in[\overline{L}],b\in\{0,1\}}$$

where the random coins $r_{j,b}^{(k)}$ are hardwired in the description of the circuit.

2. Garble $(\tilde{C}_k, (\mathsf{lab}_{j,b}^{(k)}, \overline{\mathsf{lab}}_{w,b}^{(k)})_{w\in[\ell],j\in[\overline{L}],b\in\{0,1\}}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C_k)$.

The final ciphertext consists of:

$$c := \left(\mathsf{pk}, \left(\tilde{C}_k\right)_{k\in[p]}, \left(\overline{\mathsf{lab}}_w^{(0)} := \overline{\mathsf{lab}}_{w,h_w}^{(0)}\right)_{w\in[\ell]}, \left(\mathsf{lab}_j^{(0)} := \mathsf{lab}_{j,\mathbf{y}_j}^{(0)}\right)_{j\in[\overline{L}]}\right)$$

$\mathsf{Dec}(1^T, P, \mathbf{x}, c)$: For $k = 0, \dots, p-1$, the decryption algorithm proceeds as follows:

1. Evaluate the k-th garbled circuit to obtain:

$$\left(\left(\overline{\mathsf{lab}}_w^{(k+1)}\right)_{w\in[\ell]}, (E_{k,j,b})_{\substack{j\in[\overline{L}]\\ b\in\{0,1\}}}\right) \leftarrow \mathsf{Eval}\left(\tilde{C}_k, (\mathsf{lab}_j)_{j\in[\overline{L}]}^{(k)}, (\overline{\mathsf{lab}}_w^{(k)})_{w\in[\ell]}\right)$$

2. For all $j \in [\overline{L}]$ and all $b \in \{0,1\}$, compute:

$$\widetilde{\mathsf{lab}}_{j,b}^{(k+1)} \leftarrow \mathsf{Dec}(\mathsf{pk}, E_{k,j,b}, 1^{T/p}, P)$$
$$\mathsf{lab}_j^{(k+1)} := \begin{cases} \widetilde{\mathsf{lab}}_{j,0}^{(k+1)} & \text{if } \mathsf{lab}_{j,0}^{(k+1)} \neq \perp \\ \widetilde{\mathsf{lab}}_{j,1}^{(k+1)} & \text{otherwise.} \end{cases}$$

Output $(\mathsf{lab}_j^{(p)})_{j\in[L]}$.

Figure 8: Basic Compiler – Witness Encryption for P

Decryption instead requires time $T \cdot \mathrm{poly}(\lambda)$ and memory $L \cdot \mathrm{poly}(\lambda)$.

Theorem 7.2. *Let* $(\mathsf{Garble}, \mathsf{Eval})$ *be a secure garbling scheme and let* $(\mathsf{Setup}, \mathsf{Compress}, \mathsf{Enc}, \mathsf{Dec})$ *be a secure AB-LFE scheme for RAM programs, then the construction as described above in a semantically secure witness encryption for* P*.*

Proof. We proceed by changing the distribution of the ciphertext in a series of hybrids.

- Hybrid 0: This is the original distribution, i.e., the ciphertext is computed as described in Fig. 8.

For $k = 0, \dots, p-1$, we define a series of sub-hybrids as follows:

- Hybrid $(k, 0)$: The labels:

$$\left(\mathsf{lab}^{(k)}_{j,\mathbf{y}^{(k)}_j}\right)_{j\in[\overline{L}]} \quad \text{and} \quad \left(\overline{\mathsf{lab}}^{(k)}_{w,h_w}\right)_{w\in[\ell]}$$

  and the circuit $\tilde{C}_k$ are simulated by feeding to the simulator the outputs:

$$\left(\overline{\mathsf{lab}}^{(k+1)}_{w,h_w}\right)_{w\in[\ell]} \quad \text{and} \quad \left(E_{k,j,b} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, h, \mathbf{y}^{(k)}, j, b, \mathsf{lab}^{(k+1)}_{j,b})\right)_{\substack{j\in[\overline{L}]\\ b\in\{0,1\}}}$$

  Indistinguishability follows by the security of the garbling schemes.

- Hybrid $(k, 1)$: For all $j \in [\overline{L}]$, let b_j be the j-th bit of $\mathbf{y}^{(k+1)}$. Compute:

$$\left(E_{k,j,b_j\oplus 1} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, h, \mathbf{y}^{(k)}, j, b_j \oplus 1, \mathbf{0})\right)_{j\in[\overline{L}]}$$

  and indistinguishability follows by the security of the AB-LFE. Note that this hybrid effectively erases all labels $(\mathsf{lab}^{(k+1)}_{j,1\oplus\mathbf{y}^{(k+1)}_j})_{j\in[\overline{L}]}$.

In the last hybrid, the messages $\mu_{j,b}$ such that the j-th output bit of the program is different from b are erased from the view of the distinguisher. This concludes our proof. ☐

Recursive Composition. Observe that after applying out basic compiler, the ciphertext is once again split into two parts: the labels $(\overline{\mathsf{lab}}^{(0)}_{w,h_w})_{w\in[\ell]}$, which need computations proportional to T and the remaining part, which can be computed in time $\mathrm{poly}(\lambda)$. To further reduce the running time to $\mathrm{poly}(\lambda, \log T)$, our idea is simple: instead of directly providing $(\overline{\mathsf{lab}}^{(0)}_{w,h_w})_{w\in[\ell]}$, we encrypt $(\overline{\mathsf{lab}}^{(0)}_{w,b})_{w,b}$ under our witness encryption scheme, where we evaluate D on pk for $T_0 := T/2$ steps.

We will proceed in this way for $\log T$ recursive iterations, halving the encryption time at every step. In particular, at the i-th iteration, we will augment the ciphertext with a new witness encryption which will evaluate D for

$$T_i := \frac{T_{i-1}}{2}$$

steps. To summarise, after $\log T$ recursive iterations, we obtain a witness encryption scheme for P where the encryption runtime is

$$(|x| + |P|) \cdot \mathrm{poly}(\lambda) + m \cdot \mathrm{poly}(\lambda) + \mathrm{poly}(\lambda, \log T)$$

and the space complexity is

$$(|x| + |P|) \cdot \mathrm{poly}(\lambda) + m \cdot \mathrm{poly}(\lambda) + \mathrm{poly}(\lambda, \log T).$$

Decryption requires instead $T \cdot \mathrm{poly}(\lambda) \cdot |P|$ runtime and $L \cdot \mathrm{poly}(\lambda) + |P|$ space. As a matter of fact, decryption requires us to *sequentially* evaluate a decryption for each recursive iteration (it is *not* a nested execution of decryptions): In other words, the running times of the $\log T$ decryptions will pile up *additively*.

