Adaptively Secure Laconic Function Evaluation for NC^1

Shweta Agrawal[1] and Răzvan Roşie[2]

[1] Indian Institute of Technology, Madras, India `shweta.a@cse.iitm.ac.in`
[2] University of Luxembourg, Luxembourg
`razvan.rosie@uni.lu`

Abstract. Laconic Function Evaluation (LFE) is a novel primitive introduced by Quach, Wee and Wichs (FOCS '18) that allows two parties to perform function evaluation *laconically* in the following manner: first, Alice sends a compressed "digest" of some function – say $\mathscr{C}$ – to Bob. Second, Bob constructs a ciphertext for his input M given the digest. Third, Alice, after receiving the ciphertext from Bob and having access to her circuit, can recover $\mathscr{C}(M)$ and (ideally) nothing more about Bob's message. The protocol is said to be *laconic* if the sizes of the digest and ciphertext are much smaller than the circuit size $|\mathscr{C}|$.

Quach, Wee and Wichs provided a construction of laconic function evaluation for general circuits under the learning with errors (LWE) assumption (with subexponential approximation factors), where all parameters grow polynomially with the depth but not the size of the circuit. Under LWE, their construction achieves the restricted notion of *selective* security where Bob's input M must be chosen non-adaptively before even the CRS is known.

In this work, we provide the first construction of LFE for NC^1, which satisfies *adaptive* security from the ring learning with errors assumption (with polynomial approximation factors). The construction is based on the functional encryption scheme by Agrawal and Rosen (TCC 2017). Using the compiler from Quach, Wee and Wichs which bootstraps LFE to succinct, single key FE generically, we obtain a significant *simplification* of the succinct single key FE of Agrawal and Rosen.

Keywords: functional encryption, laconic function evaluation, laconic oblivious transfer.

1 Introduction

Laconic Function Evaluation (LFE) is a novel primitive that was introduced in a recent work by Quach, Wee and Wichs [11]. Intuitively, the notion proposes a setting where two parties want to evaluate a function $\mathscr{C}$ in the following manner: Alice computes a "digest" of the circuit representation of $\mathscr{C}$ and sends this digest to Bob; Bob computes a ciphertext CT for his input M using the received digest and sends this back to Alice; finally Alice is able to compute the value $\mathscr{C}(M)$ in the clear without learning anything else about Bob's input.

The work of [11] provided an elegant construction of laconic FE for all circuits from LWE. In their construction, the size of the digest, the complexity of the encryption algorithm and the size of the ciphertext only scale with the depth but not the size of the circuit. However, under LWE, their construction only achieves the restricted notion of *selective* security where Bob's input M must be chosen non-adaptively before even the CRS is known. Moreover they must rely on LWE with subexponential modulus to noise ratio, which implies that the underlying lattice problem must be hard for subexponential approximation factors. Achieving adaptive security from LWE was left as an explicit open problem in their work.

In this work, we provide a new construction of LFE for NC^1 circuits that achieves adaptive security. Our construction relies on the ring learning with errors (RLWE) assumption with polynomial modulus to noise ratio. We start with the construction by Agrawal and Rosen for functional encryption for NC^1 circuits [4] and *simplify* it to achieve LFE. To achieve a laconic digest, we make use of the *laconic* oblivious transfer (LOT) primitive [7,12]. Our main result may be summarized as follows.

Theorem 1 (LFE for NC^1 – Informal). *Assuming the hardness of* RLWE *with polynomial approximation factors, there exists an adaptive-secure* LFE *protocol for* NC^1*.*

We also reduce the size of the digests using laconic oblivious transfer as in [11].

The authors of [11] also studied the relations between LFE and other primitives, and in particular showed that LFE implies succinct functional encryption (FE) [5, 10]. Functional encryption is a generalisation of public key encryption in which the secret key corresponds to a function, say f, rather than a user. The ciphertext corresponds to an input $\mathbf{x}$ from the domain of f. Decryption allows to recover $f(\mathbf{x})$ and nothing else. Succinctness means that the size of the ciphertext in the scheme depends only on the depth of the supported circuit rather than on its size [2, 4, 8]. We may apply the LFE to FE compiler of [11] to our new, adaptively secure LFE for NC^1, thus obtaining a much *simpler* construction of adaptively secure, succinct FE for NC^1. We believe our construction of succinct FE significantly *simplifies* the original construction of Agrawal and Rosen [4] which is quite complex.

Technical Overview. The techniques that we use exploit the algebraic structure in the construction by Agrawal and Rosen [4]. Say that the plaintext $M \in \{0,1\}^k$. If the encryption algorithm obtains a ciphertext $\mathsf{CT} := \{C_1, \ldots, C_k\}$ where each C_i encrypts a single bit M_i of plaintext, in isolation from all different M_j, we say that the ciphertext is decomposable.

The construction supports circuits of depth d and uses a tower of moduli $p_0 < p_1 < \ldots < p_d$. Building upon a particular levelled fully homomorphic encryption scheme (FHE) [6], it encrypts each bit of the plaintext, independently, as follows:

– first multiplication level – the ciphertext consists of a "Regev encoding":

$$C^1 \leftarrow a \cdot s + p_0 \cdot e + M_i \quad \in \ \mathcal{R}_{p_1}$$

where $M_i \in \mathcal{R}_{p_0}$ and $s \leftarrow_{\$} \mathcal{R}_{p_1}$ is a fixed secret term; $a \leftarrow_{\$} \mathcal{R}_{p_1}$ is provided through public parameters, while $e \leftarrow_{\chi} \mathcal{R}_{p_1}$ is the noise;

– next multiplication level – two ciphertexts are provided under the same s:

$$a' \cdot s + p_1 \cdot e' + C^1 \ \in \ \mathcal{R}_{p_2} \qquad \text{and} \qquad a'' \cdot s + p_1 \cdot e'' + (C^1 \cdot s) \ \in \ \mathcal{R}_{p_2} .$$

The computational pattern is repeated recursively up to d multiplication levels, and then for every bit of the input.

– an addition layer is interleaved between any two multiplication layers C^i and C^{i+1}: essentially it "replicates" ciphertexts in C^i and uses its modulus p_i.

– the public key (the "$\vec{a}$"s) corresponding to the last layer are also the master public key mpk of a linear functional encryption scheme $\mathsf{Lin\text{-}FE}$ [1,3].

– the decryption algorithm computes $\mathscr{C}$ obliviously over the ciphertext, layer by layer, finally obtaining:

$$\mathsf{CT}_{\mathscr{C}(M)} \leftarrow \mathscr{C}(M) \ + \ \mathsf{Noise} \ + \ \mathsf{PK}_{\mathscr{C}} \cdot s \qquad (1)$$

where $\mathsf{PK}_{\mathscr{C}}$ is a "functional public key" that depends only on the public key and the function $\mathscr{C}$, and can be viewed as a succinct representation of $\mathscr{C}$. The term $\mathsf{PK}_{\mathscr{C}} \cdot s$ occurring in (1) will be removed using the functional key $\mathsf{sk}_{\mathscr{C}}$, in order to recover $\mathscr{C}(M)$ plus noise (which can be modded out).

The crux idea when building an LFE is to set the ciphertext to be essentially the data-dependent part of the FE ciphertext, while the digest to be $\mathsf{PK}_{\mathscr{C}}$. Concretely, the mpk will form the crs. Whenever a circuit $\mathscr{C}$ is to be compressed, a circuit-dependent public value – denoted $\mathsf{PK}_{\mathscr{C}}$ – is returned as *digest*. The receiver samples its own secret s and computes recursively the Regev encodings (seen above) as well as $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \mathsf{Noise}$ (the $\mathsf{sk}_{\mathscr{C}}$-dependent term), which suffice to recover $\mathscr{C}(M)$ from (1) on the sender's side.

Our construction departs significantly from the approach in [11]. In short, the original work makes use of generic transforms for obtaining attribute-based LFEs ($\mathsf{AB\text{-}LFE}$s), obtaining $\mathsf{AB\text{-}LFE}$s supporting multi-bit outputs, using the transform of [8] to hide the "attribute" in $\mathsf{AB\text{-}LFE}$ and compressing the digests via laconic oblivious transfer [7]. On the other hand, we do not need to go via $\mathsf{AB\text{-}LFE}$, making our transformation simpler. Moreover, by relying on the adaptive security of the underlying FE scheme that we use, we obtain adaptive security.

Towards Short Digests. As in [11], we use the *laconic oblivious transfer* protocol in the following way: after getting the digest in the form of $\mathsf{PK}_{\mathscr{C}}$, we apply a second compression round, which yields:

$$(\mathsf{digest}_{\mathsf{LOT}}, \hat{\mathbf{D}}) \leftarrow \mathsf{LOT.Compress}(\mathsf{crs}_{\mathsf{LOT}}, \mathsf{PK}_{\mathscr{C}}) \ .$$

We stress that both compression methods used are deterministic. Put differently, at any point in time, the sender, in full knowledge of her circuit representation, can recreate the digest. On the receiver's side, instead of following the technique proposed in [11] – garbling an entire FHE decryption circuit and encrypting under an ABE the homomorphic ciphertext and the labels – we garble only the circuits that provides

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \mathsf{Noise}\ ,$$

while leaving the actual ciphertext intact. The advantage of such an approach resides into its conceptual simplicity, as the size of the *core* ciphertext Bob sends back to Alice remains manageable for simple $\mathscr{C}$, and is not suppressed by the size of the garbling scheme.

Paper Organization. In Section 2, we introduce the standard notations to be adopted throughout the paper, followed by the definitions of the primitives that we use as building blocks. Section 3 reviews the decomposable FE scheme proposed by Agrawal and Rosen in [4]. In Section 4, we introduce a new LFE scheme for NC^1 circuits, and show in Appendix A how to combine it with laconic oblivious transfer in order to achieve a scheme with laconic digests. Appendix B recalls the LFE to single-key succinct FE compiler.

2 Preliminaries

Notations. We denote the security parameter by $\lambda \in \mathbb{N}^*$ and we assume it is implicitly given to all algorithms in the unary representation 1^λ. An algorithm is equivalent to a Turing machine. Algorithms are assumed to be randomized unless stated otherwise; PPT stands for "probabilistic polynomial-time" in the security parameter (rather than the total length of its inputs). Given a randomized algorithm $\mathcal{A}$ we write the action of running $\mathcal{A}$ on input(s) $(1^\lambda, x_1, \dots)$ with uniform random coins r and assigning the output(s) to $(y_1, \dots)$ by $(y_1, \dots) \leftarrow_{\$} \mathcal{A}(1^\lambda, x_1, \dots; r)$. When $\mathcal{A}$ is given oracle access to some procedure $\mathcal{O}$, we write $\mathcal{A}^{\mathcal{O}}$. For a finite set S, we denote its cardinality by $|S|$ and the action of sampling an element x uniformly at random from X by $x \leftarrow_{\$} X$. We let bold variables such as $\mathbf{w}$ represent column vectors. Similarly, bold capitals stand for matrices (e.g. $\mathbf{A}$). A subscript $\mathbf{A}_{i,j}$ indicates an entry in the matrix. We overload notation for the power function and write α^u to denote that variable α is associated to some level u in a leveled construction. For any variable $k \in \mathbb{N}^*$, we define $[k] \coloneqq \{1, \dots, k\}$. A real-valued function $\textsc{Negl}(\lambda)$ is negligible if $\textsc{Negl}(\lambda) \in \mathcal{O}(\lambda^{-\omega(1)})$. We denote the set of all negligible functions by $\textsc{Negl}$. Throughout the paper $\perp$ stands for a special error symbol. We use $||$ to denote concatenation. For completeness, we recall standard algorithmic and cryptographic primitives to be used.

2.1 Computational Hardness Assumptions

The Learning With Error (LWE) search problem [13] asks to find the secret vector $\mathbf{s}$ over $\mathbb{F}_q^\ell$ given a polynomial-size set $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$, where $\mathbf{A}$ denotes a randomly sampled matrix over $\mathbb{F}_q^{k\times\ell}$, while $\mathbf{e} \in \mathbb{F}^k$ is a small error term sampled from an appropriate distribution χ. Roughly, the decision version of the problem, asks to distinguish between the aforementioned distribution as opposed to the uniform one.

