Adaptively Secure Laconic Function Evaluation for NC^1

Răzvan Roşie(✉)

JAO, Luxembourg, Luxembourg
rosie@jao.eu

Abstract. Laconic Function Evaluation (LFE) protocols, introduced by Quach *et al.* in FOCS'18, allow two parties to evaluate functions *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 getting the ciphertext from Bob and in full knowledge of 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, common reference string (crs) and ciphertext are much smaller than the circuit size $|\mathscr{C}|$.

Quach *et al.* put forward a construction of laconic function evaluation for general circuits under the learning with errors (LWE) assumption (with sub-exponential 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).

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 *et al.* [14]. 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. Put differently, an LFE protocol is a type of secure multi-party computation[4,9,11] that is Bob-optimized.

S. D. Galbraith (Ed.): CT-RSA 2022, LNCS 13161, pp. 427–450, 2022.
https://doi.org/10.1007/978-3-030-95312-6_18

The work of [14] provided an elegant construction of laconic function evaluation 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 sub-exponential modulus to noise ratio, which implies that the underlying lattice problem must be hard for sub-exponential 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 [3] and *simplify* it to achieve LFE. To achieve a laconic digest, we make use of the *laconic* oblivious transfer (LOT) primitive [8,15]. 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 adaptively-secure* LFE *protocol for* NC^1.

The authors of [14] also studied the relations between LFE and other primitives, and in particular showed that LFE implies succinct functional encryption (FE) [6,13]. 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 [1,3,10]. One can apply, as well, the LFE to FE compiler of [14] to our new, adaptively secure LFE for NC^1, and obtain an alternative variant of adaptively secure, succinct FE for NC^1.

Technical Overview. The techniques that we use exploit the algebraic structure in the construction by Agrawal and Rosen [3]. 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) [7], it encrypts each bit of the plaintext, independently, as follows:

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

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

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 $\mathsf{Lin\text{-}FE}$ [2].

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

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

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

Our Technique. 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 of $\mathsf{PK}_{\mathscr{C}}$ samples its own secret s and computes recursively the Regev encodings (seen above) as well as $\mathsf{PK}_{\mathscr{C}} \cdot s + \mathsf{Noise}'$ (the $\mathsf{sk}_{\mathscr{C}}$-dependent term). These terms suffice to recover $\mathscr{C}(M)$ from (1) on the sender's side.

Our construction departs significantly from the approach in [14]. 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 ideas behind the transform of [10] to hide the "attribute" in $\mathsf{AB\text{-}LFE}$ and compressing the digests via laconic oblivious transfer [8]. On the other hand, we do not need to go via $\mathsf{AB\text{-}LFE}$, making our transformation conceptually simpler. Moreover, by relying on the adaptive security of the underlying FE scheme that we use, we obtain adaptive security.

Towards Short Digests. As per [14], 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 [14] – garbling an entire FHE decryption circuit and encrypting under an ABE the homomorphic ciphertext and the labels – we garble only the circuit 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.

Roadmap. Section 2 puts forth the algorithmic and mathematical conventions to be adopted throughout this work, as well as the definitions of primitives we use herein. In Sect. 3 we review the decomposable FE scheme proposed by Agrawal and Rosen in [3]. In Sect. 4, we introduce a new LFE scheme for NC^1 circuits, and show in Sect. 5 how to combine it with laconic oblivious transfer in order to achieve a scheme with laconic digests.

2 Background

Algorithmic and Mathematical Notation. 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 levelled construction. For any variable $k \in \mathbb{N}^*$, we define $[k] := \{1, \dots, k\}$. 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.

Circuit Representation. We consider circuits as the main model of computation for representing (abstract) functions. Unless stated otherwise, we use k to denote the input length of the circuit and d for its depth. As we assume that circuits have a tree like structure we denote as *subcircuits* the corresponding subtrees.

2.1 Computational Hardness Assumptions

The Learning With Errors (LWE) search problem [16] asks to find the secret vector $\mathbf{s}$ over $\mathbb{F}_q^\ell$ given a polynomial-size set of tuples $(\mathbf{A}, \mathbf{A}\cdot\mathbf{s}+\mathbf{e})$, where $\mathbf{A}$ stands for a randomly sampled matrix over $\mathbb{F}_q^{k\times\ell}$, while $\mathbf{e}\in\mathbb{F}^k$ represents a small error vector sampled from an appropriate distribution χ. In rough terms, the decisional version of the LWE problems, asks any PPT adversary to distinguish between the uniform distribution and one induced by the LWE tuples.