7.3 Construction of SRE

Assuming the construction of a witness encryption for P with the above efficiency, one can construct SRE using standard techniques. Recall that the only difference between witness encryption and SRE is that SRE must also protect the privacy of the computation. To achieve this, we can appeal to an idea of [GKP+13b]: We encrypt a description of the RAM program P and its initial state $\mathbf{x}$, using a fully-homomorphic encryption scheme and we run a witness encryption for P on the homomorphic computation, evaluating a universal RAM. The encrypted messages consist of the labels obtained by garbling the FHE decryption, where the secret key is hard-coded. Witness encryption will ensure that only the labels for the right FHE ciphertext will be revealed. This achieves the desired efficiency, except that the evaluator now runs in time proportional to the circuit computing P. To achieve runtime proportional to the RAM program P, we can instead use fully-homomorphic encryption for RAM [LMW23]. Since this is a standard transformation, we omit details here and we refer to [GKP+13b] for a formal analysis.

It is even possible to adapt this approach to make the encoding rate-1 in the size of the output by relying on the rate-1 LFE sketched in Section 6.3: Instead of garbling the FHE decryption, we garble the circuit that, on input a digest h', produces

$$\mathsf{LFE.Enc}(\mathsf{pk}', h', \mathsf{sk}_{\mathsf{FHE}})$$

where $\mathsf{pk}' \overset{\$}{\leftarrow} \mathsf{LFE.Setup}(1^\lambda)$ is made public and $\mathsf{sk}_{\mathsf{FHE}}$ is the FHE secret key. The witness encryption scheme will now compute $\mathsf{LFE.Compress}(\mathsf{pk}', \mathsf{FHE.Dec}, \mathsf{ct})$ where ct is the output of the FHE homomorphic evaluation.

8 Optimal Bounded-Time ABE for RAM

In this section we present our construction of ABE for RAM.

8.1 Definition

We recall the basic definitions of ABE.

Definition 8.1 (Bounded-Time ABE for RAM). *Let $L := L(\lambda)$ and $T := T(\lambda)$ be positive integers. A bounded-time ABE scheme for RAM consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax.*

$\mathsf{Setup}(1^\lambda, 1^T)$**:** *The setup algorithm is probabilistic, it takes as input the security parameter 1^λ, a running time upper bound 1^T and outputs a master public key* mpk *and a master secret* msk.

$\mathsf{KeyGen}(\mathsf{msk}, 1^t, P)$**:** *The key generation algorithm is probabilistic, it takes as input a master secret key, a running time 1^t, where $t \leq T$, and the description of a RAM program P. The output is a secret key* $\mathsf{sk}_{P,t}$.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu)$**:** *The encryption algorithm is probabilistic, it takes as input a master public key* mpk, *an attribute $\mathbf{x} \in \mathbb{Z}_2^{\leq L}$, and a message $\mu \in \{0, 1\}$. The output is a ciphertext* ct.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}_{P,t})$**:** *The decryption procedure is deterministic, it takes as input a ciphertext* ct *and a secret key* $\mathsf{sk}_{P,t}$. *The output is a message $\mu \in \{0, 1, \perp\}$.*

For correctness, we require that there must exist a negligible function $\mathsf{negl}(\lambda)$ such that, for every sufficiently large λ, every RAM machine P, every running time $t \leq T$, every $\mathbf{x} \in \mathbb{Z}_2^{\leq L}$ such that the first entry of $P^t(\mathbf{x})$ is 0, and every message $\mu \in \{0, 1\}$, it holds that:

$$\Pr\left[\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}_{P,t}) \neq \mu\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the randomness of the procedures $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^T)$, $\mathsf{sk}_{P,t} \xleftarrow{\$} \mathsf{Hash}(\mathsf{msk}, 1^t, P)$ and $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu)$. We recall the definition of selective security.

Definition 8.2 (Selective Security). *An ABE scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is selectively secure if, for every $\mathbf{x} \in \mathbb{Z}_2^{\leq L}$, we have that the following distributions are computationally indistinguishable:*

$$(\mathsf{mpk}, \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, 0)) \approx_c (\mathsf{mpk}, \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, 1))$$

where $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^T)$, *even if the distinguisher has unrestricted oracle access to the* modified KeyGen *oracle that answers only queries (P, t) for which the first entry of $P^t(\mathbf{x})$* is not 0.

For a security parameter λ, running time upper bound T and memory upper bound L, we require that the algorithms of ABE satisfy the following efficiency requirements:

- The runtime of the Setup algorithm is bounded by $\text{poly}(\lambda, T, \log L)$.
- The runtime of the KeyGen algorithm is bounded by $\text{poly}(\lambda, T, \log L)$.

- The runtime of the Enc algorithm is bounded by $|\mathbf{x}| \cdot \mathrm{poly}(\lambda, T, \log L)$. Furthermore, the Enc algorithm is split into two subroutines:

  $$(\mathbf{d}, \tau) \leftarrow \mathsf{Hash}(\mathsf{mpk}, \mathbf{x}) \quad \text{and} \quad c \xleftarrow{\$} \mathsf{Complete}(\mathsf{mpk}, \mathbf{d}, \mu)$$

  where the runtime of $\mathsf{Complete}$ is bounded by $\mathrm{poly}(\lambda, T, \log L)$, and the final encoding consists of $\mathsf{ct} := (\mathbf{x}, c)$.

- The runtime of the Dec algorithms is bounded by $|\mathbf{x}| \cdot \mathrm{poly}(\lambda, T, \log L)$.

8.2 Construction

Let $q \geq \overline{\sigma} \cdot 2^{T \cdot \log L \cdot \omega(\log \lambda)}$, $\overline{\sigma} \geq 2^{T \cdot \log L \cdot \omega(\log \lambda)}$ and $m \geq 2k \log q$. We present our construction in Fig. 9.

