Succinct Oblivious Tensor Evaluation and Applications: Adaptively-Secure Laconic Function Evaluation and Trapdoor Hashing for All Circuits

Damiano Abram
abram.damiano@protonmail.com
Bocconi University

Giulio Malavolta
giulio.malavolta@hotmail.it
Bocconi University

Lawrence Roy
ldr709@gmail.com
Aarhus University

Abstract

We propose the notion of succinct oblivious tensor evaluation (OTE), where two parties compute an additive secret sharing of a tensor product of two vectors $\mathbf{x} \otimes \mathbf{y}$, exchanging two simultaneous messages. Crucially, the size of both messages and of the CRS is independent of the dimension of $\mathbf{x}$. We present a construction of OTE with optimal complexity from the standard learning with errors (LWE) problem. Then we show how this new technical tool enables a host of cryptographic primitives, all with security reducible to LWE, such as:

- Adaptively secure laconic function evaluation for depth-D functions $f : \{0,1\}^m \to \{0,1\}^\ell$ with communication $m + \ell + D \cdot \mathsf{poly}(\lambda)$.
- A trapdoor hash function for all functions.
- An (optimally) succinct homomorphic secret sharing for all functions.
- A rate-1/2 laconic oblivious transfer for batch messages, which is best possible.

In particular, we obtain the first laconic function evaluation scheme that is adaptively secure from the standard LWE assumption, improving upon Quach, Wee, and Wichs (FOCS 2018). As a key technical ingredient, we introduce a new notion of *adaptive lattice encodings*, which may be of independent interest.

Contents

1 Introduction

Consider the scenario where Alice holds a *long* vector $\mathbf{x}$, Bob holds a *smaller* secret vector $\mathbf{y}$ and, after a single round of simultaneous messages, they should be able to locally compute an additive secret share of the tensor product $\mathbf{x} \otimes \mathbf{y}$ while preserving the privacy of $\mathbf{y}$. That is, after one simultaneous round of messages, Alice computes α and Bob computes β such that:

$$\alpha + \beta = \mathbf{x} \otimes \mathbf{y}.$$

We refer to this problem as *non-interactive oblivious tensor evaluation* (NI-OTE). In this work, we are interested in the communication complexity of secure NI-OTE, i.e., the minimum size of the messages needed in order to compute a correct additive secret sharing, while preserving the privacy of $\mathbf{y}$. While one may intuitively expect that Alice and Bob's messages should be long enough to fully specify both the vectors, this is in fact not so. Counterintuitively, we show that it is possible to complete the above protocol with communication complexity *logarithmic* in the dimensions of the input $\mathbf{x}$.

The objective of this work is to construct explicit protocols for NI-OTE with minimal communication, and to explore the cryptographic consequences of this primitive.

1.1 Our Results

Our main technical contribution is a protocol for NI-OTE with minimal communication complexity, where the security is proven against the standard learning with errors (LWE) assumption [Reg05]. We prove this result in two steps: First, we construct an elementary (half-succinct) NI-OTE protocol where only the message of one party is short, whereas the message of the other party can depend arbitrarily on $|\mathbf{x}|$. Then we show a generic *bootstrapping* procedure that makes the scheme fully succinct, i.e., the messages of both parties are short. Overall, our main result is captured by the following informal theorem statement (treating the security parameter as constant).

Theorem 1.1 (Informal)**.** *If the LWE problem is hard, then there exists an NI-OTE protocol for* $\mathbf{x} \in \mathbb{Z}_q^m$ *and* $\mathbf{y} \in \mathbb{Z}_q^\ell$ *with communication complexity* $\ell \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, \log m)$ *and CRS of size* $\mathsf{poly}(\lambda, \log m)$*.*

Besides being a primitive of independent interest, we show that the existence of our succinct NI-OTE protocol has surprising applications in cryptography.

Application I: Trapdoor Hash Functions. For starters, we show how succinct NI-OTE, combined with recent results in laconic function evaluation [QWW18, HLL23, DHM+24], enables a construction of a trapdoor hash (TDH) function [DGI+19] for all functions (or even RAM programs, from Ring-LWE), where the size of the hash is constant, and the size of the encoding key depends only on the description of the program f, which is optimal. This improves upon prior works [DGI+19, RS21] that constructed TDHs with similar communication complexity for the class of linear functions. This is summarized by the following (informal) theorem statement.

Theorem 1.2 (Informal)**.** *If the LWE problem is hard, then there exists a TDH for functions with depth* D*, where the size of the encoding is bounded by* $(|f| + D) \cdot \mathsf{poly}(\lambda)$*. Additionally assuming the hardness of* circular *LWE, we obtain a bound on the size of the encodings of* $|f| \cdot \mathsf{poly}(\lambda)$*.*

Application II: Succint Homomorphic Secret Sharing and More. As a direct consequence of the above result, we obtain a new protocol of succinct homomorphic secret sharing [ARS24] for all functions with logarithmic communication complexity in the first party's input $\mathbf{x}$, which is optimal. Prior work [ARS24] only supported NC_1 circuits and had communication complexity proportional to $|\mathbf{x}|^\varepsilon$, for some $\varepsilon \in O(1)$. In addition, we obtain a new *batched* laconic oblivious transfer protocol [CDG+17], with constant-size receiver's message and with rate $1/2$, which is best possible.

We refer the reader to Section 1.3 for a more detailed and precise discussion on these primitives, along with additional applications of succinct NI-OTE such as spooky encryption, and pseudorandom correlation generators.

Application III: Laconic Function Evaluation. Finally, we show how to leverage succinct NI-OTE to construct a laconic function evaluation (LFE) [QWW18] protocol that is simultaneously:

- *Adaptively secure*: The attacker can choose the input adaptively, possibly depending on the public parameters.
- *Rate-1*: The size of the encoding equals the size of the input, plus the size of the output, plus an additive factor.

Prior to our work, even constructing LFE with either of the two properties was considered an open problem. The question of adaptively-secure LFE from LWE was raised in [QWW18], where they proposed a construction provable against the *adaptive* LWE assumption, whereas our construction is adaptively secure against the *standard* LWE assumption. Furthermore, we present a counterexample against the adaptive LWE assumption (Appendix 7) which translates into an adaptive attack against their scheme, underscoring the need for constructions proven adaptively secure against standard assumptions.

The question of rate-1 LFE was considered in [Wee24], where they proposed a construction from ℓ-succinct LWE, a recently-introduced variant of the LWE assumption. We improve upon this work by relying only on the standard LWE problem. Overall, our results can be summarized as follows:

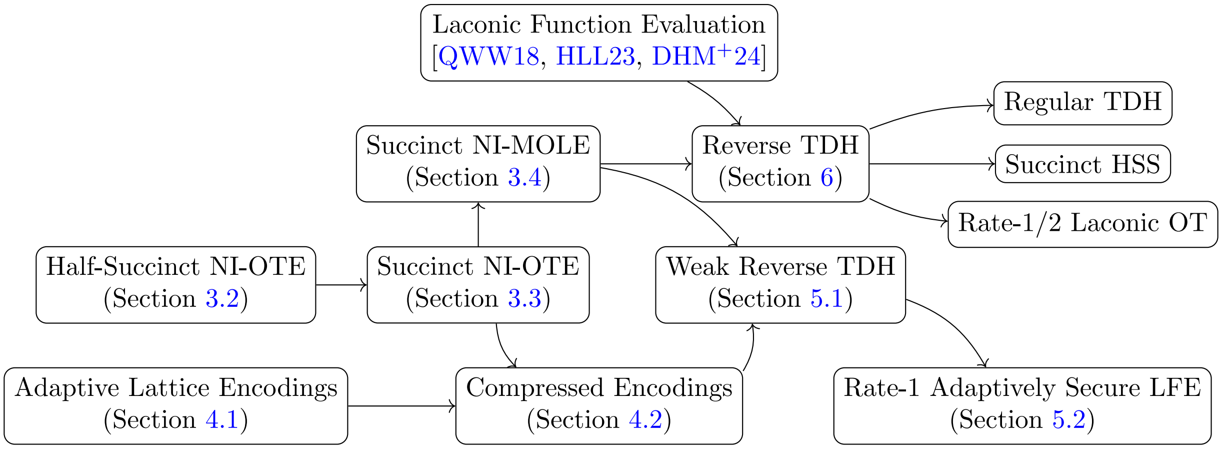

Figure 1: A schematic representation of our results, and a summary of the implications to different cryptographic primitives.

Theorem 1.3 (Informal)**.** *If the LWE problem is hard, there exists an adaptively secure LFE for depth-D functions $f : \{0,1\}^m \to \{0,1\}^\ell$ with communication $m + \ell + D \cdot \mathsf{poly}(\lambda)$.*

As a key technical ingredient, we present a new variant of *homomorphic lattice encodings* [BGG+14] that naturally supports adaptive security. This is the first construction of homomorphic lattice encodings that departs from the framework of [BGG+14], and we expect it to find other applications in the future.

A diagram summarizing our results is given in Figure 1.

1.2 Technical Outline

From now on, we call Alice *the hasher* and Bob *the encoder*. We start by presenting a half-succinct OTE, i.e., an OTE protocol where only the hasher's message is succinct in its input size. This is inspired by the work of [ARS24] and we extend their ideas in the context of tensor products. Suppose that we work over $\mathbb{Z}_q$, the hasher's input is a vector over $\mathbb{Z}_2^m$, while the encoder's input lies in $\mathbb{Z}_q^\ell$.

The construction relies on a setup that outputs a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ where $m > n$. To hash $\mathbf{x}$, we simply compute an SIS-based hash $\mathbf{d} \leftarrow \mathbf{A} \cdot \mathbf{x}$. To encode $\mathbf{y}$, on the other hand, we compute

$$\mathbf{C} \leftarrow \mathbf{A}^\intercal \cdot \mathbf{S} + \mathbf{E} + \mathbf{I}_m \otimes \mathbf{y}^\intercal$$

where $\mathbf{S} \xleftarrow{\$} \mathbb{Z}_q^{n \times (m \cdot \ell)}$ and $\mathbf{E} \xleftarrow{\$} \chi(1^\lambda)$. Above, we use $\mathbf{I}_m$ to denote the $m \times m$ identity matrix and χ to denote a low-norm distribution. Notice that, under LWE, $\mathbf{C}$ leaks no information about $\mathbf{y}$. Suppose that Alice and Bob exchanged $\mathbf{C}$ and $\mathbf{d}$. Alice can compute a "noisy" share of $\mathbf{x} \otimes \mathbf{y}$ by computing

$$\begin{aligned}\mathbf{v} := \mathbf{x}^\intercal \cdot \mathbf{C} &= \mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{S} + \mathbf{x}^\intercal \cdot \mathbf{E} + \mathbf{x}^\intercal \cdot (\mathbf{I}_m \otimes \mathbf{y}^\intercal) \\ &= \mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{S} + \mathbf{x}^\intercal \cdot \mathbf{E} + (\mathbf{x}^\intercal \otimes \mathbf{y}^\intercal).\end{aligned}$$

Bob can derive instead his own "noisy" share by computing

$$\mathbf{w} := -\mathbf{d}^\intercal \cdot \mathbf{S} = -\mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{S}.$$

It is easy to see that $\mathbf{v}+\mathbf{w}=\mathbf{x}^\intercal\otimes\mathbf{y}^\intercal+\mathbf{x}^\intercal\cdot\mathbf{E}$. Moreover, since $\mathbf{x}\in\mathbb{Z}_2^m$, the magnitude of $\mathbf{x}^\intercal\cdot\mathbf{E}$ is small. From this we can derive a fully correct, half succinct OTE over $\mathbb{Z}_p$ where p is a sufficiently small divisor of q: Instead of hashing $\mathbf{x}$, hash its bit decomposition, instead of encoding $\mathbf{y}$, encode $q/p\cdot\mathbf{y}\otimes\mathbf{g}_q^\intercal$ where $\mathbf{g}_q$ is the gadget (row) vector $(1,2,\ldots,2^{\log q})$. To reconstruct an exact secret-sharing of $\mathbf{x}^\intercal\otimes\mathbf{y}^\intercal \bmod p$, it is sufficient to apply a linear operation on the shares and round the result over $\mathbb{Z}_p$ following the ideas of [DHRW16].

An Attempt at Bootstrapping. Now, let us try to achieve full succinctness. In the construction we described above, the encoder's message has size $m\cdot\ell$, while the hash size is independent of both m and ℓ. Inspired by [ARS24], we rely on the non-interactive nature of the primitive and the linearity of the functionality. Specifically, we observe that the encoding can be reused across multiple hashes: Imagine that Alice's input is now much bigger, let's say of dimension $M\gg m,\ell$. We can split $\mathbf{x}$ into N blocks $\mathbf{x}_0,\ldots,\mathbf{x}_{N-1}$ of dimension m and hash each of them. Alice would therefore send N digests $\mathbf{d}_0,\ldots,\mathbf{d}_{N-1}$. At this point, if Bob sends a single encoding for $\mathbf{y}\in\mathbb{Z}_q^\ell$, the parties are able to derive a secret-sharing of $\mathbf{x}_i\otimes\mathbf{y}$ for every $i\in[N]$. By rearranging these, we can easily retrieve a secret sharing of $\mathbf{x}\otimes\mathbf{y}$. Notice that now the size of the encoding is sublinear in the size of $\mathbf{x}$. The communication from Alice's side has however increased proportionally to M/m. In other words, small encodings come at the price of bigger digests.

We however continue in this direction: We keep m small (a constant $t\cdot n$) and instead we find a way to compress the N digests. We rely on a property of our noisy half-succinct construction: Bob's share consists of product between the digests and the randomness $\mathbf{S}$ used in his encoding. We also observe that if Alice and Bob could obtain a secret-sharing of $\mathbf{d}_i\otimes\mathsf{vec}(\mathbf{S})$[1] for every $i\in[N]$, they would also be able to derive a secret-sharing of Bob's noisy shares: it would just suffice to apply a local linear operation on the shares. From this we could easily derive a noisy secret-sharing of $\mathbf{x}\otimes\mathbf{y}$ (which can be later rounded as we sketched above).

Then, why not to use our half-succinct OTE to derive a secret-sharing of $\mathbf{d}'\otimes\mathsf{vec}(\mathbf{S})$
where $\mathbf{d}'$ denotes the concatenation of $\mathbf{d}_0,\ldots,\mathbf{d}_{N-1}$?

With this approach, we may derive a secret-sharing of the output without asking Alice to send messages as big as $\mathbf{d}'$. Moreover, if we keep recursing, we may end up with a protocol in which Alice sends a single Merkle hash of $\mathbf{x}$, whereas Bob sends an encoding for each recursive step (at the i-th step, Bob encodes the randomness $\mathbf{S}_{i-1}$ used in the previous level). This technique could even reduce the communication to $\mathsf{polylog}(M)$! Alas, we cannot: $\mathbf{S}$ is too big. At each recursion step, the size of $\mathbf{S}$ increases by a factor of $n\cdot m$ and consequently so does the size of Bob's encodings. We need to find a way to decrease the size of $\mathbf{S}$ while preserving the linearity of Bob's share derivation procedure.

Decreasing the size of S. Instead of sampling a random $\mathbf{S}$, we generate a pseudorandom one using LWE. Specifically, include $m\cdot\ell$ random $n\times n$ matrices $\mathbf{B}_0,\ldots,\mathbf{B}_{\ell\cdot m-1}$ as part of the setup. Then, at encoding time, we sample a random vector $\mathbf{s}\xleftarrow{\$}\mathbb{Z}_q^n$ and we set the i-th row of $\mathbf{S}$ to be

[1] $\mathsf{vec}(\mathbf{S})$ denotes the vectorisation of $\mathbf{S}$.

$\mathbf{B}_i \cdot \mathbf{s} + \mathbf{e}_i$ where $\mathbf{e}_i \overset{\$}{\leftarrow} \chi(1^\lambda)$. In matrix notation, we obtain that

$$\mathbf{S} = \underbrace{\begin{pmatrix} \mathbf{B}_0 & \dots & \mathbf{B}_{\ell\cdot m-1} \end{pmatrix}}_{\mathbf{B}} \cdot \underbrace{\begin{pmatrix} \mathbf{s} & & & \\ & \mathbf{s} & & \\ & & \ddots & \\ & & & \mathbf{s} \end{pmatrix}}_{m\cdot\ell \text{ times}} + \underbrace{\begin{pmatrix} \mathbf{e}_0 & \dots & \mathbf{e}_{\ell\cdot m-1} \end{pmatrix}}_{\mathbf{E}'} = \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}) + \mathbf{E}'.$$

The encoding of $\mathbf{y}$ becomes

$$\mathbf{C} = \mathbf{A}^\intercal \cdot \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}) + \mathbf{A}^\intercal \cdot \mathbf{E}' + \mathbf{E} + \mathbf{I}_m \otimes \mathbf{y}^\intercal.$$

We also introduce another modification: Instead of sampling the entries of $\mathbf{A}$ and $\mathbf{B}$ uniformly at random over $\mathbb{Z}_q$, we sample them uniformly over $\mathbb{Z}_2$. This trick ensures that the magnitude of $\mathbf{A}^\intercal \cdot \mathbf{E}'$ remains small, while, at the same time, it does not compromise security: LWE with respect to random *binary* matrices is known to be as hard as standard LWE [BLMR13]. With these modifications to our half-succinct OTE, Alice's share becomes

$$\begin{aligned} \mathbf{v} := \mathbf{x}^\intercal \cdot \mathbf{C} &= \mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}) + \mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{E}' + \mathbf{x}^\intercal \cdot \mathbf{E} + \mathbf{x}^\intercal \cdot (\mathbf{I}_m \otimes \mathbf{y}^\intercal) \\ &= \mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}) + \mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{E}' + \mathbf{x}^\intercal \cdot \mathbf{E} + (\mathbf{x}^\intercal \otimes \mathbf{y}^\intercal). \end{aligned}$$

Bob's share becomes instead

$$\mathbf{w} := -\mathbf{d}^\intercal \cdot \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}) = -\mathbf{x}^\intercal \cdot \mathbf{A}^\intercal \cdot \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}).$$

This allows us to apply recursion without blowing up the size of $\mathbf{s}$: Since $\mathbf{w}$ is a bilinear function of $\mathbf{d}$ and $\mathbf{s}$, if the parties hold a "noisy" secret-sharing of $\mathbf{d}' \otimes \mathbf{s}$, they can easily convert it into a "noisy" secret-sharing of Bob's share by computing a local linear operation depending on $\mathbf{B}$. Moreover, since $\mathbf{B}$ is a binary matrix, this linear computation will not significantly increase the "noisiness" of the secret-sharing. At each recursion step, the size of $\mathbf{d}'$ decreases by a factor of $t(\lambda) := m/n$; the size of $\mathbf{s}$, on the other hand, remains always the same. The final result is an LWE-based succinct OTE where the digest and the CRS dimensions are $\mathsf{poly}(\lambda)$ and the encoding dimension is $O(\log M) \cdot \ell \cdot \mathsf{poly}(\lambda)$.

Succinct MOLEs and VOLEs. No, this paragraph is not about tiny mammals: It's about two cryptographic primitives called *matrix oblivious linear evaluation* and (non-interactive, half-chosen) *vector oblivious linear evaluation*. In a MOLE, Alice holds a matrix $\mathbf{M} \in \mathbb{Z}_q^{m\times\ell}$ where $m \gg \ell$, whereas Bob holds a secret vector $\mathbf{x} \in \mathbb{Z}_q^\ell$. Their goal is to derive a secret-sharing of $\mathbf{M} \cdot \mathbf{x}$ in one round and without revealing any information about $\mathbf{x}$. A VOLE corresponds to a MOLE in the special case $\ell = 1$.

We would like to minimize the communication complexity of these protocols, especially in relation to m. It is easy to see that the succinct OTE protocol we just presented gives immediately a succinct MOLE where communication scales logarithmically in m: First, we use the succinct OTE protocol to compute a secret-sharing of $\mathsf{vec}(\mathbf{M}) \otimes \mathbf{x}$, then, we apply local linear operations on the shares to obtain a secret-sharing of $\mathbf{M} \cdot \mathbf{x}$. Along the way, this solves a question left open in [ARS24]: We have just built the first non-interactive (half-chosen) VOLE with logarithmic communication in m from LWE.

Reverse Trapdoor Hashing. We observe that many laconic function evaluation schemes have a particular structure [QWW18, HLL23, DHM+24, Wee24]. First of all, their digests consist of matrices $\mathbf{A}_f$ with a constant number of columns and a number of rows proportional to the output size of f. Moreover, the encoding can be split into two parts: A function-independent pre-encoding E and an input-independent post-encoding c consisting of an LWE-like sample where the matrix is (essentially) the digest $\mathbf{A}_f$ and the secret $\mathbf{s}$ is a random vector generated by the pre-encoding procedure. Finally, the output is computed by rounding the sum $c + \mathsf{Eval}(E, f)$.

We observe that, if we ignore the noise, LWE samples are essentially matrix-vector multiplications. Therefore, by relying on the succinct MOLE we just built, the parties can derive a noisy secret-sharing of the post-encoding c in a single round of simultaneous interaction and with logarithmic communication in the output size of f. Moreover, the encoder can send the pre-encoding E along with its MOLE message. In this way, the parties can derive a secret-sharing of the output without the need for further interaction.

This yields the first rate-1 *reverse* trapdoor hashing scheme. Usually, in rate-1 trapdoor hashing, we obtain a secret-sharing of $f(x)$ by sending a digest of x and generating an encoding key for f. In reverse trapdoor hashing, we do the opposite: We send a digest for f and an encoding key for x. In our construction, the former corresponds to the MOLE hash of $\mathbf{A}_f$, whereas the latter corresponds to the pre-encoding E and the MOLE encoding of $\mathbf{s}$.[2]

Adaptive Lattice Encodings. Much of the recent advancement in lattice-based homomorphic cryptography can be traced back to a single technique introduced in 2014 by Boneh et al. [BGG+14]: BGG^+ encodings. Suppose that we are provided with a CRS consisting of a matrix $\mathbf{A} \in \mathbb{Z}_q^{k\times(\ell\cdot k\cdot\log q)}$. A BGG^+ encoding of a bit string $\mathbf{x} \in \{0,1\}^\ell$ consists of the vector

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

where $\mathbf{G} := \mathbf{I}_k \otimes \mathbf{g}_q$, $\mathbf{s} \overset{\$}{\leftarrow} \mathbb{Z}_q^k$ and $\mathbf{e} \overset{\$}{\leftarrow} \chi(1^\lambda)$. These encodings have amazing homomorphic properties: For any function $f : \{0,1\}^\ell \to \{0,1\}$, there exist efficiently computable, low-norm matrices $\mathbf{H}_f$ and $\mathbf{H}_{f,x}$ (independent of the random $\mathbf{s}$ and the noise $\mathbf{e}$ of the encoding) such that

$$\mathbf{c} \cdot \mathbf{H}_{f,x} \approx \mathbf{s}^\intercal \cdot (\mathbf{A}_f - f(x) \cdot \mathbf{G})$$

where $\mathbf{A}_f := \mathbf{A} \cdot \mathbf{H}_f$. Alas, BGG^+ encodings are secure only in the selective setting: If $\mathbf{x}$ is independent of $\mathbf{A}$, the encoding $\mathbf{c}$ looks like a random vector, if, however, $\mathbf{x}$ is adaptively chosen after seeing $\mathbf{A}$, there exists an attack that allows us to recover $\mathbf{s}$. To see why, observe that there exists a binary matrix $\mathbf{H}'$ such that, for any matrix $\mathbf{M} \in \mathbb{Z}_q^{k\times k}$, if $\mathbf{x} = \mathsf{Bits}(\mathbf{M})$, we have that

$$\mathbf{c} \cdot \mathbf{H}' \approx \mathbf{s}^\intercal \cdot (\mathbf{A} \cdot \mathbf{H}' - \mathbf{M}).$$

Suppose for convenience that q is a power of 2. To recover the i-th most significant bit of $\mathbf{s}$, it is sufficient to set $\mathbf{M} := \mathbf{A} \cdot \mathbf{H}' - 2^i \cdot \mathbf{I}_k$ and compute the most significant bit of $\mathbf{c} \cdot \mathbf{H}'$. This proves that the Adaptive LWE assumption of [QWW18] does not hold in the optimistic parameter setting in which ℓ can be arbitrarily bigger than k. Notice in the provable parameter setting (where the encodings are secure under the *subexponential* hardness of LWE), our attack fails as the bound on ℓ is too small to encode $\mathbf{M} := \mathbf{A} \cdot \mathbf{H}' - 2^i \cdot \mathbf{I}_k$.

[2] We mention that we can recover the usual notion of trapdoor hashing via universal circuits. On the other hand, the reverse implication does not seem to trivially hold, since the universal circuit would introduce an efficiency penalty in the size of the encoding.

