Succinct Randomized Encodings from Laconic Function Evaluation, Faster and Simpler

Nir Bitansky[1,2] and Rachit Garg[1]

[1]New York University
nbitansky@gmail.com, rg5134@cims.nyu.edu
[2]Tel Aviv University

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. These encodings have powerful applications, including time-lock puzzles, reducing communication in MPC, and bootstrapping advanced encryption schemes.

Until not long ago, the only known constructions were based on indistinguishability obfuscation, and in particular 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 laconic function evaluation scheme that satisfies a natural *efficiency preservation* property. Under sub-exponential assumptions, the encoding time can be further reduced to $\approx \sqrt{s}$, but at the account of a huge security loss.

As a corollary, assuming also bounded-space languages that are worst-case hard-to-parallelize, we obtain time-lock puzzles with an arbitrary polynomial gap between encoding and decoding times.

Contents

1 Introduction

Succinct randomized encodings (SREs) [BGL+15, CHJV15, KLW15] are a specific form of randomized encodings of functions [IK00]. SREs allow to encode an input x for a uniform function (e.g. Turing machine) 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+16, AMZ24]).

Existing Constructions. The works of [BGL+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+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+15, CHJV15] and achieved a (fully) succinct construction from the same assumptions.

While indistinguishability obfuscation is now known under standard assumptions [JLS21, JLS22, RVV24], constructions are still 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

We give a new construction of semi-succinct randomized encodings where the time to encode is roughly $t^{\epsilon} \cdot s$ for arbitrarily small constants $\epsilon < 1$. Our construction is based on any laconic function evaluation scheme [QWW18] that satisfies a natural *efficiency preservation* property. In laconic function evaluation, ciphertexts are allowed to grow with the function's output, but not with circuit size nor with depth [HLL23]. The efficiency preservation property roughly says that key derivation for a function that can be uniformly computed in time t and space s takes roughly the same time and space $\approx t$ and $\approx s$ (we elaborate in the technical overview). Such efficiency preservation holds in particular for the laconic function evaluation scheme constructed in [HLL23] based on the Circular LWE Assumption. Hence, under the same assumption as in [AMZ24], we obtain SREs with improved succinctness.

Theorem 1.1 (Informal). *Assume the existence of efficiency-preserving laconic function evaluation scheme. Then for any constant $\varepsilon < 1$, there exists a semi ε-succinct randomized encoding. Specifically, for a Turing*

machine computation $M(x)$ of time t and space s, and security parameter λ, the encoding time is $t^{\varepsilon} \cdot (|x| + s) \cdot \mathsf{poly}(|M|, \lambda)$. 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+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).

We also show that we can achieve full succinctness, i.e. encoding time independent of the time to run the input - if we allow sub-exponentially secure Circular LWE. However, this comes at the account of a huge security loss — the resulting construction can be broken in quasi-polynomial time.

Corollary 1.2 (Informal)**.** *Assume the existence of sub-exponentially secure efficiency-preserving laconic function evaluation scheme. There exists a fully succinct randomized encoding with a quasi-polynomial security guarantee. Specifically, for a Turing machine computation $M(x)$ of time t and space s, and security parameter λ, the encoding time is $(|x| + s) \cdot \mathsf{poly}(|M|, \lambda)$. However, there exists $c \in \mathbb{N}$, such that the scheme can be broken in time $2^{(\log \lambda)^c}$. In particular, such randomized encodings exist assuming sub-exponentially secure circular LWE.*

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 laconic function evaluation scheme, even without efficiency preservation.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 Laconic Function Evaluation. We start by recalling in more detail the notion of laconic function evaluation (LFE) [QWW18]. We focus on a depth-independent version (c.f. [HLL23]). Here given a function $f : \{0,1\}^n \to \{0,1\}^n$, represented by a circuit, we can sample a succinct public encryption key pk, such that an encryption $\mathsf{Enc}_{\mathsf{pk}}(x)$ can be (publicly) decrypted using pk and the circuit f to yield the function output $f(x)$. Furthermore, the encryption $\mathsf{Enc}_{\mathsf{pk}}(x)$ does not leak any other information about x, in the sense that, even in the presence of pk, the encryption can be efficiently simulated from the output $f(x)$. We require that the time to encrypt $x \in \{0,1\}^n$ is $n \cdot \mathsf{poly}(\lambda)$, independently of the circuit complexity of f. The size of the keys pk is accordingly $n \cdot \mathsf{poly}(\lambda)$, and we require that the time to compute them is $|f| \cdot \mathsf{poly}(\lambda)$. Here and throughout λ denotes the security parameter and $|f|$ denotes the size of the corresponding circuit.[1]

The work of [QWW18] introduced the concept of a laconic function evaluation scheme and constructed, under LWE, a scheme where both ciphertexts and keys only scale with the depth (but not the circuit size). Recently, the work of [HLL23], under Circular LWE, removes the depth-dependence, leading to a scheme with the desired efficiency. We note that LFE schemes as above stand in sharp contrast to *compact* LFE

[1]We note that in the literature a somewhat stronger version is typically considered, where givne a common random string, the function can be compressed deterministically into a corresponding public key (or digest).

schemes where the function f may have output size $m \gg n$, and yet encryption complexity is sublinear in m. Indeed, compact LFE implies compact functional encryption [QWW18], which already yield the full power of indistinguishability obfuscation [AJ15, BV15].

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

$\frac{1}{2}$-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 $\mathrm{poly}(|M|)$ factors). Then, to encode a t-step computation $M(x)$, the basic idea is to derive a public key pk for $f = E \circ S_M^{\sqrt{t}}$. The function f given an intermediate state $\mathsf{st}_i \in \{0,1\}^s$, performs $\sqrt{t}$ steps of M, and outputs an encryption $E(\mathsf{st}_{i+\sqrt{t}})$ 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 $E(\mathsf{st}_{i\sqrt{t}})$ to an LFE encryption $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_{i\sqrt{t}})$, with the exception of the last gadget $G_{\sqrt{t}}$, which decrypts $E(\mathsf{st}_t)$ to obtain the final state st_t, encoding the result of the computation. Finally, we will also include a ciphertext $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_0)$ of the initial state st_0, which encodes the input x.