To prove correctness, suppose that the first bit of $P^t(\mathbf{x})$ is $y = 0$. Due to Theorem 5.8 and Lemma 5.4, we have that

$$\mathbf{c}^\intercal = \mathbf{s}^\intercal \cdot (\mathbf{A}_{P,t} + y \cdot \mathbf{G}) + \mathbf{e}_{P,t}^\intercal$$

where $\|\mathbf{e}_{P,t}\|_\infty \leq \mathrm{poly}(\lambda^{t \cdot \log L}, B^{t \cdot \log L}) \cdot (\widetilde{B} + \sigma')$. In other words, if $y = 0$, we have that

$$\mathbf{c}^\intercal = \mathbf{s}^\intercal \cdot \mathbf{A}_{P,t} + \mathbf{e}_{P,t}^\intercal.$$

We conclude that

$$\begin{aligned}
&\left[\mathbf{v}^\intercal, \quad \mathbf{z}^\intercal, \quad \mathbf{c}^\intercal\right] \cdot \mathbf{k}_{P,t} - c' \\
&= (\mathbf{s}^\intercal \cdot \left[\mathbf{V}, \quad \mathbf{Z}, \quad \mathbf{A}_{P,t}\right] + \left[\overline{\mathbf{e}}^\intercal, \quad \tilde{\mathbf{e}}^\intercal, \quad \mathbf{e}_{P,t}^\intercal\right]) \cdot \mathbf{k}_{P,t} - \mathbf{s}^\intercal \cdot \mathbf{a} - e' + \lceil q/2 \rfloor \cdot \mu \\
&= \mathbf{s}^\intercal \cdot \left[\mathbf{V}, \quad \mathbf{Z}, \quad \mathbf{A}_{P,t}\right] \cdot \mathbf{k}_{P,t} + \left[\overline{\mathbf{e}}^\intercal, \quad \tilde{\mathbf{e}}^\intercal, \quad \mathbf{e}_{P,t}^\intercal\right] \cdot \mathbf{k}_{P,t} - \mathbf{s}^\intercal \cdot \mathbf{a} - e' + \lceil q/2 \rfloor \cdot \mu \\
&= \mathbf{s}^\intercal \cdot \mathbf{a} + \left[\overline{\mathbf{e}}^\intercal, \quad \tilde{\mathbf{e}}^\intercal, \quad \mathbf{e}_{P,t}^\intercal\right] \cdot \mathbf{k}_{P,t} - \mathbf{s}^\intercal \cdot \mathbf{a} - e' + \lceil q/2 \rfloor \cdot \mu \\
&= \lceil q/2 \rfloor \cdot \mu + \left[\overline{\mathbf{e}}^\intercal, \quad \tilde{\mathbf{e}}^\intercal, \quad \mathbf{e}_{P,t}^\intercal\right] \cdot \mathbf{k}_{P,t} - e'.
\end{aligned}$$

Notice that $\|\left[\overline{\mathbf{e}}^\intercal, \quad \tilde{\mathbf{e}}^\intercal, \quad \mathbf{e}_{P,t}^\intercal\right] \cdot \mathbf{k}_{P,t} - e'\|_\infty \leq \overline{\sigma} \cdot \mathrm{poly}(\lambda^{t \cdot \log L}, B^{t \cdot \log L}) \cdot (\widetilde{B} + \sigma')$. Given our choice of q, we obtain that $\left\lceil \left[\mathbf{v}^\intercal, \quad \mathbf{z}^\intercal, \quad \mathbf{c}^\intercal\right] \cdot \mathbf{k}_{P,t} - c' \right\rfloor_2 = \mu$.

Theorem 8.3. *Assuming the hardness of decomposed LWE, the construction in Fig. 9 is a selectively secure, bounded time, attribute-based encryption scheme for RAM.*

Proof. We proceed by means of a series of hybrids.

- Hybrid 0: This corresponds to the real world execution of the game. Specifically, we challenge chiphertext consists of the encryption of b.

- Hybrid 1: We modify the distribution of $\mathbf{M}$. First, we sample $\mathbf{R} \xleftarrow{\$} \mathbb{Z}_2^{m \times (n^2 \cdot m)}$ and then we set $\mathbf{M} \leftarrow \mathbf{V} \cdot \mathbf{R} - \mathbf{d}^\intercal \otimes \mathbf{G}$ where $\mathbf{d}$ is the hash of the challenge $\mathbf{x}$ (we can do this because we are proving *selective* security). This hybrid is statistically indistinguishable from Hybrid 0 due to the leftover hash lemma.

Optimal Bounded-Time ABE for RAM

$\mathsf{Setup}(1^\lambda, 1^T)$**:** Sample $\mathsf{pk}' := (\mathbf{A}, \overline{\mathbf{B}}, \mathbf{B}, \mathbf{Q}, \mathbf{Z}) \xleftarrow{\$} \mathsf{E.Setup}(1^\lambda)$, $\mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{k\times(n\cdot m)}$, $(\mathbf{V}, \nu) \xleftarrow{\$} \mathsf{TrapGen}(1^k, 1^m, q)$ and $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^k$. Output $\mathsf{mpk} := (\mathbf{V}, \mathbf{A}, \overline{\mathbf{B}}, \mathbf{B}, \mathbf{Q}, \mathbf{Z}, \mathbf{M}, \mathbf{a})$ and $\mathsf{msk} := \nu$.