Definition 1 (Learning With Errors). *Let $\mathcal{A}$ stand for a* PPT *adversary. The advantage of $\mathcal{A}$ in distinguishing between the following two distributions is negligible:*

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

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 [12] proposed a ring version. Let $\mathcal{R} := \mathbb{Z}[X]/(X^n+1)$ for n a power of 2, while $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$ 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 Garbling Schemes

Garbling schemes were introduced by Yao in its pioneering work [18,19], 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:*

- $(\Gamma, \mathsf{sk})\leftarrow_{\$}\mathsf{Garble}(1^\lambda, \mathscr{C})$*: takes as input the unary representation of the security parameter λ and a circuit $\mathscr{C}\in\{\mathscr{C}_k\}_\lambda$; outputs a garbled circuit Γ and a secret key* sk*.*
- $c\leftarrow_{\$}\mathsf{Enc}(\mathsf{sk}, x)$*: $x\in\{0,1\}^k$ is given as input, as well as the secret key* sk*; the encoding procedure returns an encoding c.*
- $\mathscr{C}(x)\leftarrow\mathsf{Eval}(\Gamma, c)$*: the evaluation procedure receives as inputs a garbled circuit and an encoding of x, returning $\mathscr{C}(x)$.*

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

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

Yao's Garbled Circuit [18]**.** One of the most pre-eminent types of garbling schemes is represented by the original proposal of Yao, which considers a family of circuits of n 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 $i \in [n]$. The evaluation of the circuit at point x corresponds to an evaluation of $\mathsf{Eval}(\Gamma, (L_1^{x_1}, \ldots, L_n^{x_n}))$, where x_i is the i^{th} bit of x,—thus the encoding $c := (L_1^{x_1}, \ldots, L_n^{x_n})$.

2.3 Functional Encryption Scheme - Public-Key Setting

We define the notion of functional encryption[6] 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 4 (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 FULL-SIM-FE *experiment is described in Fig. 1 (left).*

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

$\underline{\mathsf{FULL\text{-}SIM\text{-}LFE}^{\mathcal{A}}_{\mathsf{LFE}}(\lambda)}$:
$b \leftarrow_\$ \{0,1\}$
$(1^k, 1^d) \leftarrow_\$ \mathcal{A}(1^\lambda)$
$\mathsf{crs} \leftarrow_\$ \mathsf{LFE.crsGen}(1^\lambda, 1^k, 1^d)$
$(M^*, \mathscr{C}) \leftarrow_\$ \mathcal{A}(\mathsf{crs})$
$\mathsf{digest}_{\mathscr{C}} \leftarrow_\$ \mathsf{LFE.Compress}(\mathsf{crs}, \mathscr{C})$
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: FULL-SIM-FE-security defined for a functional encryption scheme in the public-key setting. Right: the simulation security experiment FULL-SIM-LFE defined for a laconic function evaluation scheme LFE.

2.4 Laconic Oblivious Transfer

Definition 5 (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:*

- $\mathsf{crs} \leftarrow_\$ \mathsf{crsGen}(1^\lambda)$*: takes as input the security parameter* λ *in unary 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* digest *as well as a state of the database* $\hat{\mathbf{D}}$*.*
- $\mathsf{CT} \leftarrow_\$ \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}, \ell, M_0, M_1)$*: the randomized encryption algorithm takes as input the common reference string* crs*, the digest* 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 *and 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:*

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

- ***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:*

$$\left| \Pr\left[\mathcal{A}(\mathsf{crs}, \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}, \ell, M_0, M_1))\right] - \Pr\left[\mathcal{A}(\mathsf{crs}, \mathcal{S}(\mathsf{crs}, \mathbf{D}, \ell, M_{\mathbf{D}[\ell]}))\right] \right| \in \text{NEGL}(\lambda)$$

2.5 Laconic Function Evaluation

We described the motivation behind LFE in Sect. 1. We proceed with its definition from [14].

Definition 6 (Laconic Function Evaluation [14]). *A laconic function evaluation scheme* LFE *for a class of circuits* $\mathfrak{C}_\lambda$ *consists of four algorithms* (crsGen, 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 FULL-SIM-LFE *is defined in Fig. 1 (right side).*

– ***Laconic outputs****: As per [14, p 13–14], 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, encryption runtime and common reference string.*

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 [3].