To circumvent the attack without relying on complexity leveraging (and therefore obtain better asymptotic parameters), we introduce a new version of lattice encodings: The encoding of $\mathbf{x}$ is now

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

where $\mathbf{s} \overset{\$}{\leftarrow} \mathbb{Z}_q^k$, $\mathbf{r} \overset{\$}{\leftarrow} \mathbb{Z}_q^k$ and $\mathbf{e} \overset{\$}{\leftarrow} \chi(1^\lambda)$. We call $\mathbf{s}$ *the encryption key* of the encoding, whereas we refer to $\mathbf{r}$ as *the authentication key.* We say that $\mathbf{c}$ is a $(\mathbf{s}, \mathbf{r})$-encoding. Observe that BGG^+ encodings correspond to the special case in which $\mathbf{r} = -\mathbf{s}$. It is easy to see also that these encodings are adaptively secure under standard LWE, no matter the value of $\mathbf{x}$ and $\mathbf{r}$.

The drawback, however, becomes clear when we look at the homomorphic properties of the modified scheme. It is easy to see that the construction is linearly homomorphic, however, we no longer know how to perform multiplications. Or at least, we do not know *when the factors are encoded using the same keys.* If instead the authentication key of the first encoding matches the encryption key of the second one, there is a solution: Suppose that we want to multiply the encodings $\mathbf{c}_x = \mathbf{s}^\intercal \cdot \mathbf{A} + \mathbf{r}^\intercal \cdot (x \cdot \mathbf{G}) + \mathbf{e}_x^\intercal$ and $\mathbf{c}_y = \mathbf{r}^\intercal \cdot \mathbf{B} + \mathbf{t}^\intercal \cdot (y \cdot \mathbf{G}) + \mathbf{e}_y^\intercal$. We observe that

$$\mathbf{c}_z := -\mathbf{c}_x \cdot \mathbf{G}^{-1}(\mathbf{B}) + x \cdot \mathbf{c}_y \approx -\mathbf{s}^\intercal \cdot \mathbf{A}\mathbf{G}^{-1}(\mathbf{B}) + \mathbf{t}^\intercal \cdot (xy \cdot \mathbf{G}).$$

In other words, we have obtained an encoding of $x \cdot y$ using $\mathbf{s}$ as encryption key and $\mathbf{t}$ as authentication key. Moreover, the matrix relative to the encoding is $-\mathbf{A} \cdot \mathbf{G}^{-1}(\mathbf{B})$. Notice that this is publicly computable, no need to know the keys or the plaintexts! To summarise, we are able to compute linear operations between $(\mathbf{s}, \mathbf{r})$-encodings obtaining other $(\mathbf{s}, \mathbf{r})$-encodings. Moreover, we are able to perform multiplications between $(\mathbf{s}, \mathbf{r})$-encodings and $(\mathbf{r}, \mathbf{t})$-encodings, obtaining $(\mathbf{s}, \mathbf{t})$-encodings as results.

Compressing Adaptive Lattice Encodings. We present a procedure to compress our adaptive lattice encodings, where a compressed encoding of $\mathbf{x} \in \{0,1\}^\ell$ will have size $\mathsf{poly}(\log \ell, \lambda)$. By leveraging the knowledge of $\mathbf{x}$, it can then be re-expanded into a standard adaptive encoding. Once again, our techniques rely on our succinct OTE protocol and the (bi)linear structure of Bob's share derivation. Specifically, the compressed encoding is composed of two parts $\mathbf{h}$ and E. The former consists of an adaptive lattice encoding of $\mathbf{d} := \mathsf{OTE.Hash}(\mathbf{x})$. Let $\mathbf{r}$ be the authentication key of $\mathbf{h}$; then E consists of an OTE encoding of a fresh authentication key $\mathbf{t}$ where $\mathbf{r}$ is used as randomness for E (this is secure as $\mathbf{h}$ looks random even when $\mathbf{r}$ is leaked). We observe that

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

In other words, $\mathbf{h}$ can be viewed as some sort of encoding of $\mathbf{d}^\intercal \otimes \mathbf{r}^\intercal$ where $\mathbf{r}$ is the randomness of E. Now, Bob's share of $\mathbf{x} \otimes \mathbf{t}$ would be $\mathbf{w} := \mathbf{P} \cdot (\mathbf{d} \otimes \mathbf{r})$, where $\mathbf{P}$ is a public low-norm matrix derived from $\mathbf{B}$. Thus, by multiplying $\mathbf{h}$ on the right by $\mathbf{P}$, we derive

$$\mathbf{h}' := \mathbf{h} \cdot \mathbf{P} \approx \mathbf{s}^\intercal \cdot \mathbf{A}\mathbf{P} + \mathbf{w}^\intercal \otimes \mathbf{g}_q.$$

Using $\mathbf{x}$ and E, we can also derive the other share $\mathbf{v}$. We conclude by observing that

$$\begin{aligned}\mathbf{h}' + \mathbf{v}^\intercal \otimes \mathbf{g}_q &\approx \mathbf{s}^\intercal \cdot \mathbf{A}\mathbf{P} + (\mathbf{x}^\intercal \otimes \mathbf{t}^\intercal \otimes \mathbf{g}_q) \\ &= \mathbf{s}^\intercal \cdot \mathbf{A}\mathbf{P} + \mathbf{t}^\intercal \cdot (\mathbf{x}^\intercal \otimes \mathbf{G})\end{aligned}$$

We have just obtained a $(\mathbf{s}, \mathbf{t})$-encoding of $\mathbf{x}$.

Rate-1 Adaptive LFE. We build our rate-1 adaptive LFE scheme using our compressed adaptive encodings in two steps: First, we construct a weak variant of (adaptive) reverse TDH for the functions that map pairs $(\mathbf{x}, \mathbf{a})$ to $f(\mathbf{x}) \otimes \mathbf{a}$ where $f \in \mathsf{NC}_1$. The scheme satisfies all the security properties of standard reverse TDH, however, in order for correctness to hold, the hasher needs to know $\mathbf{x}$ (but not $\mathbf{a}$) for the derivation of her share. On the positive side, the scheme achieves polylogarithmic communication in the size of $\mathbf{x}$ (but not $\mathbf{a}$). As a second step, we use our weak reverse TDH to build an adaptive LFE scheme for all depth-D functions $f : \{0,1\}^m \to \{0,1\}^\ell$. The size of the hash will be $\mathsf{poly}(\lambda)$ whereas the size of the encoding will be $\ell + m + D \cdot \mathsf{poly}(\lambda)$.

Step I: a Weak Reverse TDH. We start by describing the weak reverse TDH. Suppose that we want to evaluate functions of depth at most $\log d$: Each of them can be converted into an RMS program of depth d [BGI16]. We recall that an RMS program consists of an arithmetic circuit over $\mathbb{Z}$ where multiplications are allowed only if one of the factors is an input.

We start by describing the generation of encoding keys: We sample a chain of d keys $\mathsf{s}_0, \dots, \mathbf{s}_{d-1}$, we set $\mathbf{s}_d := \mathbf{a}$ and we generate compressed encodings $(\mathbf{h}_i, E_i)_{i \in [d]}$ so that, when we expand $(\mathbf{h}_i, E_i)$, we obtain a $(\mathbf{s}_i, \mathbf{s}_{i+1})$-encoding of $(\mathbf{x}, 1)$. Notice that we can homomorphically evaluate f on these encodings: The operation proceeds by levels, starting from level 1 (the inputs) to level d (the output). A level-i encoding consists of any $(\mathbf{s}_0, \mathbf{s}_i)$-encoding. We observe that $(\mathbf{h}_0, E_0)$ gives us a level-1 encoding of the inputs.

In the previous paragraphs, we showed that the levels are closed under linear operations. Moreover, we can also perform multiplications by inputs: If the first factor belongs to level i, we can multiply it by the $(\mathbf{s}_i, \mathbf{s}_{i+1})$-encoding of the other factor (the input), which can be derived from $(\mathbf{h}_i, E_i)$. Finally, we can perform hops across levels (always from level i to level $i+1$) by performing multiplications by 1[3]. To summarise, given $\mathbf{x}$ and $(\mathbf{h}_i, E_i)_{i \in [d]}$, Alice is able to derive an encoding

$$\begin{aligned} \mathbf{c}_f &\approx \mathbf{s}_0^\intercal \cdot \mathbf{A}_f + \mathbf{a}^\intercal \cdot (f(\mathbf{x})^\intercal \otimes \mathbf{G}) \\ &= \mathbf{s}_0^\intercal \cdot \mathbf{A}_f + (f(\mathbf{x})^\intercal \otimes \mathbf{a}^\intercal \otimes \mathbf{g}_q). \end{aligned}$$

To derive a secret-sharing of $f(\mathbf{x}) \otimes \mathbf{a}$, it is therefore sufficient that the parties run a succinct MOLE to compute a secret-sharing of $\mathbf{A}_f^\intercal \cdot \mathbf{s}_0$ similarly to what we did for the other reverse TDH we built. At that point, it is just a matter of applying local linear computations and rounding.

Step II: Building Rate-1 Adaptive LFE. It is finally time to talk about the rate-1 adaptive LFE scheme. Our approach is the following: We pick a *constant* d and we decompose our function f as $f_{L-1} \circ \dots \circ f_0$ where each f_i is described by an RMS program of depth d. Our goal is to use our reverse TDH to evaluate all f_i until we obtain the output. The issue is that Bob does not know what input to encode for f_i.

Instead of evaluating f_i, we evaluate a function that allows us to retrieve the encoding key for $\mathbf{x}_i := (f_i \circ \dots \circ f_0)(\mathbf{x})$. Specifically, for every $i \in [L]$, define[4]

$$\hat{f}_i : (\mathbf{x}, \mathbf{a}) \longmapsto \mathsf{OTE.Hash}(f_i(\mathbf{x})) \otimes \mathbf{a}.$$

Suppose that Alice sent a hash for $\hat{f}_i$ for every $i \in [L]$. Suppose also that we somehow managed to find a way to provide Alice with an encoding key for $(\mathbf{x}_i, \mathbf{a}_{i+1})$ where $\mathbf{a}_{i+1} \xleftarrow{\$} \mathbb{Z}_q^k$. Under these premises, the parties can derive an additive secret-sharing $\mathbf{y}_{i+1}^{\mathsf{A}} + \mathbf{y}_{i+1}^{\mathsf{B}} = \mathbf{d}_{i+1} \otimes \mathbf{a}_{i+1}$ where

[3] Remember that the expansion of $(\mathbf{h}_i, E_i)$ provides also an $(\mathbf{s}_i, \mathbf{s}_{i+1})$-encodings of 1.

[4] In our construction, $\mathsf{OTE.Hash}$ has depth 0 because it is linear.

$\mathbf{d}_{i+1} = \mathsf{OTE.Hash}(\mathbf{x}_{i+1})$. We can convert this into a compressed adaptive encoding of $\mathbf{x}_{i+1}$ by making Bob send

$$\mathbf{c}_{i+1} = \mathbf{s}_{i+1}^\intercal \cdot \mathbf{A} + \mathbf{e}_{i+1}^\intercal + (\mathbf{y}_{i+1}^\mathsf{B})^\intercal \otimes \mathbf{g}_q$$

where $\mathbf{s}_{i+1} \xleftarrow{\$} \mathbb{Z}_q^k$ and $\mathbf{e}_{i+1} \xleftarrow{\$} \chi(1^\lambda)$. By adding $(\mathbf{y}_{i+1}^\mathsf{A})^\intercal \otimes \mathbf{g}_q$ to $\mathbf{c}_{i+1}$, Alice can derived a $(\mathbf{s}_{i+1}, \mathbf{a}_{i+1})$-encoding of $\mathbf{d}_{i+1}$. Indeed,

$$\begin{aligned}\mathbf{c}_{i+1} + (\mathbf{y}_{i+1}^\mathsf{A})^\intercal \otimes \mathbf{g}_q &\approx \mathbf{s}_{i+1}^\intercal \cdot \mathbf{A} + (\mathbf{y}_{i+1}^\mathsf{B} + \mathbf{y}_{i+1}^\mathsf{A})^\intercal \otimes \mathbf{g}_q \\ &= \mathbf{s}_{i+1}^\intercal \cdot \mathbf{A} + (\mathbf{d}_{i+1}^\intercal \otimes \mathbf{a}_{i+1}^\intercal \otimes \mathbf{g}_q) \\ &= \mathbf{s}_{i+1}^\intercal \cdot \mathbf{A} + \mathbf{a}_{i+1}^\intercal \cdot (\mathbf{d}_{i+1}^\intercal \otimes \mathbf{G}).\end{aligned}$$

Bob can of course send also the other information needed to complete the encoding key for $(\mathbf{x}_{i+1}, \mathbf{a}_{i+2})$ as this is independent of f and $\mathbf{x}_{i+1}$. Notice that the size of all this material that Bob sends is independent of the input size and the output size of f. By continuing in this way, the parties end up with an additive secret-sharing of $f(\mathbf{x}) \otimes \mathbf{a}_L$ where $\mathbf{a}_L \xleftarrow{\$} \mathbb{Z}_q^k$.

There is still one matter we need to take care of: In order to perform the operations we just described, Alice needs to know $\mathbf{x}$. So, how can we achieve privacy of the input? The trick is the same as in [BTVW17]: We provide Alice with a GSW [GSW13] encryption of $\mathbf{x}$ using $\mathbf{a}_L$ as a secret key (we also send the corresponding encoding key, which is polylogarithmic in size). Then, instead of evaluating f, we evaluate $f' := \mathsf{GSW.Eval}(f, \cdot)$. At the end, the parties obtain a secret-sharing of $\mathsf{Bits}(\mathsf{ct}) \otimes \mathbf{a}_L$ where ct is a GSW encryption of $f(\mathbf{x})$. Given that $\mathbf{a}_L$ is the secret-key and the GSW decryption consists of linear operations between $\mathbf{a}_L$ and ct (followed by rounding), the parties can obtain a secret-sharing of $f(\mathbf{x})$ by applying only local operations. This immediately gives rate-1 communication in the output. What about rate-1 communication in the input? Well, instead of sending a GSW encryption of $\mathbf{x}$, send $\mathbf{z} := \mathbf{x} \oplus \mathsf{PRG}(K)$ for $K \xleftarrow{\$} \{0,1\}^\lambda$ and a GSW encryption of K. Then, during the evaluation of f', we remove the one-time-pad inside FHE.

1.3 Other Applications

We outline a few additional additional applications of our results. In particular, we discuss some of the new implications from our construction of reverse TDH.

Regular Trapdoor Hash. As an immediate implication of our reverse TDH, we obtain a (regular) TDH [DGI+19] with close to optimal parameters, that is: The size of the encoding key is $|f| \cdot \mathsf{poly}(\lambda, \log m)$ where $|f|$ denotes the size of the description of the encoded function and m denotes the size of its input. For instance, notice that if f is a point function with domain of size L, $|f| = \log L$. We achieve this with a simple application of universal circuits: Instead of hashing a function, we hash the universal circuit $U_\mathbf{x}$ with the input $\mathbf{x}$ hardwired, that on input a function f, returns $f(\mathbf{x})$. Note that the size of the digest is anyway constant (ignoring factors in the security parameter) so the complexity of the TDH only grows with the bit description of f.

To our knowledge, this (along with a concurrent work [BJSS25]) is the first trapdoor hashing schemes that support all functions. This is also the first LWE-based TDH constructions achieving nearly optimal communication for a non-trivial class of functions. To our knowledge, the only other construction where the size of the encoding key is sublinear in the dimension of the input is a recently built pairing-based TDH for point functions [BBD]. Such construction, however, pays the succinctness of the encoding key with a non-succinct CRS of size $O(m)$.

Homomorphic Secret Sharing, Spooky Encryption and Public-Key PCGs. In addition, note that the reconstruction of our reverse TDH is additive over $\mathbb{Z}_2$, thus, a reverse trapdoor hash also implies the existence of a (public-key) 2-party homomorphic secret sharing [BGI16] scheme as follows: Suppose that Alice and Bob respectively hold inputs $\mathbf{x}$ and $\mathbf{y}$ and they want to evaluate the function f. Suppose also that $\mathbf{x}$ is much longer than $\mathbf{y}$. Alice proceeds by hashing the function $f_{\mathbf{x}}$ that maps any $\mathbf{y}$ to $f(\mathbf{x}, \mathbf{y})$. She sends the digest to Bob and keeps the randomness as her part of the share of $\mathbf{x}$. Bob, on the other hand, sends an encoding key ek for $\mathbf{y}$ to Alice. He keeps the corresponding trapdoor td as his share of $\mathbf{y}$. By the correctness of reverse TDH, the parties can locally derive a secret sharing of the output. Notice also that the scheme is succinct in $\mathbf{x}$: The total communication of the protocol is $|\mathbf{y}| \cdot \mathsf{poly}(\lambda, \log|\mathbf{x}|)$!

Previously, succinct homomorphic secret sharing had been built by Abram, Roy and Scholl [ARS24] under several assumptions, including LWE, DCR and DDH over class groups. Their constructions, however, supported only a limited class of functions: Alice and Bob could only compute secret-sharings of $\mathbf{x}^{\intercal} \cdot C(\mathbf{y})$ for any circuit $C \in \mathsf{NC}_1$. Their constructions have also a second drawback: The total complexity of the protocol is $|\mathbf{y}| \cdot |\mathbf{x}|^{\varepsilon} \cdot \mathsf{poly}(\lambda)$ for a constant $\varepsilon \in (0, 1)$. Our solution instead scales polylogarithmically in $|\mathbf{x}|$. On the negative side, unlike [ARS24], our solution does not allow the parties to evaluate a function that is adaptively chosen *after* the secret-sharing phase. We can however plug our MOLE in the constructions of [ARS24] to obtain an HSS scheme that allows the evaluation of any *adaptively* chosen function $\mathbf{x}^{\intercal} \cdot C(\mathbf{y})$ with improved communication $|\mathbf{y}| \cdot \mathsf{poly}(\lambda, \log|\mathbf{x}|)$.

Observe that our succinct HSS scheme can be also viewed as a form of 2-party spooky encryption for additively shared correlation [DHRW16] where one of the parties can just send a small hash of its input instead of a full-size ciphertext. Once again, differently from [DHRW16], our construction does not allow us to choose the correlation after we committed to the inputs.

Finally, our HSS scheme can have interesting applications in the context of (public-key) pseudorandom correlation generators (PCGs) [BCG+19, OSY21, ASY22], especially when the tackled additively-shared correlation takes a long input from Alice: Let $\mathcal{C}(x)$ be the correlation function with long input. Alice can sample $s_0 \xleftarrow{\$} \{0, 1\}^{\lambda}$ and hash the function that maps y to $(\mathcal{C}(x; r_i))_{i \in [n]}$ where $(r_0, \ldots, r_{n-1}) \leftarrow \mathsf{PRG}(s_0 \oplus y)$. Bob instead picks a random $y \xleftarrow{\$} \{0, 1\}^{\lambda}$ and sends its encoding to Alice.

Rate-1/2 Laconic Oblivious Transfer. Since our construction of reverse TDH also extends to RAM programs, we obtain a new construction of laconic oblivious transfer [CDG+17] with rate 1/2 in the batch settings (which is best possible), where one transfers a set of messages with respect to different indices. The receiver hashes the function f_D that has hardwired a database D and, on input a PRF key k, does the following for all indices i and all bits b:

- If $b = D_i$: Return 0.
- Else return $\mathsf{PRF}(k, i)$.

Then the sender, on input a set of indices I and pairs of bits $\{m_{i,0}, m_{i,1}\}_{i \in I}$, samples a key k and sends $\mathsf{ek} \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, k)$ along with:

$$c_{i,b} := \{\mathsf{Dec}(\mathsf{hk}, \mathsf{td}, d)_{i,b} \oplus m_{i,b}\}_{i,b}$$

where we slightly abuse the notation and assume that $\mathsf{Dec}(\mathsf{hk}, \mathsf{td}, d)_{i,b}$ returns the (i, b)-bit of the share (which can be computed by a RAM program in time independent of the size of the database).

Note that if $b = D_i$, then $\mathsf{Dec}(\mathsf{hk}, \mathsf{td}, d)_{i,b} = \mathsf{Enc}(\mathsf{hk}, \mathsf{ek}, f_D, \rho)_{i,b}$ and therefore the receiver can recover $m_{i,b}$. Otherwise the pseudorandomness of PRF guarantees that the message is computationally hidden.

Attribute-Based Non-Interactive Key-Exchange. Finally, reverse TDH implies the existence of a non-interactive key exchange (NIKE) with the following additional property: One of the two parties can include the hash of a function f as part of their public key, whereas the other party holds an input $\mathbf{x}$. The NIKE succeeds if $f(\mathbf{x}) = 1$ and otherwise the key of either party is computationally indistinguishable from random. This can be constructed from reverse TDH in a natural manner: The former party hashes the function F_{f,k_0} that takes as input some $\mathbf{x}$ and a key k_1 and returns 0 if $f(\mathbf{x}) = 1$ and $\mathsf{PRF}(k_0, \mathbf{x}) \oplus \mathsf{PRF}(k_1, \mathbf{x})$ otherwise. If f is satisfied, then both parties hold the same share, that can be used as a shared key, otherwise the pseudorandomness of PRF protects the share of either party.

1.4 Concurrent Work

A concurrent work by Boyle et al. [BJSS25] also construct a family of TDHs for all functions $f : \{0,1\}^m \to \{0,1\}^\ell$, using a similar idea. An important difference is that they rely on the $|\mathbf{x}|^{2/3}$-succinct VOLE protocol from [ARS24], whereas we (implicitly) construct a $\mathsf{polylog}(|\mathbf{x}|)$-succinct one, derived from our OTE. This translates in different parameters for the encoding key of the TDH: The encoding key of [BJSS25] has size $(|f| + \ell^{2/3} + D) \cdot \mathsf{poly}(\lambda)$, whereas in our TDH the encoding key has size $|f| \cdot \mathsf{poly}(\lambda, \log m)$, which is close to optimal.

Besides TDHs, the results in [BJSS25] are largely orthogonal to ours. We also mention here that plugging in our OTE protocol in [BJSS25] leads to similar parameter improvements for their other applications. For instance, following the outline of [BJSS25], OTE yields a rate-1 fully homomorphic encryption (FHE) with optimal parameters. We fully credit [BJSS25] for discovering the connection, and we sketch here the transformation only for the sake of completeness. Take any FHE scheme with almost-linear decryption [BDGM19], i.e., where decryption is a linear function is the secret key $\mathbf{s}$, followed by a rounding, such as [GSW13]. Then add to the public key an MOLE encoding $\mathsf{Enc}(\mathbf{s})$. We can compress m ciphertexts $(\mathbf{c}_1, \dots, \mathbf{c}_m)$ as follows: Stack them into a matrix $\mathbf{C}$, then compute the MOLE-hash $\mathsf{Hash}(\mathbf{C})$, and run the Hash-Eval algorithm, to obtain a additive share of $\mathsf{Round}(\mathbf{C} \cdot \mathbf{s}) \in \{0,1\}^m$. Return the hash $\mathsf{Hash}(\mathbf{C})$, along with the bits of the share. Decryption works by simply running the Encoder-Eval algorithm of the MOLE, and reconstructing the output. Crucially, the compressed ciphertext consists of a hash (whose size is a fixed polynomial in the security parameter) plus m bits, i.e., it is $\mathsf{poly}(\lambda) + m$, which is optimal. This improves upon [BDGM19] since the size of the public key does not depend on m (no amortization is needed).

2 Preliminaries

Notation. We denote the security parameter by λ. We say that a function $\mathsf{negl}(\lambda)$ is negligible if it vanishes faster than any polynomial, i.e., $0 \leq \mathsf{negl}(\lambda) \leq \lambda^{-\omega(1)}$. We say that an event happens with overwhelming probability, if it occurs with probability negligibly close to 1. For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{0, 1, \dots, n-1\}$.

All vectors are denoted using lowercase bold font, whereas matrices are denoted using uppercase bold font; by default, all vectors are columns. Given a matrix $\mathbf{M}$, we denote its transpose by $\mathbf{M}^\intercal$, whereas $\mathsf{vec}(\mathbf{M})$ denotes its vectorisation, i.e., stacking the columns of $\mathbf{M}$ one underneath the other. For any $n \in \mathbb{N}$, we denote the $n \times n$ identity matrix by $\mathbf{I}_n$ and the n-dimensional row vector where all entries are equal to 1 by $\mathbf{1}^n$. We define the Kronecker product between two $n \times m$ matrices $\mathbf{A} \otimes \mathbf{B}$ to be

$$\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} a_{1,1}\mathbf{B} & \dots & a_{1,m}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & \dots & a_{n,m}\mathbf{B} \end{pmatrix}.$$

For a vector $\mathbf{x}$, we denote its infinity norm, i.e., the magnitude of its largest coordinate, by $\|\mathbf{x}\|_\infty$ and we extend this notation to matrices by taking the maximum over their columns. For any integer $q > 0$, let $\mathbf{g}_q$ be the gadget row-vector $(1, 2, \dots, 2^{\log q})$ and we omit the subscript when clear from the context. Let $\mathbf{G}^{-1}$ be the algorithm that takes as input a matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times m}$ and outputs a matrix $\mathbf{M}' \in \mathbb{Z}_2^{(n \cdot \log q) \times m}$, where each column is derived by stacking the bit-decompositions of the entries in the corresponding column of $\mathbf{M}$. Notice that $(\mathbf{I}_n \otimes \mathbf{g}_q) \cdot \mathbf{G}^{-1}(\mathbf{M}) = \mathbf{M}$. We use $\mathsf{Bits}(\mathbf{M})$ to denote $\mathsf{vec}(\mathbf{G}^{-1}(\mathbf{M}))$. Given an integer $x \in \mathbb{Z}_q$ and an integer p that divides q, we use $\lceil x \rfloor_p$ to denote rounding to $\mathbb{Z}_p$, in other words, if $x = y \cdot q/p + z$ where $z \in [-q/2p, q/2p)$, we have $\lceil x \rfloor_p = y$.