$\mathsf{KeyGen}(\mathsf{msk}, 1^t, P)$**:** Compute

$$\mathbf{A}'_{P,t} \leftarrow \mathbf{M} \cdot \mathsf{RAMEvalK}(\mathsf{pk}', 1^t, P, \mathbf{M})$$
$$\mathbf{A}_{P,t} \leftarrow \begin{bmatrix}\mathbf{A}'_{P,t}, & \mathbf{0}\end{bmatrix} \cdot \mathsf{ReadK}(\mathbf{A}, \mathbf{A}'_{P,t}, \mathbf{0}).$$

Then, derive $\mathbf{k}_{P,t} \xleftarrow{\$} \mathsf{SampleRight}(\mathsf{msk}, \begin{bmatrix}\mathbf{Z}, & \mathbf{A}_{P,t}\end{bmatrix}, \mathbf{a}, \overline{\sigma})$. Output $\mathsf{sk}_{P,t} = (1^t, P, \mathbf{A}'_{P,t}, \mathbf{k}_{P,t})$.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu)$**:** Sample $\mathbf{s} \xleftarrow{\$} \mathcal{D}^k_{\sigma'}$ and compute $(\mathbf{d}, \tau) \leftarrow \mathsf{E.Hash}(\mathsf{pk}', \mathbf{x})$. Then, compute

$$\mathbf{v}^\intercal \leftarrow \mathbf{s}^\intercal \cdot \mathbf{V} + \overline{\mathbf{e}}^\intercal$$
$$\mathbf{h}^\intercal \leftarrow \mathbf{s}^\intercal \cdot (\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal$$
$$\mathbf{z}^\intercal \leftarrow \mathbf{s}^\intercal \cdot \mathbf{Z} + \tilde{\mathbf{e}}^\intercal$$
$$c' \leftarrow \mathbf{s}^\intercal \cdot \mathbf{a} + e' - \lceil q/2 \rfloor \cdot \mu$$

where $\mathbf{e} \xleftarrow{\$} \mathcal{D}^{n^2\cdot m}_{\sigma'}$, $e' \xleftarrow{\$} \mathcal{D}_{\sigma'}$, $\overline{\mathbf{e}} \xleftarrow{\$} \mathcal{D}^m_{\sigma'}$, $\mathbf{E} \xleftarrow{\$} \mathcal{D}^{t\times(t\cdot n)}_{\widetilde{\sigma}}$ and $\tilde{\mathbf{e}} \leftarrow \mathsf{vec}(\mathbf{E} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G})))$. Output $\mathsf{ct} := (\mathbf{x}, \mathbf{v}, \mathbf{h}, \mathbf{z}, c')$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}_{P,t})$**:** If the first bit of $P^t(\mathbf{x})$ is not 0, output $\perp$. Otherwise, derive $(\mathbf{d}, \tau) \leftarrow \mathsf{E.Hash}(\mathsf{pk}', \mathbf{x})$. Then, compute

$$(\mathbf{d}', \tau') \leftarrow \mathsf{RAMEvalP}(\mathsf{pk}', 1^t, P, \mathbf{d}, \tau)$$
$$\mathbf{H}_{P,t,\mathbf{x}} \leftarrow \mathsf{RAMEvalH}(\mathsf{pk}', 1^t, P, \mathbf{z}, \mathbf{h}, \mathbf{d}, \tau)$$
$$\mathbf{h}'^\intercal \leftarrow \begin{bmatrix}\mathbf{h}^\intercal, & \mathbf{z}^\intercal\end{bmatrix} \cdot \mathbf{H}_{P,t,\mathbf{x}}$$
$$\mathbf{c}^\intercal \leftarrow \begin{bmatrix}\mathbf{h}'^\intercal, & \mathbf{0}^\intercal, & \mathbf{z}^\intercal\end{bmatrix} \mathsf{ReadH}(\mathbf{A}, \mathbf{A}'_{P,t}, \mathbf{0}, \mathbf{d}', \tau', 0).$$

Output $\left\lceil \begin{bmatrix}\mathbf{v}^\intercal & \mathbf{z}^\intercal & \mathbf{c}^\intercal\end{bmatrix} \cdot \mathbf{k}_{P,t} - c' \right\rfloor_2$.

Figure 9: Optimal Bounded-Time ABE for RAM

- Hybrid 2: We modify the distribution of the queried decryption keys. Specifically, upon receiving a query for P, t, we compute the first entry y of $P^t(x)$ and set

$$\mathbf{k}_{P,t} \xleftarrow{\$} \mathsf{SampleLeft}\left(\begin{bmatrix}\mathbf{V}, & \mathbf{Z}\end{bmatrix}, \begin{bmatrix}\mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{t^2 \cdot n}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{H}_{P,t,\mathbf{x}}, & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x}',0}, -y, \mathbf{a}, \overline{\sigma}\right)$$

where

$$\begin{aligned}(\mathbf{d}', \tau') &\leftarrow \mathsf{RAMEvalP}(\mathsf{pk}', 1^t, P, \mathbf{d}, \tau) \\ \mathbf{H}_{P,t,\mathbf{x}} &\leftarrow \mathsf{RAMEvalH}(\mathsf{pk}', 1^t, P, \mathbf{M}, \mathbf{d}, \tau) \\ \mathbf{H}_{\mathbf{x}',0} &\leftarrow \mathsf{ReadH}(\mathbf{A}, \mathbf{A}'_{P,t}, \mathbf{0}, \mathbf{d}', \tau', 0).\end{aligned}$$

Notice that

$$\begin{aligned}&\begin{bmatrix}\mathbf{V}, & \mathbf{Z}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{t^2 \cdot n}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{H}_{P,t,\mathbf{x}}, & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x}',0} - y \cdot \mathbf{G} \\ &= \begin{bmatrix}\mathbf{V} \cdot \mathbf{R}, & \mathbf{Z}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{H}_{P,t,\mathbf{x}}, & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x}',0} - y \cdot \mathbf{G} \\ &= \begin{bmatrix}\mathbf{M} + \mathbf{d}^\intercal \otimes \mathbf{G}, & \mathbf{Z}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{H}_{P,t,\mathbf{x}}, & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x}',0} - y \cdot \mathbf{G} \\ &= \begin{bmatrix}\mathbf{A}'_{P,t} + \mathbf{d}'^\intercal \otimes \mathbf{G}, & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x}',0} - y \cdot \mathbf{G} \\ &= \mathbf{A}_{P,t} + y \cdot \mathbf{G} - y \cdot \mathbf{G} \\ &= \mathbf{A}_{P,t}.\end{aligned}$$

Moreover, for every vector $\mathbf{e}$ of suitable length, it holds that

$$\left\| \begin{bmatrix}\mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{t^2 \cdot n}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{H}_{P,t,\mathbf{x}}, & \mathbf{0}\end{bmatrix} \cdot \mathbf{H}_{\mathbf{x}',0} \right\|_\infty \le \mathrm{poly}(\lambda^{t \cdot \log_\beta L}, B^{t \cdot \log_\beta L}) \le \overline{\sigma} / \omega(\sqrt{\log m}).$$

By Lemma 3.4, this hybrid is statistically indistinguishable from Hybrid 1.