3.1 The AR17 Construction for NC^1

In [3], the authors provided a functional encryption scheme supporting general circuits and a bounded number of functional keys. The first distinctive feature is represented by the supported class of functions, which are now described by arithmetic, rather than Boolean circuits. This is beneficial, as arithmetic circuits natively support multiple output bits. Second, the (input-dependent) ciphertext's size in their construction is succinct [3, Appendix E], 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 below.

Regev Encodings. We commence by recalling a simple symmetric encryption scheme due to Brakerski and Vaikuntanathan [7]. 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 [3] to recursively support levels of encodings. Henceforth, we use the name "Regev encoding" for the following map between rings $\mathcal{E}^i \colon \mathcal{R}_{p_{i-1}} \to \mathcal{R}_{p_i}$, 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 and unless stated, we write as a *superscript* of a variable the *level* to which it is 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 $\left\{\mathcal{E}^1(M_i) |\ M_i \in \mathcal{R}_{p_0} \wedge i \in [k]\right\}$, where $\mathcal{E}^1$ is the encoding mapping $\mathcal{E}^1 \colon \mathcal{R}_{p_0} \to \mathcal{R}_{p_1}$ defined in Eq. (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 Fig. 2.

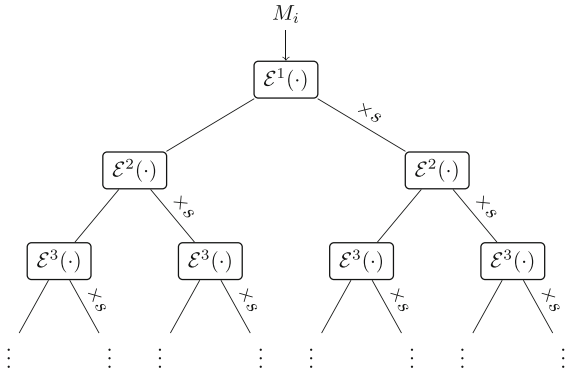

Fig. 2. The tree encoding M_i in a recursive manner corresponds to ciphertext CT_i.

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 $M_1, \ldots, M_k$. 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 $\eta \leftarrow_{\chi} \mathcal{R}_{p_d}$ 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 $\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^{d-1}\}$ 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 $\mathsf{CT}_s := \left\{\mathcal{E}^1(s), \mathcal{E}^2(s^2), \ldots, \mathcal{E}^d(s^2)\right\}$, where $s^2 := s \cdot s$, as opposed to a level superscript.

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 $\left\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^d\right\}$, 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 $\left\{\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^d\right\}$. 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 $\mathsf{PK}^i_{\mathscr{C}}$ 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 $\mathscr{C}^i(\mathbf{x})$[1] 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 *full* description of the evaluation algorithms and refer the reader to [3], but provide a *summary* after presenting decryption.

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: $(\mathrm{mod}\ p_{d-1})\ \ldots\ (\mathrm{mod}\ p_0)$. We note that correctness should follow from the description of these algorithms.

Ciphertext and Public-Key Evaluation Algorithms Let $\mathscr{C}^n$ be the circuit $\mathscr{C}$ restricted to some level n. For a better understanding of the procedures, we will denote the encoding of $\mathscr{C}^n(\mathbf{x})$ at some gate ℓ by c^n_ℓ, 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 as part of the ciphertext, that enable the decryptor to compute c^n.

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

$\underline{\mathsf{Eval}^n_{\mathsf{PK}}(\cup_{t\in[n]}\mathsf{PK}(C^t),\ell)}$ computes the label for 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), define $\mathsf{PK}(c^1_\ell) := \mathsf{PK}(a^1_\ell \cdot s + p_0 \cdot \eta^1_\ell + M_\ell) := a^1_\ell$.
   - Otherwise, let $u^{n-1}_i \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{PK}}(\cup_{t\in[n-1]}\mathsf{PK}(C^t), i)$ and $u^{n-1}_j \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{PK}}(\cup_{t\in[n-1]}\mathsf{PK}(C^t), j)$. Set $\mathsf{PK}(c^n_\ell) := u^{n-1}_i + u^{n-1}_j$.
2. Multiplication Level:
   - If $n=2$ (base case), then compute $\mathsf{PK}(c^2_\ell) := a^1_i \cdot a^1_j \cdot \mathsf{PK}(\mathcal{E}^2(s^2)) - a^1_j \cdot \mathsf{PK}(\mathcal{E}^2(c^1_i \cdot s)) - a^1_i \cdot \mathsf{PK}(\mathcal{E}^2(c^1_j \cdot s))$.
   - Otherwise, let $u^{n-1}_i \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{PK}}(\cup_{t\in[n-1]}\mathsf{PK}(C^t), i)$ and $u^{n-1}_j \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{PK}}(\cup_{t\in[n-1]}\mathsf{PK}(C^t), j)$. Set

