# Succinct Randomized Encodings from Non-compact Functional Encryption, Faster and Simpler

Nir Bitansky<sup>1</sup> and Rachit Garg<sup>1</sup>

<sup>1</sup>New York University nbitansky@gmail.com, rg5134@cims.nyu.edu

#### **Abstract**

Succinct randomized encodings allow encoding the input x of a time-t uniform computation M(x) in sub-linear time o(t). The resulting encoding  $\tilde{x}$  allows recovering the result of the computation M(x), but hides any other information about x. Such encodings are known to have powerful applications such as reducing communication in MPC, bootstrapping advanced encryption schemes, and constructing time-lock puzzles.

Until not long ago, the only known constructions were based on indistinguishability obfuscation, and in particular they were not based on standard post-quantum assumptions. In terms of efficiency, these constructions' encoding time is polylog(t), essentially the best one can hope for. Recently, a new construction was presented based on Circular Learning with Errors, an assumption similar to the one used in fully-homomorphic encryption schemes, and which is widely considered to be post-quantum resistant. However, the encoding efficiency significantly falls behind obfuscation-based scheme and is  $\approx \sqrt{t} \cdot s$ , where s is the space of the computation.

We construct, under the same assumption, succinct randomized encodings with encoding time  $\approx t^{\varepsilon} \cdot s$  for arbitrarily small constant  $\varepsilon < 1$ . Our construction is relatively simple, generic and relies on any non-compact single-key functional encryption that satisfies a natural *efficiency preservation* property.

# Contents

1	Introduction		
	1.1	This Work	
	1.2	Technical Overview	:
2	Preliminaries		
	2.1	SREs for Space Bounded Computations and Repeated Circuits	
	2.2	Single-key Functional Encryption for Repeated Circuits	,
	2.3	Garbled Circuits	9
3	Boosting Succinctness		9
	3.1	Analysis	1
A	Con	struction in [HLL23] satisfies Definition 2.7	2:

#### 1 Introduction

Succinct randomized encodings (SREs) [BGL<sup>+</sup>15, CHJV15, KLW15] are a specific form of randomized encodings of functions [IK00]. SREs allow to encode an input x for a uniform (e.g. Turing machine) function computation f(x) so that, computing the encoding  $\tilde{x}$  can be done much faster than computing f(x). The encoding  $\tilde{x}$  still allows computing f(x) (in roughly the same time as the original computation), but hides any other information about the input x. Here hiding is captured by requiring that  $\tilde{x}$  can be simulated from the output f(x) (as well as the public description of f).

The combination of fast encodings and input privacy makes SREs extremely powerful. They can be used to privately offload heavy computations to powerful parties, yielding for instance non-interactive delegation, multi-party computation with minimal communication; They can be used to bootstrap cryptographic schemes for circuits into ones for uniform machines, for example KDM encryption, functional encryption, and indistinguishability obfuscation; and they can be used to delay computation, yielding time-lock puzzles (c.f. [App11, CHJV15, KLW15, BGJ<sup>+</sup>16, AMZ24]).

**Existing Constructions.** The concurrent works of [BGL<sup>+</sup>15, CHJV15, KLW15] introduced the notion of SRE and gave constructions based on indistinguishably obfuscation and one-way functions. Unlike the work of [KLW15], the works of [BGL<sup>+</sup>15, CHJV15] only construct so called *semi-succinct* randomized encodings where the encoding time does not grow with the time t of the computation, but *does grow with the space s* of the computation (as such, they are meaningful for space-bounded computations where  $s \ll t$ ). On the other hand, these latter two works can be based on indistinguishability obfuscation with logarithmic-size input, which by now can be based on standard polynomial assumptions [JLS21]. The works of [AL18, GS18] then augmented the approach in [BGL<sup>+</sup>15, CHJV15] and achieved a (fully) succinct construction from the same assumptions.

While since the breakthrough of [JLS21], constructions of indistinguishability obfuscation from standard assumptions are advancing [JLS22, RVV24], they can still be based on very few assumptions, and in particular rely on bilinear group assumptions, which are quantumly broken. Accordingly, constructing SREs from new assumptions, and in particular *post-quantum* ones is a valuable goal. The recent work of [AMZ24] made significant progress on this front. Based on a Circular Learning with Errors Assumption [HLL23], similar to that used in fully-homomorphic encryption constructions [BGV12, GSW13], they construct semi  $\frac{1}{2}$ -succinct randomized encodings. Here the time to encode is roughly  $t^{1/2} \cdot s$ , where t and s are the time and space of the encoded computation.

#### 1.1 This Work

**Theorem 1.1** (Informal). Assume the existence of (non-compact) efficiency-preserving single-key functional encryption. Then for any constant  $\varepsilon < 1$ , there exists a semi  $\varepsilon$ -succinct randomized encoding. Specifically, for a Turing machine computation M(x) of time t and space s, and security parameter  $\lambda$ , the encoding time is  $t^{\varepsilon} \cdot (|x| + s) \cdot \text{poly}(|M|, \lambda)$ . In particular, such randomized encodings exist assuming Circular LWE.

This yields analogous results for the above mentioned applications of SREs. In particular, following the methodology of [BGJ<sup>+</sup>16] this gives the first time-lock puzzles [RSW96] with an arbitrary polynomial (as opposed to quadratic) delay under a standard post-quantum assumption (Circular LWE) and the existence of worst-case hard non-paralleling languages that can be decided in bounded-space, which is arguably a mild complexity assumption (follows for instance from the hardness of parallelizing iterated hashing).

**Simple and Generic.** Our construction follows a generic and relatively simply succinctness boosting step, which we then apply an arbitrary constant number of times to increase succinctness.

Considering a single iteration of this step yields an alternative to the semi 1/2-succinct construction of [AMZ24], which generically relies on any non-compact single-key functional encryption, even without efficiency preservation.<sup>1</sup> This is in contrast to the construction in [AMZ24], which relies on specific FHE-related algebraic techniques.

#### 1.2 Technical Overview

We now explain the main ideas behind our result.

**Recalling Functional Encryption.** We start by recalling in more detail the notion of single-key non-compact functional encryption (FE). Here given a function  $f:\{0,1\}^n \to \{0,1\}^n$ , represented by a circuit, we can generate a public encryption key pk and a functional decryption key sk, such that an encryption  $\operatorname{Enc}_{\operatorname{pk}}(x)$  can be decrypted using sk and the circuit f to yield the function output f(x). Furthermore, the encryption  $\operatorname{Enc}_{\operatorname{pk}}(x)$  does not leak any other information about x, in the sense that, even in the presence of sk, the encryption can be efficiently simulated from the output f(x). We also require the following efficiency properties: first, the time to encrypt  $x \in \{0,1\}^n$  is  $n \cdot \operatorname{poly}(\lambda)$ , independently of the circuit complexity of f, and second, the size of the keys pk, sk is  $n \cdot \operatorname{poly}(\lambda)$ , whereas the time to compute them is  $|f| \cdot \operatorname{poly}(\lambda)$ . Here and throughout  $\lambda$  denotes the security parameter and |f| denotes the size of the corresponding circuit.<sup>2</sup>

The seminal work of [GKP<sup>+</sup>13] constructed, under LWE, such schemes where the function key size grows also with the circuit size of f and encryptions grow with the circuit depth. The work of [BGG<sup>+</sup>14] constructed, also under LWE, a scheme where both ciphertexts and keys only scale with the depth (but not the circuit size), and finally the work of [HLL23], under Circular LWE, removes the depth-dependence, leading to a scheme with the desired efficiency. We note that FE schemes as above stand in sharp contrast to compact FE schemes where the function f may have output size  $m \gg n$ , and yet encryption complexity is sublinear in m. Indeed, compact FE schemes already yield the full power of indistinguishability obfuscation [AJ15, BV15].

Having recalled the notion of FE, we now move to describe our construction.

 $<sup>^{1}</sup>$ Efficiency-preservation is then required in order to achieve succinctness beyond 1/2.

<sup>&</sup>lt;sup>2</sup>Indeed, in the variant that we describe, where a single function is known at setup time, some efficiency requirement is necessary to avoid a triviality. The literature often considers a more general variant where keys for different functions can be derived from a master secret key, which will not be needed in this work.

 $\frac{1}{2}$ -Succinctness. We start by describing a construction that achieves  $\frac{1}{2}$ -succinctness. The construction is inspired by that of [AMZ24], and starts from the same basic idea. For a Turing machine M representing a function  $M:\{0,1\}^n \to \{0,1\}^n$ , with space complexity  $s(n) \geq n$ , we consider the corresponding step circuit  $S_M:\{0,1\}^s \to \{0,1\}^s$ , which performs a single step of M and has size  $\approx s$  (ignoring for simplicity poly(|M|) factors). Then, to encode a t-step computation M(x), the basic idea is to derive a public encryption key pk and function key sk for  $f = E \circ S_M^{\sqrt{t}}$ . The function f given an intermediate state  $\mathrm{st}_i \in \{0,1\}^s$ , performs  $\sqrt{t}$  steps of M, and outputs an encryption  $E(\mathrm{st}_{i+\sqrt{t}})$  of the resulting state, under some auxiliary encryption E (to be determined later). In addition, the encoding will include  $\sqrt{t}$  gadgets  $G_1,\ldots,G_{\sqrt{t}}$ , where the role of each  $G_i$  is to convert an encryption  $E(\mathrm{st}_{i\sqrt{t}})$  to an encryption  $\mathrm{Enc}_{pk}(\mathrm{st}_{i\sqrt{t}})$  under the functional encryption, with the exception of the last gadget  $G_{\sqrt{t}}$ , which decrypts  $E(\mathrm{st}_t)$  to obtain the final state  $\mathrm{st}_t$ , encoding the result of the computation. Finally, we will also include a functional encryption  $\mathrm{Enc}_{pk}(\mathrm{st}_0)$  of the initial state  $\mathrm{st}_0$ , which encodes the input x.

Given the above components  $\mathsf{sk}, G_1, \ldots, G_{\sqrt{t}}, \mathsf{Enc}_{pk}(\mathsf{st}_0)$ , an evaluator can execute the machine computation by each time starting from  $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_{i\sqrt{t}})$ , performing functional decryption to obtain  $E(\mathsf{st}_{(i+1)\sqrt{t}})$ , and then using gadget  $G_{i+1}$  to convert it to  $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_{i\sqrt{t}})$ , and so on, until the last gadget application returns the result of the computation. In terms of succinctness, as long as the circuit size of  $E(\mathsf{st} \in \{0,1\}^s)$  is  $\approx s$ , and the complexity generating each gadget  $G_i$  is  $\approx s$ , and since deriving the FE keys takes time  $\approx |E \circ S_M^{\sqrt{t}}| \approx \sqrt{t}s$ , the overall complexity of encoding would be  $\approx \sqrt{t}s$ .

The question is how to instantiate the auxiliary encryption E and gadgets  $G_i$  to yield the above efficiency properties and to be able to prove security. Here one challenge is to make sure that the size of encryptions  $E(\mathsf{st}_i \in \{0,1\}^s)$  remains of fixed size  $\approx s$ , and does not grow with the size of the functional encryption  $\mathsf{Enc}_{pk}(\mathsf{st}_i \in \{0,1\}^s)$ . Indeed, this constraint is an artifact of non-compact FE, where any increase in the size of the function output translates to an increase in the size of functional ciphertexts, which would iteratively blow up the size of the construction. To circumvent this difficulty, the work of [AMZ24] devised a lattice-based solution, which combines techniques from works on *split FHE* [BDGM20], with a notion of range-puncturable pseudo-random functions. We take a different and arguably simpler route.