Given the above components $\mathsf{pk}, G_1, \ldots, G_{\sqrt{t}}, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_0)$, an evaluator can execute the machine computation by each time starting from $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_{i\sqrt{t}})$, performing LFE 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 LFE 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 ciphertext $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_i \in \{0,1\}^s)$. Indeed, this constraint is an artifact of (non-compact) LFE, where any increase in the size of the function output translates to an increase in the size of 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 $\tilde{\mathsf{st}}_{i\sqrt{t}} = \mathsf{GC.Enc}_{\mathsf{gsk}_i}(\mathsf{st}_{i\sqrt{t}})$ under a corresponding garbling key gsk_i. The gadget G_i will be a corresponding garbled circuit $\tilde{C}_i = \mathsf{GC.Garble}_{\mathsf{gsk}_i}(C_i)$, where the circuit C_i takes as input a state $\mathsf{st}_{i\sqrt{t}}$ and outputs a ciphertext $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_{i\sqrt{t}}, \mathsf{gsk}_{i+1}; r)$, using hardwired randomness r, where gsk_{i+1} is the next garbling key (also hardwired in C_i). The function f for which we derive the LFE key is augmented accordingly, given $(\mathsf{st}, \mathsf{gsk})$, it executes $S_M^{\sqrt{t}}(\mathsf{st})$, and garbles using 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 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 LFE simulation guarantee and the garbled circuit simulation guarantee. Specifically, since our garbled circuits contain secret information (LFE encryption randomness r and next garbling key gsk_{i+1}), we'll rely on circuit-private simulation for garbled circuits where the garbled circuit $\tilde{C}_i$ and input encoding $\tilde{\mathsf{st}}_{i\sqrt{t}}$ can be efficiently simulated from the output $C_i(\mathsf{st}_{i\sqrt{t}})$ alone. This allows to simulate the garbled circuits backwards, starting from $\tilde{C}_{\sqrt{t}}, \tilde{\mathsf{st}}_{t-\sqrt{t}}$ which can be simulated from the output $M(x)$ of the entire computation. Then we can use the simulated $\tilde{\mathsf{st}}_{t-\sqrt{t}}$ to simulate the previous LFE encryption. The simulated LFE 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, $\mathsf{Encode}(M, x, t)$ 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 $\mathsf{Encode}(\mathsf{Encode}(M, \cdot, t), x, \sqrt{t}s)$ (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 $\mathsf{Encode}(M, x, t)$ of size $\sqrt{t}s$. Thus we have so far gained nothing.[2]

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 LFE key generation procedure. In particular, looking back at the previous encoding

$$\mathsf{pk}, \tilde{C}_1, \ldots, \tilde{C}_{\sqrt{t}}, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{st}_0, \mathsf{gsk}_1) \ ,$$

we note that generating the public keys 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.

Specifically, we consider an a-symmetric version of the previous solution where instead of deriving keys for the $\sqrt{t}$-step $S_M^{\sqrt{t}}$ (plus input garbling), we derive a key for the $t^{2/3}$-step circuit $S_M^{t^{2/3}}$, which now means we only need $t^{1/3}$ garbled circuits $\tilde{C}_1, \ldots, \tilde{C}_{t^{1/3}}$ for converting garbled states to LFE-encrypted states. What we gain is that now while deriving function keys pk for $S_M^{t^{2/3}}$ takes longer time $\approx t^{2/3}s$, it is a uniform computation with small output size ($\approx s$). If the space needed to generate these keys is s', then randomly encoding the key generation using the $1/2$-succinct solution, we obtain encoding time $\approx \sqrt{t^{2/3}s} \cdot s' + t^{1/3} \cdot s$. In particular, if the LFE scheme is also *efficiency preserving*, in the sense that the space s' required for generating keys for a space-s function is $\approx s$, we obtain encoding time $\approx t^{1/3}s^{3/2}$.

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^{ε} for an arbitrarily small ε. 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 LFE 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

[2] 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].

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.

Existing Efficiency Preserving LFE. Examining the LFE scheme of [HLL23], we observe that it is indeed efficiency preserving. In this scheme, which builds on the (depth-dependent) scheme of [BGG+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 λ the security parameter. We say a function f is negligible in the security parameter λ if $f = \lambda^{-\omega(1)}$. We denote this by writing $f(\lambda) = \mathsf{negl}(\lambda)$. We write $\mathsf{poly}(\lambda)$ to denote a function that is bounded by a fixed polynomial in λ. 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 λ) that run in *deterministic* polynomial time in the length of their input and takes in an advice string of $\mathsf{poly}(\lambda)$ 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 $\mathsf{M}^t(x)$ to denote the execution of a Turing machine M on input x for t steps. We denote the first k bits of a string $s \in \{0,1\}^{k+\ell}$, by the notation $s|_{1\ldots k}$.

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+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 $\mathsf{SRE} = (\mathsf{SRE.Encode}, \mathsf{SRE.Eval})$ with the following syntax:

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

$\mathsf{Eval}(\mathsf{M}, \widetilde{x}, t, s) \to y$**.** 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 $|\mathsf{M}|, |x|, t, s \leq \ell(\lambda)$ 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 $x \in \{0,1\}^n$,

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

- **Semi ε-succinctness:** There exists a polynomial poly such that for any $\lambda, \mathsf{M}, x, s$ and $t \leq 2^\lambda$, the running time of $\mathsf{Encode}(1^\lambda, \mathsf{M}, x, t, s)$ is at most $t^\varepsilon \cdot \mathsf{poly}(|\mathsf{M}|, |x|, s, \lambda)$.

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

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 evaluation. (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 $m \leq s$, 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 $\mathsf{SRE} = (\mathsf{SRE.Encode}, \mathsf{SRE.Eval})$ with the following syntax:

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

$\mathsf{Eval}(C, \widetilde{x}, t) \to y$**.** 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, $|C|, |x|, t \leq \ell(\lambda)$ 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,

$$\mathsf{Sim}\left(1^\lambda, C, C^t(x), t\right) \approx_c \mathsf{Encode}(1^\lambda, C, x, t).$$

- **Semi ε-succinctness:** There exists a polynomial poly such that for any λ, C, x and $t \leq 2^\lambda$, the running time of $\mathsf{Encode}(1^\lambda, C, x, t)$ is at most $t^\epsilon \cdot \mathsf{poly}(|C|, \lambda)$.