$$\mathsf{PK}(c^n_\ell) := u^{n-1}_i \cdot u^{n-1}_j \cdot \mathsf{PK}(\mathcal{E}^n(s^2)) - u^{n-1}_j \cdot \mathsf{PK}(\mathcal{E}^n(c^{n-1}_i \cdot s)) - u^{n-1}_i \cdot \mathsf{PK}(\mathcal{E}^n(c^{n-1}_j \cdot s)) \ .$$

$\underline{\mathsf{Eval}^n_{\mathsf{CT}}(\cup_{t\in[n]}C^t,\ell)}$ 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, set $c^1_\ell := \mathcal{E}^1(M_\ell)$.
   - Otherwise, let $c^{n-1}_i \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{CT}}(\cup_{t\in[n-1]}C^t, i)$ and $c^{n-1}_j \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{CT}}(\cup_{t\in[n-1]}C^t, j)$. Set $c^n_\ell \leftarrow c^{n-1}_i + c^{n-1}_j$.
2. Multiplication Level:
   - If $n=2$ (base case), then set $c^2_\ell := c^1_i \cdot c^1_j \cdot \mathcal{E}^2(s^2) - u^1_j \cdot \mathcal{E}^2(c^1_i \cdot s) - u^1_i \cdot \mathcal{E}^2(c^1_j \cdot s)$.
   - Else: $c^{n-1}_i \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{CT}}(\cup_{t\in[n-1]}C^t, i), c^{n-1}_j \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{CT}}(\cup_{t\in[n-1]}C^t, j)$, $u^{n-1}_i \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{PK}}(\cup_{t\in[n-1]}\mathsf{PK}(C^t), i), u^{n-1}_j \leftarrow \mathsf{Eval}^{n-1}_{\mathsf{PK}}(\cup_{t\in[n-1]}\mathsf{PK}(C^t), j)$. Set $\mathsf{CT}^n_\ell := c^{n-1}_i \cdot c^{n-1}_j + u^{n-1}_i \cdot u^{n-1}_j \cdot \mathcal{E}^n(s^2) - u^{n-1}_j \cdot \mathcal{E}^n(c^{n-1}_i \cdot s) - u^{n-1}_i \cdot \mathsf{PK}(\mathcal{E}^n(c^{n-1}_j \cdot s))$.

4 Laconic Function Evaluation for $\mathbf{NC^1}$ Circuits

We show how to instantiate an LFE protocol starting from the AR17 scheme described above. Furthermore, we show our construction achieves adaptive security (Definition 6) under RLWE (Definition 2) with polynomial approximation factors. Finally, we compare its efficiency to the scheme for general circuits proposed in [14] in Sect. 4.2.

4.1 LFE for $\mathbf{NC^1}$ 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 [3], 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 Sect. 3.1. Note that Bob does not need to access the ciphertext-independent part (the vector $\mathbf{w}$ from Sect. 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 [3], 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 [3]). *Let FE denote the functional encryption scheme for NC^1 circuits proposed in [3].*

- $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{crsGen}(1^\lambda, 1^k, 1^d)$*: the* crs *is instantiated by first running*[2] $\mathsf{FE.Setup}$

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

  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:*

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

[2] Note that we need only a part of the mpk generated by $\mathsf{FE.Setup}$, but for simplicity we call this procedure and drop the unneeded part.