Our first observation is that there is, in fact, a simple and generic way to instantiate the above approach using Yao's garbled circuits [Yao86]. Specifically, the encryption  $E(\mathsf{st}_{i\sqrt{t}})$  would generate garbled input labels  $\check{\mathsf{st}}_{i\sqrt{t}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_i}(\mathsf{st}_{i\sqrt{t}})$  under a corresponding garbling key  $\mathsf{gsk}_i$ . The gadget  $G_i$  will be a corresponding garbled circuit  $\check{C}_i = \mathsf{GC}.\mathsf{Garble}_{\mathsf{gsk}_i}(C_i)$ , where the circuit  $C_i$  takes as input a state  $\mathsf{st}_{i\sqrt{t}}$  and outputs a functional encryption  $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_{i\sqrt{t}},\mathsf{gsk}_{i+1};r)$ , using hardwired randomness r, where  $\mathsf{gsk}_{i+1}$  is the next garbling key (also hardwired in  $C_i$ ). The function f for which we derive the function key is augmented accordingly, given  $(\mathsf{st},\mathsf{gsk})$ , it executes  $S_M^{\sqrt{t}}(\mathsf{st})$ , and garbles using  $\mathsf{gsk}$  the resulting state.

Indeed, garbled circuits have the feature that when garbling circuits  $C: \{0,1\}^s \to \{0,1\}^{s'}$  where  $s' \gg s$ , computing garbled input labels takes time  $\approx s$  regardless of how big the output size s' is. Accordingly, also the function f for which we derive a function key has fixed output size throughout. In addition, each circuit  $C_i$  performs functional encryption of an input of size  $\approx s$ , which takes time  $\approx s$ , and accordingly the garbled circuit is also of size  $\approx s$  (we ignore fixed factors in the security parameter). Overall, we get the required  $\sqrt{t} \cdot s$  encoding complexity. (Note that so far we have not used any special efficiency properties of functional key generation, except that it takes time  $\approx |f|$  where f is the function for which we derive keys.)

Security follows quite directly by interchangeably applying the functional encryption simulation guarantee and the garbled circuit simulation guarantee. Specifically, since our garbled circuits contain secret information (FE encryption randomness r and next garbling key  $\operatorname{gsk}_{i+1}$ ), we'll rely on circuit-private simulation for garbled circuits where the garbled circuit  $\tilde{C}_i$  and input encoding  $\operatorname{st}_{i\sqrt{i}}$  can be efficiently

simulated from the output  $C_i(\operatorname{st}_{i\sqrt{t}})$  alone. This allows to simulate the garbled circuits backwards, starting from  $\tilde{C}_{\sqrt{t}}$ ,  $\tilde{\operatorname{st}}_{t-\sqrt{t}}$  which can be simulated from the output M(x) of the entire computation. Then we can use the simulated  $\tilde{\operatorname{st}}_{t-\sqrt{t}}$  to simulate the previous FE encryption. The simulated FE encryption, is then thought of as the output of the previous garbled circuit and is used to simulate it, and so on. The validity of this simulation is shown based on a standard hybrid argument.

**Beyond** 1/2-**Succinctness.** Aiming to improve the succinctness of our randomized encoding, a natural idea is to simply compose it with itself. That is,  $\operatorname{Encode}(M,x,t)$  which randomly encodes (M,x,t) is a uniform computation on its own of time  $\approx \sqrt{t}s$ . So let us randomly encode this computation, namely compute  $\operatorname{Encode}(\operatorname{Encode}(M,\cdot,t),x,\sqrt{t}s)$  (hardwiring the required randomness). If the space of this computation is s', we would get encoding time  $t^{\frac{1}{4}}\sqrt{s}s'$ , and if s' is not prohibitively large, then we would gain in succinctness. This, however is too good to be true; in particular, something that we have implicitly assumed in the previous solution is that the space of the underlying computation, in this case s', is at least as large as the output, which in this case is the underlying randomized encoding  $\operatorname{Encode}(M,x,t)$  of size  $\sqrt{t}s$ . Thus we have so far gained nothing.<sup>3</sup>

However, it turns out that this naïve composition idea is not completely useless. As previously noted, the 1/2-succinct solution does not take full advantage of the efficiency properties of the FE key generation procedure. In particular, looking back at the previous encoding

$$\operatorname{sk}, \tilde{C}_1, \dots, \tilde{C}_{\sqrt{t}}, \operatorname{Enc}_{pk}(\operatorname{st}_0, \operatorname{gsk}_1)$$
 ,

we note that generating the function secret and public encryption keys (sk, pk) may take time  $\sqrt{t}s$ , but results in a relatively short output (of size  $\approx s$ ). This suggests that we may be able to gain by randomly encoding only the key derivation part, which we indeed do.

By applying this step repeatedly, each time reducing  $t^{1/k}$  to  $t^{1/(k+1)}$ , we can reduce the dependence on t to  $t^{\varepsilon}$  for an arbitrarily small  $\varepsilon$ . In the body, we also show that we can maintain a linear dependence on the space s (instead of  $\approx s^{O(\log(1/\varepsilon)})$ ) by considering throughout all the iterations repeated circuits, instead of moving back and forth between Turing machines and their repeated step circuits. Another technical detail we glossed over is that previously the function encryption key pk was hardwired into the garbled circuits  $\tilde{C}_i$  whereas now it is not available in encoding time (it is only computed online by decoding the randomly encoded key derivation). This can be easily dealt with by letting the garbled circuits propagate pk between them in online time. See details in the body.

<sup>&</sup>lt;sup>3</sup>Indeed, the dependence of SREs on output size is inherent. Simulation-based SREs that are output compressing are impossible [LPST16], and indistinguishability-based output-compressing SREs already pave the way to indistinguishability obfuscation [AJ15].

**Existing Efficiency Preserving FE.** Examining the FE scheme of [HLL23], we observe that it is indeed efficiency preserving. In this scheme, which builds on the (depth-dependent) scheme of [BGG<sup>+</sup>14] and its variant in [QWW18], deriving keys for a circuit  $f: \{0,1\}^s \to \{0,1\}^s$ , essentially mimics GSW-style homomorphic evaluation over s matrices of fixed size, plus local operations meant for noise reduction.

Roughly speaking, homomorphic evaluation of a circuit f translates to a circuit of similar topology where any gate operation is replaced by a small circuit, which is essentially its homomorphic equivalent. Since the added noise-reducing steps are local, namely are performed individually on each homomorphic wire, the final key derivation circuit  $\hat{f}$  is of similar dimensions. Moreover, if we consider a repeated circuit  $f^t$ , then its homomorphic equivalent is also a repeated circuit  $\hat{f}^t$  of similar dimensions. See Appendix A for more details.

#### 2 Preliminaries

Throughout this work, we denote by  $\lambda$  the security parameter. We say a function f is negligible in the security parameter  $\lambda$  if  $f = \lambda^{-\omega(1)}$ . We denote this by writing  $f(\lambda) = \text{negl}(\lambda)$ . We write  $\text{poly}(\lambda)$  to denote a function that is bounded by a fixed polynomial in  $\lambda$ . We say an algorithm is efficient if it runs in probabilistic polynomial time (PPT) in the length of its input. Throughout this work, we consider security against *non-uniform* adversaries (indexed by  $\lambda$ ) that run in *deterministic* polynomial time in the length of their input and takes in an advice string of  $\text{poly}(\lambda)$  size.

For positive integers n < m, we denote by [n] the set  $\{1, \ldots, n\}$  and by n + [m] the set  $\{n + 1, \ldots, n + m\}$ . For some circuit  $C : \{0, 1\}^n \to \{0, 1\}^n$ , we denote t repeated evaluations of the circuit by the notation  $C^t$ , i.e., for all inputs  $x \in \{0, 1\}^n$ ,  $C^t(x) = C(C^{t-1}(x))$ , where  $C^1 = C$ . Abusing notation, we write  $M^t(x)$  to denote the execution of a Turing machine M on input x for t steps. We denote the first t bits of a string  $t \in \{0, 1\}^{k+\ell}$ , by the notation  $t \in \{0, 1\}^{k+\ell}$ , the notation  $t \in \{0, 1\}^{k+\ell}$  and  $t \in \{0, 1\}^{k+\ell}$ .

We next review the main cryptographic primitives we use in this work.

#### 2.1 SREs for Space Bounded Computations and Repeated Circuits

A succinct randomized encoding (SRE) scheme [BGL<sup>+</sup>15, CHJV15, KLW15] allows to encode a uniform computation M(x) fast, independently of the computation's complexity, while hiding the input x. We consider a restricted notion of SRE where the encoding procedure is sub-linear in the time of computation t but is allowed to scale with the space of the computation s. This notion is accordingly meaningful for space-bounded computations, where  $s \ll t$ .

**Definition 2.1** (Succinct Randomized Encodings for Space-Bounded Machines). A succinct randomized encoding for space-bounded machines consists of two algorithms SRE = (SRE.Encode, SRE.Eval) with the following syntax:

Encode  $(1^{\lambda}, M, x, t, s) \to \widetilde{x}$ . The encode algorithm takes the security parameter  $\lambda$ , Turing Machine (TM) M, input x, time bound t, and space bound s, and outputs an encoding,  $\widetilde{x}$ .

Eval $(M, \widetilde{x}, t, s) \to y$ . The evaluation algorithm takes in an encoding  $\widetilde{x}$ , Turing machine M, time bound t, and space bound s, and outputs an evaluation y.

The scheme should satisfy the following properties:

• **Correctness:** For any polynomial  $\ell(\lambda)$ , there exists a negligible function negl such that for all  $\lambda \in \mathbb{N}$  and for all Turing machines M with space bound s, inputs x, and number of steps t, such that  $|M|, |x|, t, s \le \ell(\lambda)$  we have,

$$\Pr\left[\mathsf{M}^t(x) \neq y: \begin{array}{c} \widetilde{x} \leftarrow \mathsf{Encode}(1^\lambda, \mathsf{M}, x, t, s) \\ y \leftarrow \mathsf{Eval}(\mathsf{M}, \widetilde{x}, t, s) \end{array}\right] = \mathsf{negl}(\lambda),$$

where the randomness is over the coins of Encode

• **Security:** There exists a polynomial-time simulator Sim such that for all  $\lambda \in \mathbb{N}$ , polynomially bounded t, s, Turing machines M with polynomial-size description, and polynomial-size inputs  $x \in \{0, 1\}^n$ ,

$$\operatorname{Sim}\left(1^{\lambda}, M, M(x), t, s, n\right) \approx_{c} \operatorname{Encode}(1^{\lambda}, M, x, t, s).$$

- **Semi**  $\varepsilon$ -succinctness: There exists a polynomial poly such that for any  $\lambda$ , M, x, s and  $t \le 2^{\lambda}$ , the running time of  $\operatorname{Encode}(1^{\lambda}, M, x, t, s)$  is at most  $t^{\varepsilon} \cdot \operatorname{poly}(|M|, |x|, s, \lambda)$ .
- **Semi-efficient evaluation:** There exists a polynomial poly such that for any  $\lambda$ , M, x, s and  $t \le 2^{\lambda}$ , and for any  $\widetilde{x}$  in the support of  $\text{Encode}(1^{\lambda}, M, x, t, s)$ , the running time of  $\text{Eval}(M, \widetilde{x}, t, s)$  is at most  $t \cdot \text{poly}(|M|, |x|, s, \lambda)$ .