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

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 ε-succinct randomized encoding for repeated circuits, there also exists semi ε-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 $s(n) \geq n$. We consider the uniform circuit S_M of size $s \cdot \mathsf{poly}(|M|)$, 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 Weak Laconic Functional Evaluation for Repeated Circuits

We consider a laconic functional evaluation scheme, where the function is known at setup time. 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 weak laconic functional evaluation scheme for repeated circuits consists of PPT algorithms $\mathsf{LFE} = (\mathsf{LFE.Setup}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$ with the following syntax:

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

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

$\mathsf{Dec}_{\mathsf{pk}}(C, \mathsf{ct}, t) \to y$. The decryption algorithm takes in a public key pk, 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 λ, circuit C, repetition parameter t, and inputs x, s.t. $|C|, |x|, t \le \ell(\lambda)$,
$$\Pr\left[C^t(x) \neq y : \begin{array}{c} \mathsf{pk} \leftarrow \mathsf{Setup}(1^\lambda, C, t) \\ \mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x) \\ y \leftarrow \mathsf{Dec}_{\mathsf{pk}}(C, \mathsf{ct}, t) \end{array}\right] = \mathsf{negl}(\lambda).$$
- **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,
$$\left\{\mathsf{pk}, \mathsf{Sim}_{\mathsf{pk}}(C^t(x))\right\} \approx_c \left\{\mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(x)\right\},$$
where in both distributions $\mathsf{pk} \leftarrow \mathsf{Setup}(1^\lambda, C, t)$.
- **Succinctness:** There exists a polynomial $\mathsf{poly}(\lambda)$, such that for any λ, circuit $C : \{0,1\}^n \to \{0,1\}^n$, repetition parameter t, keys pk in the support of $\mathsf{Setup}(1^\lambda, C, t)$, and ciphertext ct in the support of $\mathsf{Enc}_{\mathsf{pk}}(x \in \{0,1\}^n)$,
$$|\mathsf{pk}|, |\mathsf{ct}| \le n \cdot \mathsf{poly}(\lambda) \quad .$$
Furthermore, $\mathsf{Enc}_{\mathsf{pk}}(x \in \{0,1\}^n)$ can be computed by a uniform circuit of size $n \cdot \mathsf{poly}(\lambda)$.
- **Efficiency preservation:** There exists a polynomial $\mathsf{poly}(\lambda)$, such that for any λ, 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
$$\text{for all } r \in \{0,1\}^m: \qquad S_C^t(r) = \mathsf{Setup}(1^\lambda, C, t; r) \quad .$$
Furthermore,
$$m \le n \cdot \mathsf{poly}(\lambda), \qquad S_C \le |C| \cdot \mathsf{poly}(\lambda),$$
and S_C can be efficiently computed from C in time $O(|S_C|)$.
- **Efficient decryption:** There exists a polynomial $\mathsf{poly}(|C|, \lambda)$ such that for any λ, 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 in the support of $\mathsf{Setup}(1^\lambda, C, t)$, any ct in the support of $\mathsf{Enc}_{\mathsf{pk}}(x)$, the running time of $\mathsf{Dec}_{\mathsf{pk}}(\cdot)$ is at most $t \cdot \mathsf{poly}(|C|, \lambda)$.

Remark 2.8. In traditional laconic functional evaluation [QWW18], there is a global trusted crs, sampled which then allows a party to compute a short digest for any function deterministically. In our notion a single function is known at setup time, and the setup is randomized, accordingly no CRS is needed. Such LFE is potentially weaker than traditional LFE, and in particular may not imply collision-resistant hash functions. That said, we are not aware of simpler/better constructions than the traditional stronger notion of LFE.

2.3 Garbled Circuits

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

Definition 2.9 (Garbled Circuits)**.** A circuit garbling scheme consists of the following PPT algorithms $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Enc}, \mathsf{GC.Eval})$ with the syntax:

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

$\mathsf{Enc}_{\mathsf{gsk}}(x_i \in \{0,1\}, i \in [n]) \to \widetilde{x}_i$**.** 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 $\widetilde{x}_i$.

$\mathsf{Eval}\left(\widetilde{C}, \{\widetilde{x}_i\}_{i\in[n]}\right) \to y$**.** 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 λ, and all valid inputs $x \in \{0,1\}^n$,
$$\Pr\left[C(x) = y : \begin{array}{c} \mathsf{gsk} \leftarrow \{0,1\}^\lambda \\ \widetilde{C} \leftarrow \mathsf{Garble}_{\mathsf{gsk}}(C) \\ \forall i \in [n], \widetilde{x}_i \leftarrow \mathsf{Enc}_{\mathsf{gsk}}(x_i, i) \\ y \leftarrow \mathsf{Eval}\left(\widetilde{C}, \{\widetilde{x}_i\}_{i\in[n]}\right) \end{array}\right] = 1.$$
- **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,
$$\left\{\mathsf{Sim}\left(1^\lambda, 1^{|C|}, 1^n, C(x)\right)\right\} \approx_c \left\{\left(\widetilde{C}, \{\widetilde{x}\}_{i\in[n]}\right) : \begin{array}{c} \mathsf{gsk} \leftarrow \{0,1\}^\lambda, \\ \widetilde{C} \leftarrow \mathsf{Garble}_{\mathsf{gsk}}(C) \\ \left\{\widetilde{x}_i \leftarrow \mathsf{Enc}_{\mathsf{gsk}}(x_i, i)\right\}_{i\in[n]}. \end{array}\right\}$$
- **Efficient encoding:** There is a polynomial $\mathsf{poly}(\lambda)$, such that for any λ and circuit $C : \{0,1\}^n \to \{0,1\}^m$, where $|C|, n \le 2^\lambda$, the input encoder $\mathsf{Enc}_{\mathsf{gsk}}(\cdot)$ is computable by a uniform circuit of size at most $\mathsf{poly}(\lambda)$ and $\mathsf{Garble}_{\mathsf{gsk}}(\cdot)$ is computable by a uniform circuit of size at most $|C| \cdot \mathsf{poly}(\lambda)$.