Definition 1 (Learning With Errors). *The advantage of any* PPT *adversary* Adv *in distinguishing between the following two distributions is negligible:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{LWE}}(\lambda) \coloneqq \Pr[1 \leftarrow \mathsf{Adv}(1^\lambda, \mathbf{A}, \mathbf{u})] - \Pr[1 \leftarrow \mathsf{Adv}(1^\lambda, \mathbf{A}, \mathbf{A}\cdot\mathbf{s}+\mathbf{e})] \in \textsc{Negl}(\lambda)$$

where $\mathbf{s} \leftarrow_{\$} \mathbb{Z}_q^\ell$, $\mathbf{A} \leftarrow_{\$} \mathbb{Z}_q^{k\times\ell}$, $\mathbf{e} \leftarrow_{\chi} \mathbb{Z}_q^k$ *and* $\mathbf{u} \leftarrow_{\$} \mathbb{Z}_q^k$.

Later, Lyubashevsky, Peikert and Regev [9] proposed a ring version. Let $\mathcal{R} \coloneqq \mathbb{Z}[X]/(X^n+1)$ for n a power of 2, while $\mathcal{R}_q \coloneqq \mathcal{R}/q\mathcal{R}$ for a safe prime q satisfying $q \equiv 1 \bmod 2n$.

Definition 2 (Ring LWE). *For* $s \leftarrow_{\$} \mathcal{R}_q$, *given a polynomial number of samples that are either: (1) all of the form* $(a, a\cdot s+e)$ *for some* $a \leftarrow_{\$} \mathcal{R}_q$ *and* $e \leftarrow_{\chi} \mathcal{R}_q$ *or (2) all uniformly sampled over* $\mathcal{R}_q^2$*; the (decision)* $\mathsf{RLWE}_{q,\phi,\chi}$ *states that a* PPT*-bounded adversary can distinguish between the two settings only with negligible advantage.*

2.2 Standard Primitives

Garbling Schemes. Garbled circuits were introduced by Yao in 1982 [14, 15] to solve the famous "Millionaires' Problem". Since then, garbled circuits became a standard building-block for many cryptographic primitives. Their definition follows.

Definition 3 (Garbling Scheme). *Let* $\{\mathscr{C}_k\}_\lambda$ *be a family of circuits taking as input* k *bits. A garbling scheme is a tuple of* PPT *algorithms* $(\mathsf{Garble}, \mathsf{Enc}, \mathsf{Eval})$ *such that:*

– $(\mathit{\Gamma}, \mathsf{sk}) \leftarrow_{\$} \mathsf{Garble}(1^\lambda, \mathscr{C})$*: takes as input the unary representation of the security parameter and a circuit* $\mathscr{C} \in \{\mathscr{C}_k\}_\lambda$ *and outputs a garbled circuit* $\mathit{\Gamma}$ *and a secret key* sk.

– $c \leftarrow_{\$} \mathsf{Enc}(\mathsf{sk}, x)$*: given as input* $x \in \{0,1\}^k$ *and the secret key* sk*, the encryption procedure returns an encoding* c.

– $\mathscr{C}(x) \leftarrow \mathsf{Eval}(\mathit{\Gamma}, c)$*: the evaluation procedure receives as inputs a garbled circuit as well as an encoding of* x *returning* $\mathscr{C}(x)$.

We say that a garbling scheme $\mathit{\Gamma}$ *is* ***correct*** *if for all* $\mathscr{C} : \{0,1\}^k \to \{0,1\}^\ell$ *and for all* $x \in \{0,1\}^k$ *we have that:*

$$\Pr\left[\mathscr{C}(x) = y \;\middle|\; \begin{array}{l}(\mathit{\Gamma}, \mathsf{sk}) \leftarrow_{\$} \mathsf{GS.Garble}(1^\lambda, \mathscr{C}) \wedge \\ y \leftarrow \mathsf{GS.Eval}(\mathit{\Gamma}, \mathsf{GS.Enc}(\mathsf{sk}, x))\end{array}\right] = 1 \ .$$

Yao's Garbled Circuit [14]. An interesting type of garbled schemes is represented by the original proposal of Yao, which considers a family of circuits of k input wires and outputting a single bit. In his proposal, a circuit's secret key can be viewed as two labels (L_i^0, L_i^1) for each input wire, where $i \in [k]$. The evaluation of the circuit at point x corresponds to an evaluation of $\mathsf{Eval}(\mathit{\Gamma}, (L_1^{x_1}, \ldots, L_k^{x_k}))$, where x_i is the i^{th} bit of x — thus the encoding $c = (L_1^{x_1}, \ldots, L_k^{x_k})$.

Laconic Oblivious Transfer

Definition 4 (LOT). *A Laconic Oblivious Transfer (*LOT*) scheme consists of a tuple of* PPT *algorithms* $(\mathsf{crsGen}, \mathsf{Compress}, \mathsf{Enc}, \mathsf{Dec})$ *with the following functionality:*

– $\mathsf{crs} \leftarrow_{\$} \mathsf{crsGen}(1^\lambda, 1^k, 1^d)$*: takes as input the security parameter* λ *and outputs a common reference string* crs.
– $(\mathsf{digest}, \hat{\mathbf{D}}) \leftarrow_{\$} \mathsf{Compress}(\mathsf{crs}, \mathbf{D})$*: given a database* $\mathbf{D}$ *and the* crs*, outputs a digest of the circuit as well as a state of the database, denoted* digest *and* $\hat{\mathbf{D}}$ *respectively.*
– $\mathsf{CT} \leftarrow_{\$} \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}, \ell, M_0, M_1)$*: the randomized encryption algorithm takes as input the common reference string* crs*, the digest, an index position* ℓ*, and two messages; it returns a ciphertext* CT.
– $M \leftarrow \mathsf{Dec}(\mathsf{crs}, \mathsf{digest}, \hat{\mathbf{D}}, \mathsf{CT}, \ell)$*: the decryption algorithm takes as input* $\hat{\mathbf{D}}$*, an index location* ℓ*, the digest* digest *and the common reference string* crs*; it outputs a message* M*.*

We require an LOT *to satisfy the following properties:*

– ***Correctness****: for all* $(M_0, M_1) \in \mathcal{M} \times \mathcal{M}$*, for all* $\mathbf{D} \in \{0,1\}^k$ *and for any index* $\ell \in [k]$ *we have:*

$$\Pr\left[M = M_{\mathbf{D}[\ell]} \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow_{\$} \mathsf{LOT.crsGen}(1^\lambda, 1^k, 1^d) \wedge \\ (\mathsf{digest}, \hat{\mathbf{D}}) \leftarrow_{\$} \mathsf{LOT.Compress}(\mathsf{crs}, \mathbf{D}) \wedge \\ \mathsf{CT} \leftarrow_{\$} \mathsf{LOT.Enc}(\mathsf{crs}, \mathsf{digest}, \ell, M_0, M_1) \wedge \\ M \leftarrow \mathsf{LOT.Dec}(\mathsf{crs}, \mathsf{digest}, \hat{\mathbf{D}}, \mathsf{CT}, \ell) \end{array} \right] = 1 \ .$$

– ***Laconic Digest****: the length of the digest is a fixed polynomial in the security parameter* λ*, independent of the size of the database* $\mathbf{D}$*.*

– ***Sender Privacy against Semi-Honest Adversaries****: there exists a* PPT *simulator* $\mathcal{S}$ *such that for a correctly generated* crs *the following two distributions are computationally indistinguishable:*

$$\Big| \Pr\left[\mathcal{A}\big(\mathsf{crs}, \mathsf{LOT.Enc}(\mathsf{crs}, \mathsf{digest}, \ell, M_0, M_1)\big)\right] - \Pr\left[\mathcal{A}\big(\mathsf{crs}, \mathcal{S}(\mathsf{crs}, \mathbf{D}, \ell, M_{\mathbf{D}[\ell]})\big)\right] \Big| \in \textsc{Negl}(\lambda) \ .$$

Appendix C.1 describes the LOT construction put forth in [7].

2.3 Functional Encryption Scheme - Public-Key Setting

We define functional encryption [5] in the public key-settings and provide a simulation-based security definition. Roughly speaking, the semantic security of the functional encryption scheme guarantees the adversary cannot learn more on M as by knowing only $\mathscr{C}(M)$.

Definition 5 (Functional Encryption - Public Key Setting). *Let* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in N}$ *be an ensemble, where* $\mathcal{F}_\lambda$ *is a finite collections of functions* $\mathscr{C} : \mathcal{M}_\lambda \to Y_\lambda$*. A functional encryption scheme* FE *in the public-key setting consists of a tuple of* PPT *algorithms* (Setup, KeyGen, Enc, Dec) *such that:*

– $(\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{FE.Setup}(1^\lambda)$: *takes as input the unary representation of the security parameter* λ *and outputs a pair of master secret/public keys.*
– $\mathsf{sk}_{\mathscr{C}} \leftarrow_\$ \mathsf{FE.KeyGen}(\mathsf{msk}, \mathscr{C})$*: given the master secret key and a function* $\mathscr{C}$*, the (randomized) key-generation procedure outputs a corresponding* $\mathsf{sk}_{\mathscr{C}}$*.*
– $\mathsf{CT} \leftarrow_\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{M})$*: the randomized encryption procedure encrypts the plaintext* M *with respect to* mpk.
– $\mathsf{FE.Dec}(\mathsf{CT}, \mathsf{sk}_{\mathscr{C}})$*: decrypts the ciphertext* CT *using the functional key* $\mathsf{sk}_{\mathscr{C}}$ *in order to learn a valid message* $\mathscr{C}(\mathsf{M})$ *or a special symbol* $\perp$*, in case the decryption procedure fails.*

We say that FE *satisfies correctness if for all* $\mathscr{C} : \mathcal{M}_\lambda \to Y_\lambda$ *we have that:*

$$\Pr\left[y = \mathscr{C}(M) \;\middle|\; \begin{array}{l} (\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{FE.Setup}(1^\lambda) \wedge \\ \mathsf{sk}_{\mathscr{C}} \leftarrow_\$ \mathsf{FE.KeyGen}(\mathsf{msk}, \mathscr{C}) \wedge \\ \mathsf{CT} \leftarrow_\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{M}) \wedge \\ y \leftarrow \mathsf{FE.Dec}(\mathsf{CT}, \mathsf{sk}_{\mathscr{C}}) \end{array} \right] = 1 \; .$$

A public-key functional encryption scheme FE *is semantically secure if there exists a stateful* PPT *simulator* $\mathcal{S}$ *such that for any* PPT *adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}FE}}_{\mathcal{A},\mathsf{FE}}(\lambda) := \left| \Pr[\mathsf{FULL\text{-}SIM\text{-}FE}^{\mathcal{A}}_{\mathsf{FE}}(\lambda) = 1] - \frac{1}{2} \right|$$

is negligible, where the $\mathsf{FULL\text{-}SIM\text{-}FE}$ *experiment is described in Figure 1 (left).*

```
$\mathsf{FULL\text{-}SIM\text{-}FE}^{\mathcal{A}}_{\mathsf{FE}}(\lambda)$:
$b \leftarrow_\$ \{0,1\}$
$(\mathsf{msk}, \mathsf{mpk}) \leftarrow_\$ \mathsf{FE.Setup}(1^\lambda)$
$\mathscr{C} \leftarrow \mathcal{A}(\mathsf{mpk})$
$\mathsf{sk}_{\mathscr{C}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{msk}, \mathscr{C})$
$\mathsf{M}^* \leftarrow \mathcal{A}(\mathsf{mpk}, \mathsf{sk}_{\mathscr{C}})$
if $b = 0$:
  $\mathsf{CT}^* \leftarrow_\$ \mathsf{FE.Enc}(\mathsf{mpk}, \mathsf{M}^*)$
else:
  $\mathsf{CT}^* \leftarrow \mathcal{S}(\mathsf{mpk}, \mathsf{sk}_{\mathscr{C}}, \mathscr{C}, \mathscr{C}(\mathsf{M}^*))$
$b' \leftarrow_\$ \mathcal{A}(\mathsf{CT}^*)$
return $b' = b$
```

```
$\mathsf{FULL\text{-}SIM\text{-}LFE}^{\mathcal{A}}_{\mathsf{LFE}}(\lambda)$:
$b \leftarrow_\$ \{0,1\}$
$(1^k, 1^d) \leftarrow_\$ \mathcal{A}(1^\lambda)$
$\mathsf{crs} \leftarrow_\$ \mathsf{LFE.crsGen}(1^\lambda, 1^k, 1^d)$
$(M^*, \mathscr{C}) \leftarrow_\$ \mathcal{A}(\mathsf{crs})$
$\mathsf{digest}_{\mathscr{C}} \leftarrow_\$ \mathsf{LFE.Compress}(\mathsf{crs}, \mathscr{C})$
if $b = 0$:
  $\mathsf{CT}^* \leftarrow_\$ \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{digest}_{\mathscr{C}}, M^*)$
else
  $\mathsf{CT}^* \leftarrow_\$ \mathcal{S}(\mathsf{crs}, \mathscr{C}, \mathsf{digest}_{\mathscr{C}}, \mathscr{C}(M^*))$
$b' \leftarrow_\$ \mathcal{A}(\mathsf{CT}^*)$
return $b = b'$
```

Fig. 1. Left: $\mathsf{FULL\text{-}SIM\text{-}FE}$-security defined for a functional encryption scheme in the public-key setting. Right: the simulation security experiment $\mathsf{FULL\text{-}SIM\text{-}LFE}$ defined for a laconic function evaluation scheme LFE.