We state the following useful lemma about matrices.

Lemma 2.1 (Matrix Linearisation)**.** *There exists a deterministic polynomial-time computable function:*

$$\mathsf{Lin} : \mathbb{Z}_q^{(t \cdot \ell) \times m} \to \mathbb{Z}_q^{t \times (m \cdot \ell)}$$

such that, for any matrix $\mathbf{B} \in \mathbb{Z}_q^{(t \cdot \ell) \times m}$ and any pair of vectors $\mathbf{x} \in \mathbb{Z}_q^m$ and $\mathbf{s} \in \mathbb{Z}_q^\ell$, it holds that:

$$\mathsf{Lin}(\mathbf{B})(\mathbf{x} \otimes \mathbf{s}) = (\mathbf{I}_t \otimes \mathbf{s}^\intercal)\mathbf{B}\mathbf{x} \quad \textit{and} \quad \|\mathsf{Lin}(\mathbf{B})\|_\infty = \|\mathbf{B}\|_\infty.$$

Proof. Note that the h-th entry of $(\mathbf{I}_t \otimes \mathbf{s}^\intercal)\mathbf{B}\mathbf{x}$ is:

$$\sum_{i \in [\ell]} \sum_{j \in [m]} \mathbf{B}_{\ell \cdot h + i, j} \cdot (\mathbf{s}_i \cdot \mathbf{x}_j).$$

In other words, there exists a matrix $\mathsf{Lin}(\mathbf{B})$, obtained by rearranging the entries of $\mathbf{B}$ such that $\mathsf{Lin}(\mathbf{B})(\mathbf{x} \otimes \mathbf{s}) = (\mathbf{I}_t \otimes \mathbf{s}^\intercal)\mathbf{B}\mathbf{x}$. Since rearranging the entries does not change the infinity norm, the claim follows. □

We also recall the definition of RMS program. Essentially this consists of an algebraic circuit over $\mathbb{Z}$ where we can multiply two wires only if at least one of them is an input.

Definition 2.2 (Restricted Multiplication Straightline)**.** *A restricted multiplication straightline program (RMS) consists of a polynomial-sized family of algebraic circuits over $\mathbb{Z}$ where the only allowed gates are the following:*

- *Additions: this gate takes as input two wires x and y and outputs their sum $x + y$.*
- *Scalar multiplication: each of these gates is parametrised by a constant $\alpha \in \mathbb{Z}$. It takes as input a single wire x and outputs $\alpha \cdot x$.*
- *Multiplication: this gate takes as input two wires x and y* where y is an input to the RMS program. *The output is their product $x \cdot y$.*

We say that the program has depth d if the multiplicative depth *of the program is d.*

Let T be a positive integer. We say that an RMS program is T-bounded if, during any evaluation over binary *inputs, the absolute value of the wires never exceeds T.*

We recall the following result which states that any circuit in NC_1 can be converted into a polynomial size RMS program (paying exponentially in the depth).

Theorem 2.3 ([Bar86, BGI16]). *Let $f : \{0,1\}^n \to \{0,1\}$ be described by a boolean circuit of size s and depth d made entirely of NAND gates. Then, f can be computed using a 1-bounded RMS program of depth at most 2^d and size $O(s \cdot 2^d)$.*

2.1 Lattices and Learning with Errors

Throughout the paper, we often rely on a low-norm distribution $\chi(1^\lambda)$ over $\mathbb{Z}$. We say that $\chi(1^\lambda)$ is $B(\lambda)$-bounded if:

$$\Pr\left[|e| \le B(\lambda) \middle| e \xleftarrow{\$} \chi(1^\lambda)\right] = 1.$$

Sometimes, we abuse notation and we write $\mathbf{v} \xleftarrow{\$} \chi(1^\lambda)$, even if $\mathbf{v}$ is a vector: with this, we mean that each entry of $\mathbf{v}$ is sampled from $\chi(1^\lambda)$ independently of all the others.

We recall the learning with error assumption, introduced for the first time by Regev [Reg05].

Definition 2.4 (Learning with Errors). *Let $k := k(\lambda)$, $m := m(\lambda)$ and $q := q(\lambda)$ be positive integers. Let $\chi(1^\lambda)$ be a low-norm distribution over $\mathbb{Z}$. We say that the (χ, k, m, q)-LWE problem is hard if, for every PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that, for every $\lambda \in \mathbb{N}$, we have*

$$\left| \Pr\left[\mathcal{A}(1^\lambda, \mathbf{M}, \mathbf{u}) = 1 \middle| \begin{array}{l} \mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{m\times k}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k \\ \mathbf{e} \xleftarrow{\$} \chi^m(1^\lambda) \\ \mathbf{u} \leftarrow \mathbf{M} \cdot \mathbf{s} + \mathbf{e} \end{array}\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathbf{M}, \mathbf{u}) = 1 \middle| \begin{array}{l} \mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{m\times k} \\ \mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m \end{array}\right] \right| \le \mathsf{negl}(\lambda).$$

Suppose that $\chi(1^\lambda)$ is $B(\lambda)$-bounded. We call the quantity $\alpha := q/B$ the modulus-noise ratio.

We also define the *non-uniform LWE* assumption identically as above, except that the matrix $\mathbf{M}$ is no longer uniformly random over $\mathbb{Z}_q^{m\times k}$ but over $\mathbb{Z}_2^{m\times k}$. It is shown in [BLMR13] that non-uniform LWE is at least as hard as LWE, with a slightly increased parameter size.

Theorem 2.5 ([BLMR13]). *Assume the hardness of (χ, k, m, q)-LWE. Then, $(\chi, k \cdot \log q, m, q)$-NLWE is hard.*

2.2 Laconic Function Evaluation

Definition 2.6 (Laconic Function Evaluation [QWW18]). *Let $m := m(\lambda)$ and $\ell := \ell(\lambda)$ be positive integers. Let $\mathcal{F} = (\mathcal{F}_\lambda)_{\lambda\in\mathbb{N}}$ be a function class containing functions $f : \{0,1\}^m \to \{0,1\}^\ell$. A laconic function evaluation scheme (LFE) for $\mathcal{F}$ 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.

$\mathsf{Hash}(\mathsf{pk}, f)$: *The hashing algorithm is probabilistic, it takes as input a public key and the description of a function $f \in \mathcal{F}_\lambda$. The output is a digest h and hasher's private information ψ.*

$\mathsf{Enc}(\mathsf{pk}, h, x)$: *The encoding algorithm is probabilistic, it takes as input a public key* pk*, a digest h and an input $x \in \{0,1\}^m$. The output is an encoding E.*

$\mathsf{Dec}(\mathsf{pk}, E, f, \psi)$: *The decoding procedure is deterministic, it takes as input a public key* pk*, an encoding* E*, a function* $f \in \mathcal{F}_\lambda$ *and hasher's private information* ψ*. The output is a value* $y \in \{0,1\}^\lambda$.

Definition 2.7 (Correctness of LFE [QWW18])**.** *Let* $m := m(\lambda)$ *and* $\ell := \ell(\lambda)$ *be positive integers. Let* $\mathcal{F} = (\mathcal{F}_\lambda)_{\lambda \in \mathbb{N}}$ *be a function class containing functions* $f : \{0,1\}^m \to \{0,1\}^\ell$*. A LFE scheme for* $\mathcal{F}$ $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{Dec})$ *is correct if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every sufficiently large* λ*, every* $f \in \mathcal{F}_\lambda$ *and every* $x \in \{0,1\}^m$*, it holds that:*

$$\Pr\left[\mathsf{Dec}(\mathsf{pk}, E, f, \psi) \neq f(x)\right] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random choice of $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*,* $(h, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{pk}, f)$ *and* $E \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, x)$*.*

Definition 2.8 (Adaptive Encoder Privacy [QWW18])**.** *Consider the following experiment* $\mathsf{AEncExp}_{\mathcal{A},\mathsf{Sim}}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *and a simulator* Sim*:*

- *Sample a public key* $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*.*
- *Activate the adversary* $(x, f, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda, \mathsf{pk})$*.*
- *Compute* $(h, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{pk}, f; r)$ *where* r *denotes freshly sampled randomness.*
- *Sample a random bit* $b \xleftarrow{\$} \{0,1\}$*.*
- *If* $b = 0$ *compute* $(E_0, \phi_0) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, h, x)$*, else compute* $E_1 \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{pk}, f, r, f(x))$*.*
- *Compute* $b' \leftarrow \mathcal{A}_1(E_b, r, \mathsf{aux})$*.*
- *Return* 1 *if and only if* $b = b'$*.*

We say that an LFE scheme $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{Dec})$ *is adaptively encoder-private if there exists a PPT simulator* Sim*, such that for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left| \frac{1}{2} - \Pr\left[\mathsf{AEncExp}_{\mathcal{A},\mathsf{Sim}}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

2.3 Fully Homomorphic Encryption à la GSW

We recall the fully homomorphic encryption scheme of [GSW13], which will be used in our rate-1 LFE construction.

- Keys: the secret key and the public key are generated as follows

$$\mathsf{sk} := \begin{pmatrix} \mathbf{r} \\ -1 \end{pmatrix} \qquad \mathsf{pk} := \begin{pmatrix} \mathbf{M} \\ \mathbf{r}^\intercal \cdot \mathbf{M} + \mathbf{e}^\intercal \end{pmatrix}$$

  where $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^{k-1}$, $\mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{(k-1)\times(k\cdot\log q+\lambda)}$ and $\mathbf{e} \xleftarrow{\$} \chi(1^\lambda)$. Notice that $\mathsf{sk}^\intercal \cdot \mathsf{pk} = -\mathbf{e}^\intercal$. Moreover, under LWE, the public key is indistinguishable from a random matrix in $\mathbb{Z}_q^{k\times(k\cdot\log q+\lambda)}$.

- Encryption: to encrypt a vector $\mathbf{x} \in \mathbb{Z}_2^m$, sample $\mathbf{R} \xleftarrow{\$} \mathbb{Z}_2^{(k \cdot \log q + \lambda) \times (m \cdot k \cdot \log q)}$ and output

  $$\mathsf{ct} \leftarrow \mathsf{pk} \cdot \mathbf{R} + \mathbf{x}^\intercal \otimes \mathbf{G}$$

  where $\mathbf{G} := \mathbf{I}_k \otimes \mathbf{g}_q$. We call $\|\mathbf{R}\|_\infty$ *the noise magnitude* of the ciphertext. Notice that in any freshly generated ciphertext, this is equal to 1. Observe that if we substitute pk with a random matrix in $\mathbb{Z}_q^{k \times (k \cdot \log q + \lambda)}$, then the ciphertext ct is statistically indistinguishable from random by the Leftover Hash Lemma [ILL89].

- Decryption: suppose we have a ciphertext $\mathsf{ct} = \mathsf{pk} \cdot \mathbf{R} + \mathbf{y}^\intercal \otimes \mathbf{G}$ where the noise magnitude is $\|\mathbf{R}\|_\infty = \alpha$ and $\mathbf{y} \in \{0,1\}^\ell$. Let $\mathbf{u}$ be the last vector of the standard basis of $\mathbb{Z}_q^k$. To recover $\mathbf{y}$, we compute

  $$\begin{aligned}\mathbf{y}' :=& \mathsf{sk}^\intercal \cdot \mathsf{ct} \cdot \left(\mathbf{I}_\ell \otimes \mathbf{G}^{-1}\left(-\frac{q}{2} \cdot \mathbf{u}\right)\right) \\ =& \mathsf{sk}^\intercal \cdot \mathsf{pk} \cdot \mathbf{R} \cdot \left(\mathbf{I}_\ell \otimes \mathbf{G}^{-1}\left(-\frac{q}{2} \cdot \mathbf{u}\right)\right) + \mathsf{sk}^\intercal \cdot (\mathbf{y}^\intercal \otimes \mathbf{G}) \cdot \left(\mathbf{I}_\ell \otimes \mathbf{G}^{-1}\left(-\frac{q}{2} \cdot \mathbf{u}\right)\right) \\ =& -\mathbf{e}^\intercal \cdot \mathbf{R} \cdot \left(\mathbf{I}_\ell \otimes \mathbf{G}^{-1}\left(-\frac{q}{2} \cdot \mathbf{u}\right)\right) - \frac{q}{2} \cdot \mathsf{sk}^\intercal \cdot (\mathbf{y}^\intercal \otimes \mathbf{u}) \\ =& -\mathbf{e}^\intercal \cdot \mathbf{R} \cdot \left(\mathbf{I}_\ell \otimes \mathbf{G}^{-1}\left(-\frac{q}{2} \cdot \mathbf{u}\right)\right) - \frac{q}{2} \cdot \mathbf{y}^\intercal \otimes (\mathsf{sk}^\intercal \cdot \mathbf{u}) \\ =& \frac{q}{2} \cdot \mathbf{y}^\intercal - \mathbf{e}^\intercal \cdot \mathbf{R} \cdot \left(\mathbf{I}_\ell \otimes \mathbf{G}^{-1}\left(-\frac{q}{2} \cdot \mathbf{u}\right)\right).\end{aligned}$$

  Notice that $\|\mathbf{e}^\intercal \cdot \mathbf{R} \cdot \left(\mathbf{I}_\ell \otimes \mathbf{G}^{-1}\left(-\frac{q}{2} \cdot \mathbf{u}\right)\right)\|_\infty \leq k \cdot \alpha \cdot B$. So, if q is sufficiently big (in particular, bigger than $4k \cdot \alpha \cdot B$, we are able to recover $\mathbf{y}$ by computing the most significant bit of the entries of $\mathbf{y}'$.

- Homomorphic evaluation: there exists a deterministic polynomial-time algorithm $\mathsf{GSW.Eval}$ that, on input a function $f : \{0,1\}^m \to \{0,1\}^\ell$ and an encryption of a string $\mathbf{x} \in \{0,1\}^m$, produces an encryption of $f(\mathbf{x})$. The operation, however, increases the magnitude of the noise proportionally to the number of operations required by f. In particular, there exists a limit after which the magnitude of the noise is so high that decryption fails.

  There exist multiple ways in which $\mathsf{GSW.Eval}$ can be instantiated and not all of them are equivalent in the way they manage to keep the noise magnitude low. In this paper, we use the approach of Brakerski and Vainkuntanathan [BV14] which makes the noise magnitude grow linearly in the depth of the evaluated circuit.

3 Non-Interactive Oblivious Tensor Evaluation

In the following we define and construct a succinct Non-Interactive Oblivious Tensor Evaluation (NI-OTE) protocol.

3.1 Definitions

We begin by defining the syntax of NI-OTE.

Definition 3.1 (Non-interactive oblivious tensor evaluation)**.** *Let $m := m(\lambda)$, $\ell := \ell(\lambda)$, and $q := q(\lambda)$ be a positive integer. A NI-OTE for $\mathbb{Z}_q^m \otimes \mathbb{Z}_q^\ell$ consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{HashEval}, \mathsf{EncEval})$ *with the following syntax:*

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

$\mathsf{Hash}(\mathsf{pk}, \mathbf{x})$: *The hashing algorithm is probabilistic and takes as input a public key* pk *and the description of a vector* $\mathbf{x} \in \mathbb{Z}_q^m$. *The output is a digest* d *and hasher's private information* ψ.

$\mathsf{Enc}(\mathsf{pk}, \mathbf{y})$: *The encoding algorithm is probabilistic and takes as input a public key* pk, *and a vector* $\mathbf{y} \in \mathbb{Z}_q^\ell$. *The output is an encoding* E *and encoder's private information* ϕ.

$\mathsf{HashEval}(\mathsf{pk}, E, \psi)$: *The hasher's evaluation algorithm is deterministic and takes as input a public key* pk, *an encoding* E *and hasher's private information* ψ. *The output is a vector* $\mathbf{v} \in \mathbb{Z}_q^{m \cdot \ell}$.

$\mathsf{EncEval}(\mathsf{pk}, d, \phi)$: *The encoder's evaluation algorithm is deterministic and takes as input a public key* pk, *a digest* d *and encoder's private information* ϕ. *The output is a vector* $\mathbf{w} \in \mathbb{Z}_q^{m \cdot \ell}$.

Sometimes it will be convenient for us to fix the private information ϕ and provide it as an input to the encoding algorithm. In a slight abuse of notation, we denote this by $\mathsf{Enc}(\mathsf{pk}, \mathbf{y}, [\phi])$, in which case, the algorithm just outputs E. If the NI-OTE scheme satisfies this syntactical requirement, which in particular means that ϕ does not depend on $\mathbf{y}$, we say that the scheme is *programmable.*

Additionally, we say that an NI-OTE has a *bilinear encoder evaluation* if:

- The digest d is a vector in $\mathbb{Z}_q^n$.
- The encoder secret information ϕ is a vector in $\mathbb{Z}_q^k$.
- The encoder evaluation algorithm consists of

$$\mathsf{EncEval}(\mathsf{pk}, d, \phi) := \mathbf{P} \cdot (d \otimes \phi \otimes \mathbf{g}_q^\intercal)$$

  where the matrix $\mathbf{P} \in \mathbb{Z}_q^{(m \cdot \ell) \times (n \cdot k \cdot \log q)}$ can be publicly derived from pk.

We say that an NI-OTE is *succinct* if the size of the hash and the size of the encodings are sublinear in the size of the hasher's input. Depending on the context, we will make the dependence explicit. If only the hash is sublinear, then we say that the scheme is *half-succinct.*

Correctness. Next, we define (approximate) correctness for an NI-OTE, parametrized by an error function α. If $\alpha = 0$, then we say that the NI-OTE is perfectly correct, or simply correct.

Definition 3.2 (α-Correctness). *An NI-OTE scheme* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{HashEval}, \mathsf{EncEval})$ *is* α-*correct if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every sufficiently large* $\lambda \in \mathbb{N}$, *for every* $\mathbf{x} \in \mathbb{Z}_q^m$ *and* $\mathbf{y} \in \mathbb{Z}_q^\ell$, *we have that:*

$$\Pr\left[\|\mathsf{HashEval}(\mathsf{pk}, E, \psi) + \mathsf{EncEval}(\mathsf{pk}, d, \phi) - \mathbf{x} \otimes \mathbf{y}\|_\infty > \alpha \cdot \|\mathbf{x}\|_\infty \;\right] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random choice of $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $(d, \psi) \xleftarrow{\$} \mathsf{Hash}(\mathsf{pk}, \mathbf{x})$, *and* $(E, \phi) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathbf{y})$.

The definition of approximate correctness will be sufficient for our applications. Nevertheless, for future reference, we mention here that there is a general (standard) method to drive the correctness error down to $\alpha = 0$, which may be more convenient to work with. Specifically, let p to be a divisor of q with $p/q \in \lambda^{-\omega(\lambda)}$. Instead of encoding $\mathbf{y}$, one encodes $q/p \cdot \mathbf{y}$. Then, we return $\mathsf{HashEval}(\mathsf{pk}, E, \psi)$

and $\mathsf{EncEval}(\mathsf{pk}, d, \phi)$ rounded to the nearest multiple of q/p. By the α-correctness guarantee of the protocol, it holds that:

$$\mathsf{HashEval}(\mathsf{pk}, E, \psi) + \mathsf{EncEval}(\mathsf{pk}, d, \phi) = q/p \cdot \mathbf{y} \otimes \mathbf{x} \pm \alpha \cdot \|\mathbf{x}\|_\infty.$$

Rounding to the nearest multiple of q/p returns:

$$\left\lceil q/p \cdot \mathbf{y} \otimes \mathbf{x} \pm \alpha \cdot \|\mathbf{x}\|_\infty \right\rfloor_p = \lceil q/p \cdot \mathbf{y} \otimes \mathbf{x} \rfloor_p = \mathbf{y} \otimes \mathbf{x} \pmod{p}$$

with high probability.

Encoder Privacy. We define encoder privacy, which guarantees that the encoding is simulatable, without knowing the underlying vector. This is formalized as follows.

Definition 3.3 (Encoder Privacy). *Consider the following experiment* $\mathsf{EncExp}_{\mathcal{A},\mathsf{Sim}}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *and a simulator* Sim*:*

- *Sample a public key* $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$.
- *Activate the adversary* $(\mathbf{y}, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda, \mathsf{pk})$.
- *Sample a random bit* $b \xleftarrow{\$} \{0, 1\}$.
- *If* $b = 0$ *compute* $(E_0, \phi_0) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathbf{y})$*, else compute* $E_1 \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{pk})$.
- *Compute* $b' \leftarrow \mathcal{A}_1(E_b, \mathsf{aux})$.
- *Return* 1 *if and only if* $b = b'$.

We say that a non-interactive OTE scheme $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{HashEval}, \mathsf{EncEval})$ *is encoder-private if there exists a PPT simulator* Sim*, such that for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left| \frac{1}{2} - \Pr\left[\mathsf{EncExp}_{\mathcal{A},\mathsf{Sim}}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

3.2 Half-Succinct NI-OTE from LWE

We present our first NI-OTE protocol that only satisfies a weak form of succinctness, namely that only the size of the digest is sublinear in the size of the hasher's input. Let $n := k \cdot \log q$, where $k = \Theta(\lambda)$, and suppose that for convenience that q is a power of 2. Our protocol is described in Construction 3.4.

Construction 3.4. Half-Succinct NI-OTE

$\mathsf{Setup}(1^\lambda)$**:** Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_2^{n \times m}$ and $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_2^{n \times (m \cdot \ell \cdot n)}$, then return $\mathsf{pk} := (\mathbf{A}, \mathbf{B})$.

$\mathsf{Hash}(\mathsf{pk}, \mathbf{x})$**:** Compute $\mathbf{d} := \mathbf{A} \cdot \mathbf{x}$ and return $d := \mathbf{d}$ and $\psi := \mathbf{x}$.

$\mathsf{Enc}(\mathsf{pk}, \mathbf{y})$**:** Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^k$, $\hat{\mathbf{E}} \xleftarrow{\$} \chi(\lambda)$, and $\mathbf{E}' \xleftarrow{\$} \chi(\lambda)$. Compute

$$\mathbf{C} := \mathbf{A}^\intercal \cdot \mathbf{B} \cdot (\mathbf{I}_{m \cdot \ell} \otimes \mathbf{s} \otimes \mathbf{g}_q^\intercal) + \mathbf{A}^\intercal \cdot \mathbf{E}' + \hat{\mathbf{E}} + \mathbf{I}_m \otimes \mathbf{y}^\intercal.$$

Return $E := \mathbf{C}$ and $\phi := \mathbf{s}$.

$\mathsf{HashEval}(\mathsf{pk}, E, \psi)$**:** Return $\mathbf{v} := \mathbf{C}^\intercal \cdot \mathbf{x}$.

$\mathsf{EncEval}(\mathsf{pk}, d, \phi)$**:** Return $\mathbf{w} := -(\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}^\intercal \otimes \mathbf{g}_q) \cdot \mathbf{B}^\intercal \cdot \mathbf{d}$.

The scheme is trivially programmable, since ϕ consists of a uniformly sampled vector $\mathbf{s}$ that in particular is independent from $\mathbf{y}$. Then, we observe that the scheme has indeed a bilinear encoding evaluation algorithm. By Lemma 2.1, it holds that:

$$-(\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s}^\intercal \otimes \mathbf{g}_q) \cdot \mathbf{B}^\intercal \cdot \mathbf{d} = \mathsf{Lin}(-\mathbf{B}^\intercal)(\mathbf{d} \otimes \mathbf{s} \otimes \mathbf{g}_q^\intercal) \tag{1}$$

with $\|\mathsf{Lin}(-\mathbf{B}^\intercal)\|_\infty = \|-\mathbf{B}^\intercal\|_\infty \leq 1$. We can also bound the norm of the digest by

$$\|\mathbf{d}\|_\infty = \|\mathbf{A} \cdot \mathbf{x}\|_\infty \leq m \cdot \|\mathbf{A}\|_\infty \cdot \|\mathbf{x}\|_\infty \leq m \cdot \|\mathbf{x}\|_\infty \tag{2}$$

with a triangle inequality. Finally, it is easy to see that the scheme is half succinct, since the hash $\mathbf{d}$ is n-dimensional vector over $\mathbb{Z}_q$, whose size is in particular independent of the length of $\mathbf{x}$ and $\mathbf{y}$. On the other hand, the encoding consists of an $m \times (m \cdot \ell)$ matrix over $\mathbb{Z}_q$.