**Remark 2.2** (Semi-efficient evaluation). The above semi-efficient evaluation property can be naturally relaxed to allow a fixed polynomial blowup also in t. However, in some contexts, evaluation efficiency is a measure one tries to optimize. For example, in the context of delegation, it affects the efficiency of the server to which the computation is delegated. In the context of time-lock puzzles, it translates to the running time of the honest decryptor, and is particularly of interest, in order to tighten the relation between the time it takes to honestly solve the puzzle and the time required to maliciously solve the puzzle.

To guarantee semi-efficient evaluation, we shall also require corresponding efficient decryption/evaluation from the functional encryption. (All of these can be relaxed, to allow some fixed polynomial blowup.)

**Remark 2.3.** We require SRE that is only input-hiding, namely hides x, but not necessarily machine hiding, namely it may not hide M. This somewhat simplifies the description of our construction and is not essential. Indeed, machine hiding could be generically achieved by considering a universal Turing machine and treating the Turing machine M as part of the input.

**Remark 2.4** (Simulator Efficiency). Above, we do not make any explicit requirements on the efficiency of the simulator. In fact, even an unbounded simulator gives a meaningful notion of indistinguishability-based SRE that suffices for the applications mentioned in the intro. Since we allow the complexity of encoding to grow with s and the output size of the computation  $m \le s$ , there is a simulator that runs essentially at the same time of the encoder, encoding a dummy computation that outputs the intended output y.

**SRE for repeated circuits.** We also define a notion of SRE for repeated circuits. Here the goal is to encode a computation of a repeated circuit  $C^t$ , for some  $C: \{0,1\}^n \to \{0,1\}^n$  in time that is sub-linear in t. As we note later such SREs in particular imply corresponding SRE for space-bounded Turing machines. However, using SREs for repeated circuits will make it easier to obtain better dependence on the space in our final construction.

**Definition 2.5** (Succinct Randomized Encodings for Repeated Circuits). A succinct randomized encoding for repeated circuits consists of two algorithms SRE = (SRE.Encode, SRE.Eval) with the following syntax:

Encode  $(1^{\lambda}, C, x, t) \to \widetilde{x}$ . The encode algorithm takes the security parameter  $\lambda$ , circuit C, input x, number of repetitions t and outputs an encoding,  $\widetilde{x}$ .

Eval $(C, \widetilde{x}, t) \to y$ . The evaluation algorithm takes in an encoding  $\widetilde{x}$ , circuit C, number of repetitions t, and outputs an evaluation y.

The scheme should satisfy the following properties:

• Correctness: For any polynomial  $\ell(\lambda)$ , there exists a negligible function negl such that for all  $\lambda \in \mathbb{N}$  and for all circuits C, inputs x, and number of repetitions t, such that, |C|, |x|,  $t \le \ell(\lambda)$  we have,

$$\Pr\left[C^t(x) \neq y: \begin{array}{c} \widetilde{x} \leftarrow \mathsf{Encode}(1^{\lambda}, C, x, t) \\ y \leftarrow \mathsf{Eval}(C, \widetilde{x}, t) \end{array}\right] = \mathsf{negl}(\lambda),$$

where the randomness is over the coins of Encode.

• **Security:** There exists a polynomial-time simulator Sim such that for all  $\lambda \in \mathbb{N}$ , polynomial-size circuits C, polynomial-size inputs x, and polynomially bounded t,

$$\operatorname{Sim}\left(1^{\lambda}, C, C^{t}(x), t\right) \approx_{c} \operatorname{Encode}(1^{\lambda}, C, x, t).$$

- **Semi**  $\varepsilon$ -succinctness: There exists a polynomial poly such that for any  $\lambda$ , C, x and  $t \le 2^{\lambda}$ , the running time of  $\operatorname{Encode}(1^{\lambda}, C, x, t)$  is at most  $t^{\varepsilon} \cdot \operatorname{poly}(|C|, \lambda)$ .
- **Semi-efficient evaluation:** There exists a polynomial poly such that for any  $\lambda$ , C, x and  $t \le 2^{\lambda}$ , and for any  $\widetilde{x}$  in the support of  $\text{Encode}(1^{\lambda}, C, x, t)$ , the running time of  $\text{Eval}(C, \widetilde{x}, t)$  is at most  $t \cdot \text{poly}(|C|, \lambda)$ .

We state a general (straightforward) relation between SREs for repeated circuits and SREs for bounded-space Turing machines. Jumping forward, our SREs for repeated circuits will achieve certain encoding and evaluation, we state how these are reflected when moving between repeated circuits and Turing machines.

**Proposition 2.6.** Assuming there exists semi  $\varepsilon$ -succinct randomized encoding for repeated circuits, there also exists semi  $\varepsilon$ -succinct randomized encoding for bounded space Turing machine.

*Proof sketch.* For a space-bounded Turing machine M computing a function  $\{0,1\}^n \to \{0,1\}^n$ , we assume w.l.o.g  $s(n) \ge n$ . We consider the uniform circuit  $S_M$  of size  $s \cdot \text{poly}(|M|)$ , which performs one transition of M. Then a repeated-circuit encoding of  $(S_M, x, t)$  yields the required Turing machine randomized encoding of (M, x, t, s). □

#### 2.2 Single-key Functional Encryption for Repeated Circuits

We consider public-key functional encryption with a single function key, where the function is known at setup time (this notion can be seen as a public-key analog of reusable garbled circuits). We focus on functions that can be represented by repeated circuits  $C^t$  (jumping ahead we will later consider space-bounded Turing machine computations, which in particular can be described by repeated applications of a transition circuit). Functionality and security for repeated circuits schemes are defined as for general circuits. The repeated-circuit aspect will be relevant in the context of efficiency, where we'll require that key derivation for a repeated circuit is also a repeated circuit with roughly the same circuit size and the same input-output length.

**Definition 2.7.** A single-key functional encryption scheme for repeated circuits consists of PPT algorithms FE = (FE.Setup, FE.Enc, FE.Dec) with the following syntax:

Setup( $1^{\lambda}$ , C, t)  $\rightarrow$  (pk, sk). The setup algorithm takes as input a security parameter  $\lambda$ , a circuit  $C: \{0, 1\}^n \rightarrow \{0, 1\}^n$ , a repetition parameter t, and outputs a public key and secret key (pk, sk).

 $\operatorname{Enc}_{\operatorname{pk}}(x \in \{0,1\}^n) \to \operatorname{ct}$ . The encryption algorithm takes in the public key pk and an input x and outputs a ciphertext ct.

 $\operatorname{Dec}_{\operatorname{sk}}(C,\operatorname{ct},t)\to y$ . The decryption algorithm takes in a secret key sk, an input a circuit C, ciphertext ct, and the repeated parameter t and outputs a value y.

The scheme should satisfy the following properties:

• **Correctness:** For any polynomial  $\ell(\lambda)$ , there exists a negligible function negl such that for any  $\lambda$ , circuit C, repetition parameter t, and inputs x, s.t. |C|, |x|,  $t \le \ell(\lambda)$ ,

$$\Pr \begin{bmatrix} C^t(x) \neq y : & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^{\lambda}, C, t) \\ C^t(x) \neq y : & \mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x) \\ y \leftarrow \mathsf{Dec}_{\mathsf{sk}}(C, \mathsf{ct}, t) \end{bmatrix} = \mathsf{negl}(\lambda).$$

• **Selective security:** There exists a polynomial-time simulator Sim such that for all  $\lambda \in \mathbb{N}$ , all polynomial-size circuits  $C : \{0,1\}^n \to \{0,1\}^n$ , inputs  $x \in \{0,1\}^n$ , and polynomially bounded t,

$$\left\{ \mathsf{pk}, \mathsf{sk}, \mathsf{Sim}_{\mathsf{pk}}(C^{t}(x)) \right\} \approx_{c} \left\{ \mathsf{pk}, \mathsf{sk}, \mathsf{Enc}_{\mathsf{pk}}(x) \right\},$$

where in both distributions (pk, sk)  $\leftarrow$  Setup(1 $^{\lambda}$ , C, t).

• Succinctness: There exists a polynomial  $poly(\lambda)$ , such that for any  $\lambda$ , circuit  $C : \{0, 1\}^n \to \{0, 1\}^n$ , repetition parameter t, keys (pk, sk) in the support of  $Setup(1^{\lambda}, C, t)$ , and ciphertext ct in the support of  $Enc_{pk}(x \in \{0, 1\}^n)$ ,

$$|pk|, |sk|, |ct| \le n \cdot poly(\lambda)$$
.

Furthermore,  $\operatorname{Enc}_{pk}(x \in \{0,1\}^n)$  can be computed by a uniform circuit of size  $n \cdot \operatorname{poly}(\lambda)$ .

• **Efficiency preservation:** There exists a polynomial  $poly(\lambda)$ , such that for any  $\lambda$ , circuit  $C: \{0,1\}^n \to \{0,1\}^n$ , and repetition parameter t, there exists a circuit  $S_C: \{0,1\}^m \to \{0,1\}^m$  such that

for all 
$$r \in \{0, 1\}^m$$
:  $S_C^t(r) = \text{Setup}(1^{\lambda}, C, t; r)$ .

Furthermore,

$$m \le n \cdot \text{poly}(\lambda), \qquad S_C \le |C| \cdot \text{poly}(\lambda),$$

and  $S_C$  can be efficiently computed from C in time  $O(|S_C|)$ .

• **Efficient decryption:** There exists a polynomial poly(|C|,  $\lambda$ ) such that for any  $\lambda$ , circuit  $C: \{0, 1\}^n \to \{0, 1\}^n$ , any input  $x \in \{0, 1\}^n$ , repetition parameter t, where  $n, t \le 2^{\lambda}$ , and for any (pk, sk) in the support of Setup( $1^{\lambda}$ , C, t), any ct in the support of Enc<sub>pk</sub>(x), the running time of Dec<sub>sk</sub>( $\cdot$ ) is at most  $t \cdot \text{poly}(|C|, \lambda)$ .

#### 2.3 Garbled Circuits

We define (the circuit-private version of) garbled circuits [Yao86] with an input-efficient and a decomposable encoding property.

**Definition 2.8** (Garbled Circuits). A circuit garbling scheme consists of the following PPT algorithms GC = (GC.Garble, GC.Enc, GC.Eval) with the syntax:

Garble<sub>gsk</sub>(C)  $\to \widetilde{C}$ . The garble algorithm takes in a garbling secret key gsk  $\in \{0,1\}^{\lambda}$ , circuit  $C:\{0,1\}^n \to \{0,1\}^m$  and outputs a garbled circuit  $\widetilde{C}$ .

 $\operatorname{Enc}_{\operatorname{gsk}}(x_i \in \{0,1\}, i \in [n]) \to \widetilde{x}_i$ . The encryption algorithm takes in a garbling secret key gsk, input bit  $x_i \in \{0,1\}$ , position  $i \in [n]$  and outputs an encoding of the input  $\widetilde{x}_i$ .

Eval  $(\widetilde{C}, \{\widetilde{x}_i\}_{i \in [n]}) \to y$ . The evaluation algorithm takes in the garbled circuit  $\widetilde{C}$  and encoded inputs for each position  $\{\widetilde{x}_i\}_{i \in [n]}$  and outputs y, the evaluation the circuit C on the input x.