2.4 Laconic Function Evaluation

We described the motivation behind LFE in Section 1. We proceed with its definition from [11].

Definition 6 (Laconic Function Evaluation [11]). *A laconic function evaluation scheme* LFE *for a class of circuits* $\mathfrak{C}_\lambda$ *consists of four algorithms* (crsGen, $\mathsf{Compress}$, Enc, Dec)*:*

- $\mathsf{crs} \leftarrow_\$ \mathsf{LFE.crsGen}(1^\lambda, 1^k, 1^d)$*: assuming the input size and the depth of the circuit in the given class are* k *and* d*, a common reference string* crs *of appropriate length is generated. We assume that* crs *is implicitly given to all algorithms.*
- $\mathsf{digest}_\mathscr{C} \leftarrow_\$ \mathsf{LFE.Compress}(\mathsf{crs}, \mathscr{C})$*: the compression algorithm takes a description of the circuit* $\mathscr{C}$ *and produces a digest* $\mathsf{digest}_\mathscr{C}$*.*
- $\mathsf{CT} \leftarrow_\$ \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{digest}_\mathscr{C}, M)$*: takes as input the message* M *as well as the digest of* $\mathscr{C}$ *and produces a ciphertext* CT*.*
- $\mathsf{LFE.Dec}(\mathsf{crs}, \mathsf{CT}, \mathscr{C})$*: if the parameters are correctly generated, the decryption procedure recovers* $\mathscr{C}(M)$*, given the ciphertext encrypting* M *and circuit* $\mathscr{C}$ *or a special symbol* $\perp$*, in case the decryption procedure fails.*

We require the LFE *scheme to achieve the following properties:*

- ***Correctness*** *- for all* $\mathscr{C} : \{0,1\}^k \to \{0,1\}^\ell$ *of depth* d *and for all* $M \in \{0,1\}^k$ *we have:*

$$\Pr\left[y = \mathscr{C}(M) \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow_\$ \mathsf{LFE.crsGen}(1^\lambda, 1^k, 1^d) \wedge \\ \mathsf{digest}_\mathscr{C} \leftarrow_\$ \mathsf{LFE.Compress}(\mathsf{crs}, \mathscr{C}) \wedge \\ \mathsf{CT} \leftarrow_\$ \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{digest}_\mathscr{C}, M) \wedge \\ y \leftarrow \mathsf{LFE.Dec}(\mathsf{crs}, \mathscr{C}, \mathsf{CT}) \end{array} \right] = 1 \ .$$

- ***Security****: there exists a* PPT *simulator* $\mathcal{S}$ *such that for any stateful* PPT *adversary* $\mathcal{A}$ *we have:*

$$\mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}LFE}}_{\mathcal{A},\mathsf{LFE}}(\lambda) := \left| \Pr[\mathsf{FULL\text{-}SIM\text{-}LFE}^{\mathcal{A}}_{\mathsf{LFE}}(\lambda) = 1] - \frac{1}{2} \right|$$

is negligible, where $\mathsf{FULL\text{-}SIM\text{-}LFE}$ *is defined in Figure 1 (right side).*

- ***Laconic outputs****: As per [11, p13-14], we require the size of digest to be laconic* $|\mathsf{digest}_\mathscr{C}| \in O(poly(\lambda))$ *and we impose succinctness constraints for the sizes of the ciphertext and public parameters.*

3 Background on the AR17 FE Scheme

Overview. In this section, we recall a construction of functional encryption for circuits with logarithmic depth d in input length [4].

3.1 The AR17 Construction for NC^1

In [4], the authors provided a bounded key functional encryption scheme supporting general circuits. The first distinctive feature is represented by the supported class of functions, which are now described by arithmetic, rather than Boolean circuits. Second, the ciphertexts' size in their construction is succinct, as it grows with the depth of the circuit, rather than its size. Third, the ciphertext enjoys decomposability: assuming a plaintext is represented as a vector, each of its k elements gets encrypted independently. We describe the scheme for NC^1, and formalize it in Appendix C.2.

Regev Encodings. We commence by recalling a simple symmetric encryption scheme due to Brakerski and Vaikuntanathan [6]. Let "s" stand for an RLWE secret acting as a secret key, while a and e are the random mask and noise:

$$\begin{aligned} c_1 &\leftarrow a && \in \mathcal{R}_p \\ c_2 &\leftarrow a \cdot s + 2 \cdot e + M && \in \mathcal{R}_p \end{aligned} \tag{2}$$

Recovering the message M (a bit, in this case) is done by subtracting $c_1 \cdot s$ from c_2 and then removing the noise through the mod 2 operator. This plain scheme comes up with powerful homomorphic properties, and is generalized in [4] to recursively support levels of encodings. Henceforth, we use the name "Regev encoding" for the following map between rings $\mathcal{E}^i : \mathcal{R}_{p_{i-1}} \rightarrow \mathcal{R}_{p_i}$, where:

$$\mathcal{E}^i(M) = a^i \cdot s + p_{i-1} \cdot e^i + M \quad \in \mathcal{R}_{p_i} \ . \tag{3}$$

As a general notation, we write as an *superscript* of a variable the *level* to which it has been associated.

The NC^1 Construction. To be self-contained in the forthcoming parts, we give an informal specification of AR17's procedures.

Encryption. The encryption procedure samples a RLWE secret s, and computes a Regev encoding for each input $M_i \in \mathcal{R}_{p_0}$ independently. This step produces the following set $\left\{\mathcal{E}^1(M_i) \mid M_i \in \mathcal{R}_{p_0} \wedge i \in [k]\right\}$, where $\mathcal{E}^1$ is the encoding mapping $\mathcal{E}^1 : \mathcal{R}_{p_0} \rightarrow \mathcal{R}_{p_1}$ defined in Equation (3). This represents the *Level 1* encoding of M_i. Next, the construction proceeds recursively; the encoding of M_i corresponding to *Level 2* takes the parent node P (in this case P is $\mathcal{E}^1(M_i)$), and obtains on the left branch:

$$\mathcal{E}^2(P) = \mathcal{E}^2(\mathcal{E}^1(M_i)) = a^2_{1,i} \cdot s + p_1 \cdot e^2_{1,i} + \left((a^1_{1,i} \cdot s + p_0 \cdot e^1_{1,i} + M_i) \cdot s\right) \ ,$$

while for the right branch:

$$\mathcal{E}^2(P \cdot s) = \mathcal{E}^2(\mathcal{E}^1(M_i) \cdot s) = a^2_{2,i} \cdot s + p_1 \cdot e^2_{2,i} + \left((a^1_{1,i} \cdot s + p_0 \cdot e^1_{1,i} + M_i) \cdot s\right) \ ,$$

where $(a^2_{1,i}, a^2_{2,i}) \leftarrow_{\$} \mathcal{R}^2_{p_2}$ and noise terms are sampled from a Gaussian distribution: $(e^2_{1,i}, e^2_{2,i}) \leftarrow_{\chi} \chi^2_{p_2}$. This procedure is executed recursively up to a number of levels – denoted as d – as presented in Figure 2.

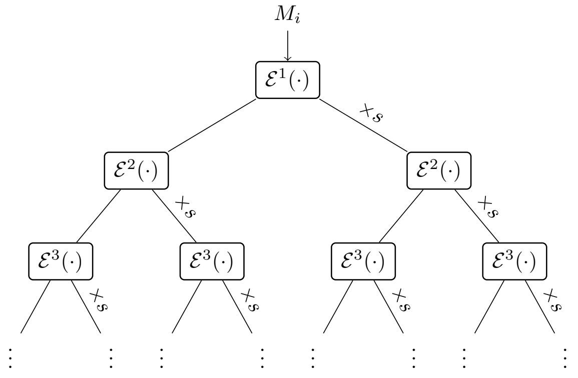

Fig. 2. The tree encoding M_i in a recursive manner corresponds to ciphertext CT_i.

Between any two successive multiplication layers, an *addition* layer is interleaved. This layer replicates the ciphertext in the previous multiplication layer (and uses its modulus). As it brings no new information, we ignore additive layers from our overview. The encoding procedure is applied for each $M_1, \dots, M_k$. In addition, Level 1 also contains $\mathcal{E}^1(s)$, while Level i (for $2 \leq i \leq d$) also contains $\mathcal{E}^i(s \cdot s)$. Hitherto, the technique used by the scheme resembles to the ones used in levelled fully homomorphic encryption. We also remind that encodings at Level i are denoted as CT^i and are included in the ciphertext. The high level idea is to compute the function $\mathscr{C}$ obliviously with the help of the encodings.

However, as we are in the FE setting, the ciphertext also contains additional information on s. This is achieved by using a linear functional encryption scheme $\mathsf{Lin\text{-}FE}$. Namely, an extra component of the form:

$$\mathbf{d} \leftarrow \mathbf{w} \cdot s + p_{d-1} \cdot \eta \tag{4}$$

is provided as an independent part of the ciphertext, also denoted as $\mathsf{CT}_{\mathsf{ind}}$, where $\eta \hookleftarrow_{\chi} \mathcal{R}_{p_d}$ stands for a noisy term and $\mathbf{w}$ is part of mpk. $\mathbf{d}$ is computed once, at top level d.

The **master secret key** of the NC^1 construction is set to be the msk for $\mathsf{Lin\text{-}FE}$. The **master public key** consists of the $\mathsf{Lin\text{-}FE.mpk}$ (the vector $\mathbf{w}$ appended to $\mathbf{a}^d$) as well as of the set of vectors $\{\mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^{d-1}\}$ that will be used by each $\mathcal{E}^i$. Once again, we *emphasize* that the vector $\mathbf{a}^d$ from $\mathsf{Lin\text{-}FE.mpk}$ coincides with the public labelling used by the mapping $\mathcal{E}^d$. It can be easily observed that the dimension of $\mathbf{a}^{i+1}$ follows from a first-order recurrence:

$$|\mathbf{a}^{i+1}| = 2 \cdot |\mathbf{a}^i| + 1 \tag{5}$$

where the initial term (the length of $\mathbf{a}^1$) is set to the length of the input. The extra 1, which is added per each layer is generated by the supplemental encodings of the key-dependent messages $\{\mathcal{E}^1(s), \mathcal{E}^2(s^2), \dots, \mathcal{E}^d(s^2)\}$.

A **functional-key** $\mathsf{sk}_{\mathscr{C}}$ is issued through the $\mathsf{Lin\text{-}FE.KGen}$ procedure in the following way: (1) using the circuit representation $\mathscr{C}$ of the considered function as well as the public set of $\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^d\}$, a publicly computable value $\mathsf{PK}_{\mathscr{C}} \leftarrow \mathsf{Eval}_{\mathsf{PK}}(\mathsf{mpk}, \mathscr{C})$ is obtained by performing $\mathscr{C}$-dependent arithmetic combinations of the values in $\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^d\}$. Then, a functional key $\mathsf{sk}_{\mathscr{C}}$ is issued for $\mathsf{PK}_{\mathscr{C}}$. The $\mathsf{Eval}_{\mathsf{PK}}(\mathsf{mpk}, \mathscr{C})$ procedure uses mpk to compute $\mathsf{PK}_{\mathscr{C}}$.