- Hybrid 3: We modify the distribution of $\mathbf{V}$. Specifically, we sample it uniformly at random. This hybrid is statistically indistinguishable from Hybrid 2.

- Hybrid 4: We modify the distribution of ct. Specifically, we sample $\mathbf{v}, \mathbf{h}$ and c' at random and we set $\mathbf{z}$ to $\mathsf{vec}(\mathbf{U} \cdot (\mathbf{I}_t \otimes \mathbf{G}^{-1}(\mathbf{Q}^{-1} \cdot \mathbf{G}))))$ where $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{t \times (t \cdot n)}$. This hybrid is indistinguishable from the previous one under the hardness of decomposed LWE. This can be showed similarly to what we did in the proof of Theorem 6.3.

Notice that in Hybrid 4 the view of the adversary contains no information about the challenge bit b. This ends the proof. $\square$

9 Rate-1 Register ABE from Rate-1 ABE

In this section we define and construct a new register ABE scheme, building on the previously constructed ABE.

9.1 Definition

We recall the definition of Register ABE.

Definition 9.1 (Bounded-Time Register ABE for RAM)**.** *Let $L := L(\lambda)$, $N := N(\lambda)$, and $T := T(\lambda)$ be positive integers. A bounded-time register ABE scheme for RAM consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax.*

$\mathsf{Setup}(1^\lambda, 1^T, 1^N)$**:** *The setup algorithm is probabilistic, it takes as input the security parameter 1^λ, a running time upper bound 1^T, a number of parties upper bound 1^N and outputs public parameters* crs.

$\mathsf{KeyGen}(\mathsf{crs}, t, P)$**:** *The key generation algorithm is probabilistic, it takes as input public parameters* crs *and a RAM program P and running time $t \leq T$. The output is a public key* pk *and a secret key* sk.

$\mathsf{Aggregate}(\mathsf{crs}, t_0, P_0, \mathsf{pk}_0, \dots, t_{n-1}, P_{N-1}, \mathsf{pk}_{N-1})$**:** *The aggregation algorithm is deterministic and takes as input public parameters* crs *and N triples $(t_i, P_i, \mathsf{pk}_i)$ where $t_i \leq T$, P_i is a RAM program and pk_i is a public key. The output is an aggregated key* mpk *and N helper keys $(\mathsf{hpk}_i)_{i \in [N]}$.*

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu)$**:** *The encryption algorithm is probabilistic, it takes as input an aggregated public key* mpk*, an attribute $\mathbf{x} \in \mathbb{Z}_2^{\leq L}$, and a message $\mu \in \{0, 1\}$. The output is a ciphertext* ct.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{hpk}_i, \mathsf{sk})$**:** *The decryption procedure is deterministic, it takes as input a ciphertext* ct*, a helper key* hpk_i *and a secret key* sk*. The output is a message $\mu \in \{0, 1, \perp\}$.*

For correctness, we require that there must exist a negligible function $\mathsf{negl}(\lambda)$ such that, for every sufficiently large λ, every list of running times $(t_j)_{j\in[N]}$ where $t_j \leq T$, every list of RAM $(P_j)_{j\in[N]}$, every $\mathbf{x} \in \mathbb{Z}_2^{\leq L}$, every $i \in [N]$ such that the first bit of $P_i^{t_i}(\mathbf{x})$ is 0, and every message $\mu \in \{0, 1\}$, it holds that:

$$\Pr\left[\mathsf{Dec}(\mathsf{ct}, \mathsf{hpk}_i, \mathsf{sk}_i) \neq \mu\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the randomness of the procedures

$$\begin{aligned} \mathsf{crs} &\xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^T, 1^N) \\ \forall j \in [N]: \qquad (\mathsf{pk}_j, \mathsf{sk}_j) &\xleftarrow{\$} \mathsf{KeyGen}(\mathsf{crs}, t_j, P_j) \\ (\mathsf{mpk}, (\mathsf{hpk}_j)_{j\in[N]}) &\leftarrow \mathsf{Aggregate}(\mathsf{crs}, t_0, P_0, \mathsf{pk}_0, \dots, t_{N-1}, P_{N-1}, \mathsf{pk}_{N-1}) \\ \mathsf{ct} &\xleftarrow{\$} \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu). \end{aligned}$$

We recall the definition of very selective security.

Definition 9.2 (Very Selective Security)**.** *A register ABE scheme* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$ *is very selectively secure if, for every polynomials $N = N(\lambda)$ and $T = T(\lambda)$, every subset of corrupted parties $C \subseteq [N]$, randomness $(r_j)_{j\in C}$, RAM programs $(P_j)_{j\in[N]}$, running* *times* $(t_j)_{j\in[N]}$ *where* $t_j \leq T$ *and every* $\mathbf{x} \in \mathbb{Z}_2^{\leq L}$ *where, for every* $j \in C$*, the first entry of* $P_j^{t_j}(\mathbf{x})$ *is not* 0*, we have that the following distributions are computationally indistinguishable:*

$$(\mathsf{crs}, (\mathsf{pk}_j)_{j\notin C}, \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, 0)) \approx_c (\mathsf{crs}, (\mathsf{pk}_j)_{j\notin C}, \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, 1))$$

where

$$\begin{aligned}
& & \mathsf{crs} &\xleftarrow{\$} \mathsf{Setup}(1^\lambda, 1^T, 1^N)\\
&\forall j \in C: & (\mathsf{pk}_j, \mathsf{sk}_j) &\leftarrow \mathsf{KeyGen}(\mathsf{crs}, t_j, P_j; r_j)\\
&\forall j \notin C: & (\mathsf{pk}_j, \mathsf{sk}_j) &\xleftarrow{\$} \mathsf{KeyGen}(\mathsf{crs}, t_j, P_j)\\
& & (\mathsf{mpk}, (\mathsf{hpk}_j)_{j\in[N]}) &\leftarrow \mathsf{Aggregate}(\mathsf{crs}, t_0, P_0, \mathsf{pk}_0, \ldots, t_{N-1}, P_{N-1}, \mathsf{pk}_{N-1}).
\end{aligned}$$