The scheme should have the following properties:

• Correctness: For any  $\lambda \in \mathbb{N}$  and for all circuits  $C : \{0,1\}^n \to \{0,1\}^m$ , polynomial in  $\lambda$ , and all valid inputs  $x \in \{0,1\}^n$ ,

$$\Pr \begin{bmatrix} \operatorname{gsk} \leftarrow \{0,1\}^{\lambda} \\ \widetilde{C} \leftarrow \operatorname{Garble}_{\operatorname{gsk}}(C) \\ C(x) = y : & \forall i \in [n], \widetilde{x}_i \leftarrow \operatorname{Enc}_{\operatorname{gsk}}(x_i, i) \\ y \leftarrow \operatorname{Eval}\left(\widetilde{C}, \{\widetilde{x}_i\}_{i \in [n]}\right) \end{bmatrix} = 1.$$

• **Security:** There exists a polynomial-time simulator Sim such that for all  $\lambda \in \mathbb{N}$ , all polynomial-size circuits C and valid inputs x,

$$\left\{ \operatorname{Sim} \left( 1^{\lambda}, 1^{|C|}, 1^{n}, C(x) \right) \right\} \approx_{c} \left\{ \left( \widetilde{C}, \left\{ \widetilde{x} \right\}_{i \in [n]} \right) : \begin{array}{c} \operatorname{gsk} \leftarrow \{0, 1\}^{\lambda}, \\ \widetilde{C} \leftarrow \operatorname{Garble}_{\operatorname{gsk}}(C) \\ \left\{ \widetilde{x}_{i} \leftarrow \operatorname{Enc}_{\operatorname{gsk}}(x_{i}, i) \right\}_{i \in [n]}. \end{array} \right\}$$

• **Efficient encoding:** There is a polynomial poly( $\lambda$ ), such that for any  $\lambda$  and circuit  $C: \{0,1\}^n \to \{0,1\}^m$ , where |C|,  $n \le 2^{\lambda}$ , the input encoder  $\operatorname{Enc}_{\operatorname{gsk}}(\cdot)$  is computable by a uniform circuit of size at most  $\operatorname{poly}(\lambda)$  and  $\operatorname{Garble}_{\operatorname{gsk}}(\cdot)$  is computable by a uniform circuit of size at most  $|C| \cdot \operatorname{poly}(\lambda)$ .

# 3 Boosting Succinctness

In this section, we present a boosting theorem that takes any semi  $\varepsilon$ -succinct SRE, for constant  $\varepsilon \le 1$ , and turns it into an semi  $\varepsilon'$ -succinct SRE for  $\varepsilon' = \varepsilon/(1+\varepsilon)$ , relying on efficiency-preserving single-key FE for repeated circuits.

Before diving into the construction, we present the following (straightforward) lemma that essentially says that a repeated circuit  $X^t$  followed by an application of another circuit Y, can be represented by another repeated circuit  $Z^t$ .

**Lemma 3.1.** Let  $X : \{0,1\}^n \to \{0,1\}^n$  and  $Y : \{0,1\}^{n+m} \to \{0,1\}^k$  be circuits. Then there exists a circuit  $Z : \{0,1\}^{n+m+\lambda} \to \{0,1\}^{n+m+\lambda}$  of size  $O(|X|+|Y|)+\operatorname{poly}(\lambda)$  such that for any  $x,y,t \in \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^{\lambda}$ ,

$$Z^{t}(x, y, t)|_{1...k} = Y(X^{t}(x), y)$$
.

Furthermore, Z is computable from X, Y,  $1^{\lambda}$  in time O(|Z|).

*Proof.* The repeated circuit Z in Fig. 1 is essentially a branching circuit that maintains a counter. For the first t-1 repetitions, the counter is set so that the circuit Z computes a repeated circuit X. On the final t-th repetition, the counter reaches the end and Z computes circuit X followed by circuit Y and outputs the resulting computation.

The circuit Z (in Fig. 1) consists of a branching circuit to check if ind  $\stackrel{?}{=}$  1, circuit description of X for one branch, circuit description of X and Y for the other branch. Thus,  $|Z| \le O(|X|, |Y|) + \text{poly}(\log t)$ , where the polynomial is the size of the branching circuit and does not depend on circuits X, Y.

Finally, for 0 < i < t,  $Z^i(x, y, t)|_{1...k}$  the input index is not equal to 1 and we compute  $(X^i(x), y, t - i)$ . On the *t*-th repetition, the input to the repeated circuit is  $(X^{t-1}(x), y, 1)$  and the circuit computes  $(Y(X^t(x), y), 0^{n+m+\lambda-k})$ .

```
Constants: Circuit descriptions X: \{0,1\}^n \to \{0,1\}^n, Y: \{0,1\}^{n+m} \to \{0,1\}^k

Inputs: x \in \{0,1\}^n, y \in \{0,1\}^m, current index ind \in \{0,1\}^\lambda

Outputs: updated parameters x' \in \{0,1\}^n, y' \in \{0,1\}^m, ind' \in \{0,1\}^\lambda
```

- 1. If ind > 1, compute  $x' \leftarrow X(x)$ . Output updated computation, (x', y, ind 1).
- 2. If ind = 1,
  - (a) Compute  $x' \leftarrow X(x)$ .
  - (b) Compute  $z \leftarrow Y(x', y)$ .
  - (c) Output updated computation,  $(z, 0^{n+m+\lambda-k})^a$ .

Figure 1: Circuit Z[X, Y](x, y, ind)

**Theorem 3.2** (Boosting succinctness of randomized encodings). Assume there exists a semi  $t^{1/c}$ -succinct randomized encoding scheme for some constant  $c \in (0,1]$  (Definition 2.1) and a single-key functional encryption scheme for repeated circuits (Definition 2.7). Then, there exists a semi  $t^{1/c+1}$ -succinct randomized encoding scheme.

**Construction 3.3** (SRE with boosted succinctness). Throughout, let  $\lambda$  be a security parameter. Some of the construction's parameters are set later in the analysis section.

**Ingredients and notation:** We consider a boolean circuit C such that  $C: \{0, 1\}^s \to \{0, 1\}^s$ .

- Let SRE = (SRE.Encode, SRE.Eval) be a  $t^{1/c}$ -succinct randomized encoding scheme according to Definition 2.1 for some constant  $c \in (0, 1]$ .
- Let FE = (FE.Setup, FE.Enc, FE.Dec) be a succinct single key functional encryption scheme according to Definition 2.7.

<sup>&</sup>quot;We assume without loss of generality, we can pad the circuit Y so  $m \ge k$  and hence,  $n + m + \lambda \ge k$ .

- Let the FE.Setup algorithm output  $pk \in \{0,1\}^p$ ,  $sk \in \{0,1\}^q$  and sample from a randomness space  $r \in \{0,1\}^{\ell_s}$ .
- Let the FE.Enc algorithm sample random coins from a space  $r \in \{0, 1\}^{\ell_e}$ .
- Let GC = (GC.Garble, GC.Enc, GC.Eval) be a garbling scheme according to Definition 2.8.
  - − The circuit we wish to garble in our construction takes an input of the form (pk, st) where  $pk \in \{0,1\}^p$ , st ∈  $\{0,1\}^s$ . For concise notation and ease of readability, we simplify our garbling encoding algorithm to write pk ←  $Enc_{gsk}(pk, [p])$ , and st ←  $Enc_{gsk}(st, p + [s])$ , where,

\* 
$$\widetilde{\mathsf{pk}} = \left\{ \widetilde{\mathsf{pk}}_j \right\}_{j \in [p]}$$
 and for every  $j \in [p]$ ,  $\widetilde{\mathsf{pk}}_j \leftarrow \mathsf{Enc}_{\mathsf{gsk}}(\mathsf{pk}_j, j)$ .  
\*  $\widetilde{\mathsf{st}} = \left\{ \widetilde{\mathsf{st}}_j \right\}_{j \in [s]}$  and for every  $j \in [s]$ ,  $\widetilde{\mathsf{st}}_j \leftarrow \mathsf{Enc}_{\mathsf{gsk}}(\mathsf{st}_j, p + j)$ .

- Additionally, let the lengths  $|\widetilde{pk}|$ ,  $|\widetilde{st}|$  be denoted by  $\widetilde{p}$ ,  $\widetilde{s}$ .

**The boosted SRE:** Let the boosted SRE be denoted by (SRE'.Encode, SRE'.Eval). Our construction of these algorithms is mentioned below.

Algorithm SRE'. Encode  $(1^{\lambda}, C, x, t)$ :

- 1. Let *C* be a boolean circuit such that  $C: \{0,1\}^s \to \{0,1\}^s$ . Let  $t_1$  denote  $t^{\frac{1}{c+1}}$  and  $t_2$  denote  $t^{\frac{c}{c+1}}$ , such that  $t = t_1 \cdot t_2$ .
- 2. Let E be the FE encryption circuit which takes as input public key pk  $\in \{0,1\}^p$ , current state st  $\in \{0,1\}^s$ , and has hardwired values gsk  $\in \{0,1\}^{\lambda}$ , randomness for encryption  $r_e \in \{0,1\}^{\ell_e}$ , repetition parameter  $t_2$ , and outputs a FE ciphertext ct (encrypting st) and a garbled public key pk (encrypting pk with respect to gsk).

Circuit  $E[gsk, r_e, t_2](pk, st)$ .

- (a) Compute ct  $\leftarrow$  FE.Enc<sub>pk</sub> (st, gsk,  $t_2$ ;  $r_e$ ).
- (b) Compute  $\widetilde{pk} \leftarrow GC.Enc_{gsk}(pk, [p])$ .
- (c) Output  $(ct, \widetilde{pk})$ .

Let  $\operatorname{gsk}_{t_1+1} \leftarrow \{0,1\}^{\lambda}$ . For  $i \in [t_1]$ ,

- (a) Sample  $\operatorname{gsk}_i \leftarrow \{0,1\}^{\lambda}$ .
- (b) Sample random coins  $r_{e,i} \leftarrow \{0,1\}^{\ell_e}$  for the FE.Enc algorithm.
- (c) Consider the circuit  $E_i = E\left[gsk_{i+1}, r_{e,i}, t_2\right]$ , where we hardcode the garbling secret key  $gsk_{i+1}$ , the random coins  $r_{e,i}$  and repetition parameter  $t_2$ .
- (d) Compute a garbled circuit,  $\widetilde{\mathbb{E}}_i \leftarrow \mathsf{GC}.\mathsf{Garble}_{\mathsf{gsk}_i}(\mathbb{E}_i)$ .

Let  $\widetilde{\mathbb{E}}_{t_1+1} \leftarrow \text{GC.Garble}_{\text{gsk}_{t_1+1}}(\mathbb{I})$ , where  $\mathbb{I}$  is the identity circuit that takes in a public key and a state  $(pk, st) \in \{0, 1\}^{p+s}$  and outputs st.

<sup>&</sup>lt;sup>4</sup>We assume these powers of t are integral, and avoid ceiling notation for ease of readability.

- 3. Let  $C: \{0,1\}^{s'} \to \{0,1\}^{s'}$ , be the repeat-then-encrypt circuit defined according to the transformation in Lemma 3.1 with the following setting.
  - The circuit *X* is equal to circuit *C*.
  - The circuit *Y* takes as input a state st  $\in \{0, 1\}^s$  and a garbling secret key gsk. It outputs  $\widetilde{st} \leftarrow GC.Enc_{gsk}(st, p + \lceil s \rceil)$ .
  - C is then the resulting circuit Z[X, Y], so that the repeated  $Z^i$  applies the repeated  $C^i$  and finally garbles the output.
- 4. Let FE.Setup  $(1^{\lambda}, C, t_2; r_s)$  be denoted by some repeated circuit  $S_C$  such that,

for all 
$$r_s$$
: FE.Setup  $\left(1^{\lambda}, C, t_2; r_s\right) = S_C^{t_2}(r_s)$ 

- 5. Let K be the functional key generation circuit defined according to the transformation in Lemma 3.1 with the following setting.
  - Circuit *X* is equal to circuit  $S_{\mathbb{C}}: \{0,1\}^{s'} \to \{0,1\}^{s'}$ .
  - Circuit *Y* takes as input state st  $\in \{0, 1\}^{s'}$ , additional input (gsk, x).