Similarly, $\mathsf{Eval}_{\mathsf{CT}}(\mathsf{mpk}, \mathsf{CT}, \mathscr{C})$ computes the value of the function $\mathscr{C}$ obliviously on the ciphertext. Both procedures are defined recursively; that is to compute $\mathsf{PK}^k_{\mathscr{C}}$ and $\mathsf{CT}^i_{\mathscr{C}(\mathbf{x})}$ at level i, $\mathsf{PK}^{i-1}_{\mathscr{C}}$ and $\mathsf{CT}^{i-1}_{\mathscr{C}(\mathbf{x})}$ are needed. For a better understanding of the procedures, we will denote the encoding of $\mathscr{C}^i(\mathbf{x})$[3] by c^i, i.e. $c^i = \mathcal{E}^i(\mathscr{C}^i(\mathbf{x}))$ and the public key or label of an encoding $\mathcal{E}^i(\cdot)$ by $\mathsf{PK}(\mathcal{E}^i(\cdot))$.

Due to space constraints, we defer the description of the evaluation algorithms to Appendix C.2.

Decryption works by evaluating the circuit of $\mathscr{C}$ (known in plain by the decryptor) over the Regev encodings forming the ciphertext. At level d, the ciphertext obtained via $\mathsf{Eval}_{\mathsf{CT}}$ has the following structure:

$$\mathsf{CT}_{\mathscr{C}(\mathbf{x})} \leftarrow \mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d + p_{d-2} \cdot \eta^{d-1} + \ldots + p_0 \cdot \eta^1 + \mathscr{C}(\mathbf{x}) \tag{6}$$

Next, based on the independent ciphertext $\mathbf{d} \leftarrow \mathbf{w} \cdot s + p_{d-1} \cdot \eta$ and on the functional key, the decryptor recovers

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta' \quad \in \mathcal{R}_{p_d} . \tag{7}$$

Finally, $\mathscr{C}(\mathbf{x})$ is obtained by subtracting (7) from (6) and repeatedly applying the *mod* operator to eliminate the noise terms: $(\bmod\ p_{d-1})\ \ldots\ (\bmod\ p_0)$.

More details on $\mathsf{Eval}_{\mathsf{PK}}$ and $\mathsf{Eval}_{\mathsf{PK}}$ are given below.

3.2 Ciphertext and Public-Key Evaluation Algorithms

Let $\mathscr{C}^n$ be the circuit representation of some function restricted to level n. For a better understanding of the procedures, we will denote the encoding of $\mathscr{C}^n(\mathbf{x})$ by c^n, i.e.

$$c^n \leftarrow \mathcal{E}^n(\mathscr{C}^n(\mathbf{x})) ,$$

and the public key or label of an encoding $\mathcal{E}^n(\cdot)$ by $\mathsf{PK}(\mathcal{E}^n(\cdot))$. Furthermore, C^n are a set of level n encodings provided by the encryptor to enable the decryptor to compute c^n.

$\mathsf{Eval}_{\mathsf{PK}}(\cup_{t\in[n]}\mathsf{PK}(\mathsf{CT}^t), \ell)$ computes the label for the ℓ^{th} wire in the n level circuit, from the i^{th} and j^{th} wires of $k-1$ level:

1. Addition Level:

[3] Here, by $\mathscr{C}^i$ we denote the restriction of the circuit $\mathscr{C}$ computing f to level i.

- If $n = 1$ (base case), then, compute $\mathsf{PK}(c_\ell^1) \leftarrow \mathsf{PK}_i + \mathsf{PK}_j$.

- Otherwise, let $a_i^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{n-1}(\cup_{t\in[n-1]}\mathsf{PK}(\mathsf{CT}^t), i)$ and $a_j^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{n-1}(\cup_{t\in[n-1]}\mathsf{PK}(\mathsf{CT}^t), j)$. Compute $\mathsf{PK}(c_\ell^n) \leftarrow a_i^{n-1} + a_j^{n-1}$.

2. Multiplication Level:
   - If $n = 2$ (base case), then compute $\mathsf{PK}(c_\ell^2) \leftarrow a_i^1 \cdot a_j^1 \mathsf{PK}(\mathcal{E}^2(s^2)) - a_j^1 \cdot \mathsf{PK}(\mathcal{E}^2(c_i^1 s)) - a_i^1 \cdot \mathsf{PK}(\mathcal{E}^2(c_j^1 s))$.

   - Otherwise, let $a_i^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{n-1}(\cup_{t\in[n-1]}\mathsf{PK}(\mathsf{CT}^t), i)$ and $a_j^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{PK}}^{n-1}(\cup_{t\in[k-1]}\mathsf{PK}(\mathsf{CT}^t), j)$.
   Compute $\mathsf{PK}(c_\ell^n) \leftarrow a_i^{n-1} \cdot a_j^{n-1} \cdot \mathsf{PK}(\mathcal{E}^n(s^2)) - a_j^{n-1} \cdot \mathsf{PK}(\mathcal{E}^n(c_i^{n-1}s)) - a_i^{n-1} \cdot \mathsf{PK}(\mathcal{E}^n(c_j^{n-1}s))$.

$\mathsf{Eval}_{\mathsf{CT}}(\cup_{t\in[n]}\mathsf{CT}^t, \ell)$ computes the encoding of the ℓ^{th} wire in the n level circuit, from the i^{th} and j^{th} wires of $n-1$ level:

1. Addition Level:
   - If $n = 1$ (base case), then, compute $c_\ell^1 \leftarrow \mathcal{E}^1(x_i) + \mathcal{E}^1(x_j)$.

   - Otherwise, let $c_i^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{n-1}(\cup_{t\in[n-1]}\mathsf{CT}^t, i)$ and $c_j^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{n-1}(\cup_{t\in[n-1]}\mathsf{CT}^t, j)$. Compute $\mathsf{CT}_\ell^n \leftarrow c_i^{n-1} + c_j^{n-1}$.

2. Multiplication Level:
   - If $n = 2$ (base case), then compute $c_\ell^2 \leftarrow c_i^1 \cdot c_j^1 \cdot \mathcal{E}^2(s^2) - a_j^1 \cdot \mathcal{E}^2(c_i^1 s) - a_i^1 \cdot \mathcal{E}^2(c_j^1 s)$.

   - Otherwise, let $c_i^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{n-1}(\cup_{t\in[n-1]}\mathsf{CT}^t, i)$ and $c_j^{n-1} \leftarrow \mathsf{Eval}_{\mathsf{CT}}^{n-1}(\cup_{t\in[n-1]}\mathsf{CT}^t, j)$. Compute $\mathsf{CT}_\ell^n \leftarrow c_i^{n-1} \cdot c_j^{n-1} + a_i^{n-1} \cdot a_j^{n-1} \mathcal{E}^n(s^2) - a_j^{n-1} \cdot \mathcal{E}^n(c_i^{n-1}s) - a_i^{n-1} \cdot \mathcal{E}^n(c_j^{n-1}s)$.

4 Laconic Function Evaluation for NC^1 Circuits

We show how to instantiate an LFE protocol starting from the AR17 scheme described above (formally summarized in Appendix C.2). Furthermore, we show our construction achieves adaptive-security under RLWE with polynomial approximation factors. Finally, we compare its efficiency to the scheme for general circuits proposed in [11].

4.1 LFE for NC^1 Circuits

The core idea behind our proposal is rooted in the design of the AR17 construction. Specifically, the mpk in AR17 acts as the crs for the LFE scheme. The *compression* procedure generates a new digest by running $\mathsf{Eval}_{\mathsf{PK}}$ on the fly, given an algorithmic description of the circuit $\mathscr{C}$ (the circuit computing the desired function). As shown in [4], the public-key evaluation algorithm can be successfully executed having knowledge of only mpk and the gate-representation of the circuit. After performing the computation, the procedure sets:

$$\mathsf{digest}_{\mathscr{C}} \leftarrow \mathsf{PK}_{\mathscr{C}} \ .$$

The digest is then handed in to the other party (say Bob).

Bob, having acquired the digest of $\mathscr{C}$ in the form of $\mathsf{PK}_{\mathscr{C}}$, encrypts his message M using the FE encryption procedure in the following way: first, a secret s is sampled from the "d-level" ring $\mathcal{R}_{p_d}$; s is used to recursively encrypt each element up to level d, thus generating a tree structure, as explained in Section 3.1. Note that Bob does not need to access the ciphertext-independent part (the vector $\mathbf{w}$ from Section 3.1) in any way. This is a noteworthy difference from the AR17 construction: an FE ciphertext is intended to be decrypted at a latter point, by (possibly) multiple functional keys. However, this constitutes an overkill when it comes to *laconic function evaluation*, as there is need to support a *single* function (for which the ciphertext is specifically created).

As a second difference from the way the ciphertext is obtained in [4], we emphasize that in our LFE protocol Bob computes directly:

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$$

where $\mathsf{PK}_{\mathscr{C}}$ is the digest Alice sent. Thus, there is no need to generate a genuine functional key and to obtain the ciphertext component that depends on $\mathbf{w}$. Directly, the two former elements constitute the ciphertext, which is sent back to Alice. Finally, the LFE *decryption* step follows immediately. Alice, after computing the *auxiliary* ciphertext in (6) "on the fly" and having knowledge of the term in (7), is able to recover $\mathscr{C}(M)$.

Formally, the construction can be defined as follows:

Definition 7 (LFE for NC^1 Circuits from [4]). *Let* FE *denote the functional encryption scheme for* NC^1 *circuits proposed in [4].*

– $\mathsf{crs} \leftarrow_{\$} \mathsf{crsGen}(1^\lambda, 1^k, 1^d)$*: the* crs *is instantiated by first running* $\mathsf{FE.Setup}$

$$(\mathsf{mpk}, \mathsf{msk}) \leftarrow_{\$} \mathsf{FE.Setup}(1^\lambda, 1^k, 1^d) \ .$$

As described, mpk *has the following elements:*

$$\{\mathbf{a}^1, \ldots, \mathbf{a}^{d-1}, (\mathbf{a}^d, \mathbf{w})\}$$

with $(\mathbf{a}^d, \mathbf{w})$ *coming from the* $\mathsf{mpk}_{\mathsf{Lin\text{-}FE}}$ *and* $\mathsf{msk} \leftarrow \mathsf{msk}_{\mathsf{Lin\text{-}FE}}$.
Set $\mathsf{crs} \leftarrow \{\mathbf{a}^1, \ldots, \mathbf{a}^d\}$ *and return it.*

– $\mathsf{digest}_{\mathscr{C}} \leftarrow \mathsf{Compress}(\mathsf{crs}, \mathscr{C})$*: the compression function, given a circuit description of some function* $\mathscr{C} : \{0,1\}^k \rightarrow \{0,1\}$ *and the* crs*, computes* $\mathsf{PK}_{\mathscr{C}} \leftarrow \mathsf{Eval}_{\mathsf{PK}}(\mathbf{a}^1, \ldots, \mathbf{a}^d, \mathscr{C})$ *and then returns:*

$$\mathsf{digest}_{\mathscr{C}} \leftarrow \mathsf{PK}_{\mathscr{C}} \ .$$

– $\mathsf{CT} \leftarrow_{\$} \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_{\mathscr{C}}, M)$*: the encryption algorithm first samples* $s \leftarrow_{\$} \mathcal{R}_{p_d}$*, randomness* R *and computes recursively the Regev encodings of each bit:*

$$(\mathsf{CT}_1, \ldots, \mathsf{CT}_k, \mathsf{CT}_{\mathsf{ind}}) \leftarrow \mathsf{FE.Enc}((\mathbf{a}^1, \ldots, \mathbf{a}^d), (M_1, \ldots, M_k); s, R)$$

Moreover, a noise η^d *is also sampled from the appropriate distribution* χ *and*

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d \tag{8}$$

is obtained. The ciphertext CT *that is returned consists of the tuple:*

$$\Bigg(\underbrace{\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d}_{\mathsf{CT}_a}, \ \underbrace{\mathsf{CT}_1, \ldots, \mathsf{CT}_k, \mathsf{CT}_{\mathsf{ind}}}_{\mathsf{CT}_b} \Bigg) \ .$$

– $\mathsf{Dec}(\mathsf{crs}, \mathscr{C}, \mathsf{CT})$*: first, the ciphertext evaluation is applied and* $\mathsf{CT}_{\mathscr{C}}$ *is obtained:*

$$\mathsf{CT}_{\mathscr{C}} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{mpk}, \mathscr{C}, \mathsf{CT}) \ . \tag{9}$$

Then $\mathscr{C}(M)$ *is obtained via the following step:*

$$\big(\mathsf{CT}_{\mathscr{C}} - (\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d)\big) \mod p_{d-1} \ \ldots \ \mod p_0$$

Proposition 1 (Correctness). *The* LFE *scheme in Definition 7 enjoys correctness.*

Proof. By the correctness of $\mathsf{Eval}_{\mathsf{CT}}$, the structure of the resulting ciphertext at level d is the following:

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d + \ldots + p_0 \cdot \eta^1 + \mathscr{C}(M) \tag{10}$$

Given the structure of the evaluated ciphertext in (10) as well as the second part of the ciphertext described by (8), the decryptor obtains $\mathscr{C}(M)$ as per the decryption procedure in (9). □

Lemma 1 (Security). *Let* FE *denote the* FULL-SIM-FE*-secure functional encryption scheme for* NC^1 *circuits described in [4]. The* LFE *scheme described in Definition 7 enjoys* FULL-SIM-LFE *security against any* PPT *adversary* $\mathcal{A}$ *such that:*

$$\mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}LFE}}_{\mathcal{A},\mathsf{LFE}}(\lambda) \leq d \cdot \mathsf{Adv}^{\mathsf{RLWE}}_{\mathcal{A}'}(\lambda) \ .$$

Proof (Lemma 1). First, we describe the internal working of our simulator. Then we show how the ciphertext can be simulated via a hybrid argument, by describing the hybrid games and their code. Third, we prove the transition between each consecutive pair of hybrids.