– $\mathsf{CT} \leftarrow_{\$} \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_{\mathscr{C}}, M)$*: the encryption algorithm first samples* $s \leftarrow_{\$} \mathcal{R}_{p_d}$*, randomness* R *and computes recursively the Regev encodings*[3] *of each bit:*

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

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:*

$$\left(\underbrace{\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d}_{\mathsf{CT}_a}, \underbrace{\mathsf{CT}_1, \ldots, \mathsf{CT}_k, \mathsf{CT}_s}_{\mathsf{CT}_b} \right) . \tag{9}$$

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

$$\mathsf{CT}_{\mathscr{C}} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{mpk}, \mathscr{C}, \mathsf{CT}) . \tag{10}$$

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

$$(\mathsf{CT}_{\mathscr{C}} - \mathsf{CT}_a) \mod p_{d-1} \ldots \mod p_0 .$$

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:

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d + \ldots + p_0 \cdot \eta^1 + \mathscr{C}(M) . \tag{11}$$

Given the structure of the evaluated ciphertext in (11) as well as the first part of the ciphertext described by (9), the decryptor obtains $\mathscr{C}(M)$ as per the decryption procedure. □

Lemma 1 (Security). *Let* FE *denote the* FULL-SIM-FE*-secure functional encryption scheme for* NC^1 *circuits described in [3]. 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.

[3] Note that we only need the Regev encodings, but for simplicity of notation, we call the FE.Enc procedure and drop the unneeded part within the FE ciphertext.

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 forming CT_b in Eq. (9) uniformly at random.
- Replaces the functional-key surrogate value $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$ with $\mathsf{Eval}_{\mathsf{CT}} - \eta^* - \mathscr{C}(M^*)$. Observe this is equivalent to running $\mathsf{Eval}_{\mathsf{CT}}$ with respect to random Regev encodings, and subtracting a lower noise term 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 FULL-SIM-LFE experiment.

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

$$\mathsf{SD}\Big(\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d, \mathsf{Eval}_{\mathsf{CT}}(\mathsf{CT}_b) - \eta^* - \mathscr{C}(M^*)\Big) \in \textsc{Negl}(\lambda) \ .$$

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 $d - i$ 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 $i \in \{0, 1, \ldots, d-1\}$.

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

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

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:

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

We can apply the result in [3, p. 27]: we sample the noise term η^* such that

$$\mathsf{SD}(\eta^* \ , \ \sum_{i=0}^{d-2} p_i \cdot \mu^{i+1}) \leq \epsilon \ .$$

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:

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

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 Fig. 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 as a special case, the (input) 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 subtracting 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: $\mathsf{Adv}^{\mathsf{FULL\text{-}SIM\text{-}LFE}}_{\mathcal{A},\mathsf{LFE}}(\lambda) \leq d \cdot \mathsf{Adv}^{\mathsf{RLWE}}_{\mathcal{A}'}(\lambda)$. □

In Sect. 5, we show how the digest can be further compressed.

4.2 Efficiency Analysis

A main benefit of the LFE scheme in Definition 7 is the simplicity of the common reference string, digest and ciphertext structures. These are essentially elements over known quotient rings of polynomial rings. The representation size of elements within rings $\mathcal{R}_{p_i}$ are decided by the prime factors $p_0, p_1, \ldots, p_d$. As stated in [3, 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[4]. However, we observe that a tighter bound may have the following form: $p_d \in O(B_1^{2^{\mathsf{Mul}}})$, where d_{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 [3] that provides guidelines for the size of the primes $p_0, p_1, \ldots, p_d$. We are only interested in the case stipulating that $|p_d|$ belongs to $\mathsf{poly}(\lambda)$. The aforementioned constraint can be achieved by imposing further restrictions on the multiplicative depth of the circuit, namely

$$d_{\mathsf{Mul}} \in O(\log(\mathsf{poly}(\lambda))),$$

meaning that the digest is short enough whenever the circuit belongs to the NC^1 class. Once we obtained a succinctness constraint for the size of elements in the fields $\mathbb{F}_{p_i}$ over which the coefficients of quotient ring elements are going to be sampled, we can proceed with an asymptotic analysis of the sizes of digest, common reference string and ciphertext.

$\mathsf{digest}_{\mathscr{C}}$ **size:** consists of n elements, each belonging to $O(B_1^{2^{d_{\mathsf{Mul}}}})$, where n denotes the degree of the quotient polynomial in the RLWE definition (Definition 2). Henceforth, $\mathsf{digest}_{\mathscr{C}} \in \mathsf{poly}(\lambda)$ given the analysis above on the size of p_d.