    Parse st as (pk, sk). Output sk and  $\widetilde{pk} \leftarrow GC.Enc_{gsk}(pk, [p])$  and  $\widetilde{x} \leftarrow GC.Enc_{gsk}(x, p + [s])$ .
  - $S_{\mathbb{C}}$  is then the resulting circuit Z[X,Y], so that the repeated  $Z^i$  applies the repeated  $S_{\mathbb{C}}^i$  and finally outputs the resulting function decryption key sk, garbled public key  $\widetilde{\mathsf{pk}}$ , and garbled  $\widetilde{x}$ .
- 6. Sample random coins  $r_s \leftarrow \{0, 1\}^{s'}$ , and compute,

$$SRE.\widetilde{x} \leftarrow SRE.Encode\left(1^{\lambda}, K\left[S_{C}\right], \left(r_{s}, \left(gsk_{1}, x\right), t_{2}\right), t_{2}\right).$$

7. Output  $SRE'.\widetilde{x} \leftarrow \left(SRE.\widetilde{x}, \left\{\widetilde{E}_i\right\}_{i \in [t_1+1]}\right)$ .

Algorithm SRE'. Eval $(C, \tilde{x}, t)$ 

- 1. Parse  $\widetilde{x} = \left( SRE.\widetilde{x}, \left\{ \widetilde{E}_i \right\}_{i \in [t_1+1]} \right)$ .
- 2. Let circuits C, C,  $S_C$ , K, variables  $t_1$ ,  $t_2$  be defined as in SRE'. Encode.
- 3. Compute  $\left(\operatorname{sk}, \widetilde{\operatorname{pk}}, \widetilde{x}\right) \leftarrow \operatorname{SRE.Eval}\left(\mathbb{K}[S_{\mathbb{C}}], \operatorname{SRE}.\widetilde{x}, t_{2}\right)|_{1...\widetilde{p}+\widetilde{s}}$ . Let  $\widetilde{\operatorname{inp}}_{1} = (\widetilde{\operatorname{pk}}, \widetilde{x})$ .
- 4. For  $i \in [t_1]$ ,
  - (a) Compute  $\left(\operatorname{ct}_{i}, \widetilde{\operatorname{pk}}_{i+1}\right) \leftarrow \operatorname{GC.Eval}\left(\widetilde{\operatorname{E}}_{i}, \widetilde{\operatorname{inp}}_{i}\right)$ .
  - (b) Compute,  $\widetilde{\mathsf{st}}_{i+1} \leftarrow \mathsf{FE.Dec}_{\mathsf{sk}} \left( \mathsf{C}, \mathsf{ct}_i, t_2 \right) |_{1...\widetilde{\mathsf{s}}}.$
  - (c) Let  $\widetilde{\mathsf{inp}}_{i+1} \leftarrow (\widetilde{\mathsf{pk}}_{i+1}, \widetilde{\mathsf{st}}_{i+1})$ .
- 5. Let  $\operatorname{st}_{t_1+1} \leftarrow \operatorname{GC.Eval}\left(\widetilde{\mathbb{E}}_{t_1+1}, \widetilde{\operatorname{inp}}_{t_1+1}\right)$ .

#### 3.1 Analysis

The correctness of the scheme follows readily from the correctness of the underlying primitives. For completeness, we include below a detailed proof. An already convinced reader may want to skip it.

**Proposition 3.4.** Assuming SRE is a correct scheme according to Definition 2.1, assuming FE is a correct scheme according to Definition 2.7, and GC is a correct scheme according to Definition 2.8, then Construction 3.3 is a correct scheme according to Definition 2.1.

*Proof.* We run through the evaluation algorithm to argue correctness.

- 1. Evaluation algorithm parses C, C,  $S_C$ , K,  $t_1$ ,  $t_2$ ,  $SRE'.\widetilde{x}$  exactly as the computation of the encode algorithm.
- 2. Since SRE is a correct scheme, with overwhelming probability, we have that SRE. Eval computes the evaluation of K on input  $(r_s, (gsk_1, x), t_2)$  after  $t_2$  repetitions. From correctness of Lemma 3.1, we have that

$$\mathsf{K}^{t_{2}}\left(r_{s},\left(\mathsf{gsk}_{1},x\right),t_{2}\right)|_{1...\widetilde{p}+\widetilde{s}}=Y\left(S_{\mathsf{C}}^{t_{2}}\left(r_{s}\right),\left(\mathsf{gsk}_{1},x\right)\right)=Y\left(\left(\mathsf{pk},\mathsf{sk}\right),\left(\mathsf{gsk}_{1},x\right)\right)$$

where the circuit Y encrypts inputs using  $gsk_1$  and the second equality holds from the correctness of the repeated circuit computation. From the description of circuit Y, we have that  $\widetilde{inp}_1 = \left(\widetilde{pk}, \widetilde{x}\right)$  is the encoding of input (pk, x) using  $gsk_1$ .

3. Next we show the invariant that for all  $j \in [t_1]$ ,  $\widetilde{\mathsf{inp}}_j$  is the encoding of input  $(\mathsf{pk}, C^{t_2 \cdot (j-1)}(x))$  using  $\mathsf{gsk}_j$  (let  $C^0(x)$  be defined as x).

We perform an induction over  $j \in [t_1 + 1]$ ,

- Base case, when j = 1: We have shown above, from evaluation of SRE.Eval, that our invariant is true for j = 1.
- Inductive step: Assuming that the invariant statement is true for  $j^* \in [t_1]$ . On loop iteration,  $i = j^*$ ,
  - (a) Since GC is perfectly correct, and from our invariant condition that the input  $(pk, C^{t_2 \cdot (j-1)}(x))$  is garbled using  $gsk_{j^*}$  and  $\widetilde{E}_{j^*}$  is garbled using  $gsk_{j^*}$ . We have that

(b) Next the algorithm performs FE decryption,

$$\widetilde{\operatorname{st}}' = \operatorname{FE.Dec}_{\operatorname{sk}} \left( \operatorname{C}, \operatorname{ct}_{j^*}, t_2 \right) |_{1...\widetilde{s}}, \text{ where}$$

$$\widetilde{\operatorname{st}}' = \operatorname{C}^{t_2} \left( C^{t_2 \cdot (j^* - 1)}(x), \operatorname{gsk}_{j^* + 1}, t_2 \right) |_{1...\widetilde{s}},$$

here the second equality holds with overwhelming probability from FE correctness.

- From definition of C,

$$C^{t_2}\left(C^{t_2\cdot(j^*-1)}(x), \mathsf{gsk}_{j^*+1}, t_2\right)|_{1...\widetilde{s}} = Y\left(C^{t_2}\left(C^{t_2\cdot(j^*-1)}(x)\right), \mathsf{gsk}_{j^*+1}\right)|_{1...\widetilde{s}}$$

$$= Y\left(C^{t_2\cdot(j^*)}(x), \mathsf{gsk}_{j^*+1}\right)|_{1...\widetilde{s}}$$

- From the description of Y, we have,  $\widetilde{\operatorname{st}}' = \operatorname{GC.Enc}_{\operatorname{gsk}_{j^*+1}}(C^{t_2\cdot(j^*)}(x), p+[s])$ .

As  $\mathsf{inp}_{j+1} \leftarrow (\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}')$ , and  $\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}'$  are garbled appropriately using  $\mathsf{gsk}_{j^*+1}$ , we've shown that our invariant condition holds.

4. Finally,  $\widetilde{\mathbf{E}}_{t_1+1}$  is a garbling of the identity circuit using  $\mathsf{gsk}_{t_1+1}$ , and from the invariant, we garbled input  $C^{t_2 \cdot t_1}(x) = C^t(x)$  using  $\mathsf{gsk}_{t_1+1}$ . Thus, from perfect evaluation of the garbled circuit,  $\mathsf{st}_{t_1+1} = C^t(x)$ .

As st<sub>t,+1</sub> =  $C^t(x)$ , correctness holds for our SRE scheme with overwhelming probability.

**Proposition 3.5.** Assuming SRE is semi  $t^{1/c}$ -succinct for some constant  $c \in (0, 1]$ , has semi-efficient evaluation according to Definition 2.1, assuming FE is succinct, efficiency preserving and has efficient decryption according to Definition 2.7, and GC has efficient encoding according to Definition 2.8, then Construction 3.3 is semi  $t^{1/c+1}$ -succinct according to Definition 2.1.

*Proof.* In the discussion below, we abuse notation and use  $poly(\cdot)$  to denote different universal polynomials (in  $\lambda$ ), such that for any  $\lambda$ , circuit  $C: \{0,1\}^s \to \{0,1\}^s$  be a circuit,  $x \in \{0,1\}^s$  an input, and  $t \le 2^{\lambda}$ , we have:

- Circuit C. Recall that  $C: \{0,1\}^{s'} \to \{0,1\}^{s'}$  is a circuit that computes C and then garbles the resulting state. From Lemma 3.1,  $|C| \le O(|C| + |Y|) + \text{poly}(\lambda)$  (where circuit Y garbles s bit input). As the garbled circuit has efficient encoding,  $|Y| \le s \cdot \text{poly}(\lambda)$  and  $|C| \le |C| \cdot \text{poly}(\lambda)$ . In particular, in what follows,  $s' \le |C| \cdot \text{poly}(\lambda)$ . Additionally, from Lemma 3.1, computing the description of circuit C takes time O(|C|).
- Functional encryption scheme FE. Recall that  $p, q, \ell_s$ , |ct|,  $\ell_e$  are the public key, secret key, randomness for the setup algorithm, ciphertext length and randomness for the encryption algorithm respectively. Since the input and output of the circuit C are of length at most s' and the underlying FE scheme is succinct, we have,  $p, q, \ell_s$ , |ct|,  $\ell_e \leq |C| \cdot \text{poly}(\lambda)$ .
- Circuit E. Recall that E is the FE encryption circuit. Since the garbled circuit has efficient encoding and garbles inputs of length p, and p, s,  $\ell_e \leq |C| \cdot \operatorname{poly}(\lambda)$ ,  $|E| \leq |C| \cdot \operatorname{poly}(\lambda)$ . Additionally, our FE scheme is succinct and computing the description of circuit E takes time O(|E|).
- Circuit  $S_C$ . Recall that this is the repeated circuit representation of the key generation algorithm on circuit C. Since our FE scheme is efficiency preserving, we have that  $S_C : \{0,1\}^m \to \{0,1\}^m$ , where  $m \le s' \cdot \text{poly}(\lambda) \le |C| \cdot \text{poly}(\lambda)$ . Additionally,  $|S_C| \le |C| \cdot \text{poly}(\lambda) \le |C| \cdot \text{poly}(\lambda)$ . Additionally, computing the description of circuit  $S_C$  takes time  $O(|S_C|)$ .
- Circuit K. Recall that K is the function key generator circuit. From Lemma 3.1,  $|K| \le O(|S_C| + |Y|) + \text{poly}(\lambda)$  where the circuit Y garbles a p + s bit input. Since the garbled circuit has efficient encoding,  $|Y| \le (p + s) \cdot \text{poly}(\lambda)$  and we have from the values set above that,  $|K| \le |C| \cdot \text{poly}(\lambda)$ . Additionally, from Lemma 3.1, computing the description of circuit K takes time O(|K|).