The Simulator. Given the digest $\mathsf{digest}_{\mathscr{C}}$, the circuit $\mathscr{C}$, the value of $\mathscr{C}(M^*)$ and the crs, our simulator $\mathcal{S}_{\mathsf{LFE}}$ proceeds as follows:

– Samples all Regev encodings uniformly at random.

– Replaces the functional-key surrogate value $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$ with $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d - \sum_{i=0}^{d-1} p_i \cdot \mu^{i+1} - \mathscr{C}(M^*)$. Observe this is equivalent to running $\mathsf{Eval}_{\mathsf{CT}}$ with respect to random Regev encodings, and subtracting lower noise terms and $\mathscr{C}(M^*)$.

The Hybrids. The way the simulator is built follows from a hybrid argument. Note that we only change the parts that represent the outputs of the intermediate simulators.

Game_0**:** Real game, corresponding to the setting $b = 0$ in the $\mathsf{FULL\text{-}SIM\text{-}LFE}$ experiment.

Game_1**:** We switch from $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$ to $\mathsf{Eval}_{\mathsf{CT}} - \mathsf{Noise} - \mathscr{C}(M^*)$, such that during decryption one recovers $\mathsf{Noise} + \mathscr{C}(M^*)$. The transition to the previous game is possible as we can sample the noise Noise η^* such that:

$$\mathsf{SD}\Big(\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d, \mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d - \sum_{i=0}^{d-1} p_i \cdot \mu^{i+1} - \mathscr{C}(M^*)\Big) \in \textsc{Negl}(\lambda) \ .$$

This game is identical to $\mathsf{Game}_{2,0}$.

$\mathsf{Game}_{2.i}$**:** We rely on the security of RLWE in order to switch all encodings on level $(d+1) - i$ with randomly sampled elements over the corresponding rings $\mathcal{R}_{p_{d+1-i}}$. Note that top levels are replaced before the bottom ones; the index $i \in [d]$.

$\mathsf{Game}_{2.d}$**:** This setting corresponds to $b = 1$ in the $\mathsf{FULL\text{-}SIM\text{-}LFE}$ experiment.

We now prove the transitions between the hybrid games.

Claim (Distance between Game_0 *and* Game_1*).* There exists a PPT simulator $\mathcal{S}_1$ such that for any stateful PPT adversary $\mathcal{A}$ we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_0 \to \mathsf{Game}_1}(\lambda) := \left| \Pr[\mathsf{Game}_0^{\mathcal{A}}(\lambda) \Rightarrow 1] - \Pr[\mathsf{Game}_1^{\mathcal{A}}(\lambda) \Rightarrow 1] \right|$$

is statistically close to 0.

Proof. Let $\mathcal{D}_0$ (respectively $\mathcal{D}_1$) be the distribution out of which CT_a is sampled in Game_0 (respectively Game_1). The statistical distance between $\mathcal{D}_0$ and $\mathcal{D}_1$ is negligible:

$$\begin{aligned}\mathsf{SD}(\mathcal{D}_0, \mathcal{D}_1) &= \frac{1}{2} \cdot \sum_{v \in \mathcal{R}_{p_d}} \Big| \Pr[v = \mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d] - \\ &\qquad \Pr[v = \mathsf{Eval}_{\mathsf{CT}}(\mathscr{C}) + \sum_{i=0}^{d-1} p_i \cdot \mu^{i+1} - \mathscr{C}(M^*)] \Big| \\ &= \frac{1}{2} \cdot \sum_{v \in \mathcal{R}_{p_d}} \Big| \Pr[v = \mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d] - \\ &\qquad \Pr[v = \mathsf{PK}_{\mathscr{C}} \cdot s + \sum_{i=0}^{d-1} p_i \cdot \mu^{i+1}] \Big|\end{aligned}$$

We apply the same technique as per [4, p. 27], and we sample a noise term η_*^d such that:

$$\mathsf{SD}(\eta_*^d \ , \ \sum_{i=0}^{d-1} p_i \cdot \nu^{i+1}) \leq \epsilon \ .$$

Thus, the advantage of any adversary in distinguishing between Game_0 and Game_1 is statistically close to 0. □

Games 1 and 2.0 are identical.

Claim (Distance between $\mathsf{Game}_{2.i}$ *and* $\mathsf{Game}_{2.i+1}$*).* There exists a PPT simulator $\mathcal{S}_1$ such that for any stateful PPT adversary $\mathcal{A}$ we have:

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_{2.i}\to\mathsf{Game}_{2.i+1}}(\lambda) &:= \left| \Pr[\mathsf{Game}_{2.i}^{\mathcal{A}}(\lambda) \Rightarrow 1] - \Pr[\mathsf{Game}_{2.i+1}^{\mathcal{A}}(\lambda) \Rightarrow 1] \right| \\ &\leq \mathsf{Adv}_{\mathcal{B}}^{\mathsf{RLWE}}(\lambda) \ .\end{aligned}$$

Proof. The reduction $\mathcal{B}$ is given as input a sufficiently large set of elements which are either: RLWE samples of the form $a \cdot s^{(d-i)} + p_{d-1-i} \cdot \eta^{(d-i)}$ or u where $u \leftarrow_{\$} \mathcal{R}_{p_{d-i}}$.

$\mathcal{B}$ constructs CT_b as follows: for upper levels $j > d-i$, $\mathcal{B}$ samples elements uniformly at random over $\mathcal{R}_{p_j}$. For each lower levels $j < d-i$, $\mathcal{R}$ samples independent secrets s^j and builds the lower level encodings correctly, as stated in Figure 2.

For challenge level $d-i$, $\mathcal{B}$ takes the challenge values of the RLWE experiment – say z –, and produces the level $d-i$ encodings from the level $d-1-i$ encodings as follows:

– for each encoding $\mathcal{E}^{d-1-i}$ in level $d-i-1$, produce a left encoding in level $d-i$:
$$z_1^{d-i} + \mathcal{E}^{d-1-i} \ .$$
– for each encoding $\mathcal{E}^{d-1-i}$ in level $d-i-1$, produce a right encoding in level $d-i$:
$$z_2^{d-i} + \mathcal{E}^{d-1-i} \cdot s^{d-1-i} \ .$$

Note that s^{d-1-i} is known in plain by $\mathcal{B}$.

Considering level 1, the message bits themselves are encrypted, as opposed to encodings from lower levels.

The first component of the ciphertext – CT_a – is computed by getting $v \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathscr{C}, \{C^i\}_{i\in[d]})$ over the encodings, and then adding the noise and the value $\mathscr{C}(M^*)$.

The adversary is provided with the simulated ciphertext. The analysis of the reduction is immediate: the winning probability in the case of $\mathcal{B}$ is identical to that of the adversary distinguishing. □

Finally, we apply the union bound and conclude with:

$$\mathsf{Adv}_{\mathcal{A},\mathsf{LFE}}^{\mathsf{FULL\text{-}SIM\text{-}LFE}}(\lambda) \leq d \cdot \mathsf{Adv}_{\mathcal{A}'}^{\mathsf{RLWE}}(\lambda) \ .$$

□

In Appendix A, we show how the digest can be further compressed.

References

1. Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015.
2. Shweta Agrawal. Stronger security for reusable garbled circuits, general definitions and attacks. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 3–35. Springer, Heidelberg, August 2017.
3. Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016.
4. Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 173–205. Springer, Heidelberg, November 2017.
5. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.
6. Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, August 2011.
7. 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, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Heidelberg, August 2017.
8. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.
9. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.
10. Adam O'Neill. Deterministic public-key encryption revisited. Cryptology ePrint Archive, Report 2010/533, 2010. `http://eprint.iacr.org/2010/533`.
11. Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.
12. Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. `http://eprint.iacr.org/2005/187`.
13. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.
14. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.
15. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Achieving a Laconic Digest

In this part, we put forth an LFE scheme with a digest independent by the size of the input, starting from the LFE construction in Section 4.1. The main idea consists in using a laconic oblivious transfer (see Appendix C.1) in order to generate a small digest. In doing so, our strategy departs from the one used in [11].

The bulk of the idea is to observe that the encryption algorithm in Section 4.1 outputs a ciphertext having two components: the first one consists of the AR17 ciphertext; the second component is literally $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$. We create an auxiliary circuit $\mathscr{C}_{aux}$ that takes as input $\mathsf{PK}_{\mathscr{C}}$ and outputs $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$. We garble $\mathscr{C}_{aux}$ using Yao's garbling scheme (Section 2.2) and use LOT to encrypt the garbling labels. Thus, instead of garbling a circuit that produces the entire ciphertext (i.e. following the template proposed by [11]) we garble $\mathscr{C}_{aux}$. Such an approach is advantageous, as the complexity of the latter circuit, essentially performing a multiplication of two elements over a polynomial ring, is in general low when compared to a circuit that reconstructs the entire ciphertext. We present our construction below:

Definition 8 (LFE for NC^1 with Laconic Digest). *Let LFE denote the laconic function evaluation scheme for NC^1 circuits presented in Section 4.1, LOT stand for a laconic oblivious transfer protocol and GS denote Yao's garbling scheme. $\overline{\mathsf{LFE}}$ stands for a laconic function evaluation scheme for NC^1 with digest of size $O(\lambda)$:*

– $\overline{\mathsf{crs}} \leftarrow_{\$} \overline{\mathsf{LFE}}.\mathsf{crsGen}(1^\lambda, 1^k, 1^d)$*: the $\overline{\mathsf{crs}}$ is instantiated by running the* $\mathsf{LFE.Setup}$

$$\mathsf{crs} \leftarrow_{\$} \mathsf{LFE.Setup}(1^\lambda, 1^k, 1^d)\ .$$

Let $\mathsf{crs}_{\mathsf{LOT}}$ stand for the common reference string of the LOT scheme.

Set $\overline{\mathsf{crs}} \leftarrow (\mathsf{crs}, \mathsf{crs}_{\mathsf{LOT}})$ and return it.

– $\mathsf{digest}_{\mathscr{C}} \leftarrow \overline{\mathsf{LFE}}.\mathsf{Compress}(\overline{\mathsf{crs}}, \mathscr{C})$*: the compression function, given a circuit description of some function $\mathscr{C} : \{0,1\}^k \to \{0,1\}$ and the $\overline{\mathsf{crs}} \leftarrow (\mathsf{crs}, \mathsf{crs}_{\mathsf{LOT}})$, computes $\mathsf{PK}_{\mathscr{C}} \leftarrow \mathsf{LFE.Compress}(\mathsf{crs}, \mathscr{C})$ and then returns:*

$$\mathsf{digest}_{\mathscr{C}} \leftarrow \mathsf{PK}_{\mathscr{C}}\ .$$

Then, using $\mathsf{crs}_{\mathsf{LOT}}$, a new digest/database pair is obtained as:

$$(\overline{\mathsf{digest}_{\mathscr{C}}}, \hat{\mathbf{D}}) \leftarrow \mathsf{LOT.Compress}(\mathsf{crs}_{\mathsf{LOT}}, \mathsf{digest}_{\mathscr{C}})\ .$$

Finally, $\overline{\mathsf{digest}_{\mathscr{C}}}$ is returned.