3 Boosting Succinctness

In this section, we present a boosting theorem that takes any semi ε-succinct SRE, for constant $\varepsilon \le 1$, and turns it into an semi ε'-succinct SRE for $\varepsilon' = \varepsilon/(1+\varepsilon)$, relying on efficiency-preserving laconic 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 $O(|X|+|Y|)+\mathsf{poly}(\lambda)$ such that for any $x, y, t \in \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^\lambda$,*

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

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 $\mathsf{ind} \stackrel{?}{=} 1$, circuit description of X for one branch, circuit description of X and Y for the other branch. Thus, $|Z| \leq O(|X|, |Y|) + \mathsf{poly}(\log t)$, where the polynomial is the size of the branching circuit and does not depend on circuits X, Y.

Finally, for $0 < i < t$, $Z^i(x,y,t)|_{1\ldots k}$ 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 $\mathsf{ind} \in \{0,1\}^\lambda$
Outputs: updated parameters $x' \in \{0,1\}^n$, $y' \in \{0,1\}^m$, $\mathsf{ind}' \in \{0,1\}^\lambda$

1. If $\mathsf{ind} > 1$, compute $x' \leftarrow X(x)$. Output updated computation, $(x', y, \mathsf{ind} - 1)$.
2. If $\mathsf{ind} = 1$,
   (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 $Z\,[X, Y]\,(x, y, \mathsf{ind})$

[a] We assume without loss of generality, we can pad the circuit Y so $m \geq k$ and hence, $n + m + \lambda \geq k$.

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

- Let the LFE.Setup algorithm output $\mathsf{pk} \in \{0,1\}^p$ and sample from a randomness space $r \in \{0,1\}^{\ell_s}$.
- Let the LFE.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 2.9.
  - The circuit we wish to garble in our construction takes an input of the form $(\mathsf{pk}, \mathsf{st})$ where $\mathsf{pk} \in \{0,1\}^p$, $\mathsf{st} \in \{0,1\}^s$. For concise notation and ease of readability, we simplify our garbling encoding algorithm to write $\widetilde{\mathsf{pk}} \leftarrow \mathsf{Enc}_{\mathsf{gsk}}(\mathsf{pk}, [p])$, and $\widetilde{\mathsf{st}} \leftarrow \mathsf{Enc}_{\mathsf{gsk}}(\mathsf{st}, p + [s])$, 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 $|\widetilde{\mathsf{pk}}|, |\widetilde{\mathsf{st}}|$ be denoted by $\widetilde{p}, \widetilde{s}$.

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

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

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 $t = t_1 \cdot t_2$.[3]
2. Let $\mathtt{E}$ be the FE encryption circuit which takes as input public key $\mathsf{pk} \in \{0,1\}^p$, current state $\mathsf{st} \in \{0,1\}^s$, and has hardwired values $\mathsf{gsk} \in \{0,1\}^\lambda$, randomness for encryption $r_e \in \{0,1\}^{\ell_e}$, repetition parameter t_2, and outputs a LFE ciphertext ct (encrypting st) and a garbled public key $\widetilde{\mathsf{pk}}$ (encrypting pk with respect to gsk).
   Circuit $\mathtt{E}[\mathsf{gsk}, r_e, t_2](\mathsf{pk}, \mathsf{st})$.
   (a) Compute $\mathsf{ct} \leftarrow \mathsf{LFE.Enc}_{\mathsf{pk}}(\mathsf{st}, \mathsf{gsk}, t_2; r_e)$.
   (b) Compute $\widetilde{\mathsf{pk}} \leftarrow \mathsf{GC.Enc}_{\mathsf{gsk}}(\mathsf{pk}, [p])$.
   (c) Output $\left(\mathsf{ct}, \widetilde{\mathsf{pk}}\right)$.

   Let $\mathsf{gsk}_{t_1+1} \leftarrow \{0,1\}^\lambda$. For $i \in [t_1]$,
   (a) Sample $\mathsf{gsk}_i \leftarrow \{0,1\}^\lambda$.
   (b) Sample random coins $r_{e,i} \leftarrow \{0,1\}^{\ell_e}$ for the LFE.Enc algorithm.
   (c) Consider the circuit $\mathtt{E}_i = \mathtt{E}\left[\mathsf{gsk}_{i+1}, r_{e,i}, t_2\right]$, 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, $\widetilde{\mathtt{E}}_i \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}_i}(\mathtt{E}_i)$.

   Let $\widetilde{\mathtt{E}}_{t_1+1} \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}_{t_1+1}}(\mathtt{I})$, 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 st.

[3] We assume these powers of t are integral, and avoid ceiling notation for ease of readability.

3. Let $\mathtt{C} : \{0,1\}^{s'} \to \{0,1\}^{s'}$, 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 $\mathsf{st} \in \{0,1\}^s$ and a garbling secret key gsk. It outputs $\widetilde{\mathsf{st}} \leftarrow \mathsf{GC.Enc}_{\mathsf{gsk}}(\mathsf{st}, p + [s])$.
   - $\mathtt{C}$ 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 $\mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2; r_s\right)$ be denoted by some repeated circuit $S_{\mathtt{C}}$ such that,
$$\text{for all } r_s : \mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2; r_s\right) = S_{\mathtt{C}}^{t_2}(r_s)$$
5. Let $\mathtt{K}$ 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 $S_{\mathtt{C}} : \{0,1\}^{s'} \to \{0,1\}^{s'}$.
   - Circuit Y takes as input state $\mathsf{st} \in \{0,1\}^{s'}$, additional input (gsk, x). Parse st as pk. Output pk and $\widetilde{\mathsf{pk}} \leftarrow \mathsf{GC.Enc}_{\mathsf{gsk}}(\mathsf{pk}, [p])$ and $\widetilde{x} \leftarrow \mathsf{GC.Enc}_{\mathsf{gsk}}(x, p + [s])$.
   - $S_{\mathtt{C}}$ is then the resulting circuit $Z[X, Y]$, so that the repeated Z^i applies the repeated $S_{\mathtt{C}}^i$ and finally outputs the resulting public key pk, garbled public key $\widetilde{\mathsf{pk}}$, and garbled $\widetilde{x}$.
6. Sample random coins $r_s \leftarrow \{0,1\}^{s'}$, and compute,
$$\mathsf{SRE}.\widetilde{x} \leftarrow \mathsf{SRE.Encode}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(r_s, \left(\mathsf{gsk}_1, x\right), t_2\right), t_2\right).$$
7. Output $\mathsf{SRE}'.\widetilde{x} \leftarrow \left(\mathsf{SRE}.\widetilde{x}, \left\{\widetilde{\mathtt{E}}_i\right\}_{i \in [t_1+1]}\right)$.