Next, we argue that the scheme satisfies approximate correctness. Indeed, let us rewrite

$$\mathbf{C} = \mathbf{A}^\intercal \cdot \mathbf{Z} + \widetilde{\mathbf{E}} + \mathbf{I}_m \otimes \mathbf{y}^\intercal,$$

where $\mathbf{Z} := \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s} \otimes \mathbf{g}_q^\intercal)$ and $\widetilde{\mathbf{E}} := \mathbf{A}^\intercal \cdot \mathbf{E}' + \hat{\mathbf{E}}$. Substituting, we obtain

$$\begin{aligned}
\mathbf{v} + \mathbf{w} &= \mathbf{C}^\intercal \cdot \mathbf{x} - \mathbf{Z}^\intercal \cdot \mathbf{d} \\
&= (\mathbf{Z}^\intercal \cdot \mathbf{A}) \cdot \mathbf{x} + \widetilde{\mathbf{E}}^\intercal \cdot \mathbf{x} + (\mathbf{I}_m \otimes \mathbf{y}) \cdot \mathbf{x} - \mathbf{Z}^\intercal \cdot \mathbf{d} \\
&= \mathbf{Z}^\intercal \cdot \mathbf{d} + \widetilde{\mathbf{E}}^\intercal \cdot \mathbf{x} + \mathbf{x} \otimes \mathbf{y} - \mathbf{Z}^\intercal \cdot \mathbf{d} \\
&= \mathbf{x} \otimes \mathbf{y} + \widetilde{\mathbf{E}}^\intercal \cdot \mathbf{x}.
\end{aligned}$$

Since $\mathbf{A}$ is a matrix in $\mathbb{Z}_2$ with n rows, the entries of $\widetilde{\mathbf{E}} = \mathbf{A}^\intercal \cdot \mathbf{E}' + \hat{\mathbf{E}}$ is obtained by adding at most $n+1$ entries of the vectors $\mathbf{E}'$ and $\hat{\mathbf{E}}$, which are $B(\lambda)$-bounded. In other words,

$$\left\|\mathbf{A}^\intercal \cdot \mathbf{E}' + \hat{\mathbf{E}}\right\|_\infty \leq (n+1) \cdot B(\lambda).$$

Therefore we can bound the correctness error by

$$\left\|\widetilde{\mathbf{E}}^\intercal \cdot \mathbf{x}\right\|_\infty \leq m \cdot (n+1) \cdot B(\lambda) \cdot \|\mathbf{x}\|_\infty. \tag{3}$$

with a triangle inequality. Finally, we show that the scheme satisfies encoder privacy.

Theorem 3.5 (Encoder Privacy)**.** *Assuming the hardness of LWE, Construction 3.4 satisfies encoder privacy.*

Proof. Consider the following sequence of hybrids.

- Hybrid $\mathcal{H}_0$: This is the original distribution.
- Hybrid $\mathcal{H}_1$: We define
$$\mathbf{C} := \mathbf{A}^\intercal \cdot \mathbf{U} + \hat{\mathbf{E}} + (\mathbf{I}_m \otimes \mathbf{y}^\intercal)$$
where $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n\times(m\cdot\ell)}$.

We claim that this hybrid is indistinguishable from the previous once, by the LWE assumption. To see why, observe that the encoding provided to the adversary in the previous hybrid equals:

$$\mathbf{C} = \mathbf{A}^\intercal \cdot \mathbf{U} + \hat{\mathbf{E}} + (\mathbf{I}_m \otimes \mathbf{y}^\intercal),$$

where $\mathbf{U} := \mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell} \otimes \mathbf{s} \otimes \mathbf{g}_q^\intercal) + \mathbf{E}'$ for $\hat{\mathbf{E}} \xleftarrow{\$} \chi(\lambda)$ and $\mathbf{E}' \xleftarrow{\$} \chi(\lambda)$. This last term consists of an LWE sample with secret $\mathbf{s}$ and public matrix obtained by splitting $\mathbf{B} \cdot (\mathbf{I}_{m\cdot\ell\cdot k} \otimes \mathbf{g}_q^\intercal)$ in $m \cdot \ell$ blocks in $\mathbb{Z}_2^{n\times k}$ and stacking them one underneath the other.

In more details, consider a reduction that receives a matrix $\mathbf{M} \in \mathbb{Z}_q^{(n\cdot m\cdot\ell)\times k}$ and a vector $\mathbf{u} \in \mathbb{Z}_q^{n\cdot m\cdot\ell}$ that is either an LWE sample with respect to $\mathbf{M}$ or a uniformly random vector. The reduction splits the rows of $\mathbf{M}$ into $\ell \cdot m$ blocks in $\mathbb{Z}_q^{n\times k}$, denoted by

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_0 \\ \vdots \\ \mathbf{M}_{m\cdot\ell-1} \end{pmatrix}.$$

For every $j \in [m \cdot \ell]$, set $\mathbf{B}_j := \left(\mathbf{G}^{-1}(\mathbf{M}_j^\intercal)\right)^\intercal$. Notice that $\mathbf{B}_j$ is a random matrix over $\mathbb{Z}_2^{n\times n}$, since $\mathbf{M}_j$ is uniformly sampled and q is a power of 2. Finally, construct $\mathbf{B}$ as

$$\mathbf{B} = (\mathbf{B}_0, \ldots, \mathbf{B}_{m\cdot\ell-1})$$

and generate the columns in $\mathbf{U}$ by splitting $\mathbf{u}$ into $m \cdot \ell$ vectors in $\mathbb{Z}_q^n$. Then set $\mathbf{C}$ to $\mathbf{A}^\intercal \cdot \mathbf{U} + \hat{\mathbf{E}} + (\mathbf{I}_m \otimes \mathbf{y}^\intercal)$.

- Hybrid $\mathcal{H}_2$: We sample $\mathbf{C} \xleftarrow{\$} \mathbb{Z}_q^{(m)\times(m\cdot\ell)}$.

  Indistinguishability from the previous hybrid follows by another reduction to the LWE problem. Indeed, notice that $\mathbf{A}^\intercal \cdot \mathbf{U} + \hat{\mathbf{E}}$ is a batch of $m \cdot \ell$ LWE samples with $\mathbf{A}^\intercal$ as public matrix. The LWE secrets consist of the columns of $\mathbf{U}$.

The proof is concluded by defining the simulator Sim to output a uniformly random $\mathbf{C} \xleftarrow{\$} \mathbb{Z}_q^{(m)\times(m\cdot\ell)}$ as the encoding. □

3.3 Bootstrapping to Fully-Succinct NI-OTE

We now show a bootstrapping procedure to turn the NI-OTE constructed in Section 3.2 into a fully succinct NI-OTE for $\mathbb{Z}_q^m \otimes \mathbb{Z}_q^\ell$. We assume that $m := t^r \cdot n$, for some $t, r \in \mathbb{Z}$, and we assume the existence of a half-succinct α-correct $\mathsf{hOTE} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{HashEval}, \mathsf{EncEval})$ for $\mathbb{Z}_q^{t\cdot n} \otimes \mathbb{Z}_q^n$ with bilinear encoder evaluation, where $d \in \mathbb{Z}_q^n$ and $\phi \in \mathbb{Z}_q^k$, with $n := k \cdot \log q$. We describe our scheme in Construction 3.6.

Construction 3.6. Fully Succinct NI-OTE

$\mathsf{Setup}(1^\lambda)$**:** Return $\mathsf{pk} \xleftarrow{\$} \mathsf{hOTE.Setup}(1^\lambda)$.

$\mathsf{Hash}(\mathsf{pk}, \mathbf{x})$**:** Set $\mathbf{x}_0 := \mathbf{x}$. Then for every $i \in [r]$ proceed as follows:

- Parse $\mathbf{x}_i$ as the vertical concatenation of $(\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,t^{r-i-1}})$ where $\mathbf{x}_{i,j} \in \mathbb{Z}_q^{t\cdot n}$.
- For every $j \in [t^{r-i-1}]$, compute

$$(\mathbf{d}_{i,j}, \psi_{i,j}) \xleftarrow{\$} \mathsf{hOTE.Hash}(\mathsf{pk}, \mathbf{x}_{i,j}).$$

- Define $\mathbf{x}_{i+1}$ to be the vertical concatenation of the $\mathbf{d}_{i,j}$.

Return $d := \mathbf{x}_r$ and $\psi := \{\psi_{i,j}\}_{i\in[r],j\in[t^{r-i-1}]}$.

$\mathsf{Enc}(\mathsf{pk}, \mathbf{y})$**:** Set $\mathbf{y}_0 := \mathbf{y}$, then for $i \in [r]$, compute

$$(E_i, \phi_{i+1}) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathbf{y}_i) \quad \text{and} \quad \mathbf{y}_{i+1} := \phi_{i+1} \otimes \mathbf{g}_q^{\intercal}$$

Output $E := \{E_i\}_{i\in[r]}$ and $\phi := \phi_r$.

$\mathsf{HashEval}(\mathsf{pk}, E, \psi)$**:** Let $\mathbf{P}$ be the public matrix derived from pk. For all $i \in [r]$ and $j \in [t^{r-i-1}]$, compute

$$\mathbf{P}_i := (\mathbf{I}_{t^{r-i}} \otimes \mathbf{P}) \quad \text{and} \quad \mathbf{v}_{i,j} := \mathsf{hOTE.HashEval}(\mathsf{pk}, E_i, \psi_{i,j}).$$

Let $\mathbf{v}_i$ be the vertical concatenation of $\{\mathbf{v}_{i,j}\}_{j\in[t^{r-i-1}]}$. Return $\mathbf{v} := \sum_{i\in[r]} \left(\prod_{k=1}^{i} \mathbf{P}_k\right) \cdot \mathbf{v}_i$.

$\mathsf{EncEval}(\mathsf{pk}, d, \phi)$**:** Let $\mathbf{P}_i$ be defined as above. Return

$$\mathbf{w} := \left(\prod_{i=1}^{r-1} \mathbf{P}_i\right) \cdot \mathsf{hOTE.EncEval}(\mathsf{pk}, d, \phi_r).$$

The scheme is clearly programmable, if the underlying hOTE is. Furthermore, note that the scheme maintains a bilinear encoder evaluation, since, by Equation (1) we have that:

$$\mathsf{EncEval}(\mathsf{pk}, d, \phi) = \left(\prod_{i=1}^{r-1} \mathbf{P}_i\right) \cdot \mathbf{P} \cdot (d \otimes \phi \otimes \mathbf{g}_q^{\intercal}). \tag{4}$$

To bound the norm of the public matrix, recall that $\mathbf{P} \in \mathbb{Z}_q^{n^2 \times (t \cdot n^2)}$ and $\|\mathbf{P}\|_\infty \leq 1$. So, for every $i \in [r]$, each row of $\mathbf{P}_i = (\mathbf{I}_{t^{r-i}} \otimes \mathbf{P})$ has at most n^2 non-zero entries. Thus, multiplying by $\mathbf{P}_i$ increases the infinity norm at most by a factor n^2. Thus we have that:

$$\left\| \prod_{i=1}^{r-1} \mathbf{P}_i \cdot \mathbf{P} \right\|_\infty \leq n^{2r}. \tag{5}$$

On the other hand, we can bound the norm of the digest by recalling that $\mathbf{x}_{i,j} \in \mathbb{Z}_q^{t\cdot n}$ and by Equation (2) the norm of the digest is increased by a factor $t \cdot n$ each time it is hashed. Thus:

$$\|d\|_\infty = \|\mathbf{x}_r\|_\infty \leq (tn)^r \cdot \|\mathbf{x}\|_\infty. \tag{6}$$

Furthermore, the dimension of the digest is n, and therefore independent of the dimensions of the input vector, and its bit-length is at most $n \cdot (r \cdot \log tn + \log\|\mathbf{x}\|_\infty)$. The size of the encoding is $r = O(\log m)$ times the size of an encoding of hOTE (whose size is independent of m). Thus, the scheme is fully succinct.

Approximate Correctness. Towards proving approximate correctness, we begin by observing that, by the α-correctness of hOTE, for every $i \in [r]$ and $j \in [t^{r-i-1}]$, we have:

$$\mathsf{hOTE.EncEval}(\mathsf{pk}, \mathbf{d}_{i,j}, \phi_{i+1}) + \mathsf{hOTE.HashEval}(\mathsf{pk}, E_i, \psi_{i,j}) = \mathbf{x}_{i,j} \otimes \mathbf{y}_i + \mathbf{e}_{i,j}, \tag{7}$$

where $\|\mathbf{e}_{i,j}\|_\infty \leq \alpha \cdot \|\mathbf{x}_{i,j}\|_\infty \leq \alpha \cdot \|\mathbf{x}_i\|_\infty = tn(n+1)B(\lambda) \cdot \|\mathbf{x}_i\|_\infty$, by Equation (3). To establish correctness, it suffices to prove the following.

Lemma 3.7 (Approximate Correctness)**.** *For every $i \in [r]$:*

$$\sum_{j=i}^{r-1} \left(\prod_{k=1}^{j} \mathbf{P}_k \right) \cdot \mathbf{v}_j + \mathbf{w} = \left(\prod_{k=1}^{i} \mathbf{P}_k \right) \cdot (\mathbf{x}_i \otimes \mathbf{y}_i) + \widetilde{\mathbf{e}}_i$$

where $\|\widetilde{\mathbf{e}}_i\|_\infty \leq \alpha \cdot \left(\sum_{k=i}^{r-1} (t \cdot n^3)^k \right) \cdot \|\mathbf{x}\|_\infty$*.*

Before proceeding with the proof, we can indeed see that, setting $i = 0$, we obtain:

$$\mathbf{v} + \mathbf{w} = (\mathbf{x} \otimes \mathbf{y}) + \mathbf{e}$$

where $\|\mathbf{e}\|_\infty \leq \alpha \cdot (t \cdot n^3)^r \cdot \|\mathbf{x}\|_\infty$. Thus, the scheme satisfies $\alpha(t \cdot n^3)^r$-correctness.

Proof. We proceed by induction on the index i, starting from $i := r - 1$. For the base case, we have We proceed by induction starting from $i = r - 1$ and going all the way down to $i = 0$. It is easy to see that $\mathbf{v}_{r-1} = \mathsf{hOTE.HashEval}(\mathsf{pk}, E_{r-1}, \psi_{r-1,0})$, whereas $d = \mathbf{d}_{r-1,0}$. Therefore, by Equation (7), we obtain:

$$\begin{aligned}
\left(\prod_{k=1}^{r-1} \mathbf{P}_k \right) \cdot \mathbf{v}_{r-1} + \mathbf{w} &= \left(\prod_{k=1}^{r-1} \mathbf{P}_k \right) \cdot \Big(\mathsf{hOTE.HashEval}(\mathsf{pk}, E_{r-1}, \psi_{r-1,0}) + \mathsf{hOTE.EncEval}(\mathsf{pk}, d, \phi_r) \Big) \\
&= \left(\prod_{k=1}^{r-1} \mathbf{P}_k \right) \cdot (\mathbf{x}_{r-1,0} \otimes \mathbf{y}_{r-1}) + \left(\prod_{k=1}^{r-1} \mathbf{P}_k \right) \cdot \mathbf{e}_{r-1,0} \\
&= \left(\prod_{k=1}^{r-1} \mathbf{P}_k \right) \cdot (\mathbf{x}_{r-1} \otimes \mathbf{y}_{\mathbf{r-1}}) + \widetilde{\mathbf{e}}_{r-1}
\end{aligned}$$

where $\widetilde{\mathbf{e}}_{r-1} := \left(\prod_{k=1}^{r-1} \mathbf{P}_k \right) \cdot \mathbf{e}_{r-1,0}$. Notice that

$$\|\widetilde{\mathbf{e}}_{r-1}\|_\infty \leq (n^2)^{r-1} \cdot \|\mathbf{e}_{r-1,0}\|_\infty \leq (n^2)^{r-1} \cdot \alpha \cdot \|\mathbf{x}_{r-1}\|_\infty \leq \alpha \cdot (t \cdot n^3)^{r-1} \cdot \|\mathbf{x}\|_\infty$$

by Equation (5) and Equation (6). Then, by induction hypothesis:

$$\begin{aligned}
\sum_{j=i-1}^{r-1} \left(\prod_{k=1}^{j} \mathbf{P}_k \right) \cdot \mathbf{v}_j + \mathbf{w} &= \left(\prod_{k=1}^{i-1} \mathbf{P}_k \right) \cdot \mathbf{v}_{i-1} + \left(\prod_{k=1}^{i} \mathbf{P}_k \right) \cdot (\mathbf{x}_i \otimes \mathbf{y}_i) + \widetilde{\mathbf{e}}_i \\
&= \left(\prod_{k=1}^{i-1} \mathbf{P}_k \right) \cdot \Big(\mathbf{v}_{i-1} + \mathbf{P}_i \cdot (\mathbf{x}_i \otimes \mathbf{y}_i) \Big) + \widetilde{\mathbf{e}}_i.
\end{aligned}$$

Moreover:

$$\mathbf{P}_i \cdot (\mathbf{x}_i \otimes \mathbf{y}_i) = \begin{pmatrix} \mathbf{P} \cdot (\mathbf{d}_{i-1,0} \otimes \phi_i \otimes \mathbf{g}_q^\intercal) \\ \mathbf{P} \cdot (\mathbf{d}_{i-1,1} \otimes \phi_i \otimes \mathbf{g}_q^\intercal) \\ \vdots \\ \mathbf{P} \cdot (\mathbf{d}_{i-1,t^{r-i}} \otimes \phi_i \otimes \mathbf{g}_q^\intercal) \end{pmatrix} = \begin{pmatrix} \mathsf{hOTE.EncEval}(\mathsf{pk}, \mathbf{d}_{i-1,0}, \phi_i) \\ \mathsf{hOTE.EncEval}(\mathsf{pk}, \mathbf{d}_{i-1,1}, \phi_i) \\ \vdots \\ \mathsf{hOTE.EncEval}(\mathsf{pk}, \mathbf{d}_{i-1,t^{r-i}}, \phi_i) \end{pmatrix}.$$

We also observe that:

$$\mathbf{v}_{i-1} = \begin{pmatrix} \mathbf{v}_{i-1,0} \\ \mathbf{v}_{i-1,1} \\ \vdots \\ \mathbf{v}_{i-1,t^{r-i}} \end{pmatrix} = \begin{pmatrix} \mathsf{hOTE.EncHash}(\mathsf{pk}, E_{i-1}, \psi_{i-1,0}) \\ \mathsf{hOTE.EncHash}(\mathsf{pk}, E_{i-1}, \psi_{i-1,1}) \\ \vdots \\ \mathsf{hOTE.EncHash}(\mathsf{pk}, E_{i-1}, \psi_{i-1,t^{r-i}}) \end{pmatrix}.$$

Furthermore, by Equation (7), we have:

$$\mathbf{v}_{i-1}+\mathbf{P}_i\cdot(\mathbf{x}_i\otimes\mathbf{y}_i)=\begin{pmatrix}(\mathbf{x}_{i-1,0}\otimes\mathbf{y}_{i-1})+\mathbf{e}_{i-1,0}\\(\mathbf{x}_{i-1,1}\otimes\mathbf{y}_{i-1})+\mathbf{e}_{i-1,1}\\\vdots\\(\mathbf{x}_{i-1,t^{r-i}}\otimes\mathbf{y}_{i-1})+\mathbf{e}_{i-1,t^{r-i}}\end{pmatrix}=(\mathbf{x}_{i-1}\otimes\mathbf{y}_{i-1})+\mathbf{e}_{i-1},\ \text{where } \mathbf{e}_{i-1}:=\begin{pmatrix}\mathbf{e}_{i-1,0}\\\mathbf{e}_{i-1,1}\\\vdots\\\mathbf{e}_{i-1,t^{r-i}}\end{pmatrix}.$$

Notice that $\|\mathbf{e}_{i-1}\|_\infty\leq\alpha\cdot\|\mathbf{x}_{i-1}\|_\infty\leq\alpha\cdot(tn)^{i-1}\cdot\|\mathbf{x}\|_\infty$ by Equation (5) and Equation (6). We conclude that

$$\sum_{j=i-1}^{r-1}\left(\prod_{k=1}^{j}\mathbf{P}_k\right)\cdot\mathbf{v}_j+\mathbf{w}=\left(\prod_{k=1}^{i-1}\mathbf{P}_k\right)\cdot(\mathbf{x}_{i-1}\otimes\mathbf{y}_{i-1})+\widetilde{\mathbf{e}}_{i-1}$$

where $\widetilde{\mathbf{e}}_{i-1}:=\widetilde{\mathbf{e}}_i+\left(\prod_{k=1}^{i-1}\mathbf{P}_k\right)\cdot\mathbf{e}_{i-1}$. Observe that

$$\|\widetilde{\mathbf{e}}_{i-1}\|_\infty\leq\|\widetilde{\mathbf{e}}_i\|_\infty+\alpha\cdot(t\cdot n^3)^{i-1}\cdot\|\mathbf{x}\|_\infty\leq\alpha\cdot\left(\sum_{k=i-1}^{r-1}(t\cdot n^3)^k\right)\cdot\|\mathbf{x}\|_\infty$$

as desired. □

Encoder Privacy. The following theorem establishes encoder privacy.

Theorem 3.8. *Assuming that* hOTE *is encoder private, Construction 3.6 is encoder private.*

Proof. The proof follows by a standard hybrid argument, where we gradually substitute the encodings $\{E_j\}_{j\in[r]}$ with the outputs of the simulator $\mathsf{Sim}(1^\lambda,\mathsf{pk})$, provided by the definition of encoder privacy of hOTE. □

How to Build a Fully Succinct OTE for $\mathbb{Z}_q^m\otimes\mathbb{Z}_q^\ell$. Construction 3.6 presents a non-interactive OTE for $\mathbb{Z}_q^m\otimes\mathbb{Z}_q^n$ where the hash size is a vector in $\mathbb{Z}_q^n$ and the encoding consists of r matrices of size $(tn)\times(tn^2)$. A trivial way to build a fully succinct OTE for $\mathbb{Z}_q^m\otimes\mathbb{Z}_q^\ell$ is to instantiate Construction 3.6 with $n:=\ell$. If $\ell=O(\lambda\cdot\log q)$, this is secure, however, the encoder size would scale as $r\cdot\ell^3$ while the hash size would be linear in ℓ. We can do better: just split $\mathbf{y}$ into blocks $\mathbf{y}_0,\dots,\mathbf{y}_N$ of size n. Then, compute an OTE encoding of each of them using Construction 3.6 and send them to the hasher. Given the hasher's digest (notice, a single digest is sufficient for all $\mathbf{y}_0,\dots,\mathbf{y}_N$), we can then derive shares for $\mathbf{x}\otimes\mathbf{y}_0,\dots,\mathbf{x}\otimes\mathbf{y}_N$. By reordering these, we obtain a secret-sharing of $\mathbf{x}\otimes\mathbf{y}$. In this way, the digest dimension is n, while the encoding dimension is $\ell\cdot r\cdot t^2\cdot n^2$.

3.4 From Succinct NI-OTE to Succinct NI-MOLE

We will also consider an modification of NI-OTE where instead of computing a tensor product, we compute a matrix-vector product. More specifically, we let the hasher take as input a matrix $\mathbf{M}$ and the encoder a vector $\mathbf{y}$ and we require that

$$\mathsf{HashEval}(\mathsf{pk},E,\psi)+\mathsf{EncEval}(\mathsf{pk},d,\phi)\approx\mathbf{My}.$$

Furthermore, we require the protocol to be succinct, in the sense that the communication complexity should be independent of the number of rows of the matrix $\mathbf{M}$. We refer to this protocol as Non-Interactive Matrix Oblivious Linear Evaluation (NI-MOLE).