**Encoding Time.** We analyze the efficiency of the resulting SRE'. Encode algorithm.

- We've shown above that the circuits C, E,  $S_C$ , K are computable by SRE'. Encode in time O(|C|), O(|E|),  $O(|S_C|)$ , O(|K|) respectively and each of the circuits are bounded by size  $|C| \cdot \text{poly}(\lambda)$ . Hence, these circuits can be computed in time atmost  $|C| \cdot \text{poly}(\lambda)$ .
- Succinct randomized encoding scheme (SRE.Encode, SRE.Eval) is run on K with number of steps  $t_2$ . Since SRE is semi  $t^{1/c}$ -succinct, the runtime of running the inner SRE is at most  $t_2^{1/c} \cdot \text{poly}(|K|, \lambda) \le t_1 \cdot \text{poly}(|C|, \lambda)$ .
- The time taken to garble  $t_1$  circuits of size  $|E| \le |C| \cdot \text{poly}(\lambda)$  is  $t_1 \cdot |C| \cdot \text{poly}(\lambda)$ .

Since  $t_2^{1/c} = t_1 = t^{1/(c+1)}$ , the resulting scheme is semi  $t^{1/(c+1)}$ -succinct. The total encoding time is  $t^{1/(c+1)} \cdot \text{poly}(|C|, \lambda)$ .

**Evaluation Time.** We analyze the efficiency of our SRE'. Eval algorithm.

- Succinct randomized encoding scheme (SRE.Encode, SRE.Eval) is run on K with number of steps  $t_2$ . Since SRE has semi-efficient evaluation, the runtime of running the inner SRE is at most  $t_2 \cdot \text{poly}(|K|, \lambda) \le t_2 \cdot \text{poly}(|C|, \lambda)$ .
- Resulting evaluation procedure. For  $i \in [t_1]$ ,
  - We compute a garbled circuit evaluation, where the garbled inputs and circuit are at most  $poly(|C|, \lambda)$ .
  - The resulting FE, has efficient decryption and can decrypt the computation of C in time  $t_2$  · poly( $|C|, \lambda$ ).

- The size of  $\widetilde{\text{inp}}_{i+1}$  is at most poly( $|C|, \lambda$ ) (as size of K is at most poly( $|C|, \lambda$ )).

Running time of our computation is  $t_2 \cdot \text{poly}(|C|, \lambda) + t_2 \cdot t_1 \cdot \text{poly}(|C|, \lambda) \le t \cdot \text{poly}(|C|, \lambda)$ .

Claim 3.6. If SRE is semi  $t^{1/c}$ -succinct for some constant  $c \in (0,1]$  where the running time of SRE.Encode is at most  $t^{1/c} \cdot |C| \cdot \text{poly}(\lambda)$  (namely, linear dependence on |C|), then the running time of the resulting SRE'. Encode is at most  $t^{1/(c+1)} \cdot |C| \cdot \text{poly}(\lambda)$ .

*Proof.* The only place in the running time analysis where the dependence is not necessarily linear is in the time to encode K using SRE.Encode. Since  $|K| \le |C| \cdot \text{poly}(\lambda)$ , linear dependence of SRE.Encode on |K| immediately translates to linear dependence of SRE'.Encode on |C|.

In the full version , we show further optimizations to the evaluation time, making the evaluation linear in time t and size of the repeated circuit |C|. This more careful analysis pertains to the simulation complexity of circuits on a Turing machine, and uses the fact that we are considering circuits C that are step-circuits of Turing machines.

**Proposition 3.7.** Assuming SRE is secure according to Definition 2.1, assuming FE is secure according to Definition 2.7, and GC is secure according to Definition 2.8, then SRE' given by Construction 3.3 is secure according to Definition 2.1.

*Proof.* Simulator SRE'.Sim $(1^{\lambda}, C, C(x), t)$  outputs SRE'. $\widetilde{x}$  and is defined below.

Let the circuits C, E, C,  $S_C$ , K and parameters  $t_1$ ,  $t_2$  and FE scheme parameters be defined similarly to SRE'. Encode. Below we restate the notation.

Let  $C: \{0,1\}^s \to \{0,1\}^s$ . Let  $t_1, t_2$ , be such that  $t=t_1 \cdot t_2$ . Let  $E[gsk, r, t_2]$  be the FE encryption circuit and u be the size of the circuit. Let C be the repeat-then-encrypt circuit, and let  $S_C$  be the repeated circuit corresponding to FE.Setup  $(1^\lambda, C, t_2; \cdot)$ . Let  $K[S_C]$  be the key-generation circuit. Let  $p, q, \ell_s, |ct|, \ell_e$  be the public key, secret key, randomness for the setup algorithm, ciphertext length and randomness for the encryption algorithm respectively.

- 1. Compute (pk, sk)  $\leftarrow$  FE.Setup (1 $^{\lambda}$ , C,  $t_2$ ).
- 2. Let  $(\widetilde{\mathbb{E}}_{t_1+1}, \widetilde{\mathsf{inp}}_{t_1+1}) \leftarrow \mathsf{GC}.\mathsf{Sim}(1^{\lambda}, 1^{|\mathbb{I}|}, 1^{p+s}, (C^t(x)))$ , where  $\mathbb{I}$  is the identity circuit that takes in a public key and a state  $(\mathsf{pk}, \mathsf{st}) \in \{0, 1\}^{p+s}$  and outputs  $\mathsf{st}$ . For  $i \in \{t_1, \dots, 1\}$ ,
  - Let  $(\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}') = \widetilde{\mathsf{inp}}_{i+1}$ . Compute  $\mathsf{ct}_i \leftarrow \mathsf{FE}.\mathsf{Sim}_{\mathsf{pk}} \left(\widetilde{\mathsf{st}'}, 0 \dots 0\right)$ .
  - Compute  $(\widetilde{E}_i, \widetilde{\mathsf{inp}}_i) \leftarrow \mathsf{GC.Sim}\left(1^{\lambda}, 1^{u}, 1^{p+s}, \left(\mathsf{ct}_i, \widetilde{\mathsf{pk}}'\right)\right)$ .
- 3. Simulate SRE,

$$SRE.\widetilde{x} \leftarrow SRE.Sim\left(1^{\lambda}, \mathbb{K}\left[S_{\mathbb{C}}\right], \left(\mathsf{sk}, \widetilde{\mathsf{inp}}_{1}\right), t_{2}\right).$$

4. Output  $SRE'.\widetilde{x} = \left(SRE.\widetilde{x}, \left\{\widetilde{E}_i\right\}_{i \in [t_1+1]}\right)$ .

We define a sequence of hybrids starting with a hybrid that captures real encodings produced by SRE'. Encode and ending with simulated encodings produced by SRE'. Sim.

**Hybrid** −1. The Encode algorithm presented in Construction 3.3.

- 1. Let the circuits C, E, C,  $S_C$ , K and parameters  $t_1$ ,  $t_2$  and FE scheme parameters be defined similarly to SRE'. Encode.
- 2. Let  $\operatorname{gsk}_{t_1+1} \leftarrow \{0,1\}^{\lambda}$ . For  $i \in [t_1]$ , let E be the FE encryption circuit,
  - (a) Sample  $gsk_i \leftarrow \{0, 1\}^{\lambda}$ .
  - (b) Sample random coins  $r_{e,i} \leftarrow \{0,1\}^{\ell_e}$  for the FE.Enc algorithm.
  - (c) Consider the circuit  $E_i = E[gsk_{i+1}, r_{e,i}, t_2]$ , where we hardcode the garbling secret key  $gsk_{i+1}$ , the random coins  $r_{e,i}$  and repetition parameter  $t_2$ .
  - (d) Compute a garbled circuit,  $\widetilde{\mathbf{E}}_i \leftarrow \mathsf{GC}.\mathsf{Garble}_{\mathsf{gsk}_i}(\mathbf{E}_i)$ .

Let  $\widetilde{\mathbb{E}}_{t_1+1} \leftarrow \text{GC.Garble}_{\mathsf{gsk}_{t_1+1}}(\mathbb{I})$ , where  $\mathbb{I}$  is the identity circuit that takes in a public key and a state  $(\mathsf{pk},\mathsf{st}) \in \{0,1\}^{p+s}$  and outputs st.

3. Let  $C: \{0,1\}^{s'} \to \{0,1\}^{s'}$ , be the repeat-then-encrypt circuit. Let FE.Setup  $(1^{\lambda}, C, t_2; r_s)$  be denoted by some repeated circuit  $S_C(r_s)$ , and K be the functional key generation circuit. Sample random coins  $r_s \leftarrow \{0,1\}^{s'}$ , and compute,

$$SRE.\widetilde{x} \leftarrow SRE.Encode\left(1^{\lambda}, K\left[S_{C}\right], \left(r_{s}, \left(gsk_{1}, x\right), t_{2}\right), t_{2}\right).$$

4. Output SRE'. $\widetilde{x} \leftarrow \left( SRE.\widetilde{x}, \left\{ \widetilde{E}_i \right\}_{i \in \lceil t_1 + 1 \rceil} \right)$ .

**Hybrid** 0. We sample  $SRE.\tilde{x}$  using the simulator SRE.Sim instead of the real encoder SRE.Encode.

3. Let  $C: \{0,1\}^{s'} \to \{0,1\}^{s'}$ , be the repeat-then-encrypt circuit and K be the functional key generation circuit.

Compute (pk, sk)  $\leftarrow$  FE.Setup (1 $^{\lambda}$ , C,  $t_2$ ).

Compute  $\widetilde{\mathsf{pk}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_1}(\mathsf{pk}, [p])$  and  $\widetilde{\mathsf{st}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_1}(x, p + [s])$ . Let  $\widetilde{\mathsf{inp}}_1 = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$ . Simulate SRE,

 $SRE.\widetilde{x} \leftarrow SRE.Sim\left(1^{\lambda}, K\left[S_{C}\right], \left(sk, \widetilde{inp}_{1}\right), t_{2}\right).$ 

**Hybrid** j, **for**  $j \in [t_1]$ . For  $i \in [j]$ , we sample simulated garbled circuits (and input encodings), and for all i > j, we sampled real garbled circuits (and input encodings).

- 2. Let  $\operatorname{gsk}_{t_1+1} \leftarrow \{0,1\}^{\lambda}$ . For  $i \in \{j+1,\ldots,t_1\}$ , let E be the FE encryption circuit,
  - (a) Sample  $gsk_i \leftarrow \{0, 1\}^{\lambda}$ .
  - (b) Sample random coins  $r_{e,i} \leftarrow \{0,1\}^{\ell_e}$  for the FE.Enc algorithm.
  - (c) Consider the circuit  $E_i = E[gsk_{i+1}, r_{e,i}, t_2]$ , where we hardcode the garbling secret key  $gsk_{i+1}$ , the random coins  $r_{e,i}$  and repetition parameter  $t_2$ .
  - (d) Compute a garbled circuit,  $\widetilde{E}_i \leftarrow GC.Garble_{gsk_i}(E_i)$ .

Let  $\widetilde{\mathbb{E}}_{t_1+1} \leftarrow \mathsf{GC}.\mathsf{Garble}_{\mathsf{gsk}_{t_1+1}}(\mathtt{I})$ , where  $\mathtt{I}$  is the identity circuit that takes in a public key and a state  $(\mathsf{pk},\mathsf{st}) \in \{0,1\}^{p+s}$  and outputs  $\mathsf{st}.$ 

3. Let  $C:\{0,1\}^{s'} \to \{0,1\}^{s'}$ , be the repeat-then-encrypt circuit and K be the functional key generation circuit.