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

1. Parse $\widetilde{x} = \left(\mathsf{SRE}.\widetilde{x}, \left\{\widetilde{\mathtt{E}}_i\right\}_{i \in [t_1+1]}\right)$.
2. Let circuits $C, \mathtt{C}, S_{\mathtt{C}}, \mathtt{K}$, variables t_1, t_2 be defined as in $\mathsf{SRE}'.\mathsf{Encode}$.
3. Compute $\left(\mathsf{pk}, \widetilde{\mathsf{pk}}, \widetilde{x}\right) \leftarrow \mathsf{SRE.Eval}\left(\mathtt{K}[S_{\mathtt{C}}], \mathsf{SRE}.\widetilde{x}, t_2\right)|_{1 \ldots \widetilde{p}+\widetilde{s}}$. Let $\widetilde{\mathsf{inp}}_1 = (\widetilde{\mathsf{pk}}, \widetilde{x})$.
4. For $i \in [t_1]$,
   (a) Compute $\left(\mathsf{ct}_i, \widetilde{\mathsf{pk}}_{i+1}\right) \leftarrow \mathsf{GC.Eval}\left(\widetilde{\mathtt{E}}_i, \widetilde{\mathsf{inp}}_i\right)$.
   (b) Compute, $\widetilde{\mathsf{st}}_{i+1} \leftarrow \mathsf{LFE.Dec}_{\mathsf{pk}}\left(\mathtt{C}, \mathsf{ct}_i, t_2\right)|_{1 \ldots \widetilde{s}}$.
   (c) Let $\widetilde{\mathsf{inp}}_{i+1} \leftarrow (\widetilde{\mathsf{pk}}_{i+1}, \widetilde{\mathsf{st}}_{i+1})$.
5. Let $\mathsf{st}_{t_1+1} \leftarrow \mathsf{GC.Eval}\left(\widetilde{\mathtt{E}}_{t_1+1}, \widetilde{\mathsf{inp}}_{t_1+1}\right)$.

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* LFE *is a correct scheme according to Definition 2.7, and* GC *is a correct scheme according to Definition 2.9, 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 $C, \mathtt{C}, S_{\mathtt{C}}, \mathtt{K}, t_1, t_2, \mathsf{SRE}'.\widetilde{x}$ 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 $\mathtt{K}$ on input $\left(r_s, \left(\mathsf{gsk}_1, x\right), t_2\right)$ after t_2 repetitions. From correctness of Lemma 3.1, we have that
$$\mathtt{K}^{t_2}\left(r_s, \left(\mathsf{gsk}_1, x\right), t_2\right)|_{1\ldots\widetilde{p}+\widetilde{s}} = Y\left(S_{\mathtt{C}}^{t_2}\left(r_s\right), \left(\mathsf{gsk}_1, x\right)\right) = Y\left(\mathsf{pk}, \left(\mathsf{gsk}_1, x\right)\right)$$
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 $\widetilde{\mathsf{inp}}_1 = \left(\widetilde{\mathsf{pk}}, \widetilde{x}\right)$ 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 $\left(\mathsf{pk}, C^{t_2\cdot(j-1)}(x)\right)$ using 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 $\left(\mathsf{pk}, C^{t_2\cdot(j-1)}(x)\right)$ is garbled using gsk_{j^*} and $\widetilde{\mathsf{E}}_{j^*}$ is garbled using gsk_{j^*}. We have that
     $$\left(\mathsf{ct}_{j^*}, \widetilde{\mathsf{pk}}'\right) = \mathtt{E}\left[\mathsf{gsk}_{j^*+1}, r_{e,j^*}\right]\left(\mathsf{pk}, C^{t_2\cdot(j^*-1)}(x)\right), \text{ where}$$
     $$\mathsf{ct}_{j^*} = \mathsf{LFE.Enc}_{\mathsf{pk}}\left(C^{t_2\cdot(j^*-1)}(x), \mathsf{gsk}_{j^*+1}, t_2\right), \text{ and } \widetilde{\mathsf{pk}}' = \mathsf{GC.Enc}_{\mathsf{gsk}_{j^*+1}}(\mathsf{pk}, [p])$$
     (b) Next the algorithm performs LFE decryption,
     $$\widetilde{\mathsf{st}}' = \mathsf{LFE.Dec}_{\mathsf{pk}}\left(\mathtt{C}, \mathsf{ct}_{j^*}, t_2\right)|_{1\ldots\widetilde{s}}, \text{ where}$$
     $$\widetilde{\mathsf{st}}' = \mathtt{C}^{t_2}\left(C^{t_2\cdot(j^*-1)}(x), \mathsf{gsk}_{j^*+1}, t_2\right)|_{1\ldots\widetilde{s}},$$
     here the second equality holds with overwhelming probability from LFE correctness.
     – From definition of $\mathtt{C}$,
     $$\begin{aligned}\mathtt{C}^{t_2}\left(C^{t_2\cdot(j^*-1)}(x), \mathsf{gsk}_{j^*+1}, t_2\right)|_{1\ldots\widetilde{s}} &= Y\left(C^{t_2}\left(C^{t_2\cdot(j^*-1)}(x)\right), \mathsf{gsk}_{j^*+1}\right)|_{1\ldots\widetilde{s}} \\ &= Y\left(C^{t_2\cdot(j^*)}(x), \mathsf{gsk}_{j^*+1}\right)|_{1\ldots\widetilde{s}}\end{aligned}$$

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

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

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

As $\mathsf{st}_{t_1+1} = C^t(x)$, 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* LFE *is succinct, efficiency preserving and has efficient decryption according to Definition 2.7, and* GC *has efficient encoding according to Definition 2.9, 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 $\mathsf{poly}(\cdot)$ to denote different universal polynomials (in λ), such that for any λ, circuit $C : \{0,1\}^s \to \{0,1\}^s$, $x \in \{0,1\}^s$ an input, and $t \le 2^\lambda$, we have:

- Circuit $\mathtt{C}$. Recall that $\mathtt{C} : \{0,1\}^{s'} \to \{0,1\}^{s'}$ is a circuit that computes C and then garbles the resulting state. From Lemma 3.1, $|\mathtt{C}| \le O(|C| + |Y|) + \mathsf{poly}(\lambda)$ (where circuit Y garbles s bit input). As the garbled circuit has efficient encoding, $|Y| \le s \cdot \mathsf{poly}(\lambda)$ and $|\mathtt{C}| \le |C| \cdot \mathsf{poly}(\lambda)$. In particular, in what follows, $s' \le |C| \cdot \mathsf{poly}(\lambda)$. Additionally, from Lemma 3.1, computing the description of circuit $\mathtt{C}$ takes time $O(|\mathtt{C}|)$.