It is easy to see that one can generically construct an $(\ell \cdot \alpha)$-correct NI-MOLE from an α-correct NI-OTE: Simply hash $\mathbf{x} := \mathbf{G}^{-1}(\mathsf{vec}(\mathbf{M}))$ and encode $\mathbf{y} \otimes \mathbf{g}_q$. Then the hasher and the encoder return

$$\mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot \mathsf{HashEval}(\mathsf{pk}, E, \psi) \quad \text{and} \quad \mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot \mathsf{EncEval}(\mathsf{pk}, d, \phi)$$

respectively. By the correctness of the NI-OTE we have:

$$\begin{aligned} \mathbf{v} + \mathbf{w} &= \mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot (\mathbf{x} \otimes \mathbf{y} \otimes \mathbf{g}_q^{\intercal}) + \mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot \mathbf{e} \\ &= (\mathbf{I}_m \otimes \mathbf{y}^{\intercal} \otimes \mathbf{g}_q) \cdot \mathbf{x} + \mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot \mathbf{e} \\ &= (\mathbf{I}_m \otimes \mathbf{y}^{\intercal}) \cdot \mathsf{vec}(\mathbf{M}^{\intercal}) + \mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot \mathbf{e} \\ &= \mathbf{M} \cdot \mathbf{y} + \mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot \mathbf{e} \\ &\approx \mathbf{M} \cdot \mathbf{y} \end{aligned}$$

ignoring low-order error terms, since $\|\mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q})\|_\infty = 1$ by Lemma 2.1. Notice that

$$\|\mathsf{Lin}(\mathbf{I}_{m \cdot \ell \cdot \log q}) \cdot \mathbf{e}\|_\infty \leq \ell \cdot \alpha \cdot \|\mathbf{x}\|_\infty = \ell \cdot \alpha.$$

Thus, henceforth we will assume the existence of succinct NI-OTE and succinct NI-MOLE interchangeably, with the understanding the succinct NI-OTE implies the existence of both.

4 Adaptive Lattice Encodings

In the following we present our new construction of adaptive lattice encodings. Let $k := k(\lambda)$ and $q := q(\lambda)$ be positive integers, and let $\mathbf{G}$ be the k-dimensional gadget matrix $\mathbf{G} := \mathbf{I}_k \otimes \mathbf{g}_q$. For element $x \in \mathbb{Z}_q$, vectors $\mathbf{s}, \mathbf{r} \in \mathbb{Z}_q^k$, matrix $\mathbf{A} \in \mathbb{Z}_q^{k \times (k \cdot \log q)}$ and noise term $\mathbf{e} \in \mathbb{Z}^{k \cdot \log q}$, we define the corresponding *adaptive lattice encoding* as:

$$\mathsf{LEnc}_{\mathbf{A}}(x; \mathbf{s}, \mathbf{r}, \mathbf{e}) := \mathbf{s}^{\intercal}\mathbf{A} + x \cdot \mathbf{r}^{\intercal}\mathbf{G} + \mathbf{e}^{\intercal}.$$

Note that, for an appropriately sampled $\mathbf{A}$, $\mathbf{s}$, and $\mathbf{e}$. It is straightforward to see that the encoding is computationally close to uniform under the LWE assumption. Next, we demonstrate the homomorphic properties of such encodings.

4.1 Homomorphic Operations

We show that our lattice encodings support addition, scalar multiplication, and even multiplication, provided that the encodings are encrypted with correlated secrets. We present a formal description of the algorithms below.

Construction 4.1. Homomorphic Operations on Lattice Encodings

Addition: For every $x_0, x_1 \in \mathbb{Z}_q$, vectors $\mathbf{s}, \mathbf{r} \in \mathbb{Z}_q^k$, matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{k \times (k \cdot \log q)}$ and noise terms $\mathbf{e}_0, \mathbf{e}_1 \in \mathbb{Z}_q^{k \cdot \log q}$, we have:

$$\mathsf{LEnc}_{\mathbf{A}_0}(x_0; \mathbf{s}, \mathbf{r}, \mathbf{e}_0) + \mathsf{LEnc}_{\mathbf{A}_1}(x_1; \mathbf{s}, \mathbf{r}, \mathbf{e}_1) = \mathsf{LEnc}_{\mathbf{A}_0 + \mathbf{A}_1}(x_0 + x_1; \mathbf{s}, \mathbf{r}, \mathbf{e}_0 + \mathbf{e}_1).$$

Scalar Multiplication: For every $x, \delta \in \mathbb{Z}_q$, vectors $\mathbf{s}, \mathbf{r} \in \mathbb{Z}_q^k$, matrix $\mathbf{A} \in \mathbb{Z}_q^{k \times (k \cdot \log q)}$ and noise terms $\mathbf{e} \in \mathbb{Z}_q^{k \cdot \log q}$, we have:

$$\mathsf{LEnc}_{\mathbf{A}}(x; \mathbf{s}, \mathbf{r}, \mathbf{e}) \cdot \mathbf{G}^{-1}(\delta \cdot \mathbf{G}) = \mathsf{LEnc}_{\mathbf{A} \cdot \mathbf{G}^{-1}(\delta \cdot \mathbf{G})}(x \cdot \delta; \mathbf{s}, \mathbf{r}, \mathbf{G}^{-1}(\delta \cdot \mathbf{G})^{\intercal} \cdot \mathbf{e}).$$

Multiplication: For every $x_0, x_1 \in \mathbb{Z}_q$, vectors $\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}_q^k$, matrices $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{k\times(k\cdot\log q)}$ and noise terms $\mathbf{e}_0, \mathbf{e}_1 \in \mathbb{Z}_q^{k\cdot\log q}$, we have:

$$\begin{aligned}&-\mathsf{LEnc}_{\mathbf{A}_0}(x_0; \mathbf{s}_0, \mathbf{s}_1, \mathbf{e}_0)\cdot\mathbf{G}^{-1}(\mathbf{A}_1) + x_0\cdot\mathsf{LEnc}_{\mathbf{A}_1}(x_1; \mathbf{s}_1, \mathbf{s}_2, \mathbf{e}_1)\\&\quad= \mathsf{LEnc}_{-\mathbf{A}_0\cdot\mathbf{G}^{-1}(\mathbf{A}_1)}(x_0\cdot x_1; (\mathbf{s}_0, \mathbf{s}_2, -\mathbf{G}^{-1}(\mathbf{A}_1)^\intercal\cdot\mathbf{e}_0 + x_0\cdot\mathbf{e}_1)).\end{aligned}$$

We elaborate on the correctness of the claimed homomorphic operations. For addition, we observe that:

$$\begin{aligned}\mathsf{LEnc}_{\mathbf{A}_0}(x_0; \mathbf{s}, \mathbf{r}, \mathbf{e}_0) + \mathsf{LEnc}_{\mathbf{A}_1}(x_1; \mathbf{s}, \mathbf{r}, \mathbf{e}_1) &= \mathbf{s}^\intercal\mathbf{A}_0 + x_0\cdot\mathbf{r}^\intercal\mathbf{G} + \mathbf{e}_0^\intercal + \mathbf{s}^\intercal\mathbf{A}_1 + x_1\cdot\mathbf{r}^\intercal\mathbf{G} + \mathbf{e}_1^\intercal\\&= \mathbf{s}^\intercal\cdot(\mathbf{A}_0+\mathbf{A}_1) + (x_0+x_1)\cdot\mathbf{r}^\intercal\mathbf{G} + (\mathbf{e}_0^\intercal + \mathbf{e}_1^\intercal)\\&= \mathsf{LEnc}_{\mathbf{A}_0+\mathbf{A}_1}(x_0+x_1; \mathbf{s}, \mathbf{r}, \mathbf{e}_0+\mathbf{e}_1).\end{aligned}$$

For scalar multiplication, we have:

$$\begin{aligned}\mathsf{LEnc}_{\mathbf{A}}(x; \mathbf{s}, \mathbf{r}, \mathbf{e})\cdot\mathbf{G}^{-1}(\delta\cdot\mathbf{G}) &= \mathbf{s}^\intercal\mathbf{A}\mathbf{G}^{-1}(\delta\cdot\mathbf{G}) + x\cdot\mathbf{r}^\intercal\mathbf{G}\mathbf{G}^{-1}(\delta\cdot\mathbf{G}) + \mathbf{e}^\intercal\mathbf{G}^{-1}(\delta\cdot\mathbf{G})\\&= \mathbf{s}^\intercal\mathbf{A}\mathbf{G}^{-1}(\delta\cdot\mathbf{G}) + (x\cdot\delta)\cdot\mathbf{r}^\intercal\mathbf{G} + \mathbf{e}^\intercal\mathbf{G}^{-1}(\delta\cdot\mathbf{G})\\&= \mathsf{LEnc}_{\mathbf{A}\cdot\mathbf{G}^{-1}(\delta\cdot\mathbf{G})}(x\cdot\delta; \mathbf{s}, \mathbf{r}, \mathbf{G}^{-1}(\delta\cdot\mathbf{G})^\intercal\cdot\mathbf{e}).\end{aligned}$$

Finally, for homomorphic multiplication, we have:

$$\begin{aligned}&-\mathsf{LEnc}_{\mathbf{A}_0}(x_0; \mathbf{s}_0, \mathbf{s}_1, \mathbf{e}_0)\cdot\mathbf{G}^{-1}(\mathbf{A}_1) + x_0\cdot\mathsf{LEnc}_{\mathbf{A}_1}(x_1; \mathbf{s}_1, \mathbf{s}_2, \mathbf{e}_1)\\&\quad= -(\mathbf{s}_0^\intercal\mathbf{A}_0 + x_0\cdot\mathbf{s}_1^\intercal\mathbf{G} + \mathbf{e}_0^\intercal)\cdot\mathbf{G}^{-1}(\mathbf{A}_1) + x_0\cdot(\mathbf{s}_1^\intercal\mathbf{A}_1 + x_1\cdot\mathbf{s}_2^\intercal\mathbf{G} + \mathbf{e}_1^\intercal)\\&\quad= \mathbf{s}_0^\intercal\cdot(-\mathbf{A}_0\mathbf{G}^{-1}(\mathbf{A}_1)) - x_0\cdot\mathbf{s}_1^\intercal\mathbf{A}_1 + x_0\cdot\mathbf{s}_1^\intercal\mathbf{A}_1 + (x_0\cdot x_1)\cdot\mathbf{s}_2^\intercal\mathbf{G} - \mathbf{e}_0^\intercal\mathbf{G}^{-1}(\mathbf{A}_1) + x_0\cdot\mathbf{e}_1^\intercal\\&\quad= \mathsf{LEnc}_{-\mathbf{A}_0\cdot\mathbf{G}^{-1}(\mathbf{A}_1)}(x_0\cdot x_1; (\mathbf{s}_0, \mathbf{s}_2, -\mathbf{G}^{-1}(\mathbf{A}_1)^\intercal\cdot\mathbf{e}_0 + x_0\cdot\mathbf{e}_1)).\end{aligned}$$

Evaluating RMS Programs. From the homomorphic operations described above, one can derive a general routine to evaluate any T-bounded (i.e., the maximum norm of an intermediate variable of the computation is bounded by T) RMS program of depth d. Recall that in an RMS program, one can sum any two variables, whereas multiplication can be done only so long as one of the two variables is an input. Before discussing the evaluation algorithm, let us generalize the notation described above to vectors, with:

$$\mathsf{LEnc}_{\mathbf{A}}(\mathbf{x}; \mathbf{s}, \mathbf{r}, \mathbf{e}) := \mathbf{s}^\intercal\mathbf{A} + \mathbf{r}^\intercal(\mathbf{x}^\intercal\otimes\mathbf{G}) + \mathbf{e}^\intercal$$

by increasing the dimensions of the components appropriately. We refer to $\mathbf{s}$ as the *encryption key* and $\mathbf{r}$ as the *authentication key.* Homomorphic operations can be extended to vectors in a straightforward manner. We are now ready to state the algorithm to evaluate RMS programs.

Lemma 4.2 (Evaluation of RMS Programs)**.** *Let* $\mathbf{A} \in \mathbb{Z}_q^{k\times(m\cdot k\cdot\log q)}$ *be a matrix,* $\mathbf{x} \in \mathbb{Z}_q^{m-1}$ *be an input, and let f be a T-bounded depth-$d(\lambda)$ RMS program. Define $\hat{\mathbf{x}}$ as the vertical concatenation of $\mathbf{x}$ and 1. For all $i \in [d]$, let:*

$$\mathbf{c}_i := \mathsf{LEnc}_{\mathbf{A}}(\hat{\mathbf{x}}; \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{e}_i)$$

with $\mathbf{s}_i \in \mathbb{Z}_q^k$ *and such that* $\max_i \|\mathbf{e}_i\|_\infty \le \beta$*. Then there exist two polynomial-time algorithms* $\mathsf{EvalRMSK}$ *and* $\mathsf{EvalRMSC}$ *such that:*

$$\mathbf{A}_f \leftarrow \mathsf{EvalRMSK}(\mathbf{A}, f) \quad \textit{and} \quad \tilde{\mathbf{c}} \leftarrow \mathsf{EvalRMSC}(\mathbf{A}, f, \mathbf{x}, \{\mathbf{c}_i\}_{i\in[d]})$$

with $\tilde{\mathbf{c}} \in \mathsf{LEnc}_{\mathbf{A}_f}(f(\mathbf{x}), \mathbf{s}_0, \mathbf{s}_d, \tilde{\mathbf{e}})$ *such that* $\|\tilde{\mathbf{e}}\|_\infty \le \beta\cdot O(T\cdot(k\cdot\log q)^d)$.

Proof. We refer to any encoding using $\mathbf{s_i}$ as encryption key and $\mathbf{s_j}$ as authentication key a level-(i, j) encoding. Observe that splitting a level-$(i, i+1)$ encoding $\mathbf{c}_i$ into blocks of dimension $k \cdot \log q$, we obtain level-$(i, i+1)$ encodings:

$$\mathsf{LEnc}_{\mathbf{A}_j}(x_j; \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{e}_{i,j})$$

where x_j is the j-th entry of $\hat{\mathbf{x}}$, $\mathbf{A}_j$ is the j-th block of $\mathbf{A}$, and $\mathbf{e}_{i,j}$ is the j-th block of $k \cdot \log q$ entries in $\widetilde{\mathbf{e}}_i$).

Using the addition and multiplication by a constant, we can apply linear operations over encodings lying on the same level (i, j). In this way, we obtain a level-(i, j) encoding of the result. On the other hand, the operation increases the norm of the noise in the encodings: If the linear operation is described by a vector $\boldsymbol{\ell}$, the noise magnitude increases by a factor $\sum_v \log \ell_v$. We can also homomorphically compute multiplications between any encoding on level (i, j) and any encoding on level $(j, j+1)$. In this way, we obtain a level-$(i, j+1)$ encoding of the product. This time the noise magnitude increases by a factor $O(k \cdot \log q)$ and by an additive term $T \cdot \beta$ for each multiplication. Finally, we observe that we can convert a level-(i, j) encoding into a level $(i, j+1)$ encoding by simply multiplying by a level-$(j, j+1)$ encoding of 1. Notice that the latter is know given that the last entry of $\hat{\mathbf{x}}$ is a 1. Overall, the growth of the noise norm is bounded by a factor $\beta \cdot O(T \cdot (k \cdot \log q)^d)$.

We also highlight that for all these operations we described, we are able to derive the matrix underlying the output encodings from the matrices underlying the input encodings. Thus both algorithms are well-defined. □

4.2 Compressing Lattice Encodings

We describe a procedure to compress and decompress lattice encodings. Formally, this consists of a triple of algorithms $(\mathsf{Setup}, \mathsf{Compress}, \mathsf{Expand})$ that allows one to sample a compressed version of a lattice encoding, that can be later on expanded into the format described above.

Let $n := n(\lambda), k := k(\lambda)$, $m := m(\lambda)$, and $q := q(\lambda)$ be positive integers. We are going to assume the existence of an α-correct succinct NI-OTE protocol $\mathsf{OTE} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{HashEval}, \mathsf{EncEval})$ with bilinear encoder evaluation for $\mathbb{Z}_q^m \otimes \mathbb{Z}_q^{k \cdot \log q}$ where digests are vectors in $\mathbb{Z}_q^n$ and the encoder private information consists of a vector in $\mathbb{Z}_q^k$. Let $\mathbf{G} := \mathbf{I}_k \otimes \mathbf{g}_q$. Our scheme is formally described in Construction 4.3. Observe that if we instantiate OTE with Construction 3.6, the size of the compressed encoding scales logarithmically in the size of its input.

Construction 4.3. Compression of Lattice Encodings

$\mathsf{Setup}(1^\lambda)$**:** Return $\mathsf{ck} := \mathsf{pk} \xleftarrow{\$} \mathsf{OTE.Setup}(1^\lambda)$.

$\mathsf{Compress}(\mathsf{ck}, \mathbf{A}, \mathbf{x}, \mathbf{s}_0, \mathbf{s}_1, \mathbf{r})$**:** Compute $(\mathbf{d}, \psi) := \mathsf{OTE.Hash}(\mathsf{pk}, \mathbf{x})$, then sample $\mathbf{e} \xleftarrow{\$} \chi(\lambda)$. Compute

$$\mathbf{h} := \mathbf{s}_0^\intercal \mathbf{A} + \mathbf{r}^\intercal(\mathbf{d}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal$$

and set $E \xleftarrow{\$} \mathsf{OTE.Enc}(\mathsf{pk}, \mathbf{s}_1 \otimes \mathbf{g}_q^\intercal, \mathbf{r})$. Return $(\mathbf{h}, E)$.

$\mathsf{Expand}(\mathsf{ck}, \mathbf{A}, \mathbf{h}, E, \mathbf{x})$**:** Recompute the hash $(\mathbf{d}, \psi) := \mathsf{OTE.Hash}(\mathsf{pk}, \mathbf{x})$ and set $\mathbf{v} := \mathsf{OTE.HashEval}(\mathsf{pk}, E, \psi)$. Let $\mathbf{P}$ be the matrix of the OTE protocol that can be publicly derived from pk. Return

$$\mathbf{c} := \mathbf{h}\mathbf{P}^\intercal + \mathbf{v}^\intercal.$$

We show that the expansion algorithm indeed leads to well-formed lattice encoding. By the correctness of the NI-OTE protocol, we have that:

$$\mathbf{w} := \mathsf{OTE.EncEval}(\mathsf{pk}, \mathbf{d}, \mathbf{s}) = \mathbf{P} \cdot (\mathbf{d} \otimes \mathbf{s} \otimes \mathbf{g}_q^\intercal).$$

Then let us rewrite:

$$\begin{aligned}
\mathbf{c} &= \mathbf{h} \cdot \mathbf{P}^\intercal + \mathbf{v}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{A} + \mathbf{r}^\intercal \cdot (\mathbf{d}^\intercal \otimes \mathbf{G}) \cdot \mathbf{P}^\intercal + \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{v}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{A} \cdot \mathbf{P}^\intercal + \mathbf{r}^\intercal \cdot (\mathbf{d}^\intercal \otimes \mathbf{I}_k \otimes \mathbf{g}_q) \cdot \mathbf{P}^\intercal + \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{v}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{A} \cdot \mathbf{P}^\intercal + (\mathbf{d}^\intercal \otimes \mathbf{r}^\intercal \otimes \mathbf{g}_q) \cdot \mathbf{P}^\intercal + \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{v}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{A} \cdot \mathbf{P}^\intercal + \mathbf{w}^\intercal + \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{v}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{A} \cdot \mathbf{P}^\intercal + \mathbf{x}^\intercal \otimes (\mathbf{s}_1^\intercal \otimes \mathbf{g}_q) + \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{e}'^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{A} \cdot \mathbf{P}^\intercal + \mathbf{s}_1^\intercal \cdot (\mathbf{x}^\intercal \otimes \mathbf{I}_k \otimes \mathbf{g}_q) + \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{e}'^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{A} \cdot \mathbf{P}^\intercal + \mathbf{s}_1^\intercal \cdot (\mathbf{x}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{e}'^\intercal \\
&= \mathsf{LEnc}_{\mathbf{AP}^\intercal}(\mathbf{x}; \mathbf{s}_0, \mathbf{s}_1, \mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{e}'^\intercal)
\end{aligned}$$

by the α-correctness of the NI-OTE. We can bound the norm of the noise term by:

$$\|\mathbf{e}^\intercal \cdot \mathbf{P}^\intercal + \mathbf{e}'^\intercal\|_\infty \leq k \cdot n \cdot \log q \cdot \|\mathbf{e}\|_\infty \cdot \|\mathbf{P}\|_\infty + \|\mathbf{e}'\|_\infty \leq k \cdot n^{2r+1} \cdot \log q \cdot B(\lambda) + \alpha \cdot \|\mathbf{x}\|_\infty.$$

by Equation (5) and by the α-correctness of the NI-OTE. We prove that the compressed encodings satisfy a notion of simulation that we define below.

Theorem 4.4 (Simulatability). *Consider the following experiment* $\mathsf{CompExp}_{\mathcal{A},\mathsf{CSim}}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *and a simulator* CSim*:*

- *Sample* $\mathsf{ck} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k \times (n \cdot k \cdot \log q)}$, *and* $\mathbf{s}_0, \mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^k$.
- *Activate the adversary* $(\mathbf{x}, \mathbf{s}_1, \mathsf{aux}) \leftarrow \mathcal{A}_0(1^\lambda, \mathsf{ck}, \mathbf{A})$.
- *Sample* $b \xleftarrow{\$} \{0,1\}$. *If* $b = 0$ *compute:*

  $$(\mathbf{h}_0, E_0) \xleftarrow{\$} \mathsf{Compress}(\mathsf{ck}, \mathbf{A}, \mathbf{x}, \mathbf{s}_0, \mathbf{s}_1, \mathbf{r})$$

  whereas if $b = 1$ *compute:*

  $$(\mathbf{h}_1, E_1) \xleftarrow{\$} \mathsf{CSim}(1^\lambda, \mathsf{ck}, \mathbf{A}).$$

- *Obtain* $b' \leftarrow \mathcal{A}_1(\mathbf{h}_b, E_b, \mathsf{aux})$ *and return* 1 *if and only if* $b = b'$.

Then assuming the hardness of LWE and that OTE *is encoder private, there exists a PPT simulator* CSim*, such that for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left| \frac{1}{2} - \Pr\left[\mathsf{CompExp}_{\mathcal{A},\mathsf{CSim}}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of the experiment.

Proof. Consider the following sequence of hybrid experiments.

- Hybrid $\mathcal{H}_0$: This is the original experiment.
- Hybrid $\mathcal{H}_1$: We provide the adversary $\mathcal{A}_1$ with a pair $(\mathbf{h}, E)$ where $\mathbf{h} \xleftarrow{\$} \mathbb{Z}_q^{n\cdot k\cdot \log q}$ and $E \xleftarrow{\$} \mathsf{OTE.Enc}(\mathsf{pk}, \mathbf{s}_1 \otimes \mathbf{g}_q^{\intercal}, \mathbf{r})$ for $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^k$.

  Indistinguishability from the previous hybrid follows by a direct reduction to LWE. Indeed, in Hybrid $\mathcal{H}_0$, we have that $\mathbf{h}^{\intercal} = \mathbf{A}^{\intercal} \cdot \mathbf{s}_0 + \mathbf{e} + (\mathbf{d} \otimes \mathbf{G}^{\intercal}) \cdot \mathbf{r}$, where in particular $\mathbf{A}^{\intercal} \cdot \mathbf{s}_0 + \mathbf{e}$ is an LWE sample.
- Hybrid $\mathcal{H}_2$: We also simulate $E \xleftarrow{\$} \mathsf{OTE.Sim}(1^{\lambda}, \mathsf{pk})$.

  Indistinguishability follows from a straightforward reduction to the encoder privacy of OTE.

We conclude by observing that in Hybrid $\mathcal{H}_2$ the pair $(\mathbf{h}, E)$ given to $\mathcal{A}_1$ is independent of the values $\mathbf{x}$ and $\mathbf{s}_1$ chosen by $\mathcal{A}_0$. This concludes the proof. □

5 Rate-1 Adaptive LFE for all Bounded-Depth Functions

5.1 Almost Optimal, Weak Reverse Trapdoor Hashing for RMS

As a stepping stone, we present a variant of a trapdoor hashing (TDH) scheme [DGI+19] with reversed syntax, achieving almost optimal encoding key size but supporting a restricted family of functions. The syntax is *reversed* from the original definition from [DGI+19] where a TDH hashes inputs and encodes functions, whereas we do the opposite.

Let $m := m(\lambda)$, $\ell := \ell(\lambda)$, $p := p(\lambda)$, $d := d(\lambda)$, $T := T(\lambda)$, $k := k(\lambda)$ and $t := t(\lambda)$ be positive integers. Construction 5.1 allows the evaluation of functions that map any pair $(\mathbf{x}, \mathbf{a})$, where $\mathbf{x} \in \{0,1\}^m$ and $\mathbf{a} \in \mathbb{Z}_p^{t\cdot k}$, into $f(\mathbf{x}) \otimes \mathbf{a}$ where $f : \{0,1\}^m \to \{0,1\}^{\ell}$ is described by a depth-d, T-bounded RMS program.

Our construction only achieves a weak mix of (adaptive) privacy and correctness: Although the encoding key leaks no information about $\mathbf{x}$ and $\mathbf{a}$, in order to successfully run the encoding procedure, we are required to know $\mathbf{x}$ (but not $\mathbf{a}$). Although this properties may seem artificial, they will later be useful in combination with techniques of [BTVW17], to build adaptive LFE with rate-1 encodings (Section 5.2).