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

CT **size:** consists of k "tree-like" structures (Fig. 2), k standing for input length; each tree is a perfect binary tree having of 2^d-1 ring elements. Therefore, the ciphertext's size is upper bounded $k \cdot 2^d \cdot (n \cdot \mathsf{poly}(\lambda))$ and fulfils succinctness requirements whenever $d_{\mathsf{Mul}} \in O(\log(\mathsf{poly}(\lambda)))$.

Enc **runtime:** the encryption runtime is essentially proportional to the size of the ciphertext, up to a constant factor needed to perform inner operations when each node of the tree is built.

crs **size:** the crs has the size identical to the size of the ciphertext, and inherits similar succinctness properties.

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 $\lfloor \log_2(p_0) \rfloor = \ell$. 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.

Remark 1. Another potentially beneficial point is the ability of the scheme in Definition 7 (and also Definition 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 $\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d$.

Comparison with [14]. Reflecting on the construction presented by Quach *et al.*, one can note that its core stems from the attribute-based encryption[17] scheme in [5], and the post-processing steps of their attribute-based LFE rely on techniques from [10]; Quach *et al.* provide a comparison between their AB-LFE and the underlying ABE; the size of their crs consists of k LWE matrices, where k is the input length, each entry belonging to $\mathbb{Z}/q\mathbb{Z}$. The size of q can be tracked by inspecting $\mathsf{Eval}_{\mathsf{PK}}$ ([14, Claim 2.4] and [5, p. 6, 23]): as the noise grows exponentially with the depth of the circuit, q has to grow accordingly: $q \approx 2^{\mathsf{poly}(\lambda,d)}$; in [3] the size of p_d grows exponentially with the multiplicative depth of the circuit, which limits the circuit class our scheme can support.

5 Further Compression for 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 Sect. 4.1. The main idea consists in using a laconic oblivious transfer (Definition 5) in order to generate a small digest.

The bulk of the idea is to observe that the encryption algorithm in Sect. 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 [18] 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 [14]) 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 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 Sect. 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 $(\mathsf{crs}_{\mathsf{LOT}}, \mathsf{crs})$[5], the encryption algorithm first samples $s \leftarrow_{\$} \mathcal{R}_{p_d}$, randomness R and computes recursively the Regev encodings of each bit:*

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

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

$$\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d \tag{12}$$

  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:

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

[5] Even if we omit that explicitly, note that crs includes the bulk of mpk in [3], consistently with the execution of the subroutine $\mathsf{LFE.Setup}$.

and the labels are encrypted under LOT*:*

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

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:*

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

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:*

$$\mathsf{CT}_{\mathscr{C}} \leftarrow \mathsf{Eval}_{\mathsf{CT}}(\mathsf{mpk}, \mathscr{C}, \mathsf{CT}) \ . \tag{13}$$

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

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

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 $\mathscr{C}(M)$.

□

In the remaining part, we prove the scheme above achieves simulation security, assuming the same security level from the underlying primitive.

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

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

*Proof. (*Theorem 2*).* We prove the scheme enjoys adaptive security. Our proof makes use of the LFE simulator, and of the simulation security of the garbling protocol and of the LOT scheme.

Simulator. Our simulator $\mathcal{S}_{\overline{\mathsf{LFE}}}$ is obtained in the following manner: first, we run the simulator of the underlying LFE scheme that will provide the bulk of the ciphertext. Independently, we run the simulator of the garbled circuit $\mathcal{S}_{\mathsf{GS}}$. In the end, we employ the simulator of the LOT scheme ($\mathcal{S}_{\mathsf{LOT}}$) on the labels obtained from the $\mathcal{S}_{\mathsf{GS}}$.

The proof presented herein follows from a hybrid argument. The games are described below, and the game hops are motivated afterwards:

Game_0: corresponds to the FULL-SIM-LFE experiment, having the 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 proceed with the following change: we employ the garbled scheme simulator GS for generating the labels corresponding to the second component contained within the ciphertext. The game distance to the previous one is bounded by the simulation security of the garbling scheme.
Game_3: we switch the main part of ciphertext to one provided by the $\mathcal{S}_{\mathsf{LFE}}$ simulator, the distance to the previous hybrid being bounded by the advantage in Lemma 1.

Claim (Transition between Game_0 *and* Game_1*).* The advantage of any PPT adversary to distinguish 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 $\left(\mathsf{PK}_{\mathscr{C}} \cdot s + p_{d-1} \cdot \eta^d, \{L_{i,i0}, L_{i,1}\}_{i \in |\mathsf{digest}_{\mathscr{C}}|}\right)$.