- Laconic Functional Evaluation scheme LFE. Recall that $p, \ell_s, |\mathsf{ct}|, \ell_e$ 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 $\mathtt{C}$ are of length at most s' and the underlying LFE scheme is succinct, we have, $p, \ell_s, |\mathsf{ct}|, \ell_e \le |C| \cdot \mathsf{poly}(\lambda)$.

- Circuit $\mathtt{E}$. Recall that $\mathtt{E}$ is the LFE encryption circuit. Since the garbled circuit has efficient encoding and garbles inputs of length p, and $p, s, \ell_e \le |C| \cdot \mathsf{poly}(\lambda)$, $|\mathtt{E}| \le |C| \cdot \mathsf{poly}(\lambda)$. Additionally, our LFE scheme is succinct and computing the description of circuit $\mathtt{E}$ takes time $O(|\mathtt{E}|)$.

- Circuit $S_{\mathtt{C}}$. Recall that this is the repeated circuit representation of the key generation algorithm on circuit $\mathtt{C}$. Since our LFE scheme is efficiency preserving, we have that $S_{\mathtt{C}} : \{0,1\}^m \to \{0,1\}^m$, where $m \le s' \cdot \mathsf{poly}(\lambda) \le |C| \cdot \mathsf{poly}(\lambda)$. Additionally, $|S_{\mathtt{C}}| \le |\mathtt{C}| \cdot \mathsf{poly}(\lambda) \le |C| \cdot \mathsf{poly}(\lambda)$. Additionally,computing the description of circuit $S_{\mathtt{C}}$ takes time $O(|S_{\mathtt{C}}|)$.

- Circuit $\mathtt{K}$. Recall that $\mathtt{K}$ is the function key generator circuit. From Lemma 3.1, $|\mathtt{K}| \le O(|S_{\mathtt{C}}| + |Y|) + \mathsf{poly}(\lambda)$ where the circuit Y garbles a $p + s$ bit input. Since the garbled circuit has efficient encoding, $|Y| \le (p + s) \cdot \mathsf{poly}(\lambda)$ and we have from the values set above that, $|\mathtt{K}| \le |C| \cdot \mathsf{poly}(\lambda)$. Additionally, from Lemma 3.1, computing the description of circuit $\mathtt{K}$ takes time $O(|\mathtt{K}|)$.

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

- We've shown above that the circuits $\mathtt{C}, \mathtt{E}, S_{\mathtt{C}}, \mathtt{K}$ are computable by SRE′.Encode in time $O(|\mathtt{C}|)$, $O(|\mathtt{E}|)$, $O(|S_{\mathtt{C}}|)$, $O(|\mathtt{K}|)$ respectively and each of the circuits are bounded by size $|C| \cdot \mathsf{poly}(\lambda)$. Hence, these circuits can be computed in time at most $|C| \cdot \mathsf{poly}(\lambda)$.

- Succinct randomized encoding scheme $(\mathsf{SRE.Encode}, \mathsf{SRE.Eval})$ is run on $\mathtt{K}$ with number of steps t_2. Since SRE is semi $t^{1/c}$-succinct, the runtime of running the inner SRE is at most $t_2^{1/c} \cdot \mathsf{poly}(|\mathtt{K}|, \lambda) \leq t_1 \cdot \mathsf{poly}(|C|, \lambda)$.
- The time taken to garble t_1 circuits of size $|\mathtt{E}| \leq |C| \cdot \mathsf{poly}(\lambda)$ is $t_1 \cdot |C| \cdot \mathsf{poly}(\lambda)$.

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 $t^{1/(c+1)} \cdot \mathsf{poly}(|C|, \lambda)$.

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

- Succinct randomized encoding scheme $(\mathsf{SRE.Encode}, \mathsf{SRE.Eval})$ is run on $\mathtt{K}$ with number of steps t_2. Since SRE has semi-efficient evaluation, the runtime of running the inner SRE is at most $t_2 \cdot \mathsf{poly}(|\mathtt{K}|, \lambda) \leq t_2 \cdot \mathsf{poly}(|C|, \lambda)$.
- Resulting evaluation procedure. For $i \in [t_1]$,
  - We compute a garbled circuit evaluation, where the garbled inputs and circuit are at most $\mathsf{poly}(|C|, \lambda)$.
  - The resulting LFE, has efficient decryption and can decrypt the computation of $\mathtt{C}$ in time $t_2 \cdot \mathsf{poly}(|C|, \lambda)$.
  - The size of $\widetilde{\mathsf{inp}}_{i+1}$ is atmost $\mathsf{poly}(|C|, \lambda)$ (as size of $\mathtt{K}$ is at most $\mathsf{poly}(|C|, \lambda)$).

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

□

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* $t^{1/c} \cdot |C| \cdot \mathsf{poly}(\lambda)$ *(namely, linear dependence on* $|C|$*), then the running time of the resulting* SRE′.Encode *is at most* $t^{1/(c+1)} \cdot |C| \cdot \mathsf{poly}(\lambda)$.

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

In Appendix B , 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, where we run the evaluation algorithm in a RAM model.

Proposition 3.7. *Assuming* SRE *is secure according to Definition 2.1, assuming* LFE *is secure according to Definition 2.7, and* GC *is secure according to Definition 2.9, then* SRE′ *given by Construction 3.3 is secure according to Definition 2.1.*

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

Let the circuits $C, \mathtt{E}, \mathtt{C}, S_{\mathtt{C}}, \mathtt{K}$ and parameters t_1, t_2 and LFE 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 $\mathtt{E}[\mathsf{gsk}, r, t_2]$ be the LFE encryption circuit and u be the size of the circuit. Let $\mathtt{C}$ be the repeat-then-encrypt circuit, and let $S_{\mathtt{C}}$ be the repeated circuit

corresponding to $\mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2; \cdot\right)$. Let $\mathtt{K}[S_{\mathtt{C}}]$ be the key-generation circuit. Let $p, \ell_s, |\mathsf{ct}|, \ell_e$ be the public key, secret key, randomness for the setup algorithm, ciphertext length and randomness for the encryption algorithm respectively.