Compute  $(pk, sk) \leftarrow FE.Setup(1^{\lambda}, C, t_2).$ 

Compute  $\widetilde{\mathsf{pk}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_{j+1}}(\mathsf{pk}, [p])$  and  $\widetilde{\mathsf{st}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_{j+1}}\left(C^{t_2 \cdot j(x)}, p + [s]\right)$ , where  $C^0(x) \leftarrow x$ . Let  $\widetilde{\mathsf{inp}}_{j+1} = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$ .

For  $i \in \{j, ..., 1\}$ ,

- (a) Let  $(\widetilde{pk}', \widetilde{st}') = \widetilde{inp}_{i+1}$ .
- (b) Compute  $ct_i \leftarrow FE.Sim_{pk} (\widetilde{st'}, 0...0)$ .
- (c) Compute  $\left(\widetilde{\mathbf{E}}_{i},\widetilde{\mathsf{inp}}_{i}\right) \leftarrow \mathsf{GC.Sim}\left(\mathbf{1}^{\lambda},\mathbf{1}^{u},\mathbf{1}^{p+s},\left(\mathsf{ct}_{i},\widetilde{\mathsf{pk}}'\right)\right)$ .

Compute  $\widetilde{\mathsf{pk}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_1}(\mathsf{pk}, [p])$  and  $\widetilde{\mathsf{st}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_1}(x, p + [s])$ . Let  $\widetilde{\mathsf{inp}}_1 = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$ . Simulate SRE,

$$SRE.\widetilde{x} \leftarrow SRE.Sim\left(1^{\lambda}, K\left[S_{C}\right], \left(sk, \widetilde{inp}_{1}\right), t_{2}\right).$$

Observe that if we set, j = 0 in the hybrid above, our steps would be identical to  $Hyb_0$ .

**Hybrid**  $t_1 + 1$ . The simulated algorithm. Here we simulate garbled circuit  $t_1 + 1$ .

1. Let the circuits C, E, C,  $S_C$ , K be defined similarly to step 1 of hybrid -1. Compute  $(pk, sk) \leftarrow FE.Setup(1^{\lambda}, C, t_2)$ . 2. Let  $(\widetilde{\mathbb{E}}_{t_1+1}, \widetilde{\mathsf{inp}}_{t_1+1}) \leftarrow \mathsf{GC.Sim}(1^{\lambda}, 1^{|\mathbb{I}|}, 1^{p+s}, (C^t(x)))$ , where  $\mathbb{I}$  is the identity circuit that takes in a public key and a state  $(\mathsf{pk}, \mathsf{st}) \in \{0, 1\}^{p+s}$  and outputs  $\mathsf{st.}$ 

For  $i \in \{t_1, ..., 1\}$ ,

- (a) Let  $(\widetilde{pk}', \widetilde{st}') = \widetilde{inp}_{i+1}$ .
- (b) Compute  $ct_i \leftarrow FE.Sim_{pk} (\widetilde{st'}, 0...0)$ .
- (c) Compute  $\left(\widetilde{\mathbf{E}}_{i},\widetilde{\mathsf{inp}}_{i}\right) \leftarrow \mathsf{GC}.\mathsf{Sim}\left(1^{\lambda},1^{u},1^{p+s},\left(\mathsf{ct}_{i},\widetilde{\mathsf{pk}}'\right)\right)$ .
- 3. Simulate SRE,

$$SRE.\widetilde{x} \leftarrow SRE.Sim\left(1^{\lambda}, \mathbb{K}\left[S_{\mathbb{C}}\right], \left(\mathsf{sk}, \widetilde{\mathsf{inp}}_{1}\right), t_{2}\right).$$

4. Output SRE'. $\widetilde{x} = \left( SRE.\widetilde{x}, \left\{ \widetilde{E}_i \right\}_{i \in [t_1 + 1]} \right)$ .

Observe that our steps are identical to  $Hyb_{t_1}$ .

We write  $\mathsf{Hyb}_i$  to denote the output distribution of hybrid i. We now show that each pair of adjacent distributions defined above are computationally indistinguishable.

**Claim 3.8.** Suppose SRE is secure according to Definition 2.1. Then,

$$\mathsf{Hyb}_{-1} \approx_c \mathsf{Hyb}_0$$
.

*Proof.* The main difference between the hybrids is the following: In  $Hyb_{-1}$ , we compute a real SRE encoding of the functional key generation computation. In  $Hyb_0$ , we perform the functional key generation computation explicitly, and then simulate the SRE encoding using the output of the resulting output. Indistinguishability follow from the security of the underlying SRE. Details follow.

Recall that for all  $r_s$ ,

FE.Setup 
$$(1^{\lambda}, C[C, t_2], t_2; r_s) = S_C^{t_2}(r_s),$$

and hence the two processes have the exact same output distribution (pk, sk). Accordingly also the garbled input encoding pk is distributed the same in both hybrids and so is  $inp_1 = (pk, st)$ .

Hence, from indistinguishability of the underlying SRE, we have,

SRE.Encode 
$$(1^{\lambda}, \mathbb{K}[S_{\mathbb{C}}], (r_s, x, gsk_1), t_2) \approx_c SRE.Sim(1^{\lambda}, \mathbb{K}[S_{\mathbb{C}}], (sk, \widetilde{inp}_1), t_2),$$

which concludes the proof.

**Claim 3.9.** Suppose FE is selectively-secure according to Definition 2.7 and GC is secure according to Definition 2.8. Then, for  $j \in [t_1]$ ,

$$\mathsf{Hyb}_{i-1} \approx_c \mathsf{Hyb}_i$$
.

*Proof.* We define the following sub-hybrid, where we simulate the  $j^{th}$  garbled circuit in the computation.

**Hybrid** (j-1).5. For  $i \in [j-1]$ , we sample simulated garbled circuits (and input encodings), and for all i > j, we sampled real garbled circuits (and input encodings). For iteration we simulate the garbled circuit, but run the real FE encryption algorithm.

3. Let  $C: \{0,1\}^{s'} \to \{0,1\}^{s'}$ , be the repeat-then-encrypt circuit and K be the functional key generation circuit.

Compute (pk, sk)  $\leftarrow$  FE.Setup (1 $^{\lambda}$ , C,  $t_2$ ).

Compute  $\widetilde{\mathsf{pk}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_{j+1}}(\mathsf{pk}, [p])$  and  $\widetilde{\mathsf{st}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_{j+1}}\left(C^{t_2 \cdot j}(x), p + [s]\right)$ , where  $C^0(x) \leftarrow x$ . Let  $\widetilde{\mathsf{inp}}_{j+1} = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$ .

For i = j,

(a) Sample some random coins  $r_{e,j} \in \{0,1\}^{\ell_e}$ . Let  $\operatorname{ct}_j \leftarrow \mathsf{FE.Enc}_{\mathsf{pk}}\left(C^{t_2\cdot(j-1)}(x),\operatorname{gsk}_{j+1},t_2;r_{e,j}\right)$ , where  $C^0(x) \leftarrow x$ .

(b) Let 
$$(\widetilde{E}_i, \widetilde{\mathsf{inp}}_i) \leftarrow \mathsf{GC.Sim}\left(1^{\lambda}, 1^{u}, 1^{p+s}, \left(\mathsf{ct}_j, \widetilde{\mathsf{pk}}'\right)\right)$$
, where  $(\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}') = \widetilde{\mathsf{inp}}_{i+1}$ .

For  $i \in \{j-1, ..., 1\}$ ,

(a) Let 
$$(\widetilde{E}_i, \widetilde{\mathsf{inp}}_i) \leftarrow \mathsf{GC.Sim}(1^{\lambda}, 1^u, 1^{p+s}, (\mathsf{FE.Sim}_{\mathsf{pk}}(\widetilde{\mathsf{st'}}, 0 \dots 0), \widetilde{\mathsf{pk'}}))$$
, where  $(\widetilde{\mathsf{pk'}}, \widetilde{\mathsf{st'}}) = \widetilde{\mathsf{inp}}_{i+1}$ .

Compute  $\widetilde{\mathsf{pk}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_1}(\mathsf{pk}, [p])$  and  $\widetilde{\mathsf{st}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_1}(x, p + [s])$ . Let  $\widetilde{\mathsf{inp}}_1 = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$ . Simulate SRE,

$$SRE.\widetilde{x} \leftarrow SRE.Sim\left(1^{\lambda}, K\left[S_{C}\right], \left(sk, \widetilde{inp}_{1}\right), t_{2}\right).$$

**Claim 3.10.** Suppose GC is secure according to Definition 2.8. Then, for  $j \in [t_1]$ ,

$$\mathsf{Hyb}_{(j-1)} \approx_c \mathsf{Hyb}_{(j-1).5}$$
.

*Proof.* Observe that the only difference between the hybrids  $\mathsf{Hyb}_{(j-1)}$  and  $\mathsf{Hyb}_{(j-1).5}$  is how the j-th garbled circuit  $\widetilde{\mathsf{E}}_j$  and garbled input  $\mathsf{inp}_j$  are computed. The main difference between the hybrids is the following: In  $\mathsf{Hyb}_{(j-1)}$ , we garble the real circuit. In  $\mathsf{Hyb}_{(j-1).5}$ , we perform the circuit computation explicitly, and then simulate the garbled circuit encoding using the output of the resulting output. Indistinguishability follow from the security of garbled circuits. Details follow.

- In  $Hyb_{(i-1)}$ .
  - We sample  $gsk_i, \ldots, gsk_t$ .
  - We compute  $\widetilde{\mathsf{pk}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_j}(\mathsf{pk},[p]), \ \widetilde{\mathsf{st}} = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_j}(\mathsf{st},p+[s]) \ \text{where } \mathsf{st} = C^{t_2\cdot(j-1)}(x). \ \mathsf{Let} \ \widetilde{\mathsf{inp}}_j \leftarrow \left(\widetilde{\mathsf{pk}},\widetilde{\mathsf{st}}\right).$
  - We compute  $\widetilde{E}_j$  as follows.
    - 1. Consider the circuit  $E_j = E\left[ \operatorname{gsk}_{j+1}, r_{e,j}, t_2 \right]$ , where we hardcode, the garbling secret key  $\operatorname{gsk}_{j+1}$ , and the random coins  $r_{e,j}$ .
    - 2. Compute a garbled circuit,  $\widetilde{\mathbb{E}}_j \leftarrow GC.Garble_{gsk_i}(\mathbb{E}_j)$ .
- In  $Hyb_{(j-1).5}$ .
  - We sample  $\mathsf{gsk}_{j+1},\dots,\mathsf{gsk}_{t_1}$  (or we no longer sample  $\mathsf{gsk}_j).$
  - We compute  $\widetilde{E}_j$  as follows.

1. Compute 
$$\operatorname{ct}_j \leftarrow \operatorname{\mathsf{FE.Enc}}_{\operatorname{\mathsf{pk}}} \left( C^{t_2 \cdot (j-1)}(x), \operatorname{\mathsf{gsk}}_{j+1}, t_2; r_{e,j} \right)$$
, where  $C^0(x) \leftarrow x$ .

2. Compute 
$$(\widetilde{E}_j, \widetilde{\mathsf{inp}}_j) \leftarrow \mathsf{GC}.\mathsf{Sim}\left(1^{\lambda}, 1^{u}, 1^{p+s}, \left(\mathsf{ct}_j, \widetilde{\mathsf{pk}}'\right)\right)$$
. Where  $\widetilde{\mathsf{pk}}' = \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_{j+1}}(\mathsf{pk}, [p])$ .