Concerning efficiency, the digest size in our construction is logarithmic in m, t ℓ and d. Moreover, the size of the encoding key is $d^2 \cdot t \cdot \mathsf{poly}(\lambda, \log m, \log \ell)$. Notice that we can achieve logarithmic dependency in m only because the encoding procedure needs $\mathbf{x}$ in order to run successfully.

The Construction. We now invite the reader to take a look at Construction 5.1. The scheme relies on the compressed, adaptive lattice encodings of Section 4. We instantiate them over $\mathbb{Z}_q$, where $q = \Delta \cdot p$ where $\Delta := \Delta(\lambda)$ is a positive integer. We use $(\mathsf{Setup}, \mathsf{Compress}, \mathsf{Expand})$ to denote the algorithms of Construction 4.3 and we use $n := n(\lambda)$ to denote the digest size of the α-correct OTE with which it is instantiated. Let $\mathbf{P}$ be the matrix used for the hasher evaluation in the OTE protocol. We also rely on an $\hat{\alpha}$-correct succinct NI-MOLE protocol $\mathsf{MOLE} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{HashEval}, \mathsf{EncEval})$ for matrices of size $\mathbb{Z}_q^{(t\cdot\ell\cdot k\cdot \log q)\times k}$.

Construction 5.1. Almost Optimal, Weak Reverse TDH for RMS

$\mathsf{Setup}(1^{\lambda})$**:** Run the setup for the MOLE and the compressed lattice encodings

$$\mathsf{ck} \xleftarrow{\$} \mathsf{Setup}(1^{\lambda}), \qquad \mathsf{mpk} \xleftarrow{\$} \mathsf{MOLE.Setup}(1^{\lambda}).$$

Then, sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k\times(n\cdot k\cdot\log q)}$, $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{k\times(t\cdot k\cdot\log q)}$ and return $\mathsf{hk} := (\mathsf{ck}, \mathsf{mpk}, \mathbf{A}, \mathbf{B})$.

$\mathsf{Hash}(\mathsf{hk}, f)$**:** Compute:

$$\mathbf{A}_f \leftarrow \mathsf{EvalRMSK}(\mathbf{A}\mathbf{P}^\intercal, f), \qquad \mathbf{F} \leftarrow -\mathbf{A}_f \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})).$$

Then output $(\mathbf{d}, \psi) \xleftarrow{\$} \mathsf{MOLE.Hash}(\mathsf{mpk}, \mathbf{F}^\intercal)$.

$\mathsf{Gen}\left(\mathsf{hk}, \mathbf{x}, \mathbf{a}; \{\mathbf{r}_i\}_{i\in[d]}\right)$**:** Let $\hat{\mathbf{x}}$ be the vertical concatenation of $\mathbf{x}$ and 1. For every $i \in [d+1]$, sample $\mathbf{s}_i \xleftarrow{\$} \mathbb{Z}_q^k$ and set:

$$(\mathbf{h}_i, E_i) \xleftarrow{\$} \mathsf{Compress}(\mathsf{ck}, \mathbf{A}, \hat{\mathbf{x}}, \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{r}_i).$$

Next, sample $\mathbf{e} \xleftarrow{\$} \chi(1^\lambda)$ and set $\mathbf{b} \leftarrow \mathbf{s}_d^\intercal\mathbf{B} + \mathbf{a}^\intercal(\mathbf{I}_t \otimes \mathbf{G}) + \mathbf{e}^\intercal$, where $\mathbf{G} = \mathbf{I}_k \otimes \mathbf{g}_q$. Proceed by generating $(C, \phi) \xleftarrow{\$} \mathsf{MOLE.Enc}(\mathsf{mpk}, \mathbf{s}_0)$ and sampling $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$. Output $\mathsf{ek} := (C, \mathbf{b}, \{\mathbf{h}_i, E_i\}_{i\in[d]}, \mathsf{seed})$ and $\mathsf{td} := (\phi, \mathsf{seed})$

$\mathsf{Enc}(\mathsf{hk}, \mathsf{ek}, \psi, \mathbf{x}, f)$**:** For every $i \in [d]$, compute $\mathbf{c}_i \leftarrow \mathsf{Expand}(\mathsf{ck}, \boldsymbol{A}, \mathbf{h}_i, E_i, \hat{\mathbf{x}})$ and set

$$\mathbf{c} \leftarrow \mathsf{EvalRMSC}(\mathbf{A}\mathbf{P}^\intercal, f, \mathbf{x}, \{\mathbf{c}_i\}_{i\in[d]})$$

Next, derive $\mathbf{z} \leftarrow -\mathbf{c} \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{b}$. Finally, compute

$$\mathbf{v} \leftarrow \mathsf{MOLE.HashEval}(\mathsf{mpk}, C, \psi), \qquad \mathbf{u}' \leftarrow (\mathbf{z} - \mathbf{v}^\intercal) \cdot (\mathbf{I}_{t\cdot\ell\cdot k} \otimes \mathbf{G}^{-1}(\Delta)),$$
$$\mathbf{y}' \leftarrow \lceil \mathbf{u}' + \mathsf{PRG}(\mathsf{seed}) \rfloor_p$$

Output $\mathbf{y}'$.

$\mathsf{Dec}(\mathsf{hk}, \boldsymbol{d}, \mathsf{td})$**:** Compute

$$\mathbf{w} \leftarrow \mathsf{MOLE.EncEval}(\mathsf{mpk}, \mathbf{d}, \phi), \qquad \mathbf{u} \leftarrow \mathbf{w}^\intercal \left(\mathbf{I}_{t\cdot\ell\cdot k} \otimes \mathbf{G}^{-1}(\Delta)\right),$$
$$\mathbf{y} \leftarrow \lceil -\mathbf{u} - \mathsf{PRG}(\mathsf{seed}) \rfloor_p$$

Then, output $\mathbf{y}$.

Correctness. In order for the construction to be fully correct, we need to choose a sufficiently large modulus q. Specifically, it must hold that

$$p \cdot \frac{\|\mathbf{P}\|_\infty \cdot T \cdot B \cdot (k \cdot \log q)^d + \alpha \cdot T \cdot (k \cdot \log q)^d + \hat{\alpha}}{q} = 2^{-\omega(\log \lambda)}.$$

Notice that in the OTE of Section 3.3, it holds that $\|\mathbf{P}\|_\infty = n^{2r}$ where $r = O(\log m)$[5]. We begin by observing that, by the correctness of Construction 4.3, for every $i \in [d]$:

$$\mathbf{c}_i = \mathbf{s}_i^\intercal \mathbf{A}\mathbf{P}^\intercal + \mathbf{s}_{i+1}^\intercal(\hat{\mathbf{x}}^\intercal \otimes \mathbf{G}) + \mathbf{e}_i^\intercal$$

where $\|\mathbf{e}_i\|_\infty \leq k \cdot n \cdot \|\mathbf{P}\|_\infty \cdot \log q \cdot B(\lambda) + \alpha$. Therefore, by Lemma 4.2, we have that:

$$\mathbf{c} = \mathsf{EvalRMSC}(\mathbf{A}\mathbf{P}^\intercal, f, \mathbf{x}, \{\mathbf{c}_i\}_{i\in[d]}) = \mathbf{s}_0^\intercal \cdot \mathbf{A}_f + \mathbf{s}_d^\intercal \cdot (f(\mathbf{x})^\intercal \otimes \mathbf{G}) + \widetilde{\mathbf{e}}^\intercal,$$

[5] By a careful choice of parameters, r can also be made a constant:e.g., by setting $t = \lambda$ in Construction 3.6.

where $\|\widetilde{\mathbf{e}}\|_\infty \leq n \cdot \|\mathbf{P}\|_\infty \cdot B \cdot T \cdot (k \cdot \log q)^{d+1} + \alpha \cdot T \cdot (k \cdot \log q)^d$. We obtain that:

$$\begin{aligned}
\mathbf{z} &= -\mathbf{c} \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{b} \\
&= -\mathbf{s}_0^\intercal \cdot \mathbf{A}_f \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) - \mathbf{s}_d^\intercal \cdot (f(\mathbf{x})^\intercal \otimes \mathbf{G}) \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{b} \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{F} - \mathbf{s}_d^\intercal \cdot (f(\mathbf{x})^\intercal \otimes \mathbf{B}) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \left(\mathbf{s}_d^\intercal \cdot \mathbf{B} + \mathbf{a}^\intercal \cdot (\mathbf{I}_t \otimes \mathbf{G}) + \mathbf{e}^\intercal\right) \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{F} - f(\mathbf{x})^\intercal \otimes (\mathbf{s}_d^\intercal \cdot \mathbf{B}) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \left(\mathbf{s}_d^\intercal \cdot \mathbf{B} + \mathbf{a}^\intercal \cdot (\mathbf{I}_t \otimes \mathbf{G}) + \mathbf{e}^\intercal\right) \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{F} + f(\mathbf{x})^\intercal \otimes (\mathbf{a}^\intercal \cdot (\mathbf{I}_t \otimes \mathbf{G})) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{e}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{F} + \mathbf{a}^\intercal \cdot (f(\mathbf{x})^\intercal \otimes \mathbf{I}_t \otimes \mathbf{G}) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{e}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{F} + (f(\mathbf{x})^\intercal \otimes \mathbf{a}^\intercal \otimes \mathbf{g}_q) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{e}^\intercal
\end{aligned}$$

Finally, by the $\hat{\alpha}$-correctness of the MOLE, we observe that

$$\begin{aligned}
\mathbf{z} - \mathbf{v}^\intercal - \mathbf{w}^\intercal &= \mathbf{s}_0^\intercal \cdot \mathbf{F} + (f(\mathbf{x})^\intercal \otimes \mathbf{a}^\intercal \otimes \mathbf{g}_q) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{e}^\intercal - \mathbf{v}^\intercal - \mathbf{w}^\intercal \\
&= \mathbf{s}_0^\intercal \cdot \mathbf{F} + (f(\mathbf{x})^\intercal \otimes \mathbf{a}^\intercal \otimes \mathbf{g}_q) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{e}^\intercal - \mathbf{s}_0^\intercal \cdot \mathbf{F} - \hat{\mathbf{e}}^\intercal \\
&= (f(\mathbf{x})^\intercal \otimes \mathbf{a}^\intercal \otimes \mathbf{g}_q) - \widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{e}^\intercal - \hat{\mathbf{e}}^\intercal
\end{aligned}$$

where $\|\hat{\mathbf{e}}\|_\infty \leq \hat{\alpha}$. Let $\boldsymbol{\varepsilon} := -\widetilde{\mathbf{e}}^\intercal \cdot (\mathbf{I}_\ell \otimes \mathbf{G}^{-1}(\mathbf{B})) + f(\mathbf{x})^\intercal \otimes \mathbf{e}^\intercal - \hat{\mathbf{e}}^\intercal$. It holds that:

$$\|\boldsymbol{\varepsilon}\|_\infty \leq n \cdot \|\mathbf{P}\|_\infty \cdot B \cdot T \cdot (k \cdot \log q)^{d+2} + \alpha \cdot T \cdot (k \cdot \log q)^d + B + \hat{\alpha}.$$

We observe that:

$$(\mathbf{z} - \mathbf{v}^\intercal - \mathbf{w}^\intercal) \cdot (\mathbf{I}_{t \cdot \ell \cdot k} \otimes \mathbf{G}^{-1}(\Delta)) = \Delta \cdot (f(\mathbf{x})^\intercal \otimes \mathbf{a}^\intercal) + \boldsymbol{\varepsilon} \cdot (\mathbf{I}_{t \cdot \ell \cdot k} \otimes \mathbf{G}^{-1}(\Delta)).$$

Notice also that $\|\boldsymbol{\varepsilon} \cdot (\mathbf{I}_{t \cdot \ell \cdot k} \otimes \mathbf{G}^{-1}(\Delta))\|_\infty \leq \log q \cdot \|\boldsymbol{\varepsilon}\|_\infty$.

Finally, we observe that $\mathbf{y}' + \mathbf{y} \neq f(\mathbf{x})^\intercal \otimes \mathbf{a}^\intercal \bmod p$ only if one of the entries of $\mathbf{u} - \mathsf{PRG}(\mathsf{seed})$ is less than $\log q \cdot \|\boldsymbol{\varepsilon}\|_\infty$ away from an odd multiple of $\Delta/2$. Since seed is independent of $\mathbf{u}$ and by the security of the PRG, the probability of this event is at most $p \cdot \log q \cdot \|\boldsymbol{\varepsilon}\|_\infty / q + \mathsf{negl}(\lambda)$. By hypothesis, this quantity is negligible.

Adaptive Privacy. We now show that our construction satisfies adaptive privacy: under the LWE assumption with superpolynomial modulus-noise ratio, we can simulate $(C, \mathbf{b}, \{\mathbf{h}_i, E_i\}_{i \in [d]}, \mathsf{seed})$ without knowing any information about $\mathbf{x}$ and $\mathbf{a}$ even if these are chosen by the adversary after seeing the output of the setup.

Theorem 5.2 (Adaptive Privacy)**.** *Consider the following experiment* $\mathsf{AdaptivePrivacy}_{\mathcal{A},\mathsf{Sim}}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *and a simulator* Sim*:*

- *Sample* $\mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$.
- *Activate the adversary* $(\mathbf{x}, \mathbf{a}, \mathsf{aux}) \leftarrow \mathcal{A}_0(1^\lambda, \mathsf{hk})$.
- *Sample* $b \xleftarrow{\$} \{0, 1\}$. *If* $b = 0$, *sample* $\boldsymbol{r}_i \xleftarrow{\$} \mathbb{Z}_q^k$ *for every* $i \in [d]$. *Then, compute*

$$(\mathsf{ek}_0, \mathsf{td}_0) \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, \mathbf{x}, \mathbf{a}; \{\mathbf{r}_i\}_{i \in [d]})$$

If $b = 1$ *compute:*

$$\mathsf{ek}_1 \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{hk}).$$

- *Obtain* $b' \leftarrow \mathcal{A}_1(\mathsf{ek}_b, \mathsf{aux})$ *and return* 1 *if and only if* $b = b'$.

Then assuming the hardness of LWE with superpolynomial modulus-noise ratio and that MOLE *and* OTE *are encoder private, there exists a PPT simulator* Sim*, such that for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left| \frac{1}{2} - \Pr\left[\mathsf{AdaptivePrivacy}_{\mathcal{A},\mathsf{Sim}}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of the experiment.

Proof. We prove the theorem through a series of indistinguishable hybrids.

- Hybrid $\mathcal{H}_0$: This is the original experiment: We provide the adversary $\mathcal{A}_1$ with $(C, \mathbf{b}, \{\mathbf{h}_i, E_i\}_{i\in[d]}, \mathsf{seed})$ where $((C, \mathbf{b}, \{\mathbf{h}_i, E_i\}_{i\in[d]}, \mathsf{seed}), \mathsf{td}) \xleftarrow{\$} \mathsf{Gen}(\mathsf{pk}, \mathbf{x}, \mathbf{a})$.

- Hybrid $\mathcal{H}_1$: In this hybrid, we generate C using $\mathsf{MOLE.Sim}(1^\lambda, \mathsf{mpk})$. The rest remains as in the previous hybrid.

  Indistinguishability from Hybrid $\mathcal{H}_0$ follows from the encoder privacy of MOLE.

- Hybrid $\mathcal{H}_2^i$: In this hybrid, for every $j < i$, we generate $(\mathbf{h}_j, E_j) \xleftarrow{\$} \mathsf{CSim}(1^\lambda, \mathsf{ck}, \mathbf{A})$, where CSim is the simulator of Theorem 4.4. The rest remains as in the previous hybrid.

  We observe that Hybrid $\mathcal{H}_1$ is identical to Hybrid $\mathcal{H}_2^0$. Moreover, for every $i \in [d]$, Hybrid $\mathcal{H}_2^i$ is computationally indistinguishable from Hybrid $\mathcal{H}_2^{i+1}$ due to Theorem 4.4 (notice that in Hybrid $\mathcal{H}_2^i$, the pair $(\mathbf{h}_{i-1}, E_{i-1})$ is independent of $\mathbf{s}_i$).

- Hybrid $\mathcal{H}_3$: In this hybrid, we sample $\mathbf{b} \xleftarrow{\$} \mathbb{Z}_q^{t\cdot k\cdot \log q}$. The rest remains as in the previous hybrid.

  Hybrid $\mathcal{H}_2^d$ is computationally indistinguishable from $\mathcal{H}_3$ under LWE. Notice that in Hybrid $\mathcal{H}_2^d$, the terms $C, \{\mathbf{h}_i, E_i\}_{i\in[d]}$ no longer contain information about $\mathbf{s}_d$. This allows us to reduce indistinguishability to LWE. Indeed, all information about $\mathbf{a}^\intercal \cdot (\mathbf{I}_t \otimes \mathbf{G})$ in $\mathbf{b}$ is masked by

  $$\mathbf{s}_d^\intercal \cdot \mathbf{B} + \mathbf{e}^\intercal$$

  This can be viewed as an LWE sample with respect to the matrix $\mathbf{B}^\intercal$ and secret $\mathbf{s}_d$.

Notice that in Hybrid $\mathcal{H}_3$ all material provided to $\mathcal{A}_1$ is independent of $\mathbf{x}$ and $\mathbf{a}$.

□

5.2 Rate-1 Adaptive LFE for all Bounded-Depth Functions

We now present our adaptive LFE scheme for bounded-depth functions. We described it in Construction 5.3. Let $\ell := \ell(\lambda)$, $m := m(\lambda)$, and $D := D(\lambda)$ be positive integers. The construction allows the evaluation of any function $f : \{0,1\}^m \to \{0,1\}^\ell$ described by a polynomial-sized circuit of depth at most $D(\lambda)$. The digest size is $\mathsf{poly}(\lambda, \log m, \log \ell, \log D)$, whereas the size of the encodings is $m + \ell + D \cdot \mathsf{poly}(\lambda, \log m, \log \ell, \log D)$.

The construction builds upon the reverse trapdoor hashing scheme RTDH := (Setup, Hash, Gen, Enc, Dec) of Construction 5.1. We assume the latter outputs a secret-sharing over $\mathbb{Z}_p$ where $p = 2^{\omega(\log \lambda)}$. Moreover, we assume that RTDH is instantiated using the OTE protocols in Construction 3.6 and 3.4. The basic idea is the following: we pick a constant $d = O(1)$ and we rewrite f as the composition of $L = D/\log d$ functions of depth at most $\log d$. Each of these can be regarded as a depth-d RMS program, which can therefore be evaluate using RTDH.

Construction 5.3. Rate-1 Adaptive LFE for Bounded-Depth Circuits

$\mathsf{Setup}(1^\lambda)$**:** Output $\mathsf{pk} := \mathsf{hk} \xleftarrow{\$} \mathsf{RTDH.Setup}(1^\lambda)$

$\mathsf{Hash}(\mathsf{pk}, f)$**:** For any vector $\mathbf{z}$, let $f_{\mathbf{z}}$ be the function that maps a key K to $f(\mathbf{z} \oplus \mathsf{PRG}(K))$. Let f' be the function that maps a pair $(\mathsf{ct}, \mathbf{z})$ to $\mathsf{GSW.Eval}(\mathsf{ct}, f_{\mathbf{z}})$ where the evaluation is performed as in [BV14]. Rewrite f' as

$$f' = f'_{L-1} \circ f'_{L-2} \circ \cdots \circ f'_0$$

where each f'_i is described by a T-bounded depth-d RMS program, all with the same input and output size. For every $i \in [L-1]$, let $\hat{f}_i$ be the function that maps $\mathbf{x}$ to $\mathsf{OTE.Hash}(\mathsf{tpk}, f_i(\mathbf{x}))$ and set $\hat{f}_{L-1} \leftarrow f'_{L-1}$. Notice that $\mathsf{OTE.Hash}$ is linear so its depth is 0[a]. Here tpk denotes the OTE public key that is used in Construction 4.3. Let $\hat{f}$ the function that maps any vector $\mathbf{x}$ to $(\hat{f}_0(\mathbf{x}), \ldots, \hat{f}_{L-1}(\mathbf{x}))$. Output $(\mathbf{d}, \psi) \leftarrow \mathsf{RTDH.Hash}(\mathsf{hk}, \hat{f})$.

$\mathsf{Enc}(\mathsf{pk}, h, \mathbf{x})$**:** Sample $\mathbf{M} \xleftarrow{\$} \mathbb{Z}_q^{(k-1)\times(k\cdot\log q+\lambda)}$, $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^{k-1}$ and $\mathbf{e} \xleftarrow{\$} \chi(1^\lambda)$ and generate a GSW key

$$\mathsf{sk}_{\mathsf{GSW}} := \begin{pmatrix} \mathbf{r} \\ -1 \end{pmatrix} \qquad\qquad \mathsf{pk}_{\mathsf{GSW}} := \begin{pmatrix} \mathbf{M} \\ \mathbf{r}^\intercal \cdot \mathbf{M} + \mathbf{e}^\intercal \end{pmatrix}$$

Sample $K \xleftarrow{\$} \{0,1\}^\lambda$ and encrypt the input

$$\mathbf{z} \leftarrow \mathbf{x} \oplus \mathsf{PRG}(K), \qquad\qquad \mathsf{ct} \xleftarrow{\$} \mathsf{GSW.Enc}(\mathsf{pk}_{\mathsf{GSW}}, K).$$

For every $i \in [L], j \in [d]$ sample randomness $\mathbf{r}_{i,j} \xleftarrow{\$} \mathbb{Z}_q^k$ and define:

$$\mathbf{a}_i := \begin{pmatrix} \mathbf{G}^{-1}(\mathbf{r}_{i,0}) \\ \vdots \\ \mathbf{G}^{-1}(\mathbf{r}_{i,d-1}) \end{pmatrix} \qquad\qquad \mathbf{a}_L \leftarrow \mathbf{G}^{-1}(\mathsf{sk}_{\mathsf{GSW}})$$

Then, set $\mathbf{x}_0 := (\mathsf{Bits}(\mathsf{ct}), \mathbf{z})$ and compute

$$\begin{aligned} &(\mathsf{ek}_0, \mathsf{td}_0) \xleftarrow{\$} \mathsf{RTDH.Gen}(\mathsf{hk}, \mathbf{x}_0, \mathbf{a}_1; \{\mathbf{r}_{0,j}\}_{j\in[d]}) \\ \forall i > 0: \quad &(\mathsf{ek}'_i, \mathsf{td}_i) \xleftarrow{\$} \mathsf{RTDH.Gen}(\mathsf{hk}, \mathbf{0}, \mathbf{a}_{i+1}; \{\mathbf{r}_{i,j}\}_{j\in[d]}) \end{aligned}$$

For every $i \in [L]$, let $\mathsf{ek}'_i = (C_i, \mathbf{b}_i, \{\mathbf{h}'_{i,j}, E_{i,j}\}_{j\in[d]}, \mathsf{seed}_i)$. Compute

$$\mathbf{y}_i \leftarrow -\mathsf{RTDH.Dec}(\mathsf{hk}, \mathbf{d}, \mathsf{td}_i) \bmod p$$

and set $\overline{\mathsf{ek}}_i := (C_i, \mathbf{b}_i, \{E_{i,j}\}_{j\in[d]}, \mathsf{seed}_i)$. Let n be the digest size of OTE. For every $i \in [L-1]$, take the block of $\mathbf{y}_i$ corresponding to the evaluation of $\hat{f}_i$ and split it into $d \cdot n$ subblocks $\hat{\mathbf{y}}_{i,0}, \ldots, \hat{\mathbf{y}}_{i,n\cdot d-1}$ of dimension $k \cdot \log q$ and set

$$\begin{aligned} \mathbf{y}_{i,j} &\leftarrow \left(\hat{\mathbf{y}}_{i,j} \,\|\, \hat{\mathbf{y}}_{i,d+j} \,\|\, \ldots \,\|\, \hat{\mathbf{y}}_{i,d\cdot(n-1)+j}\right) \\ \mathbf{w}_{i,j} &\leftarrow \mathbf{h}'_{i+1,j} - (\mathbf{y}_{i,j} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n\cdot k} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_{\log q}) \bmod q \end{aligned}$$

Let $\mathbf{u}$ be the last element of the standard basis of $\mathbb{Z}_q^k$ and let $\overline{\mathbf{y}}_{L-1}$ be the block of $\mathbf{y}_{L-1}$ corresponding to the output of $\hat{f}_{L-1}$. Compute

$$\mathbf{w}_{L-1} \leftarrow -\overline{\mathbf{y}}_{L-1} \cdot (\mathbf{I}_{\ell\cdot k^2\cdot\log q} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_k \otimes \mathbf{g}_q^\intercal) \cdot \mathsf{Lin}(\mathbf{I}_{\ell\cdot k^2\cdot\log q})^\intercal \cdot \mathsf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \bmod q.$$

Sample $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$ and set $\mathbf{w} \leftarrow \lceil \mathbf{w}_{L-1} + \mathsf{PRG}(\mathsf{seed}) \rfloor_2$.