– $\mathsf{CT} \leftarrow_{\$} \overline{\mathsf{LFE}}.\mathsf{Enc}(\overline{\mathsf{crs}}, \overline{\mathsf{digest}_{\mathscr{C}}}, M)$*: after parsing the* $\overline{\mathsf{crs}}$ *as* $(\mathsf{crs}_{\mathsf{LOT}}, \mathsf{crs})$[4]*, the encryption algorithm first samples* $s \leftarrow_{\$} \mathcal{R}_{p_d}$*, randomness* R *and computes recursively the Regev encodings of each bit:*

$$\mathsf{CT}_{\mathsf{LFE}} \leftarrow \mathsf{LFE}.\mathsf{Enc}(\mathsf{crs}, (M_1, \ldots, M_k); s, R)$$

Moreover, a noise η^{d-1} *is also sampled from the appropriate distribution* χ *and an auxiliary circuit* $\mathscr{C}_{aux}$ *that returns*

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d \tag{11}$$

is obtained, where $\mathsf{PK}_{\mathscr{C}}$ *constitutes the input and* $s, p_{d-1} \cdot \eta^d$ *are hardwired.*

Then, $\mathscr{C}_{aux}$ *is garbled by Yao's garbling scheme:*

$$(\Gamma, \{L_i^0, L_i^1\}_{i=1}^t) \leftarrow_{\$} \mathsf{GS}.\mathsf{Garble}(\mathscr{C}_{aux})$$

and the labels are encrypted under LOT*:*

$$\overline{L_i} \leftarrow_{\$} \mathsf{LOT}.\mathsf{Enc}(\mathsf{crs}_{\mathsf{LOT}}, \overline{\mathsf{digest}_{\mathscr{C}}}, i, L_i^0, L_i^1), \forall\, i \in [t]$$

The ciphertext CT *is set to be the tuple* $(\mathsf{CT}_{\mathsf{LFE}}, \Gamma, \overline{L_1}, \ldots, \overline{L}_t)$.

– $\overline{\mathsf{LFE}}.\mathsf{Dec}(\overline{\mathsf{crs}}, \mathscr{C}, \mathsf{CT})$*: First, the labels* L_i *corresponding to the binary decomposition of* $\mathsf{PK}_{\mathscr{C}}$ *are obtained:*

$$L_i^{\mathsf{PK}_{\mathscr{C}}[i]} \leftarrow \mathsf{LOT}.\mathsf{Dec}(\mathsf{crs}_{\mathsf{LOT}}, \mathsf{digest}_{\mathscr{C}}, i, \overline{L_i})$$

When feeding Γ *with* L_i*, the decryptor recovers* $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^{d-1}$.

Then, the ciphertext evaluation is applied and $\mathsf{CT}_{\mathscr{C}}$ *is obtained:*

$$\mathsf{CT}_{\mathscr{C}} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{mpk}, \mathscr{C}, \mathsf{CT})\ . \tag{12}$$

Then $\mathscr{C}(M)$ *is obtained via the following step:*

$$\Big(\mathsf{CT}_{\mathscr{C}} - (\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d)\Big) \mod p_{d-1} \ \ldots \ \mod p_0$$

Proposition 2 (Correctness). *The laconic function evaluation scheme in Definition 8 is correct.*

Proof. By the correctness of the LOT scheme, the correct labels corresponding to the value of $\mathsf{PK}_{\mathscr{C}}$ are recovered. By the correctness of the garbling scheme, when fed with the correct labels, the value of $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$ is obtained. Finally, by the correctness of LFE, we recover $\mathscr{C}(M)$. □

[4] Even if we omit that explicitly, note that crs includes mpk, consistently with the execution of the subroutine $\mathsf{FE}.\mathsf{Enc}()$.

In what follows, we show the scheme achieves simulation security, assuming the same security level from the underlying primitive.

Theorem 2 (Security). *Let $\mathfrak{C}_\lambda$ denote a class of circuits and let $\overline{\mathsf{LFE}}$ denote the laconic function evaluation scheme defined Definition 8. Let GS and LOT denote the underlying garbling scheme, respectively laconic oblivious transfer scheme. The advantage of any* PPT *bounded adversary $\mathcal{A}$ against the adaptive simulation security of the $\overline{\mathsf{LFE}}$ scheme is bounded as follows:*

$$\mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}LFE}}_{\overline{\mathsf{LFE}},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}GS}}_{\mathsf{GS},\mathcal{B}_1}(\lambda)+\mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}LOT}}_{\mathsf{LOT},\mathcal{B}_2}(\lambda)+\mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}LFE}}_{\mathsf{LFE},\mathcal{B}_3}(\lambda) .$$

Proof (Theorem 2). We show the construction achieves adaptive security. In our proof we make use of the LFE simulator as well as the simulation security of the garbling protocol and of the LOT scheme.

Simulator. We build the simulator $\mathcal{S}_{\overline{\mathsf{LFE}}}$ as follows: first, we run the simulator of the underlying LFE scheme in order to obtain the bulk of the ciphertext. Independently, we run the simulator of the garbled circuit $\mathcal{S}_{\mathsf{GS}}$. Finally, we run the simulator of the LOT scheme ($\mathcal{S}_{\mathsf{LOT}}$) on the labels obtained from the $\mathcal{S}_{\mathsf{GS}}$.

The proof requires a hybrid argument. We present below the games and the transitions between them:

Game_0**:** corresponds to the $\mathsf{FULL\text{-}SIM\text{-}LFE}$ game where b is set to 0.

Game_1**:** in this game, we use the simulator $\mathcal{S}_{\mathsf{LOT}}$ to simulate the corresponding component of the ciphertext (i.e. $\overline{L_i}$) . The distance to the previous game is bounded by the simulation security of the LOT protocol.

Game_2**:** in this game, we switch to the usage of the garbled scheme simulator GS in order to compute the labels corresponding to the second part of the ciphertext. The game hop is bounded by the simulation security of the garbling scheme.

Game_3**:** we change the bulk ciphertext to the one obtained by the $\mathcal{S}_{\mathsf{LFE}}$ simulator, the distance to the previous game being bounded by the advantage in Lemma 1.

Claim (Transition between Game_0 *and* Game_1*).* The advantage of any PPT adversary in distinguishing between Game_0 and Game_1 is bounded as follows:

$$\mathsf{Adv}^{\mathsf{Game}_0\to\mathsf{Game}_1}_{\mathcal{A}_1}(\lambda) \leq \mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}LOT}}_{\mathsf{LOT},\mathcal{B}_1}(\lambda) .$$

Proof ($\mathsf{Game}_0 \to \mathsf{Game}_1$*).* We provide a reduction to the LOT security experiment, which initially samples and publishes $\mathsf{crs}_{\mathsf{LOT}}$. Let $\mathcal{B}_1$ denote the reduction. $\mathcal{B}_1$ simulates the LFE game in front of the PPT bounded adversary $\mathcal{A}_1$: (1) First, it samples $\mathsf{crs}_{\mathsf{LFE}}$, and publishes $(\mathsf{crs}_{\mathsf{LFE}}, \mathsf{crs}_{\mathsf{LOT}})$; (2) Next, $\mathcal{B}_1$ receives from the

LFE adversary $\mathcal{A}_1$ the tuple $(\mathscr{C}, \mathscr{C}(M^*))$; (3) computes the digest $\mathsf{digest}_{\mathscr{C}}$, the underlying LFE ciphertext, and the garbled circuits with the associated labels.

Next, $\mathcal{B}_1$ impersonates an adversary against the LOT security game, by submitting the tuple $\big(\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d,\ \{L_{i,i0}, L_{i,1}\}_{i \in |\mathsf{digest}_{\mathscr{C}}|}\big)$.

The LOT game picks a random $b \in \{0,1\}$ and provides the adversary with either a correctly generated LOT ciphertext encrypting the labels L_i^0, L_i^1 under position i, or by a simulated ciphertext. The latter ciphertext is generated by $\mathcal{S}_{\mathsf{LOT}}$.

Thus $\mathcal{B}_1$ obtains the entire $\overline{\mathsf{LFE}}$ ciphertext, which is passed to $\mathcal{A}_1$. $\mathcal{A}_1$ returns a bit, indicating its current setting. It is clear that any PPT adversary able to distinguish between the two settings breaks the LOT security of the underlying LOT scheme. □

Claim (Transition between Game_1 *and* Game_2*).* The advantage of any PPT adversary to distinguish between Game_1 and Game_2 is bounded as follows:

$$\mathsf{Adv}_{\mathcal{A}_2}^{\mathsf{Game}_1 \to \mathsf{Game}_2}(\lambda) \leq \mathsf{Adv}_{\mathsf{GS},\mathcal{B}_2}^{\mathsf{FULL\text{-}SIM\text{-}GS}}(\lambda) \ .$$

Proof ($\mathsf{Game}_1 \to \mathsf{Game}_2$*).* By the input and circuit privacy of the garbled circuit, there exists $\mathcal{S}_{\mathsf{GS}}$ that produces a tuple $(\tilde{\Gamma}, \{\tilde{L}_i^0, \tilde{L}_i^1\}_{i \in [t]})$.

Let $\mathcal{B}_2$ denote the reduction, and let $\mathcal{A}_2$ denote the adversary against the LFE game. As for the previous game, $\mathcal{B}_2$ samples and publishes $\mathsf{crs}_{\overline{\mathsf{LFE}}}$, $\mathcal{A}_2$ provides $(\mathscr{C}, \mathscr{C}(M^*))$. $\mathcal{B}_2$ builds the LFE ciphertext.

Next, $\mathcal{B}_2$ impersonates an adversary against the GS security experiment. $\mathcal{B}_2$ provides the GS game with $(\mathscr{C}_{aux}, \mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d)$, and receives either correctly generated garbled circuit and labels or a simulated garbled circuit and the simulated labels.

Thus, if $\mathcal{A}_2$ distinguishes between the two games, $\mathcal{B}_2$ distinguishes between the two distributions of labels in the GS game. □

Claim (Transition between Game_2 *and* Game_3*).* The advantage of any PPT adversary to distinguish between Game_2 and Game_3 is bounded as follows:

$$\mathsf{Adv}_{\mathcal{A}_3}^{\mathsf{Game}_2 \to \mathsf{Game}_3}(\lambda) \leq \mathsf{Adv}_{\mathsf{LFE},\mathcal{B}_3}^{\mathsf{FULL\text{-}SIM\text{-}LFE}}(\lambda) \ .$$

Proof ($\mathsf{Game}_2 \to \mathsf{Game}_3$*).* Finally, we use the security of the LFE scheme in order to switch elements of the first component of the ciphertext to simulated ones. The advantage of any adversary noticing the transition is bounded by the advantage of winning the $\mathsf{FULL\text{-}SIM\text{-}LFE}$ security of the underlying LFE.

The $\mathsf{FULL\text{-}SIM\text{-}LFE}$ experiment generates $\mathsf{crs}_{\mathsf{LFE}}$. $\mathcal{B}_3$ samples $\mathsf{crs}_{\mathsf{LOT}}$ and provides the resulting $\overline{\mathsf{crs}}$ to $\mathcal{A}_3$. The adversary returns $(\mathscr{C}, \mathscr{C}(M^*))$, which are forwarded to $\mathsf{FULL\text{-}SIM\text{-}LFE}$. The experiment generates the ciphertext either correctly or using $\mathcal{S}_{\mathsf{LFE}}$.

Then $\mathcal{B}_3$ employs $\mathcal{S}_{\mathsf{GS}}$ and $\mathcal{S}_{\mathsf{LOT}}$ to obtain the remaining $\overline{\mathsf{LFE}}$ ciphertext components and runs $\mathcal{A}_3$ on the full ciphertext. $\mathcal{B}_3$ returns the corresponding output to the setting indicated by $\mathcal{A}_3$.