9.2 Construction

Our construction is based on a rate-1 ABE satisfying some additional properties:

- For any polynomial $m = m(\lambda)$, the procedure Hash can be split into three parts $\mathsf{Hash}_0, \mathsf{Hash}_1, \mathsf{Hash}_2$, given a (suitably padded) string

  $$\mathbf{x} := (\mathbf{x}_{0,0}, \ldots, \mathbf{x}_{0,m-1}, \mathbf{x}_{1,0}, \ldots, \mathbf{x}_{1,m-1}),$$

  we have that $\mathsf{Hash}(\mathsf{mpk}, \mathbf{x}) = (\mathbf{d}, \tau)$ where

  $$\begin{aligned}
  &\forall i \in [m], \alpha \in \{0,1\}: & (\mathbf{d}^{(0)}_{\alpha,i}, \tau^{(0)}_{\alpha,i}) &\leftarrow \mathsf{Hash}_0(\mathsf{mpk}, \mathbf{x}_{\alpha,i})\\
  &\forall \alpha \in \{0,1\}: & (\mathbf{d}^{(1)}_{\alpha}, \tau^{(1)}_{\alpha}) &\leftarrow \mathsf{Hash}_1(\mathsf{mpk}, (\mathbf{d}^{(0)}_{\alpha,0}, \ldots, \mathbf{d}^{(0)}_{\alpha,m-1}))\\
  & & (\mathbf{d}, \tau^{(2)}) &\leftarrow \mathsf{Hash}_2(\mathsf{mpk}, (\mathbf{d}^{(1)}_0, \mathbf{d}^{(1)}_1)).
  \end{aligned}$$

  and for every $\Gamma \subseteq \{0,1\} \times [m]$, we have that

  $$\mathsf{Path}(\tau, \Gamma) = \bigcup_{(\alpha,i)\in\Gamma} (\tau^{(2)} \cup \tau^{(1)}_\alpha \cup \tau^{(0)}_{\alpha,i}).$$

  Moreover, if, for every $j \in \{0,1,2\}$ we denote the output size of Hash_j by out_j and the input size by in_j, we require that $\mathsf{out}_j = \mathrm{poly}(\lambda, \log(\mathsf{in}_j))$. Notice that we are essentially asking that Hash has a Merkle tree structure as the construction in Section 8. In the register ABE scheme we are about to present $m = 2N$.

- The decryption procedure succeeds even if we do not provide the full ciphertext. Specifically, suppose our secret key is associated with the RAM program P^t and $P^t(\mathbf{x}) = 0$. Assume also that the ciphertext ct is split in $(\mathbf{x}, c)$ where c is produced by $\mathsf{Complete}(\mathsf{mpk}, \mathbf{d}, \mu)$, $(\mathbf{d}, \tau) \leftarrow \mathsf{Hash}(\mathsf{mpk}, \mathbf{x})$. We require decryption to succeed (with overwhelming probability) even if we are just provided with $(\tau_{P,t}, \mathbf{d}, c)$ where $\tau_{P,t} \leftarrow \mathsf{Path}(\tau, P^t)$. Notice that our ABE scheme in Section 8 satisfies this property as $\mathsf{RAMEvalP}$ and $\mathsf{RAMEvalH}$ work even if they receive as input $\tau_{P,t}$ instead of τ (see Theorem 5.8).

For every $i \in [N]$, we define the RAM program U_i that performs the following operations:

1. It parses its tape as $((s_0, t_0, P_0), r_0, \ldots, (s_{N-1}, t_{N-1}, P_{N-1}), r_{N-1}, \mathbf{x})$ where for every $j \in [N]$, P_j is the description of a RAM program, $t_j \leq T$, r_j and s_j are binary strings and $\mathbf{x}$ can be further split into $2N$ chunks $\mathbf{x}_0, \ldots, \mathbf{x}_{2N-1}$.
2. It checks whether $\mathsf{Ext}(s_i, r_i) = 1$ where Ext is a strong randomness extractor. In such case, it outputs 1.
3. Otherwise, it computes $P_i^{t_i}(\mathbf{x})$.

Let $\overline{T}$ be the running time of U_i. Notice that U_i never touches $(s_j, t_j, P_j, r_j)_{j \neq i}$. The scheme is specified in Fig. 10.

Correctness easily follows from the special correctness properties of our ABE scheme: We observe that τ_i coincides with $\mathsf{Path}(\tau, U_i^{\overline{T}})$ where

$$(\mathbf{d}, \tau) = \mathsf{Hash}(\mathsf{mpk}, ((s_0, t_0, P_0), r_0, \ldots, (s_{N-1}, t_{N-1}, P_{N-1}), r_{N-1}, \mathbf{x})).$$

Notice also that, since $\mathsf{Ext}(s_i, r_i) = 0$ and the first entry of $P_i^{t_i}(\mathbf{x})$ is 0, we have

$$U_i^{\overline{T}}((s_0, t_0, P_0), r_0, (s_1, t_1, P_1), r_1, \ldots, (s_{N-1}, t_{N-1}, P_{N-1}), r_{N-1}, \mathbf{x}))$$

has the first entry equal to 0, so decryption succeeds.

Theorem 9.3. *Suppose that* $\mathsf{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Complete}, \mathsf{Dec})$ *is a selectively secure, bounded-time rate-1 ABE scheme for RAM with the properties described above. Let* Ext *be a* $(\lambda, \mathsf{negl}(\lambda))$*-strong randomness extractor and assume that* $\mathsf{in}_0 - \mathsf{out}_0 \geq \lambda$*. Then, the construction in Fig. 10 is a very selectively secure, bounded-time, registered ABE scheme for RAM.*

Proof. We proceed by means of a series of hybrids.