Output $E = (\mathsf{ct}, \mathbf{z}, \mathsf{seed}, \mathsf{ek}_0, \{\overline{\mathsf{ek}}_i\}_{i\in[L]}, \{\mathbf{w}_{i,j}\}_{i\in[L-1],j\in[d]}, \mathbf{w})$

$\mathsf{Dec}(\mathsf{pk}, E, f, \psi)$**:** Initially, set $\mathbf{x}_0 \leftarrow (\mathsf{Bits}(\mathsf{ct}), \mathbf{z})$. Then, for $i = 0, \ldots, L-2$:

- compute $\mathbf{y}'_i \leftarrow \mathsf{RTDH.Enc}(\mathsf{hk}, \mathsf{ek}_i, \psi, \mathbf{x}_i, \hat{f})$
- take the block of $\mathbf{y}'_i$ corresponding to the evaluation of $\hat{f}_i$ and split it into $d \cdot n$ subblocks $\hat{\mathbf{y}}'_{i,0}, \ldots, \hat{\mathbf{y}}'_{i,n\cdot d-1}$ of dimension $k \cdot \log q$ and set

$$\mathbf{y}'_{i,j} \leftarrow \left(\hat{\mathbf{y}}'_{i,j} \parallel \hat{\mathbf{y}}'_{i,d+j} \parallel \ldots \parallel \hat{\mathbf{y}}'_{i,d\cdot(n-1)+j}\right)$$
$$\mathbf{h}_{i+1,j} \leftarrow \mathbf{w}_{i,j} + (\mathbf{y}'_{i,j} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n\cdot k} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_{\log q}) \bmod q$$

- set $\mathsf{ek}_{i+1} \leftarrow (\overline{\mathsf{ek}}_{i+1}, \{\mathbf{h}_{i+1,j}\}_{j\in[d]})$
- compute the input to the next RMS program $\mathbf{x}_{i+1} \leftarrow f'_i(\mathbf{x}_i)$

Finally, let $\mathbf{u}$ be the last element of the standard basis of $\mathbb{Z}_q^k$. Derive

$$\mathbf{y}'_{L-1} \leftarrow \mathsf{RTDH.Enc}(\mathsf{hk}, \mathsf{ek}_{L-1}, \psi, \mathbf{x}_{L-1}, \hat{f})$$
$$\mathbf{w}'_{L-1} \leftarrow \overline{\mathbf{y}}'_{L-1} \cdot (\mathbf{I}_{\ell\cdot k^2\cdot\log q} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_k \otimes \mathbf{g}_q^\intercal) \cdot \mathsf{Lin}(\mathbf{I}_{\ell\cdot k^2\cdot\log q})^\intercal \cdot \mathsf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \bmod q.$$

where $\overline{\mathbf{y}}'_{L-1}$ denotes the block of $\mathbf{y}'_{L-1}$ corresponding to the evaluation of $\hat{f}_{L-1}$.

Output $\lceil (\mathbf{w}'_{L-1} - \mathsf{PRG}(\mathsf{seed}) \rfloor_2 \oplus \mathbf{w}$

[a] See Construction 3.4 and Construction 3.6.

Correctness. Towards proving correctness, we prove the following lemma, which states that the term $\mathbf{h}_{i,j}$ computed by the server during the decoding procedure is an adaptive lattice encoding of $\mathsf{OTE.Hash}(\mathsf{tpk}, \mathbf{x}_i)$. In other words, ek_i is an encoding key for $\mathbf{x}_i = (f'_{i-1} \circ \cdots \circ f'_0)(\mathsf{Bits}(\mathsf{ct}), \mathbf{z})$.

Lemma 5.4. *For every $i \in [L]$ and $j \in [d]$, we have that*

$$\mathbf{h}_{i,j} = \mathbf{s}_{i,j}^\intercal \cdot \mathbf{A} + \mathbf{r}_{i,j}^\intercal \cdot (\mathbf{d}_i'^\intercal \otimes \mathbf{G}) + \mathbf{e}_i^\intercal$$

where $(\mathbf{d}'_i, \psi'_i) \leftarrow \mathsf{OTE.Hash}(\mathsf{tpk}, \mathbf{x}_i)$, $\mathbf{s}_{i,j}$ is the encryption key used in the encoding $\mathbf{h}'_{i,j}$ ($\mathbf{h}_{0,j}$, if $i = 0$) and $\|\mathbf{e}_i\|_\infty \leq B(\lambda)$.

Proof. We proceed by induction over i. The claim is trivially true for $i = 0$. Now, we show that if it holds for i, it holds also for $i + 1$.

We observe that

$$\begin{aligned}
\mathbf{h}_{i+1,j} &= \mathbf{w}_{i,j} + (\mathbf{y}'_{i,j} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n\cdot k} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_{\log q}) \bmod q \\
&= \mathbf{h}'_{i+1,j} + (\mathbf{y}'_{i,j} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n\cdot k} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_{\log q}) - (\mathbf{y}_{i,j} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n\cdot k} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_{\log q}) \bmod q \\
&= \mathbf{h}'_{i+1,j} + ((\mathbf{y}'_{i,j} - \mathbf{y}_{i,j}) \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n\cdot k} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_{\log q}) \bmod q.
\end{aligned}$$

Now, by Theorem 5.2 and the inductive hypothesis, we recall that $\mathbf{y}'_i - \mathbf{y}_i = \hat{f}(\mathbf{x}_i)^\intercal \otimes \mathbf{a}_{i+1}^\intercal \bmod p$. We also recall that this subtractive secret-sharing was initially over $\mathbb{Z}_q$, i.e. the parties held vectors $\mathbf{u}'_i$ and $\mathbf{u}_i$ such that $\mathbf{u}'_i - \mathbf{u}_i = q/p \cdot \hat{f}(\mathbf{x}_i)^\intercal \otimes \mathbf{a}_{i+1}^\intercal + \boldsymbol{\varepsilon}_i$ where $\|\boldsymbol{\varepsilon}_i\|_\infty$ is low. This secret-sharing was rerandomised using $\mathsf{PRG}(\mathsf{seed}_i)$ and then rounded. We argue that $\mathbf{y}_i$ is pseudorandom. In particular, since $p = 2^{\omega(\log \lambda)}$, with overwhelming probability, all entries of $\mathbf{y}_i$ are in the interval $[-p/2, p/2-1)$. Since $\hat{f}(\mathbf{x}_i)^\intercal \otimes \mathbf{a}_{i+1}^\intercal$ has all entries in $\{0,1\}$, we conclude that (with overwhelming probability) $\mathbf{y}'_i$ and $\mathbf{y}_i$ is a subtractive secret-sharing of $\hat{f}(\mathbf{x}_i)^\intercal \otimes \mathbf{a}_{i+1}^\intercal$ even over the integers. Continuing, we observe that the i-th n-entry block of $\hat{f}(\mathbf{x}_i)$ coincides with $\mathbf{d}'_{i+1}$ where $\mathbf{d}'_{i+1}$ is the output of $\mathsf{OTE.Hash}(\mathsf{tpk}, f'_i(\mathbf{x}_i)) = \mathsf{OTE.Hash}(\mathsf{tpk}, \mathbf{x}_{i+1})$. Moreover, we observe that, by the way we constructed $\mathbf{a}_{i+1}$, $\mathbf{y}'_{i,j}$ and $\mathbf{y}_{i,j}$, we have that

$$\mathbf{y}'_{i,j} - \mathbf{y}_{i,j} = {\mathbf{d}'_{i+1}}^\intercal \otimes \mathbf{G}^{-1}(\mathbf{r}_{i+1,j})^\intercal$$

Putting everything together, we obtain

$$\begin{aligned}
\mathbf{h}_{i+1,j} &= \mathbf{h}'_{i+1,j} + ({\mathbf{d}'_{i+1}}^\intercal \otimes \mathbf{G}^{-1}(\mathbf{r}_{i+1,j})^\intercal \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n\cdot k} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_{\log q}) \\
&= \mathbf{h}'_{i+1,j} + ({\mathbf{d}'_{i+1}}^\intercal \otimes \mathbf{G}^{-1}(\mathbf{r}_{i+1,j})^\intercal \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n} \otimes \mathbf{G}^\intercal \otimes \mathbf{I}_{\log q}) \\
&= \mathbf{h}'_{i+1,j} + {\mathbf{d}'_{i+1}}^\intercal \otimes \mathbf{r}_{i+1,j}^\intercal \otimes \mathbf{g}_q \\
&= \mathbf{h}'_{i+1,j} + \mathbf{r}_{i+1,j}^\intercal \cdot ({\mathbf{d}'_{i+1}}^\intercal \otimes \mathbf{G}). \qquad (8)
\end{aligned}$$

Since $\mathbf{h}'_{i+1,j} = \mathbf{s}_{i+1,j}^\intercal \cdot \mathbf{A} + \mathbf{e}_{i+1}^\intercal$ where $\|\mathbf{e}_{i+1}\|_\infty \leq B(\lambda)$, we obtain that

$$\mathbf{h}_{i+1,j} = \mathbf{s}_{i+1,j}^\intercal \cdot \mathbf{A} + \mathbf{r}_{i+1,j}^\intercal \cdot ({\mathbf{d}'_{i+1}}^\intercal \otimes \mathbf{G}) + \mathbf{e}_{i+1}^\intercal$$

This ends the proof of the lemma. □

Continuing with our analysis, by Lemma 5.4, we have that $\mathbf{y}'_{L-1}$ and $\mathbf{y}_{L-1}$ form a subtractive secret sharing of $\hat{f}(\mathbf{x}_{L-1})^\intercal \otimes \mathbf{a}_L^\intercal$ over $\mathbb{Z}_p$. Furthermore, similarly to how we argued in the proof of Lemma 5.4, it is possible to prove that the probability that any entry of $\mathbf{y}_{L-1}$ lies outside of $[-p/2, p/2-1)$ is negligible. Given that all entries of $\hat{f}(\mathbf{x}_{L-1})^\intercal \otimes \mathbf{a}_L^\intercal$ belong to $\{0,1\}$, we conclude that, with overwhelming probability, $\mathbf{y}'_{L-1}$ and $\mathbf{y}_{L-1}$ form a subtractive secret sharing of $\hat{f}(\mathbf{x}_{L-1})^\intercal \otimes \mathbf{a}_L^\intercal$ even over $\mathbb{Z}$. By the way we constructed $\hat{f}, \overline{\mathbf{y}}_{L-1}$ and $\overline{\mathbf{y}}'_{L-1}$, we infer that $\overline{\mathbf{y}}'_{L-1} - \overline{\mathbf{y}}_{L-1}) = \hat{f}_{L-1}(\mathbf{x}_{L-1})^\intercal \otimes \mathbf{a}_L^\intercal$

We also notice that $\mathbf{a}_L = \mathbf{G}^{-1}(\mathsf{sk}_{\mathsf{GSW}})$, whereas

$$\hat{f}_{L-1}(\mathbf{x}_{L-1}) = f'_{L-1}(\mathbf{x}_{L-1}) = (f'_{L-1} \circ \cdots \circ f'_0)(\mathbf{x}_0) = \mathsf{Bits}(f'(\mathsf{ct}, \mathbf{z})) = \mathsf{Bits}(\mathsf{vec}(\mathsf{GSW.Eval}(\mathsf{ct}, f_{\mathbf{z}}))).$$

Let $\hat{\mathsf{ct}} := \mathsf{GSW.Eval}(\mathsf{ct}, f_{\mathbf{z}})$. We obtain that

$$\begin{aligned}
\mathbf{w}'_{L-1} + \mathbf{w}_{L-1} &= ((\overline{\mathbf{y}}'_{L-1} - \overline{\mathbf{y}}_{L-1}) \cdot (\mathbf{I}_{\ell\cdot k^2 \cdot \log q} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_k \otimes \mathbf{g}_q^\intercal) \cdot \mathsf{Lin}(\mathbf{I}_{\ell\cdot k^2\cdot\log q})^\intercal \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \\
&= (\hat{f}_{L-1}(\mathbf{x}_{L-1})^\intercal \otimes \mathbf{a}_L^\intercal) \cdot (\mathbf{I}_{\ell\cdot k^2 \cdot \log q} \otimes \mathbf{g}_q^\intercal \otimes \mathbf{I}_k \otimes \mathbf{g}_q^\intercal) \cdot \mathsf{Lin}(\mathbf{I}_{\ell\cdot k^2\cdot\log q})^\intercal \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \\
&= (\mathsf{vec}(\hat{\mathsf{ct}})^\intercal \otimes \mathsf{sk}_{\mathsf{GSW}}^\intercal) \cdot \mathsf{Lin}(\mathbf{I}_{\ell\cdot k^2\cdot\log q})^\intercal \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \\
&= \mathsf{vec}(\hat{\mathsf{ct}})^\intercal \cdot (\mathbf{I}_{\ell\cdot k\cdot\log q} \otimes \mathsf{sk}_{\mathsf{GSW}}) \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \\
&= \mathsf{sk}_{\mathsf{GSW}}^\intercal \cdot \hat{\mathsf{ct}} \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})).
\end{aligned}$$

By the correctness of GSW evaluation, we have that $\mathsf{sk}_{\mathsf{GSW}}^{\intercal} \cdot \hat{\mathsf{ct}} = \mathsf{sk}_{\mathsf{GSW}}^{\intercal} \cdot (f_{\mathbf{z}}(K)^{\intercal} \otimes \mathbf{G}) + \hat{\mathbf{e}}^{\intercal}$ where $\|\hat{\mathbf{e}}\|_\infty \leq B \cdot D \cdot \mathsf{poly}(\lambda)$. Notice that $f_{\mathbf{z}}(K) = f(\mathbf{z} \oplus \mathsf{PRG}(K)) = f(\mathbf{x})$. We conclude that

$$\begin{aligned}\mathbf{w}'_{L-1} + \mathbf{w}_{L-1} &= (\mathsf{sk}_{\mathsf{GSW}}^{\intercal} \cdot (f(\mathbf{x})^{\intercal} \otimes \mathbf{G}) + \hat{\mathbf{e}}^{\intercal}) \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \\ &= -\Delta \cdot \mathsf{sk}_{\mathsf{GSW}}^{\intercal} \cdot (f(\mathbf{x})^{\intercal} \otimes \mathbf{u}) + \hat{\mathbf{e}}^{\intercal} \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \\ &= -\Delta \cdot f(\mathbf{x})^{\intercal} \otimes (\mathsf{sk}_{\mathsf{GSW}}^{\intercal} \cdot \mathbf{u}) + \hat{\mathbf{e}}^{\intercal} \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})) \\ &= \Delta \cdot f(\mathbf{x})^{\intercal} + \hat{\mathbf{e}}^{\intercal} \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u})). \end{aligned} \tag{9}$$

Notice that $\|\hat{\mathbf{e}}^{\intercal} \cdot \mathbf{G}^{-1}(-\Delta \cdot (\mathbf{I}_\ell \otimes \mathbf{u}))\|_\infty \leq \|\hat{\mathbf{e}}\|_\infty$. So, unless any of the entries of $\mathbf{w}_{L-1} + \mathsf{PRG}(\mathsf{seed})$ is less than $\|\hat{\mathbf{e}}\|_\infty$ away from $q/4$ or $-q/4$, we have that

$$f(\mathbf{x}) = \lceil (\mathbf{w}'_{L-1} - \mathsf{PRG}(\mathsf{seed}) \rfloor_2 \oplus \lceil \mathbf{w}_{L-1} + \mathsf{PRG}(\mathsf{seed}) \rfloor_2.$$

Since seed is sampled independently of $\mathbf{w}_{L-1}$, the probability of the bad event is at most $\mathsf{poly}(\lambda) \cdot \|\hat{\mathbf{e}}\|_\infty / q) + \mathsf{negl}(\lambda)$. Given our choice of q for RTDH, this is a negligible amount.

Security. Next, we prove that our construction is encoder private.

Theorem 5.5. *Assume the hardness of LWE with superpolynomial modulus-noise ratio. Then, Construction 5.3 is an adaptively encoder private LFE.*

Proof. We prove our claim by relying on a series of indistinguishable hybrids.

- Hybrid $\mathcal{H}_0$: This hybrid corresponds to the original game: we provide the adversary with a tuple $(\mathsf{ct}, \mathbf{z}, \mathsf{seed}, \{\mathbf{h}_{0,j}\}_{j \in [d]}, \{\overline{\mathsf{ek}}_i\}_{i \in [L]}, \{\mathbf{w}_{i,j}\}_{i \in [L-1], j \in [d]}, \mathbf{w})$ generated using $\mathsf{LFE.Enc}(\mathsf{pk}, h, \mathbf{x})$.
- Hybrid $\mathcal{H}_1$: In this hybrid, we change the distribution of $\mathbf{w}$: using pk, f, $(\mathsf{ct}, \mathbf{z}, \mathsf{seed}, \{\mathbf{h}_{0,j}\}_{j \in [d]}, \{\overline{\mathsf{ek}}_i\}_{i \in [L]}, \{\mathbf{w}_{i,j}\}_{i \in [L-1], j \in [d]})$ and following the same operations as in the decoding procedure, we compute $\mathbf{w}'_{L-1}$. Then, we set $\mathbf{w} \leftarrow f(\mathbf{x}) \oplus \lceil \mathbf{w}'_{L-1} - \mathsf{PRG}(\mathsf{seed}) \rfloor_2$.

  This hybrid is statistically indistinguishable from Hybrid $\mathcal{H}_0$ due to the correctness of the primitive.
- Hybrid $\mathcal{H}_2$: In this hybrid, we change the distribution of $\mathbf{w}_{i,j}$ for every $i \in [L-1]$ and $j \in [d]$: using pk, f, $(\mathsf{ct}, \mathbf{z}, \{\mathbf{h}_{\mathbf{0},\beta}\}_{\beta \in [d]}, \{\overline{\mathsf{ek}}_\gamma\}_{\gamma \in [i]}, \{\mathbf{w}_{\gamma,\beta}\}_{\gamma \in [i], \beta \in [d]})$ and following the same operations as in the decoding procedure, we compute $\mathbf{y}'_{i,j}$. Then, we set

  $$\mathbf{w}_{i,j} \leftarrow \mathbf{h}'_{i+1,j} + \mathbf{r}_{i+1,j}^{\intercal} \cdot (\mathbf{d}'_{i+1}{}^{\intercal} \otimes \mathbf{G}) - (\mathbf{y}'_{i,j} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n \cdot k} \otimes \mathbf{g}_q^{\intercal} \otimes \mathbf{I}_{\log q}), \tag{10}$$

  where $(\mathbf{d}'_{i+1}, \psi'_{i+1}) \leftarrow \mathsf{OTE.Hash}(\mathsf{tpk}, \mathbf{x}_{i+1})$ and $\mathbf{x}_{i+1} \leftarrow (f'_i \circ \cdots \circ f'_0)(\mathsf{Bits}(\mathsf{ct}), \mathbf{z})$.

  This hybrid is statistically indistinguishable from Hybrid $\mathcal{H}_1$. Indeed, $\mathbf{w}_{i,j}$ satisfies the relation in (10) even in $\mathcal{H}_1$. This is highlighted in Lemma 5.4, specifically in (8).
- Hybrid $\mathcal{H}_3$: In this hybrid, for every $i \in [L] \setminus \{0\}$, we compute

  $$(\mathsf{ek}_i, \mathsf{td}_i) \xleftarrow{\$} \mathsf{RTDH.Gen}(\mathsf{hk}, \mathbf{x}_{i+1}, \mathbf{a}_{i+1}; \{\mathbf{r}_{i,j}\}_{j \in [d]})$$

  where $\mathbf{x}_{i+1} \leftarrow (f'_i \circ \cdots \circ f'_0)(\mathsf{Bits}(\mathsf{ct}), \mathbf{z})$. Let $\mathsf{ek}_i = (C_i, \mathbf{b}_i, \{\mathbf{h}_{i,j}, E_{i,j}\}_{j \in [d]}, \mathsf{seed}_i)$. For every $i \in [L-1]$ and $j \in [d]$, we set

  $$\mathbf{w}_{i,j} \leftarrow \mathbf{h}_{i+1,j} - (\mathbf{y}'_{i,j} \otimes \mathbf{g}_q) \cdot (\mathbf{I}_{n \cdot k} \otimes \mathbf{g}_q^{\intercal} \otimes \mathbf{I}_{\log q}),$$

We also argue that $\mathcal{H}_2$ is perfectly indistinguishable from Hybrid $\mathcal{H}_3$. Indeed, the only difference between the output of $\mathsf{RTDH.Gen}(\mathsf{hk}, \mathbf{x}_{i+1}, \mathbf{a}_{i+1}; \{\mathbf{r}_{i,j}\}_{j \in [d]})$ and $\mathsf{RTDH.Gen}(\mathsf{hk}, \mathbf{0}, \mathbf{a}_{i+1}; \{\mathbf{r}_{i,j}\}_{j \in [d]})$ is that, in the first case, the encoding $\mathbf{h}_{i,j}$ is "shifted" by $\mathbf{r}_{i+1,j}^{\intercal} \cdot (\mathbf{d}_{i+1}'^{\intercal} \otimes \mathbf{G})$ where $(\mathbf{d}'_{i+1}, \psi'_{i+1}) \leftarrow \mathsf{OTE.Hash}(\mathsf{tpk}, \mathbf{x}_{i+1})$ (see Construction 4.3). In the second case, no shift is essentially applied. This is because, by the linearity of our OTE hashing (see Construction 3.4 and Construction 3.6), $\mathsf{OTE.Hash}(\mathsf{tpk}, \mathbf{0})$ outputs $\mathbf{0}$.

- Hybrid $\mathcal{H}_4^\iota$: In this hybrid, for every $i < \iota$, we generate

$$\mathsf{ek}_i := (C_i, \mathbf{b}_i, \{\mathbf{h}_{i,j}, E_{i,j}\}_{j \in [d]}, \mathsf{seed}_i) \xleftarrow{\$} \mathsf{RTDH.Sim}(1^\lambda, \mathsf{hk})$$

  We observe that Hybrid $\mathcal{H}_3$ is identical to $\mathcal{H}_4^0$. Moreover, by Theorem 5.2, for every $\iota \in [L-1]$, we have that $\mathcal{H}_4^\iota$ is computationally indistinguishable from $\mathcal{H}_4^{\iota+1}$. Notice that here we are implicitly relying on the fact that, in Hybrid $\mathcal{H}_4^\iota$, the tuple $\{\mathsf{ek}_i\}_{i<\iota}$ no longer contains information about $\{\mathbf{r}_{\iota,j}\}_{j \in [d]}$.

- Hybrid $\mathcal{H}_5$: In this hybrid, we sample $\mathsf{pk}_{\mathsf{GSW}} \xleftarrow{\$} \mathbb{Z}_q^{k \times (k \cdot \log q + \lambda)}$.

  We argue that $\mathcal{H}_4^t$ is computationally indistinguishable from $\mathcal{H}_5$. Indeed, under LWE, the term $\mathbf{r}^{\intercal} \cdot \mathbf{M} + \mathbf{e}^{\intercal}$ is indistinguishable from random. Here, we are implicitly relying on the fact that, in $\mathcal{H}_4^t$, the tuple $\{\mathsf{ek}_i\}_{i \in [L]}$ no longer contains information about $\mathbf{r}$.

- Hybrid $\mathcal{H}_6$: In this hybrid, we generate $\mathsf{ct} \xleftarrow{\$} \mathbb{Z}_q^{k \times (\lambda \cdot k \cdot \log q)}$.

  Hybrid $\mathcal{H}_6$ is statistically indistinguishable from $\mathcal{H}_5$. Indeed, now $\mathsf{pk}_{\mathsf{GSW}}$ is a uniformly random matrix, so we can apply the leftover hash lemma to argue that ct is indistinguishable from random.

- Hybrid $\mathcal{H}_7$: In this hybrid, we sample $\mathbf{z} \xleftarrow{\$} \mathbb{Z}_2^m$.

  Since ct no longer contains information about the PRG seed K, we can argue that $\mathcal{H}_6$ is computationally indistinguishable from $\mathcal{H}_7$ thanks to the security of the PRG.

Notice that in $\boldsymbol{\mathcal{H}_7}$ all the material provided to the adversary can be computed by a simulator with no information about $\mathbf{x}$ except $f(\mathbf{x})$. This ends the proof of security.

□

6 Reverse Trapdoor Hashing for all Functions

We construct reverse TDHs for all functions. Note that, for an expressive enough class of functions, in terms of functionality reverse TDHs are identical to the standard notion of TDH [DGI+19], since one can always encode the universal circuit as the input, and vice-versa. However, since the encoding key grows with the size of the input, embedding a universal circuit introduces a dependency in the size of the function. Thus, we pay a price in succinctness, when going from TDH to reverse TDH. On the other hand, the opposite direction (reverse TDH $\implies$ TDH) has no such problem, since the size of the hash is anyway constant. Thus, reverse TDH appears to be a more powerful abstraction.