We also note that this setting simulates the $\mathsf{FULL\text{-}SIM\text{-}LFE}$ experiment with $b = 1$. □

This completes the proof of Theorem 2. □

Size of parameters. A main benefit of the LFE schemes in Definitions 7 and 8 is the *simplicity* of the crs, $\mathsf{digest}_{\mathscr{C}}$ and ciphertext structures. These are essentially elements over known rings of polynomials. The sizes of the rings $\mathcal{R}_{p_i}$ are enforced by the prime factors $p_0, p_1, \ldots, p_d$. As stated in [4, Appendix E] $p_d \in O(B_1^{2^d})$ where B_1 denotes the magnitude of the noise used for Level-1 encodings and being bounded by $p_1/4$ in order to ensure correct decryption[5]. However, we observe that a tighter bound may have the following form: $p_d \in O(B_1^{2^{\mathsf{Mul}}})$, where Mul denotes the number of multiplicative levels in the circuit. From the point of view of space complexity, we rely on the original analysis of [4] that provides guidelines for the size of the primes $p_0, p_1, \ldots, p_d$. As we are only interested in the case stipulating that $|p_d|$ belongs to $O(\mathsf{poly}(\lambda))$. The condition can be achieved by imposing further restrictions on the multiplicative depth of the circuit, namely $d \in O(\log(\mathsf{poly}(\lambda)))$, meaning that the digest is short enough whenever the circuit belong to NC^1 class. The size of the digest in Definition 8 is laconic, given the fact that LOT produces laconic digests.

Handling multiple bits of output. Regarding the ability to support multiple output bits (say ℓ), this is inherent by using an arithmetic circuit and setting $\lfloor \log(p_0) \rfloor = \ell$. If binary circuits are needed, then ℓ public evaluation can be obtained through $\mathsf{Eval}_{\mathsf{PK}}$, and the scheme modified by having the garbling circuit producing ℓ outputs.

Another potentially beneficial point of comparison is the ability of the schemes in Definitions 7 and 8 to support a bounded number of circuits without changing the data-dependent ciphertext. This happens when the second party involved is stateful: given two functions f, g represented through circuits $\mathscr{C}_f, \mathscr{C}_g$, and assuming the receiver stores s (it is stateful), the Enc algorithm may simply recompute $\mathsf{PK}_g \cdot s + p_{d-1} \cdot \mu^d$ in the same manner it has already computed $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$.

B Bootstrapping LFE to succinct single-key FE

Quash *et al.* [11] proposed a compiler from an LFE protocol to a succinct single-key functional encryption scheme. We recall their result below.

Theorem 3 (LFE to FE Compiler from [11]). *Assuming the existence of a selectively (respectively adaptively) secure* LFE *scheme, there exists a succinct, selective (respectively adaptive), simulation-secure, single-key, functional encryption scheme.*

If an LFE*, for a circuit family* $\{\mathscr{C}k, d\}_{k\in\mathbb{N},d\in\mathbb{N}}$ *with circuit parameters* (k, d)*, has an encryption circuit of size* $T = T(\lambda, k, d)$ *(where the inputs to the encryption circuit are both the message and the randomness for the* LFE *encryption),*

[5] Our scheme is intended to support Boolean circuits, thus p_0 is set to 2.

then the resulting FE *has encryption time and ciphertext size* $T \cdot \mathsf{poly}(\lambda)$*.*

Their transformation follows:

- $\overline{\mathsf{FE}}.\mathsf{Setup}(1^\lambda)$*: compute* $\mathsf{crs}_{\mathsf{LFE}} \leftarrow \mathsf{LFE}.\mathsf{crsGen}(1^\lambda)$ *and* $(\mathsf{mpk}_{\mathsf{FE}}, \mathsf{msk}_{\mathsf{FE}}) \leftarrow_{\$} \mathsf{FE}.\mathsf{Setup}(1^\lambda)$*.*
  Return:
  $$\overline{\mathsf{mpk}} \leftarrow \mathsf{mpk}_{\mathsf{FE}}$$
  $$\overline{\mathsf{msk}} \leftarrow (\mathsf{crs}_{\mathsf{LFE}}, \mathsf{msk}_{\mathsf{FE}})$$
- $\overline{\mathsf{FE}}.\mathsf{KeyGen}(\overline{\mathsf{msk}}, \mathscr{C})$*: let* $\mathscr{C}_{\mathsf{LFE}}$ *be a circuit that takes as input the plaintext* M *and some randomness* R *and returns an* LFE *ciphertext:* $\mathsf{LFE}.\mathsf{Enc}(\mathsf{crs}_{\mathsf{LFE}}, \mathsf{digest}_{\mathscr{C}_{\mathsf{LFE}}}, M; R)$*. Issue a functional key:*
  $$\mathsf{sk}_{\mathscr{C}_{\mathsf{LFE}}} \leftarrow_{\$} \mathsf{FE}.\mathsf{KeyGen}(\mathsf{msk}_{\mathsf{FE}}, \mathscr{C}_{\mathsf{LFE}})\ .$$
  Return:
  $$\overline{\mathsf{sk}_{\mathscr{C}}} \leftarrow (\mathsf{sk}_{\mathscr{C}_{\mathsf{LFE}}}, \mathscr{C}, \mathsf{crs}_{\mathsf{LFE}})\ .$$
- $\overline{\mathsf{FE}}.\mathsf{Enc}(\overline{\mathsf{mpk}}, M)$*: sample randomness* R *and return*
  $$\overline{\mathsf{CT}} \leftarrow \mathsf{FE}.\mathsf{Enc}(\mathsf{mpk}_{\mathsf{FE}}, (M, R))\ .$$
- $\overline{\mathsf{FE}}.\mathsf{Dec}(\overline{\mathsf{sk}_{\mathscr{C}}}, \overline{\mathsf{CT}})$*: parse* $\overline{\mathsf{sk}_{\mathscr{C}}}$ *as* $(\mathsf{sk}_{\mathscr{C}_{\mathsf{LFE}}}, \mathscr{C}, \mathsf{crs}_{\mathsf{LFE}})$*, obtain* $\mathsf{digest}_{\mathscr{C}}$ *from* $\mathscr{C}$*, then return*
  $$\mathsf{LFE}.\mathsf{Dec}(\mathsf{crs}_{\mathsf{LFE}}, \mathsf{digest}_{\mathscr{C}}, \mathsf{FE}.\mathsf{Dec}(\mathsf{sk}_{\mathscr{C}_{\mathsf{LFE}}}, \overline{\mathsf{CT}}))\ .$$

B.1 A New FE from the "LFE to FE" Compiler

In this part, we consider a novel, single-key and succinct FE, obtained through the transform above, using as building blocks: (1) the LFE scheme proposed in Appendix A, and (2) the FE scheme from [4]. In our analysis, we consider the size of the ciphertext. For the case of the original AR17 construction described in [4], its size grows exponentially within the multiplicative level, but also linearly with the length of the plaintext.

On the contrary, considering the $\overline{\mathsf{FE}}$ obtained through the compiler above, the ciphertext should support $\mathscr{C}_{\mathsf{LFE}}$. Since $\mathscr{C}_{\mathsf{LFE}}$ outputs three main components: a) an AR17 ciphertext, b) a garbled circuit, c) a set of LOT labels, the multiplicative depth of the circuit computing these components will be lower: the depth-dominating component is the one *computing* the AR17 ciphertext. However, $\mathscr{C}_{\mathsf{LFE}}$ can handle computing all Regev encodings in parallel; thus, the circuit $\mathscr{C}_{\mathsf{LFE}}$ has a smaller depth compared to $\mathscr{C}$[6].

We assume we bootstrap using the FE from Theorem 3. Therefore, the size of the $\overline{\mathsf{FE}}$ ciphertext (i.e. AR17 ciphertext) is enforced by the depth $\mathscr{C}_{\mathsf{LFE}}$, which is lower compared to the depth of the original $\mathscr{C}$. As a consequence, the ciphertext corresponding to $\overline{\mathsf{FE}}$ is asymptotically more efficient.

[6] This is also guaranteed by the laconic requirement of an LFE, which stipulates the time of computing the ciphertext should be lower than the size of the original circuit.

C Further Definitions

C.1 Construction of LOT in [7]

In [7], Cho *et al.* first present an LOT construction based on the DDH assumption, for which the hash function compresses a database of length 2λ into a digest of length λ (factor-2 compression). Afterwards, the authors explain how to build an LOT scheme for a database of arbitrarily $\mathsf{poly}(\lambda)$ length M.

For the factor-2 compression LOT, the construction is based on two primitives called *Somewhere Statistically Binding Hash Functions* (SSB) and *Hash Proof System* (HPS). We recall their definitions.

Definition 9 (Somewhere Statistically Binding Hash). *An SSB hash function* SSBH *consists of three algorithms* crsGen*,* $\mathsf{bindingCrsGen}$ *and* Hash *with the following syntax.*

– $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$*: takes the security parameter λ as input and outputs a common reference string* crs.
– $\mathsf{crs} \leftarrow \mathsf{bindingCrsGen}(1^\lambda, i)$*: takes as input the security parameter λ and an index $i \in [2\lambda]$, and outputs a binding common reference string* crs.
– $y \leftarrow \mathsf{Hash}(\mathsf{crs}, x)$*. For some domain $\mathcal{D}$, it takes as input a common reference string* crs *and a string $x \in \mathcal{D}^{2\lambda}$, and outputs a string $y \in \{0,1\}^\lambda$.*

We require the following properties from an SSB hash function:

1. **Statistically Binding at Position i**: For every $i \in [2\lambda]$ and an overwhelming fraction of crs in the support of $\mathsf{bindingCrsGen}(1^\lambda, i)$ and every $x \in \mathcal{D}^{2\lambda}$, we have that $(\mathsf{crs}, \mathsf{Hash}(\mathsf{crs}, x))$ uniquely determines x_i. More formally, for all $x' \in \mathcal{D}^{2\lambda}$ such that $x_i \neq x'_i$ we have that $\mathsf{Hash}(\mathsf{crs}, x') \neq \mathsf{Hash}(\mathsf{crs}, x)$.

2. **Index Hiding**: It holds for all $i \in [2\lambda]$ that $\mathsf{crsGen}(1^\lambda) \approx_c \mathsf{bindingCrsGen}(1^\lambda, i)$, i.e., common reference strings generated by crsGen and $\mathsf{bindingCrsGen}$ are computationally indistinguishable.

Definition 10 (Hash Proof System). *Let $\mathcal{L}_z \subseteq \mathcal{M}_z$ be an $\mathbb{NP}$-language residing in a universe $\mathcal{M}_z$, both parameterized by some parameter z. Moreover, let $\mathcal{L}_z$ be characterized by an efficiently computable witness-relation $\mathcal{R}$, namely, for all $x \in \mathcal{M}_z$ it holds that $x \in \mathcal{L}_z \Leftrightarrow \exists\ w : \mathcal{R}(x, w) = 1$. A hash proof system* HPS *for $\mathcal{L}_z$ consists of three algorithms* KeyGen*,* $\mathsf{Hpublic}$ *and* $\mathsf{Hsecret}$ *with the following syntax.*

– $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, z)$*: takes as input the security parameter λ and a parameter z, and outputs a public-key and secret key pair* $(\mathsf{pk}, \mathsf{sk})$.
– $y \leftarrow \mathsf{Hpublic}(\mathsf{pk}, x, w)$*. Takes as input a public key* pk*, an instance $x \in \mathcal{L}_z$, and a witness w, and outputs a value y.*
– $y \leftarrow \mathsf{Hsecret}(\mathsf{sk}, x)$*: takes as input a secret key* sk *and an instance $x \in \mathcal{M}_z$, and outputs a value y.*

We require the following properties of a hash proof system.

1. **Perfect Completeness**: For all z, for all $(\mathsf{pk}, \mathsf{sk})$ in the generated by $\mathsf{KeyGen}(1^\lambda, z)$, and for all $x \in \mathcal{L}_z$ having witness w (i.e., $\mathcal{R}(x, w) = 1$), it holds that:

$$\mathsf{Hpublic}(\mathsf{pk}, x, w) = \mathsf{Hsecret}(\mathsf{sk}, x)$$

2. **Perfect Soundness**: For every z and every $x \in \mathcal{M}_z$, let $(\mathsf{pk}, \mathsf{sk})$ generated by $\mathsf{KeyGen}(1^\lambda, z)$, then it holds that:

$$(z, pk, \mathsf{Hsecret}(\mathsf{sk}, x)) \equiv (z, \mathsf{pk}, u)$$

where u is distributed uniformly random in the range of $\mathsf{Hsecret}$.