- Hybrid 0: This corresponds to the real world execution of the game.
- Hybrid 1: We modify the distribution of the public keys of the honest parties. Specifically, at each query, we sample the strings s and r at random conditioned on $\mathsf{Ext}(s, r) = 1$. Since Hash_0 is compressing, by the chain rule of min entropy,

  $$\tilde{\mathsf{H}}_\infty(r|\mathbf{d}^{(0)}) \geq \mathsf{in}_0 - \mathsf{out}_0 \geq \lambda.$$

  So, since Ext is a $(\lambda, \mathsf{negl}(\lambda))$-strong extractor, this hybrid is statistically indistinguishable from Hybrid 0.
- Hybrid 2: We modify the distribution of the ciphertexts. Specifically, instead of encrypting the challenge, we always encrypt 0. This hybrid is computationally indistinguishable from Hybrid 1 thanks to the special security properties of the ABE scheme. Indeed, notice that for every $i \in [N]$ (including the indexes associated with honest parties), the first entry of

  $$U_i^{\overline{T}}((s_0, t_0, P_0), r_0, (s_1, t_1, P_1), r_1, \ldots, (s_{N-1}, t_{N-1}, P_{N-1}), r_{N-1}, \mathbf{x})$$

  is different from 0.

Rate-1 Register ABE for RAM

$\mathsf{Setup}(1^\lambda, 1^T, 1^N)$**:** Compute $(\mathsf{mpk}', \mathsf{msk}') \xleftarrow{\$} \mathsf{ABE.Setup}(1^\lambda, 1^{\overline{T}})$ and, for every $j \in [N]$, $k_i \xleftarrow{\$} \mathsf{ABE.KeyGen}(\mathsf{msk}', 1^{\overline{T}}, U_j)$. Output $\mathsf{crs} := (\mathsf{mpk}', (k_j)_{j\in[N]})$.

$\mathsf{KeyGen}(\mathsf{crs}, t, P)$**:** Sample $s \xleftarrow{\$} \{0,1\}^\lambda$ and $r \xleftarrow{\$} \{0,1\}^{\mathsf{in}_0}$ conditioned on $\mathsf{Ext}(s,r) = 0$. Then, compute $(\mathbf{d}^{(0)}, \tau^{(0)}) \leftarrow \mathsf{ABE.Hash}_0(\mathsf{mpk}', r)$. Output $\mathsf{pk} := (s, \mathbf{d}^{(0)})$ and $\mathsf{sk} := \tau^{(0)}$.

$\mathsf{Aggregate}(\mathsf{crs}, t_0, P_0, \mathsf{pk}_0, \ldots, t_{N-1}, P_{N-1}, \mathsf{pk}_{N-1})$**:** For every $j \in [N]$, rewrite pk_j as $(s_j, \mathbf{d}^{(0)}_{2j+1})$ and compute $(\mathbf{d}^{(0)}_{0,2j}, \tau^{(0)}_{0,2j}) \leftarrow \mathsf{ABE.Hash}_0(\mathsf{mpk}', (s_j, t_j, P_j))$. Then, set

$$\begin{aligned} (\mathbf{d}^{(1)}_0, \tau^{(1)}_0) &\leftarrow \mathsf{ABE.Hash}_1(\mathsf{mpk}', (\mathbf{d}^{(0)}_{0,0}, \mathbf{d}^{(0)}_{0,1}, \ldots, \mathbf{d}^{(0)}_{0,2N-1})) \\ \forall j \in [N]: \qquad \tau^{(1)}_{0,j} &\leftarrow \mathsf{Path}(\tau^{(1)}_0, 2j, 2j+1). \end{aligned}$$

Output $\mathsf{mpk} := (\mathsf{mpk}', \mathbf{d}^{(1)}_0)$ and, for every $j \in [N]$, $\mathsf{hpk}_j := (j, k_j, \tau^{(1)}_{0,j}, \tau^{(0)}_{0,2j})$.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu)$**:** Rewrite $\mathbf{x}$ as $(\mathbf{x}_0, \ldots, \mathbf{x}_{2N-1})$ and compute

$$\begin{aligned} \forall j \in [2N]: \qquad (\mathbf{d}^{(0)}_{1,i}, \tau^{(0)}_{1,j}) &\leftarrow \mathsf{ABE.Hash}_0(\mathsf{mpk}', \mathbf{x}_j) \\ (\mathbf{d}^{(1)}_1, \tau^{(1)}_1) &\leftarrow \mathsf{ABE.Hash}_1(\mathsf{mpk}', (\mathbf{d}^{(0)}_{1,0}, \mathbf{d}^{(0)}_{1,1}, \ldots, \mathbf{d}^{(0)}_{1,2N-1})) \\ (\mathbf{d}, \tau^{(2)}) &\leftarrow \mathsf{ABE.Hash}_2(\mathsf{mpk}', (\mathbf{d}^{(1)}_0, \mathbf{d}^{(1)}_1)). \end{aligned}$$

Output $\mathsf{ct} := (\mathbf{x}, c)$ where $c \xleftarrow{\$} \mathsf{ABE.Complete}(\mathsf{mpk}', \mathbf{d}, \mu)$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{hpk}_i, \mathsf{sk})$**:** Rewrite $\mathbf{x}$ as $(\mathbf{x}_0, \ldots, \mathbf{x}_{2N-1})$ and compute

$$\begin{aligned} \forall j \in [2N]: \qquad (\mathbf{d}^{(0)}_{1,j}, \tau^{(0)}_{1,j}) &\leftarrow \mathsf{ABE.Hash}_0(\mathsf{mpk}', \mathbf{x}_j) \\ (\mathbf{d}^{(1)}_1, \tau^{(1)}_1) &\leftarrow \mathsf{ABE.Hash}_1(\mathsf{mpk}', (\mathbf{d}^{(0)}_{1,0}, \mathbf{d}^{(0)}_{1,1}, \ldots, \mathbf{d}^{(0)}_{1,2N-1})) \\ (\mathbf{d}, \tau^{(2)}) &\leftarrow \mathsf{ABE.Hash}_2(\mathsf{mpk}', (\mathbf{d}^{(1)}_0, \mathbf{d}^{(1)}_1)) \end{aligned}$$

and set $\tau_i \leftarrow \tau^{(2)} \cup \tau^{(1)}_{0,i} \cup \tau^{(1)}_1 \cup \tau^{(0)}_{0,2i} \cup \mathsf{sk} \cup \bigcup_{j\in[2N]} \tau^{(0)}_{1,j}$. Output $\mathsf{ABE.Dec}((\tau_i, c), k_i)$.