1. Compute $\mathsf{pk} \leftarrow \mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2\right)$.
2. Let $\left(\widetilde{\mathtt{E}}_{t_1+1}, \widetilde{\mathsf{inp}}_{t_1+1}\right) \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{|\mathtt{I}|}, 1^{p+s}, \left(C^t(x)\right)\right)$, 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 st.
   For $i \in \{t_1, \ldots, 1\}$,
   - Let $\left(\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}'\right) = \widetilde{\mathsf{inp}}_{i+1}$. Compute $\mathsf{ct}_i \leftarrow \mathsf{LFE.Sim}_{\mathsf{pk}}\left(\widetilde{\mathsf{st}}', 0 \ldots 0\right)$.
   - Compute $\left(\widetilde{\mathtt{E}}_i, \widetilde{\mathsf{inp}}_i\right) \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^u, 1^{p+s}, \left(\mathsf{ct}_i, \widetilde{\mathsf{pk}}'\right)\right)$.
3. Simulate SRE,
$$\mathsf{SRE}.\widetilde{x} \leftarrow \mathsf{SRE.Sim}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(\mathsf{sk}, \widetilde{\mathsf{inp}}_1\right), t_2\right).$$
4. Output $\mathsf{SRE}'.\widetilde{x} = \left(\mathsf{SRE}.\widetilde{x}, \left\{\widetilde{\mathtt{E}}_i\right\}_{i \in [t_1+1]}\right)$.

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

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

1. Let the circuits $C, \mathtt{E}, \mathtt{C}, S_{\mathtt{C}}, \mathtt{K}$ and parameters t_1, t_2 and LFE scheme parameters be defined similarly to $\mathsf{SRE}'.\mathsf{Encode}$.
2. Let $\mathsf{gsk}_{t_1+1} \leftarrow \{0,1\}^\lambda$. For $i \in [t_1]$, let $\mathtt{E}$ be the FE encryption circuit,
   (a) Sample $\mathsf{gsk}_i \leftarrow \{0,1\}^\lambda$.
   (b) Sample random coins $r_{e,i} \leftarrow \{0,1\}^{\ell_e}$ for the LFE.Enc algorithm.
   (c) Consider the circuit $\mathtt{E}_i = \mathtt{E}\left[\mathsf{gsk}_{i+1}, r_{e,i}, t_2\right]$, 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, $\widetilde{\mathtt{E}}_i \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}_i}\left(\mathtt{E}_i\right)$.

   Let $\widetilde{\mathtt{E}}_{t_1+1} \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}_{t_1+1}}(\mathtt{I})$, 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 st.
3. Let $\mathtt{C} : \{0,1\}^{s'} \to \{0,1\}^{s'}$, be the repeat-then-encrypt circuit. Let $\mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2; r_s\right)$ be denoted by some repeated circuit $S_{\mathtt{C}}(r_s)$, and $\mathtt{K}$ be the functional key generation circuit.
   Sample random coins $r_s \leftarrow \{0,1\}^{s'}$, and compute,
$$\mathsf{SRE}.\widetilde{x} \leftarrow \mathsf{SRE.Encode}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(r_s, \left(\mathsf{gsk}_1, x\right), t_2\right), t_2\right).$$
4. Output $\mathsf{SRE}'.\widetilde{x} \leftarrow \left(\mathsf{SRE}.\widetilde{x}, \left\{\widetilde{\mathtt{E}}_i\right\}_{i \in [t_1+1]}\right)$.

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

3. Let $\mathtt{C} : \{0,1\}^{s'} \to \{0,1\}^{s'}$, be the repeat-then-encrypt circuit and $\mathtt{K}$ be the functional key generation circuit.
   Compute $\mathsf{pk} \leftarrow \mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2\right)$.
   Compute $\widetilde{\mathsf{pk}} = \mathsf{GC.Enc}_{\mathsf{gsk}_1}(\mathsf{pk}, [p])$ and $\widetilde{\mathsf{st}} = \mathsf{GC.Enc}_{\mathsf{gsk}_1}(x, p + [s])$. Let $\widetilde{\mathsf{inp}_1} = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$.
   Simulate SRE,
$$\mathsf{SRE}.\widetilde{x} \leftarrow \mathsf{SRE.Sim}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(\mathsf{pk}, \widetilde{\mathsf{inp}_1}\right), t_2\right).$$

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 $\mathsf{gsk}_{t_1+1} \leftarrow \{0,1\}^\lambda$. For $i \in \{j+1, \ldots, t_1\}$, let $\mathtt{E}$ be the FE encryption circuit,
   (a) Sample $\mathsf{gsk}_i \leftarrow \{0,1\}^\lambda$.
   (b) Sample random coins $r_{e,i} \leftarrow \{0,1\}^{\ell_e}$ for the LFE.Enc algorithm.
   (c) Consider the circuit $\mathtt{E}_i = \mathtt{E}\left[\mathsf{gsk}_{i+1}, r_{e,i}, t_2\right]$, 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, $\widetilde{\mathtt{E}}_i \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}_i}(\mathtt{E}_i)$.
   Let $\widetilde{\mathtt{E}}_{t_1+1} \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}_{t_1+1}}(\mathtt{I})$, 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 st.
3. Let $\mathtt{C} : \{0,1\}^{s'} \to \{0,1\}^{s'}$, be the repeat-then-encrypt circuit and $\mathtt{K}$ be the functional key generation circuit.
   Compute $\mathsf{pk} \leftarrow \mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2\right)$.
   Compute $\widetilde{\mathsf{pk}} = \mathsf{GC.Enc}_{\mathsf{gsk}_{j+1}}(\mathsf{pk}, [p])$ and $\widetilde{\mathsf{st}} = \mathsf{GC.Enc}_{\mathsf{gsk}_{j+1}}\left(C^{t_2 \cdot j(x)}, p + [s]\right)$, where $C^0(x) \leftarrow x$.
   Let $\widetilde{\mathsf{inp}_{j+1}} = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$.
   For $i \in \{j, \ldots, 1\}$,
   (a) Let $(\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}') = \widetilde{\mathsf{inp}}_{i+1}$.
   (b) Compute $\mathsf{ct}_i \leftarrow \mathsf{LFE.Sim}_{\mathsf{pk}}\left(\widetilde{\mathsf{st}}', 0 \ldots 0\right)$.
   (c) Compute $\left(\widetilde{\mathtt{E}}_i, \widetilde{\mathsf{inp}}_i\right) \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^u, 1^{p+s}, \left(\mathsf{ct}_i, \widetilde{\mathsf{pk}}'\right)\right)$.
   Compute $\widetilde{\mathsf{pk}} = \mathsf{GC.Enc}_{\mathsf{gsk}_1}(\mathsf{pk}, [p])$ and $\widetilde{\mathsf{st}} = \mathsf{GC.Enc}_{\mathsf{gsk}_1}(x, p + [s])$. Let $\widetilde{\mathsf{inp}_1} = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$.
   Simulate SRE,
$$\mathsf{SRE}.\widetilde{x} \leftarrow \mathsf{SRE.Sim}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(\mathsf{pk}, \widetilde{\mathsf{inp}_1}\right), t_2\right).$$