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 by:

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

Proof ($\mathsf{Game}_1 \to \mathsf{Game}_2$*).* By the input and circuit privacy of the garbling scheme, 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 $\overline{\mathsf{LFE}}$ 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 experiment. □

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

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

Proof ($\mathsf{Game}_2 \to \mathsf{Game}_3$*).* During the last hop, we rely on the security of the LFE component for changing elements of the first component of the ciphertext to simulated ones. It is easy to see that the advantage any adversary has in distinguishing the transition is bounded by the advantage of winning the $\mathsf{FULL\text{-}SIM\text{-}LFE}$ security experiment against 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 $\mathcal{A}_3$.

Finally, we remark that this setting simulates exactly the $\mathsf{FULL\text{-}SIM\text{-}LFE}$ experiment with $b = 1$. □

This completes the proof of Theorem 2. □

6 Concluding Remarks

As for the concluding remarks, we note that our work introduces new constructions of laconic function evaluation for the NC^1 class. The schemes we propose follow from the FE scheme in [3] and exploit standard cryptographic tools, such as laconic oblivious transfer and garbled circuits in order to further reduce the size of digest.

As open questions, apart from having adaptive security for general circuits, we would like to see a candidate having a short common reference string, as our parameters are impractical for high-depth circuits. Another fruitful research direction would investigate if schemes fulfilling the syntax of special homomorphic encodings with succinct ciphertexts can be generically converted into LFEs.

Acknowledgements. The author is grateful to Shweta Agrawal for numerous valuable comments and improvement suggestions on this work. Most work was done while the author was affiliated with the University of Luxembourg. The author was supported in part by the ERC grant CLOUDMAP 787390.

References

1. Agrawal, S.: Stronger security for reusable garbled circuits, general definitions and attacks. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. Part I. LNCS, vol. 10401, pp. 3–35. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_1

2. Agrawal, S., Libert, B., Stehlé, D.: Fully secure functional encryption for inner products, from standard assumptions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. Part III. LNCS, vol. 9816, pp. 333–362. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_12
3. Agrawal, S., Rosen, A.: Functional encryption for bounded collusions, revisited. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. Part I. LNCS, vol. 10677, pp. 173–205. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_7
4. Badrinarayanan, S., Jain, A., Ostrovsky, R., Visconti, I.: UC-secure multiparty computation from one-way functions using stateless tokens. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. Part II. LNCS, vol. 11922, pp. 577–605. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_21
5. Boneh, D., et al.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_30
6. Boneh, D., Sahai, A., Waters, B.: Functional encryption: definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19571-6_16
7. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_29
8. Cho, C., Döttling, N., Garg, S., Gupta, D., Miao, P., Polychroniadou, A.: Laconic oblivious transfer and its applications. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. Part II. LNCS, vol. 10402, pp. 33–65. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_2
9. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing, 25–27 May, pp. 218–229. ACM Press, New York (1987)
10. Goldwasser, S., Tauman Kalai, Y., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th Annual ACM Symposium on Theory of Computing, Palo Alto, CA, USA, 1–4 June 2013, pp. 555–564. ACM Press (2013)
11. Halevi, S., Hazay, C., Polychroniadou, A., Venkitasubramaniam, M.: Round-optimal secure multi-party computation. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. Part II. LNCS, vol. 10992, pp. 488–520. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_17
12. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_1
13. O'Neill, A.: Deterministic public-key encryption revisited. Cryptology ePrint Archive, Report 2010/533 (2010). http://eprint.iacr.org/2010/533
14. Quach, W., Wee, H., Wichs, D.: Laconic function evaluation and applications. In: Thorup, M. (ed.) 59th Annual Symposium on Foundations of Computer Science, Paris, France, 7–9 October 2018, pp. 859–870. IEEE Computer Society Press (2018)
15. Rabin, M.O.: How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187 (2005). http://eprint.iacr.org/2005/187
16. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th Annual ACM Symposium on Theory of Computing, Baltimore, MA, USA, 22–24 May 2005, pp. 84–93. ACM Press (2005)

17. Waters, B.: Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 53–70. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19379-8_4
18. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, 3–5 November 1982, pp. 160–164. IEEE Computer Society Press (1982)
19. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Ontario, Canada, 27–29 October 1986, pp. 162–167. IEEE Computer Society Press (1986)