The difference is in how  $\widetilde{\mathbf{E}}_j$ ,  $\widetilde{\mathsf{inp}}_j$  are computed. Observe that, for circuit  $\mathbf{E}\left[\mathsf{gsk}_{j+1}, r_{e,j}, t_2\right]$ , and input  $\left(\mathsf{pk}, \left(C^{t_2\cdot(j-1)}(x)\right)\right)$ , the real circuit performs the same computation as the simulated circuit. The circuit  $\mathbf{E}$  computes,  $\mathsf{ct}_j \leftarrow \mathsf{FE}.\mathsf{Enc}_{\mathsf{pk}}\left(C^{t_2\cdot(j-1)}(x), \mathsf{gsk}_{j+1}, t_2; r_{e,j}\right)$ , where  $C^0(x) \leftarrow x$  and computes  $\widetilde{\mathsf{pk}}' \leftarrow \mathsf{GC}.\mathsf{Enc}_{\mathsf{gsk}_{j+1}}(\mathsf{pk}, [p])$ . Thus, security follows from the security of the garbled circuit.

**Claim 3.11.** Suppose FE is secure according to Definition 2.7. Then, for  $j \in [t_1]$ ,

$$\mathsf{Hyb}_{(j-1).5} \approx_c \mathsf{Hyb}_j$$
.

*Proof.* Observe that the only difference between the hybrids  $\mathsf{Hyb}_{(j-1).5}$  and  $\mathsf{Hyb}_{(j)}$  is how the ciphertext on j-th iteration is computed. In the former hybrid we explicitly perform the computation, while in the latter we simulate the computation using the FE simulator. Security follows from the security of the functional encryption scheme. The details are sketched below.

• In  $Hyb_{(j-1).5}$ .

- We set, 
$$\operatorname{ct}_j \leftarrow \mathsf{FE.Enc_{pk}}\left(C^{t_2\cdot(j-1)}(x), \operatorname{gsk}_{j+1}, t_2; r_{e,j}\right)$$
 (where  $C^0(x) \leftarrow x$ ).

In Hyb<sub>(i)</sub>.

- We set 
$$\operatorname{ct}_j \leftarrow \operatorname{FE.Sim}_{\operatorname{pk}}\left(\widetilde{\operatorname{st}}', 0 \dots 0\right)$$
, where  $\widetilde{\operatorname{st}}' = \operatorname{GC.Enc}_{\operatorname{gsk}_{j+1}}\left(C^{t_2 \cdot j}(x), p + [s]\right)$ .

Observe that the functional encryption scheme has input  $\left(C^{t_2\cdot(j-1)}(x), \operatorname{gsk}_{j+1}, t_2\right)$  for the repeated circuit C with  $t_2$  repetitions.

Running  $C^{t_2}$  on our input (Lemma 3.1), we compute circuit C for  $t_2$  repetitions and then garble state using  $gsk_{j+1}$ . Thus computing  $t_2$  repetitions of C, we compute  $\left(C^{t_2\cdot(j-1)+t_2}(x)\right)$  and we garble using  $gsk_{j+1}$  to obtain output  $\left(\widetilde{st}',0\ldots 0\right)$  exactly as used in the above FE simulation. From the underlying indistinguishability of the functional encryption scheme, the two hybrids are indistinguishable.

Combining the above two claims, we have that  $\mathsf{Hyb}_{(i-1)} \approx_c \mathsf{Hyb}_i$ .

Claim 3.12. Suppose GC is secure according to Definition 2.8.

$$\mathsf{Hyb}_{t_1} \approx_c \mathsf{Hyb}_{t_1+1}$$
.

*Proof.* Observe that the only difference between the hybrids  $\mathsf{Hyb}_{t_1}$  and  $\mathsf{Hyb}_{t_1+1}$  is how the garbled circuit corresponding to I and input encoding  $\mathsf{inp}_{t_1+1}$  is computed. Thus security follows from the security of garbled circuits. In  $\mathsf{Hyb}_{t_1}$ , we compute  $\mathsf{inp}_{t_1+1}$  as garbled encryptions of inputs  $\mathsf{pk}$ ,  $C^t(x)$  using key  $\mathsf{gsk}_{t_1+1}$ . The real computation outputs  $C^t(x)$  on the evaluation of circuit I. In  $\mathsf{Hyb}_{t_1+1}$ , we simulate the garbled inputs  $\mathsf{inp}_{t_1+1}$  using the output  $C^t(x)$ .

From inspection, note that  $\mathsf{Hyb}_{-1}$  is the real algorithm,  $\mathsf{Hyb}_{t_1+1}$  is the simulated algorithm. Thus, from the claims proved above, the security of our randomized encoding scheme holds true.

## Acknowledgments

We thank Giulio Malavolta for a helpful discussion and encouragement to look into this problem. N. Bitansky was supported in part by the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No. 101042417, acronym SPP).

#### References

- [AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology CRYPTO 2015 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2015.
- [AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. *IACR Cryptol. ePrint Arch.*, page 759, 2018.
- [AMZ24] Shweta Agrawal, Giulio Malavolta, and Tianwei Zhang. Time-lock puzzles from lattices. In *Annual International Cryptology Conference*, pages 425–456. Springer, 2024.
- [App11] Benny Applebaum. Key-dependent message security: Generic amplification and completeness. In Kenneth G. Paterson, editor, Advances in Cryptology EUROCRYPT 2011 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings, volume 6632 of Lecture Notes in Computer Science, pages 527–546. Springer, 2011.
- [BDGM20] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate io from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology EUROCRYPT 2020 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I,* volume 12105 of *Lecture Notes in Computer Science*, pages 79–109. Springer, 2020.
- [BGG<sup>+</sup>14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, Advances in Cryptology EUROCRYPT 2014 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings, volume 8441 of Lecture Notes in Computer Science, pages 533–556. Springer, 2014.
- [BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 345–356. ACM, 2016.

- [BGL<sup>+</sup>15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 439–448, 2015.
- [BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325. ACM, 2012.
- [BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190. IEEE Computer Society, 2015.
- [CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for ram programs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 429–437, 2015.
- [GKP<sup>+</sup>13] 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, *Symposium on Theory of Computing Conference*, STOC'13, Palo Alto, CA, USA, June 1-4, 2013, pages 555–564. ACM, 2013.
- [GS18] Sanjam Garg and Akshayaram Srinivasan. A simple construction of io for turing machines. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*, volume 11240 of *Lecture Notes in Computer Science*, pages 425–454. Springer, 2018.
- [GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, Advances in Cryptology CRYPTO 2013 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I, volume 8042 of Lecture Notes in Computer Science, pages 75–92. Springer, 2013.
- [HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 415–434. IEEE, 2023.
- [IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA, pages 294–304. IEEE Computer Society, 2000.
- [JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 60–73. ACM, 2021.
- [JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over \$\mathbb{F}\_p\$, dlin, and prgs in nc<sup>0</sup>. In Orr Dunkelman and Stefan Dziembowski, editors,

Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I, volume 13275 of Lecture Notes in Computer Science, pages 670–699. Springer, 2022.

- [KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 419–428, 2015.
- [LPST16] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography* 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I, volume 9562 of Lecture Notes in Computer Science, pages 96–124. Springer, 2016.
- [QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS), pages 859–870. IEEE, 2018.
- [RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. In *MIT/LCS/TR-684*, volume 20, page 5, 1996.
- [RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. *IACR Cryptol. ePrint Arch.*, page 856, 2024.
- [Yao86] Andrew Yao. How to generate and exchange secrets. In FOCS, 1986.

### A Construction in [HLL23] satisfies Definition 2.7

**Theorem A.1** ([HLL23]). Assuming that the circular LWE assumption holds, there exists a selective 1-key simulation secure FE scheme that satisfies Definition 2.7.

*Proof.* The theorem statement in [HLL23], proved correctness, security, and succinctness as defined in Definition 2.7.

The authors [HLL23] showed that their construction satisfies a stronger computational correctness notion. Since in our setting, we can choose the function and the input selectively, any non-uniform computational attacker will imply our correctness definition. More concretely, for every  $\lambda$ , fix  $|C_{\lambda}|$ ,  $x_{\lambda}$ ,  $t_{\lambda} \le \ell(\lambda)$  that maximize the probability of winning. These parameters can be selectively output by the non-uniform computational attacker.

The only remaining proof we need to show is that their scheme is additionally efficiency-preserving. We show that the input independent homomorphic evaluation procedure from [HLL23] is efficiency preserving.

Claim A.2. Algorithm UEvalC (construction 2 in [HLL23]) is efficiency preserving.

*Proof.* Let  $C: \{0,1\}^s \to \{0,1\}^s$  be a boolean circuit,  $\mathbf{A}_{\mathsf{attr}} \in \mathbb{Z}_q^{(n+1) \times (s \cdot m)}, \mathbf{A}_{\mathsf{circ}} \in \mathbb{Z}_q^{(n+1) \times m}$ , where n, m, q are some polynomially bounded functions (in  $\lambda$ ).

Let  $\mathsf{UEvalC}_C$  be a circuit that runs  $\mathsf{UEval}$  on input  $\mathsf{A}_{\mathsf{attr}}, \mathsf{A}_{\mathsf{circ}}, C$ , and computes  $\mathsf{A}_C = (\mathsf{A}_{C:1}, \ldots, \mathsf{A}_{C:s})$  where  $(C:1,\ldots,C:s)$  are the output wires of circuit C.  $\mathsf{UEvalC}_C$  outputs  $(\mathsf{A}_C,\mathsf{A}_{\mathsf{circ}})$ . We claim that,  $|\mathsf{UEvalC}_C| \leq |C| \cdot \mathsf{poly}(\lambda)$ , can be computed in from C in time  $O(|\mathsf{UEvalC}_C|)$  and,

$$\forall t \in \mathbb{N}, \;\; \mathsf{UEvalC}_C^t(\mathbf{A}_{\mathsf{attr}}, \mathbf{A}_{\mathsf{circ}}) = \left(\mathsf{UEvalC}\left(\mathbf{A}_{\mathsf{attr}}, \mathbf{A}_{\mathsf{circ}}, C^t\right), \mathbf{A}_{\mathsf{circ}}\right).$$

It is easy to see that  $|\text{UEvalC}_C| \leq |C| \cdot \text{poly}(\lambda)$  and can be computed in time  $O(|\text{UEvalC}_C|)$  as the algorithm garbles gate-by-gate and performs  $\text{poly}(\lambda, n, m, q)$  operations on each gate. We show the functionality claim by inducting on t. Clearly, the above claim holds for t = 1 by definition. Assuming it holds for some t, our circuit outputs matrices, which are associated with output wires  $C:1,\ldots,C:s$ . These output wires are the inputs to the new circuit when performing a repeated computation. Hence these can be considered as  $A_{\text{attr}}$  for the (t+1)-th computation.

The succinct FE scheme in [HLL23], is constructed by a generic transformation from a laconic function evaluation scheme and a non-succinct functional encryption scheme [QWW18]. This transformation, compresses a circuit using the UEvalC procedure described in construction 2 of [HLL23]. This is the primary dependence of the circuit in the key generation algorithm for the succinct FE scheme. Once the digest (which is the output of UEvalC) is computed, the algorithm only performs computations on this digest. Note that the digest is bounded by  $|C| \cdot \text{poly}(\lambda)$ .

We can finish the proof of the claim using Lemma 3.1 where the repeated circuit X is UEvalC and the computation at the end, denoted by circuit Y primarily depends on the digest and has circuit size at most  $|C| \cdot \text{poly}(\lambda)$ . Combining all the elements, we have that our functional encryption scheme is efficient preserving. We defer a complete proof of the claim to the full version.