6.1 Definition

Definition 6.1 (Reverse Trapdoor Hashing)**.** *A reverse trapdoor hashing scheme for the function class $\mathcal{F} = (\mathcal{F}_\lambda)_{\lambda \in \mathbb{N}}$ with input size $m(\lambda)$ and output of size $\ell(\lambda)$ consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda)$**:** *The setup algorithm is randomised and takes as input the security parameter* 1^λ*. The output is a hash key* hk*.*

$\mathsf{Hash}(\mathsf{hk}, f)$**:** *The hashing algorithm is randomised takes as input a hash key* hk *and a function* $f \in \mathcal{F}_\lambda$*. The output is a digest* d *and hasher's private information* ρ*.*

$\mathsf{Gen}(\mathsf{hk}, \mathbf{x})$**:** *The generation algorithm is randomised and takes as input an hash key* hk*, an element* $\mathbf{x} \in \mathbb{Z}_2^m$*. The output is an encoding key* ek *and a trapdoor* td*.*

$\mathsf{Enc}(\mathsf{hk}, \mathsf{ek}, f, \rho)$**:** *The encoding algorithm is deterministic and takes as input a hash key* hk*, an encoding key* ek *and a function* $f \in \mathcal{F}_\lambda$ *and hasher's private information* ρ*. The output is an encoding* $\mathbf{e} \in \mathbb{Z}_2^\ell$*.*

$\mathsf{Dec}(\mathsf{hk}, \mathsf{td}, d)$**:** *The decoding procedure is deterministic and takes as input a hash key* hk*, a trapdoor* td *and a digest* d*. The output is an encoding* $\mathbf{e}' \in \mathbb{Z}_2^\ell$*.*

Definition 6.2 (Correctness)**.** *We say that a reverse TDH scheme is correct if there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, function* $f \in \mathcal{F}_\lambda$ *and* $\mathbf{x} \in \mathbb{Z}_2^m$*, it holds that:*

$$\Pr\left[\mathsf{Enc}(\mathsf{hk}, \mathsf{ek}, f, \rho) \oplus \mathsf{Dec}(\mathsf{hk}, \mathsf{td}, d) \neq f(\mathbf{x})\right] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random choice of $\mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*,* $(d, \rho) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, f)$*, and* $(\mathsf{ek}, \mathsf{td}) \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, \mathbf{x})$*.*

Definition 6.3 (Function privacy of reverse trapdoor hashing)**.** *Consider the following experiment* $\mathsf{FuncExp}_\mathcal{A}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*:*

- *Sample a hash key* $\mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*.*
- *Activate the adversary* $(f_0, f_1, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$*.*
- *Sample a random bit* $b \xleftarrow{\$} \{0, 1\}$*.*
- *Compute* $(d, \rho) \xleftarrow{\$} \mathsf{Hash}(\mathsf{hk}, f_b)$*.*
- *Compute* $b' \leftarrow \mathcal{A}_1(\mathsf{hk}, d, \mathsf{aux})$*.*
- *Return* 1 *if and only if* $b = b'$*.*

We say that a reverse trapdoor hashing scheme $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is function private if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left|\frac{1}{2} - \Pr\left[\mathsf{FuncExp}_\mathcal{A}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

If the above property holds for every adversary (even computationally unbounded ones) we say that the scheme is statistically function private.

Definition 6.4 (Input privacy of reverse trapdoor hashing)**.** *Consider the following experiment* $\mathsf{InpExp}_{\mathcal{A},\mathsf{Sim}}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *and a simulator* Sim*:*

- *Sample a hash key* $\mathsf{hk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*.*
- *Activate the adversary* $(\mathbf{x}, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$*.*

- *Sample a random bit* $b \xleftarrow{\$} \{0,1\}$.
- *If* $b = 0$ *compute* $(\mathsf{ek}_0, \mathsf{td}_0) \xleftarrow{\$} \mathsf{Gen}(\mathsf{hk}, \mathbf{x})$, *else compute* $\mathsf{ek}_1 \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{hk})$.
- *Compute* $b' \leftarrow \mathcal{A}_1(\mathsf{hk}, \mathsf{ek}_b, \mathsf{aux})$.
- *Return* 1 *if and only if* $b = b'$.

We say that a reverse trapdoor hashing scheme $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is function-private if there exists a PPT simulator* Sim*, such that for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left|\frac{1}{2} - \Pr\left[\mathsf{InpExp}_{\mathcal{A},\mathsf{Sim}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

6.2 Laconic Function Evaluation with Pre-Encoding

We now define a new variant of LFE. We ask that the encoding procedure is split into two parts: First, the encoder sends a function-independent pre-encoding of the input, then, once the hash of the function is revealed, it sends an input-independent post-encoding. We also require that the function digest and the post-encoding have a particular structure: the former consists of a matrix, the latter consists of a *LWE-like* sample. Finally, we ask that the output of the decoding procedure is produced by rounding the sum between the post-encoding and a (possibly non-linear) function of the pre-encoding. The reader may have already noticed that these properties are satisfied by many LFE schemes studied in the literature [QWW18, HLL23, DHM+24, Wee24].

Definition 6.5 (Laconic Function Evaluation with Pre-Encoding)**.** *Let* $\mathcal{F} := (\mathcal{F}_\lambda)_{\lambda \in \mathbb{N}}$ *be a family of functions, an LFE scheme consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

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

$\mathsf{Hash}(\mathsf{pk}, f)$**:** *The hashing algorithm takes as input a public key* pk *and the description of a function* $f \in \mathcal{F}_\lambda$ *with* $f : \{0,1\}^m \to \{0,1\}$*. The output is a digest* $\mathbf{A}_f \in \mathbb{Z}_q^{k \times (k \cdot \log q)}$ *and hasher's private information* ψ.

$\mathsf{Enc}(\mathsf{pk}, \mathbf{A}_f, \mathbf{x})$**:** *The encoding algorithm takes as input a public key* pk *a hash* $\mathbf{A}_f$*, and an input string* $\mathbf{x} \in \{0,1\}^m$*. The algorithm is divided into two subroutines.*

$\mathsf{PreEnc}(\mathsf{pk}, \mathbf{x})$**:** *The pre-encoding algorithm is independent of the hash: it takes as input a public key* pk*, an input* $\mathbf{x}$ *(and sometimes a vector* $\mathbf{r} \in \mathbb{Z}_q^{k-1}$*). It returns a pre-encoding of the input* E*, along with a private information* $\mathbf{s} \in \mathbb{Z}_q^k$.

$\mathsf{PostEnc}(\mathsf{pk}, \mathbf{A}_f, \mathbf{s})$**:** *The post-encoding algorithm does not depend on the input and returns a post-encoding defined as:*

$$c := \mathbf{s}^\intercal \cdot \mathbf{A}_f \cdot \mathbf{t} + e \in \mathbb{Z}_p$$

where $e \xleftarrow{\$} \chi(1^\lambda)$ *and* $\mathbf{t}$ *is sampled over* $\mathbb{Z}_2^{k \cdot \log q}$ *(not necessarily at random).*

The algorithm outputs an encoding $(E, c, \mathbf{t})$.

$\mathsf{Dec}(\mathsf{pk}, (E, c, \mathbf{t}), \psi)$**:** *The decoding algorithm takes as input a public key* pk*, an encoding* $(E, c, \mathbf{t})$ *and hasher's private information* ϕ*. The output is a bit*

$$\lceil \mathsf{Eval}(E, \psi) \cdot \mathbf{t} + c \rfloor_2$$

where Eval *is a polynomial-time deterministic algorithm that returns a vector in* $\mathbb{Z}_q^{k \cdot \log q}$*.*

We define a particular version of correctness that applies in most scheme as in Definition 6.5

Definition 6.6 (Special correctness)**.** *Let* $\widetilde{B} := \widetilde{B}(\lambda)$ *be a function of the security parameter. An LFE scheme with pre-encoding satisfies* $\widetilde{B}(\lambda)$*-special correctness if, for every* $\lambda \in \mathbb{N}$*, function* $f \in \mathcal{F}_\lambda$ *and input* $\mathbf{x} \in \mathbb{Z}_2^m$*, we have*

$$\mathsf{Eval}(E, \psi) = \mathbf{s}^\intercal \cdot \mathbf{A}_f + \mathbf{r}^\intercal \cdot f(\mathbf{x}) \cdot \mathbf{G} + \widetilde{\mathbf{e}}^\intercal$$

where $\mathbf{G} = \mathbf{I}_k \otimes \mathbf{g}_q$*,* $\widetilde{\mathbf{e}}$ *is such that* $\|\widetilde{\mathbf{e}}\|_\infty \leq \widetilde{B}(\lambda)$ *and the last entry of* $\mathbf{r}$ *is* -1*.*

We recall the notion of hasher privacy.

Definition 6.7 (Hasher privacy)**.** *Consider the following experiment* $\mathsf{HashExp}_\mathcal{A}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*:*

- *Sample a public key* $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*.*
- *Activate the adversary* $(f_0, f_1, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$*.*
- *Sample a random bit* $b \xleftarrow{\$} \{0,1\}$*.*
- *Compute* $(d, \rho) \xleftarrow{\$} \mathsf{Hash}(\mathsf{pk}, f_b)$*.*
- *Compute* $b' \leftarrow \mathcal{A}_1(\mathsf{pk}, d, \mathsf{aux})$*.*
- *Return* 1 *if and only if* $b = b'$*.*

We say that an LFE scheme with pre-encoding $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{PreEnc}, \mathsf{PostEnc}, \mathsf{Dec})$ *is hasher-private if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left| \frac{1}{2} - \Pr\left[\mathsf{HashExp}_\mathcal{A}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

If the above property holds for every adversary (even computationally unbounded ones) we say that the scheme is statistically hasher private.

Finally we define a slightly weaker version of the standard encoder privacy, which however suffices for our purposes. Namely, we only require security against a distinguisher that sees the pre-encoding information (and we do not pose any requirement on the post-encoding).

Definition 6.8 (Pre-Encoding privacy)**.** *Consider the following experiment* $\mathsf{PreEncExp}_{\mathcal{A},\mathsf{Sim}}(1^\lambda)$ *parametrized by an adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *and a simulator* Sim*:*

- *Sample a public key* $\mathsf{pk} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$*.*
- *Activate the adversary* $(\mathbf{x}, \mathsf{aux}) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$*.*
- *Sample a random bit* $b \xleftarrow{\$} \{0,1\}$*.*

- *If* $b = 0$*, compute* $(E_0, \phi_0) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathbf{x})$*. Otherwise, compute* $E_1 \xleftarrow{\$} \mathsf{Sim}(1^\lambda, \mathsf{pk})$*.*
- *Compute* $b' \leftarrow \mathcal{A}_1(\mathsf{pk}, E_b, \mathsf{aux})$*.*
- *Return* 1 *if and only if* $b = b'$*.*

We say that an LFE scheme with pre-encoding $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{PreEnc}, \mathsf{PostEnc}, \mathsf{Dec})$ *is pre-encoder private if for every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $\lambda \in \mathbb{N}$*, we have that:*

$$\left| \frac{1}{2} - \Pr\left[\mathsf{PreEncExp}_{\mathcal{A}}(1^\lambda) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

Most known LFE schemes satisfy (or can be adapted to satisfy) the above syntactical requirements. We summarize the state of the art in the following:

- Assuming the hardness of LWE, there exists an LFE scheme with pre-encoding for all bounded-depth circuits [QWW18, Appendix E].
- Assuming the hardness of small-secret circular LWE, there exists an LFE scheme with pre-encoding for all (no bound on the depth) circuits [HLL23].
- Assuming the hardness of Ring-LWE (small-secret circular Ring-LWE, resp.) there exists an LFE scheme with pre-encoding for all bounded-depth (unbounded-depth, resp.) RAM programs [DHM+24].

For our purposes, the details of these constructions will be irrelevant, provided that they satisfy the above syntax. Henceforth, we just assume that such an LFE exists, with the understanding that the exact efficiency guarantees of our construction will depend on the building block used to instantiate it.

6.3 Construction

In the following we define our construction for a reverse TDH for functions

$$f : \{0,1\}^m \to \{0,1\}^\ell.$$

We will use the following ingredients instantiating them over the ring $\mathbb{Z}_q$:

- The α-correct, succinct MOLE protocol (Protocol 3.4). $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{HashEval}, \mathsf{EncEval})$
- An LFE scheme with pre-encoding. $(\mathsf{Setup}, \mathsf{Hash}, \mathsf{PreEnc}, \mathsf{PostEnc}, \mathsf{Dec})$. Suppose that the scheme satisfies $\widetilde{B}$-special correctness.
- A pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \mathbb{Z}_q^\ell$ that can be evaluated uniformly (e.g., instantiated via a pseudorandom function).

For convenience, we assume that q is even. Moreover, we assume that $\alpha, \widetilde{B} \leq q \cdot \mathsf{negl}(\lambda)$. The protocol description is presented below.

Construction 6.9. Reverse Trapdoor Hash

$\mathsf{Setup}(1^\lambda)$**:** Sample keys

$$\mathsf{pk} \xleftarrow{\$} \mathsf{LFE.Setup}(1^\lambda), \qquad \mathsf{mpk} \xleftarrow{\$} \mathsf{MOLE.Setup}(1^\lambda).$$

Then, sample $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$ and output $\mathsf{hk} := (\mathsf{pk}, \mathsf{mpk}, \mathsf{seed})$.

$\mathsf{Gen}(\mathsf{hk},\mathbf{x})$**:** Compute

$$(E,\mathbf{s}) \xleftarrow{\$} \mathsf{LFE.PreEnc}(\mathsf{pk},\mathbf{x}), \qquad (E',\phi) \xleftarrow{\$} \mathsf{MOLE.Enc}(\mathsf{mpk},\mathbf{s})$$

Output $\mathsf{ek} := (E, E')$ and $\mathsf{td} := \phi$.

$\mathsf{Hash}(\mathsf{hk}, f)$**:** Let $\mathbf{u}$ be the last element of the standard basis over $\mathbb{Z}_q^k$. Let $(f_0,\dots,f_{\ell-1})$ be the functions that compute the i-th bit of the output of f. For every $i \in [\ell]$ compute

$$(\mathbf{A}_{f_i}, \psi_i) \xleftarrow{\$} \mathsf{LFE.Hash}(\mathsf{pk}, f_i).$$

Finally, compute

$$\begin{aligned}\mathbf{A} &\leftarrow \left(\mathbf{A}_{f_0}\cdot \mathbf{G}^{-1}\left(-\frac{q}{2}\cdot\mathbf{u}\right) \Big\| \dots \Big\| \mathbf{A}_{f_{\ell-1}}\cdot \mathbf{G}^{-1}\left(-\frac{q}{2}\cdot\mathbf{u}\right)\right)\\ (d,\psi) &\xleftarrow{\$} \mathsf{MOLE.Hash}(\mathsf{mpk}, \mathbf{A}^\intercal)\end{aligned}$$

and output d and $\rho := (\psi, \psi_0, \dots, \psi_{\ell-1})$.

$\mathsf{Enc}(\mathsf{hk},\mathsf{ek}, f, \rho)$**:** Let $\mathbf{u}$ be the last element of the standard basis over $\mathbb{Z}_q^k$. Compute

$$\mathbf{v} \leftarrow \mathsf{MOLE.HashEval}(\mathsf{mpk}, E', \psi).$$

Parse $\mathbf{v}$ as the vertical concatenation of $(v_0,\dots,v_{\ell-1})$ and let $(r_0,\dots,r_{\ell-1}) \leftarrow \mathsf{PRG}(\mathsf{seed})$. For every $i \in [\ell]$, compute

$$e_i \leftarrow \lceil r_i + \mathsf{LFE.Eval}(E,\psi_i)\cdot \mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - v_i \rfloor_2$$

Finally, output $\mathbf{e} := (e_0,\dots,e_{\ell-1})$.

$\mathsf{Dec}(\mathsf{hk}, d, \mathsf{td})$**:** Compute

$$\mathbf{w} \leftarrow \mathsf{MOLE.EncEval}(\mathsf{mpk}, d, \phi).$$

Parse $\mathbf{w}$ as the vertical concatenation of $(w_0,\dots,w_{\ell-1})$ and derive $(r_0,\dots,r_{\ell-1}) \leftarrow \mathsf{PRG}(\mathsf{seed})$. Then, for all $i \in [\ell]$, compute

$$e_i' \leftarrow \lceil -r_i - w_i \rfloor_2$$

Finally, output $\mathbf{e}' := (e_0',\dots,e_{\ell-1}')$.

Correctness. For correctness, observe that for all $i \in [\ell]$ we have that:

$$\begin{aligned}&\underbrace{r_i + \mathsf{LFE.Eval}(E,\psi_i)\cdot \mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - v_i}_{s_0} + \underbrace{-r_i - v_i'}_{s_1}\\ &= \mathsf{LFE.Eval}(E,\psi_i)\cdot \mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - (v_i + v_i')\\ &= (\mathbf{s}^\intercal\cdot\mathbf{A}_{f_i} + \mathbf{r}^\intercal\cdot f_i(\mathbf{x})\cdot\mathbf{G} + \widetilde{\mathbf{e}}^\intercal)\cdot \mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - \mathbf{s}^\intercal\cdot\mathbf{A}_{f_i}\cdot\mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - e_i\\ &= -q/2\cdot f_i(\mathbf{x})\cdot\mathbf{r}^\intercal\cdot\mathbf{u} + \widetilde{\mathbf{e}}^\intercal\cdot\mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - e_i\\ &= q/2\cdot f_i(\mathbf{x}) + \hat{e}_i\end{aligned}$$

where $\hat{e}_i := \widetilde{\mathbf{e}}^\intercal\cdot\mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - e_i$ and the second equality follows from the α-correctness of the MOLE and the special correctness of the LFE. Notice that $|\widetilde{\mathbf{e}}^\intercal\cdot\mathbf{G}^{-1}(-q/2\cdot\mathbf{u}) - e_i| \le \alpha(\lambda) + \widetilde{B}(\lambda)$.

Thus we have that s_0 and s_1 is a noisy secret sharing of $f_i(\mathbf{x})$. Furthermore, it holds that

$$f_i(\mathbf{x}) = \lceil s_0 + s_1 \rfloor_2 = \lceil s_0 \rfloor_2 \oplus \lceil s_1 \rfloor_2$$

except if $s_0 \in [q/4 - |\hat{e}_i|, q/4 + |\hat{e}_i|] \cup [3q/4 - |\hat{e}_i|, 3q/4 + |\hat{e}_i|]$. Note that the size of the interval is at most $4 \cdot |\hat{e}_i|$, which is a negligible fraction of q. Thus, by the pseudorandomness of PRG, this event happens with negligible probability, concluding our proof of correctness.

Security. We prove security in the following.

Theorem 6.10. *Suppose that* MOLE *is encoder private. If* LFE *is pre-encoding secure, Construction 6.9 is input private. Finally, if* LFE *is (statistically) function private, the reverse tradpoor hashing scheme is (statistically) function private.*

Proof. It is easy to see that the construction is (statistically) function private if LFE is (statistically) function private.

As for input privacy, we proceed by a series of indistinguishable hybrids.

- Hybrid $\mathcal{H}_0$: This hybrid corresponds to the original game: We provide the adversary $\mathcal{A}_1$ with an encoding key (E, E') generated using $\mathsf{Gen}(\mathsf{hk}, \mathbf{x})$.
- Hybrid $\mathcal{H}_1$: In this hybrid, we generate E' using $\mathsf{MOLE.Sim}(1^\lambda, \mathsf{mpk})$. The rest remains as in the previous hybrid.

  Hybrid $\mathcal{H}_0$ and Hybrid $\mathcal{H}_1$ are computationally indistinguishable thanks to the encoder privacy of the MOLE.
- Hybrid $\mathcal{H}_2$: In this hybrid, we generate E using $\mathsf{LFE.Sim}(1^\lambda, \mathsf{pk})$. The rest remains as in the previous hybrid.

  Hybrid $\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable under the input privacy of LFE.

Notice that in Hybrid $\mathcal{H}_2$, the pair (E, E') provided to the adversary contains no information about $\mathbf{x}$. From this, we can easily derive a simulator. This ends the proof. □

7 A Counterexample for Adaptive LWE

We present a counterexample to a conjecture from [QWW18]. The conjecture (adaptive LWE) says that the probability of that any polynomial-time attacker wins the following experiment is negligibly close to 1/2.

- The challenger samples random matrices $\{\mathbf{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n \times m}\}_{i \in [k]}$ and sends them to the attacker.
- The attacker chooses an $x \in \{0,1\}^{k-1}$ and sends it to the challenger. Let $\hat{x} := (x, 0)$.
- The challenger samples an $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and a bit $b \xleftarrow{\$} \{0,1\}$.
- If $b = 0$ it computes:
  $$\{\mathbf{b}_i := \mathbf{s}^\intercal(\mathbf{A}_i - \hat{x}_i \cdot \mathbf{G}) + \mathbf{e}_i^\intercal\}_{i \in [k]}$$
  where $\mathbf{e}_i \xleftarrow{\$} \chi(\lambda)^m$.
- If $b = 1$ it samples $\{\mathbf{b}_i \xleftarrow{\$} \mathbb{Z}_q^m\}_{i \in [k]}$.

- The attacker wins if, given $\{\mathbf{b}_i\}_{i\in[k]}$, it correctly guesses b.

We show that, if k is allowed to grow independently of the LWE parameters (n, q, χ), the assumption is false. This roughly corresponds to the *optimistic parameter* settings proposed in [QWW18]. We sketch the attack in the following. We assume for convenience that q is a power of 2, but the attack can be adapted to other moduli.

Let $\mathbf{A} := (\mathbf{A}_0 \parallel \ldots \parallel \mathbf{A}_{k-1})$. By the homomorphic properties of the lattice encodings, we know that there exists a matrix $\mathbf{H}$ with $\|\mathbf{H}\|_\infty = 1$ such that, for any matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times n}$, if $\mathbf{x} := \mathsf{Bits}(\mathbf{M})$, we have that:

$$(\mathbf{s}^\intercal(\mathbf{A} - \mathbf{x} \otimes \mathbf{G}) + \mathbf{e}^\intercal) \cdot \mathbf{H} = \mathbf{s}^\intercal(\mathbf{A} \cdot \mathbf{H} - \mathbf{M}) + \mathbf{e}^\intercal \cdot \mathbf{H} \approx \mathbf{s}^\intercal(\mathbf{A} \cdot \mathbf{H} - \mathbf{M})$$

since $\|\mathbf{e}^\intercal\mathbf{H}\|_\infty \approx 0$. To recover the i-th most significant bit of (each component of) $\mathbf{s}$, it is sufficient to set $\mathbf{M}_i := \mathbf{A}\cdot\mathbf{H} - 2^i \cdot \mathbf{I}_n$ and $\mathbf{x}_i := \mathsf{Bits}(\mathbf{M}_i)$. Then compute the (component-wise) most significant bit of:

$$(\mathbf{s}^\intercal(\mathbf{A} - \mathbf{x}_i \otimes \mathbf{G}) + \mathbf{e}^\intercal) \cdot \mathbf{H} \approx \mathbf{s}^\intercal(\mathbf{A} \cdot \mathbf{H} - \mathbf{M}_i) = 2^i \cdot \mathbf{s}^\intercal.$$

We can then recover the entire secret key $\mathbf{s}$, by setting the input $\mathbf{x}$ to be the concatenation of $(\mathbf{x}_1, \ldots, \mathbf{x}_{\log q})$. This shows that the conjecture is false if k is allowed to be arbitrarily bigger than n. Notice in the provable parameter setting (where the encodings are secure under the *subexponential* hardness of LWE), our attack fails as the bound on k is too small to encode $\mathbf{M}_i := \mathbf{A} \cdot \mathbf{H} - 2^i \cdot \mathbf{I}_n$.

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

[ARS24] Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 301–330, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 790–820, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[Bar86] David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 1–5, 1986.

[BBD] Rishabh Bhadauria, Pedro Branco, and Nico Döttling. Rate-1 registration-based encryption and laconic oblivious transfer. (Personal Communication).

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[BJSS25] Elette Boyle, Abhishek Jain, Sacha Servan-Schreiber, and Akshayaram Srinivasan. Simultaneous-message and succinct secure computation. In *EUROCRYPT 2025*, 2025.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 264–302, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014: 5th Conference on Innovations in Theoretical Computer Science*, pages 1–12, Princeton, NJ, USA, January 12–14, 2014. Association for Computing Machinery.

[CDG$^+$17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[DGI$^+$19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[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, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 107–142, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th Annual Symposium on Foundations of Computer Science*, pages 415–434, Santa Cruz, CA, USA, November 6–9, 2023. IEEE Computer Society Press.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st Annual ACM Symposium on Theory of Computing*, pages 12–24, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 859–870, Paris, France, October 7–9, 2018. IEEE Computer Society Press.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology –*

CRYPTO 2021, Part III, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\mathrm{depth}, \lambda)$-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 178–209, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.