Figure 10: Rate-1 Register ABE for RAM

Notice that in Hybrid 2, the view of the adversary is independent of the challenge bit. This ends the proof. □

Acknowledgments

D.A. and G.M. are supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation). G.M. is also funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972. L.R. is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement number 101124977 (DECRYPSIS) and the Danish Independent Research Council under Grant-ID DFF-0165-00107B (C3PO).

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Berlin, Heidelberg, May / June 2010. 18

[ACFQ22] Prabhanjan Ananth, Kai-Min Chung, Xiong Fan, and Luowen Qian. Collusion-resistant functional encryption for RAMs. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 160–194. Springer, Cham, December 2022. 3

[AKY24a] Shweta Agrawal, Simran Kumari, and Shota Yamada. Attribute based encryption for turing machines from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 352–386. Springer, Cham, August 2024. 7

[AKY24b] Shweta Agrawal, Simran Kumari, and Shota Yamada. Compact pseudorandom functional encryption from evasive LWE. Cryptology ePrint Archive, Paper 2024/1719, 2024. 7

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 455–472. Springer, Cham, November 2018. 4

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Oblivious tensor evaluation and its applications. In *57th ACM STOC*, 2025. 7, 11, 43

[AMZ24] Shweta Agrawalr, Giulio Malavolta, and Tianwei Zhang. Time-lock puzzles from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 425–456. Springer, Cham, August 2024. 4, 12

[AP09] Joel Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, 2009. 19

[ARS24] Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 301–330. Springer, Cham, May 2024. 43

[AY20] Shweta Agrawal and Shota Yamada. Optimal broadcast encryption from pairings and LWE. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 13–43. Springer, Cham, May 2020. 5

[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. 3, 4

[BDGM22] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for IO: Circular-secure LWE suffices. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl, July 2022. 3

[BG25] Nir Bitansky and Rachit Garg. Succinct randomized encodings from laconic function evaluation, faster and simpler. In *EUROCRYPT 2025*, 2025. 4, 12

[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, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Berlin, Heidelberg, May 2014. 3, 7, 9, 13, 18, 19, 20, 35, 36, 37, 42

[BGL$^+$15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 439–448, 2015. 3, 4

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012. 22

[BJSS25] Elette Boyle, Abhishek Jain, Sacha Servan-Shreiber, and Akshayaran Srinivasan. Simultaneous-message and succinct secure computation. In *Cryptology ePrint Archive, Paper 2025/096*, 2025. 43

[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, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Cham, November 2017. 42

[BÜW24] Chris Brzuska, Akin Ünal, and Ivy K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In *ASIACRYPT 2024*, 2024. 7

[BV16] Zvika Brakerski and Vinod Vaikuntanathan. Circuit-ABE from LWE: Unbounded attributes and semi-adaptive security. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 363–384. Springer, Berlin, Heidelberg, August 2016. 7

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for ram programs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 429–437, 2015. 3, 4

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 523–552. Springer, Berlin, Heidelberg, May / June 2010. 18

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. Cryptology ePrint Archive, Paper 2025/044, 2025. 5

[CW23] Valerio Cini and Hoeteck Wee. ABE for circuits with poly (λ)-sized keys from LWE. In *64th FOCS*, pages 435–446. IEEE Computer Society Press, November 2023. 7

[CW25] Valerio Cini and Hoeteck Wee. Faster abe for turing machines from circular evasive lwe, 2025. 3, 7

[DHM$^+$24] Fangqi Dong, Zihan Hao, Ethan Mook, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and ABE for RAMs from (ring-)LWE. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 107–142. Springer, Cham, August 2024. 3, 4, 5, 7

[DHMW24] Fangqi Dong, Zihan Hao, Ethan Mook, and Daniel Wichs. Laconic function evaluation, functional encryption and obfuscation for RAMs with sublinear computation. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 190–218. Springer, Cham, May 2024. 3

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Berlin, Heidelberg, August 2016. 43

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008. 21

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Berlin, Heidelberg, May 2004. 21

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. 6, 24

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. 4, 43

[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, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Berlin, Heidelberg, August 2013. 3

[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 ACM STOC*, pages 555–564. ACM Press, June 2013. 12, 13, 42, 48

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 736–749, 2021. 3

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309. 5

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. 18, 19

[GS18] Sanjam Garg and Akshayaram Srinivasan. A simple construction of iO for Turing machines. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 425–454. Springer, Cham, November 2018. 4

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013. 11, 21, 42

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013. 7

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015. 9

[HHWW19] Ariel Hamlin, Justin Holmgren, Mor Weiss, and Daniel Wichs. On the plausibility of fully homomorphic encryption for RAMs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 589–619. Springer, Cham, August 2019. 3

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, November 2023. 4, 5, 7, 11, 13, 20, 32, 37

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 511–542. Springer, Cham, April 2023. 5

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021. 3

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Cham, May / June 2022. 3

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015. 3, 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, *55th ACM STOC*, pages 595–608. ACM Press, June 2023. 3, 13, 42, 48

[LV22] Alex Lombardi and Vinod Vaikuntanathan. Correlation-intractable hash functions via shift-hiding. In Mark Braverman, editor, *ITCS 2022*, volume 215, pages 102:1–102:16. LIPIcs, January / February 2022. 9

[MV24] Daniele Micciancio and Vinod Vaikuntanathan. SoK: Learning with errors, circular security, and fully homomorphic encryption. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14604 of *LNCS*, pages 291–321. Springer, Cham, April 2024. 24

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018. 4, 9, 11, 38, 42

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 2009. 19

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Berlin, Heidelberg, May 2005. 5

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 195–221. Springer, Cham, December 2022. 7

[Wee22] Hoeteck Wee. Optimal broadcast encryption and CP-ABE from evasive lattice assumptions. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 217–241. Springer, Cham, May / June 2022. 7, 15

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\text{depth}, \lambda)$-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 178–209. Springer, Cham, August 2024. 4, 6, 7, 19

[Wee25] Hoeteck Wee. Almost optimal kp and cp-abe for circuits from succinct lwe. In *EUROCRYPT 2025*, 2025. 5, 7

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 127–156. Springer, Cham, October 2021. 3

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. 22