Furthermore, we recall some notions, which are used in the description of the LOT construction in [7]. We denote by $\hat{\mathbf{M}} = g^{\mathbf{M}} \in \mathbb{G}^{m \times n}$ the element-wise exponentiation of g with the elements of $\mathbf{M}$. We also define $\hat{\mathbf{L}} = \hat{\mathbf{H}}^{\mathbf{M}} \in \mathbb{G}^{m \times k}$, where $\hat{\mathbf{H}} \in \mathbb{G}^{m \times n}$ and $\mathbf{M} \in \mathbb{Z}_p^{n \times k}$ as follows: each element $\hat{\mathbf{L}}_{i,j} = \prod_{k=1}^{n} \hat{\mathbf{H}}_{i,k}^{\mathbf{M}_{k,j}}$ This is an abuse of notation, but intuitively this operation corresponds to matrix multiplication "in the exponent".

LOT with Factor-2 Compression. In this part, we will present initially the constructions presented in [7] of SSBH and HPS, followed by their factor-2 compression LOT.

SSB Hash Function. Let n be an integer such that $n = 2\lambda$, and let $(\mathbf{G}, \cdot)$ be a cyclic group of order p and with generator g. Let $\mathbf{T}_i \in \mathbb{Z}_p^{2 \times n}$ be a matrix which is zero everywhere except the i-th column, which is equal to $\mathbf{t} = (0, 1)^\top$. The three algorithms of the SSB hash function are defined below.

```
$\mathsf{crsGen}(1^\lambda)$:
$\mathbf{H} \leftarrow\!\!\$\; \mathbb{Z}_p^{2 \times n}$
$\hat{\mathbf{H}} \leftarrow g^{\mathbf{H}}$
return $\hat{\mathbf{H}}$
```

```
$\mathsf{bindingCrsGen}(1^\lambda, i)$:
$\mathbf{w} = \{(1, w_2)^\top | w_2 \leftarrow\!\!\$\; \mathbb{Z}_p\}$
$\mathbf{a} \leftarrow\!\!\$\; \mathbb{Z}_p^n$
$\mathbf{A} \leftarrow \mathbf{w} \cdot \mathbf{a}^\top$
$\mathbf{H} \leftarrow \mathbf{T}_i + \mathbf{A}$
$\hat{\mathbf{H}} \leftarrow g^{\mathbf{H}}$
return $\hat{\mathbf{H}}$
```

```
$\mathsf{Hash}(\mathsf{crs}, x)$:
$\mathbf{y} \leftarrow \hat{\mathbf{H}}^{\mathbf{x}}$
return $y$
```

Note that in the definition of Hash, $x \in \{0,1\}^{2\lambda}$ needs to be parsed in a vector $\mathbf{x} \in \mathbb{Z}_p^n$. Furthermore, the output $\mathbf{y} \in \mathbb{G}^2$ has to be returned parsed as a binary string y. Thus, in order to achieve factor-2 compression, i.e. $y \in \{0,1\}^\lambda$, a bit-representation for a group element in $\mathbb{G}$ of size $\frac{\lambda}{2}$ is required.

Hash Proof System. Fix a matrix $\hat{\mathbf{H}} \in \mathbb{G}^{2 \times n}$ and an index $i \in [n]$. Let $\mathsf{HPS} = (\mathsf{KeyGen}, \mathsf{Hpublic}, \mathsf{Hsecret})$ be defined for the following language $\mathcal{L}_{\hat{\mathbf{H}},i}$:

$$\mathcal{L}_{\hat{\mathbf{H}},i} = \{(\hat{y}, b) \in \mathbb{G}^2 \times \{0,1\} | \exists\, x \in \mathbb{Z}_p^n \; s.t. \; \hat{y} = \hat{\mathbf{H}}^x \wedge x_i = b\}$$

Furthermore, for the ease of explanation, it would be convenient to work with a matrix $\hat{\mathbf{H}}' = \begin{pmatrix} \hat{\mathbf{H}} \\ g^{e_i^\top} \end{pmatrix}$ where $g^{e_i^\top} \in \mathbb{Z}_p^n$ is the i-th unit vector, with all elements equal to 0 except the i-th one which is equal to 1. The three algorithms of the HPS are defined as follows.

```
KeyGen(1^λ, (Ĥ, i)):
    $r \leftarrow_{\$} \mathbb{Z}_p^3$
    $\hat{h} \leftarrow ((\hat{\mathbf{H}}')^\top)^r$
    $pk \leftarrow \hat{h},\ sk \leftarrow r$
    return $(pk, sk)$

Hpublic(pk, (ĥ, b), x):
    $\hat{z} \leftarrow (pk^\top)^x = (\hat{h}^\top)^x$
    return $\hat{z}$

Hsecret(sk, (ĥ, b)):
    $\hat{y}' \leftarrow \begin{pmatrix} \hat{y} \\ g^b \end{pmatrix}$
    $\hat{z} \leftarrow ((\hat{y}')^\top)^{sk} = ((\hat{y}')^\top)^r$
    return $\hat{z}$
```

LOT Scheme. Let SSBH := (SSBH.crsGen, SSBH.bindingCrsGen, SSBH.Hash) be the SSB hash function with domain $\mathcal{D} = \mathbb{Z}_p$. Also, abstractly let the associated hash proof system be HPS = (HPS.KeyGen, HPS.Hpublic, HPS.Hsecret) for the language $\mathcal{L}_{\mathsf{crs},i} = \{(\mathsf{digest}, b) \in \{0,1\}^\lambda \times \{0,1\} \mid \exists\, D \in \mathcal{D}^{2\lambda} : \mathsf{SSBH.Hash}(\mathsf{crs}, D) = \mathsf{digest} \wedge D[i] = b\}$.

The LOT scheme $\ell\mathrm{OT}$:= (crsGen, Compress, Enc, Dec) follows:

```
– crsGen(1^λ):
      crs ← SSBH.crsGen(1^λ)
      return crs
– Compress(crs, $D \in \{0,1\}^{2\lambda}$):
      digest ← SSBH.Hash(crs, $D$)
      $\hat{D} \leftarrow (D, \mathsf{digest})$
      return $(\hat{D}, \mathsf{digest})$
– Enc(crs, digest, $L$, $m_0$, $m_1$):
      Let HPS be the hash-proof system for the language $\mathcal{L}_{\mathsf{crs},L}$
      $(pk, sk) \leftarrow$ HPS.KeyGen($1^\lambda$, (crs, $L$))
      $c_0 \leftarrow m_0 \oplus$ HPS.Hsecret($sk$, (digest, 0))
      $c_1 \leftarrow m_1 \oplus$ HPS.Hsecret($sk$, (digest, 1))
      $e \leftarrow (pk, c_0, c_1)$
      return $e$
– Dec$^{\hat{D}}$(crs, $e$, $L$):
      $e = (pk, c_0, c_1),\ \hat{D} = (D, \mathsf{digest})$
      $b \leftarrow D[L]$
      $m \leftarrow c_b \oplus$ HPS.Hpublic($pk$, (digest, $b$), $D$)
      return $m$
```

The above LOT protocol has factor-2 compression and satisfies the correctness and sender privacy requirements.

LOT for arbitrarily large input databases. The authors then provide a construction to bootstrap an ℓOT scheme with factor-2 compression into an LOT scheme with an arbitrary compression factor, which can compress a database of an arbitrary (a priori unbounded polynomial in λ) size M. They achieved such generalization through *Merkle Trees.*

We briefly recall their approach. Assume for ease of exposition that $M = 2d \cdot \lambda$. First, the construction partitions the arbitrarily large database $D \in \{0,1\}^M$ into strings of length 2λ (*leaves*). Then each of these strings is compressed into a new string of length λ via the factor-2 LOT $\mathsf{Compress}$ algorithm (*node*). Next, such strings are paired, forming new strings of length 2λ, which can be compressed, in turn, through Hash, and so on. This process forms a Merkle tree structure. The $\mathsf{Compress}$ function of the general LOT scheme will exploit such structure, in fact it will compute $(\hat{D}, \mathsf{digest})$, where $\hat{D}$ is the entire Merkle tree, and digest is the root of the tree.

Now, it is easy to see that, in order to verify that a database D, with hash root digest, has a certain value b at a location L, there is no need to provide the entire Merkle tree. Instead, it is sufficient to provide a path of siblings from the digest to the leaf that contains location L. It can then be easily verified if the hash values from the leaf to the root are correct.

C.2 Formal Description for NC^1 Construction in [4]

Formal description.

- $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\!\!\$\ \mathsf{FE.Setup}(1^\lambda, 1^k, 1^d)$: let d stand for the circuit depth, k stand for the length of the supported inputs and λ for the security parameter.

$$\textbf{for } i \leftarrow 1, \ldots, d-1 : \\ \mathbf{a}^i \leftarrow\!\!\$\ \mathcal{R}_{p_i}^{L^i}$$

  The set $\{p_i : i \leftarrow 1, \ldots, d\}$ stands for a set of d primes, while L^i denotes the size of an encoding (we assume they are a priori known). Then, a $\mathsf{Lin\text{-}FE}$ scheme is instantiated:

$$(\mathsf{Lin\text{-}FE.mpk}, \mathsf{Lin\text{-}FE.msk}) \leftarrow\!\!\$\ \mathsf{Lin\text{-}FE.Setup}(1^\lambda)$$

  where $\mathsf{Lin\text{-}FE.mpk} \leftarrow (\mathbf{w}, \mathbf{a}^d)$. The following variables are set and returned:

$$\mathsf{mpk} \leftarrow (\mathbf{a}^1, \ldots, \mathbf{a}^d, \mathbf{w}) \\ \mathsf{msk} \leftarrow \mathsf{Lin\text{-}FE.msk}$$

- $\mathsf{FE.KGen}(\mathsf{msk}, \mathscr{C})$: given a function $\mathscr{C}$ represented as a circuit of depth d:

$$\mathsf{PK}_{\mathscr{C}} \leftarrow\!\!\$\ \mathsf{Eval}_{\mathsf{PK}}(\mathsf{mpk}, \mathscr{C})$$

Invoke key-derivation for $\mathsf{Lin\text{-}FE}$ to obtain $\mathsf{sk}_{\mathscr{C}}$[7] and return it:

$$\mathsf{sk}_{\mathscr{C}} \leftarrow_{\$} \mathsf{Lin\text{-}FE.KGen}(\mathsf{msk}, \mathsf{PK}_{\mathscr{C}})$$

– $\mathsf{FE.Enc}(\mathsf{mpk}, M = [M_1, \ldots, M_k])$: first, for each M_j, compute its encodings, recursively, as shown in Figure 2 for all multiplicative levels $1 \to d$:

$$\mathcal{C}_j^i \leftarrow_{\$} \mathcal{E}^i(\ldots \mathcal{E}^1(M_j) \ldots \cdot s \cdot \ldots), \ \forall i \in [d] \ .$$

Then, set $\mathbf{d}$ as follows:

$$\mathbf{d} \leftarrow \mathbf{w} \cdot s + p_{d-1} \cdot \mu \ .$$

Finally, set the ciphertext corresponding to M as $\mathsf{CT}_M \leftarrow \big(\{\mathcal{C}^i\}_{i \in [d]}, \mathbf{d}\big)$.

– $\mathsf{FE.Dec}(\mathsf{sk}_{\mathscr{C}}, \mathsf{CT}_M)$: compute

$$\mathsf{CT}_{\mathscr{C}(M)} \leftarrow \mathsf{Eval}_{\mathscr{C}}(\{\mathcal{C}^i\}_{i \in [d]}, \mathscr{C})$$

which is equivalent to:

$$\mathsf{CT}_{\mathscr{C}(M)} \leftarrow \mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^{d-1} + \ldots + p_0 \cdot \eta^0 + \mathscr{C}(M) \quad \in \ \mathcal{R}_{p_d} \ .$$

Compute also:

$$(\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta') \leftarrow \mathsf{Lin\text{-}FE.Dec}(\mathsf{sk}_{\mathscr{C}}, \mathbf{d}) \ .$$

Subtract the last two equations and remove the noise terms to obtain $\mathscr{C}(M)$.

More details on $\mathsf{Eval}_{\mathsf{PK}}$ and $\mathsf{Eval}_{\mathsf{PK}}$ are given below. We note that correctness should follow from the description of these algorithms.

[7] Note that $\mathbf{w}^{\top} \cdot \mathsf{sk}_{\mathscr{C}} = \mathsf{PK}_{\mathscr{C}}$.