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 $C, \mathtt{E}, \mathtt{C}, S_{\mathtt{C}}, \mathtt{K}$ be defined similarly to step 1 of hybrid -1.
   Compute $\mathsf{pk} \leftarrow \mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2\right)$.
2. Let $\left(\widetilde{\mathtt{E}}_{t_1+1}, \widetilde{\mathsf{inp}}_{t_1+1}\right) \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{|\mathtt{I}|}, 1^{p+s}, \left(C^t(x)\right)\right)$, 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 st.
   For $i \in \{t_1, \ldots, 1\}$,

(a) Let $(\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}') = \widetilde{\mathsf{inp}}_{i+1}$.

(b) Compute $\mathsf{ct}_i \leftarrow \mathsf{LFE.Sim}_{\mathsf{pk}}\left(\widetilde{\mathsf{st}}', 0 \dots 0\right)$.

(c) Compute $\left(\widetilde{\mathrm{E}}_i, \widetilde{\mathsf{inp}}_i\right) \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^u, 1^{p+s}, \left(\mathsf{ct}_i, \widetilde{\mathsf{pk}}'\right)\right)$.

3. Simulate SRE,
$$\mathsf{SRE}.\widetilde{x} \leftarrow \mathsf{SRE.Sim}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(\mathsf{pk}, \widetilde{\mathsf{inp}}_1\right), t_2\right).$$

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

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

We write 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,
$$\mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}\left[C, t_2\right], t_2; r_s\right) = S_{\mathtt{C}}^{t_2}(r_s),$$
and hence the two processes have the exact same output distribution pk. Accordingly also the garbled input encoding $\widetilde{\mathsf{pk}}$ is distributed the same in both hybrids and so is $\widetilde{\mathsf{inp}}_1 = (\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}})$.

Hence, from indistinguishability of the underlying SRE, we have,
$$\mathsf{SRE.Encode}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(r_s, x, \mathsf{gsk}_1\right), t_2\right) \approx_c \mathsf{SRE.Sim}\left(1^\lambda, \mathtt{K}\left[S_{\mathtt{C}}\right], \left(\mathsf{pk}, \widetilde{\mathsf{inp}}_1\right), t_2\right),$$
which concludes the proof. □

Claim 3.9. *Suppose* LFE *is selectively-secure according to Definition 2.7 and* GC *is secure according to Definition 2.9. Then, for* $j \in [t_1]$,
$$\mathsf{Hyb}_{j-1} \approx_c \mathsf{Hyb}_j.$$

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 LFE encryption algorithm.

3. Let $\mathtt{C} : \{0,1\}^{s'} \to \{0,1\}^{s'}$, be the repeat-then-encrypt circuit and $\mathtt{K}$ be the functional key generation circuit.
Compute $\mathsf{pk} \leftarrow \mathsf{LFE.Setup}\left(1^\lambda, \mathtt{C}, t_2\right)$.
Compute $\widetilde{\mathsf{pk}} = \mathsf{GC.Enc}_{\mathsf{gsk}_{j+1}}(\mathsf{pk}, [p])$ and $\widetilde{\mathsf{st}} = \mathsf{GC.Enc}_{\mathsf{gsk}_{j+1}}\left(C^{t_2\cdot j}(x), p + [s]\right)$, 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 $\mathsf{ct}_j \leftarrow \mathsf{LFE.Enc}_{\mathsf{pk}}\left(C^{t_2\cdot(j-1)}(x), \mathsf{gsk}_{j+1}, t_2; r_{e,j}\right)$, where $C^0(x) \leftarrow x$.

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

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

(a) Let $\left(\widetilde{\mathsf{E}}_i, \widetilde{\mathsf{inp}}_i\right) \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^u, 1^{p+s}, \left(\mathsf{LFE.Sim}_{\mathsf{pk}}\left(\widetilde{\mathsf{st}}', 0\ldots0\right), \widetilde{\mathsf{pk}}'\right)\right)$, where $(\widetilde{\mathsf{pk}}', \widetilde{\mathsf{st}}') = \widetilde{\mathsf{inp}}_{i+1}$.

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

$$\mathsf{SRE}.\widetilde{x} \leftarrow \mathsf{SRE.Sim}\left(1^\lambda, \mathtt{K}\left[S_{\mathsf{C}}\right], \left(\mathsf{pk}, \widetilde{\mathsf{inp}}_1\right), t_2\right).$$

Claim 3.10. *Suppose* GC *is secure according to Definition 2.9. 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 $\widetilde{\mathsf{inp}}_j$ 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 $\mathsf{Hyb}_{(j-1)}$.
  - We sample $\mathsf{gsk}_j, \ldots, \mathsf{gsk}_{t_1}$.
  - We compute $\widetilde{\mathsf{pk}} = \mathsf{GC.Enc}_{\mathsf{gsk}_j}(\mathsf{pk}, [p])$, $\widetilde{\mathsf{st}} = \mathsf{GC.Enc}_{\mathsf{gsk}_j}(\mathsf{st}, p + [s])$ where $\mathsf{st} = C^{t_2\cdot(j-1)}(x)$. Let $\widetilde{\mathsf{inp}}_j \leftarrow \left(\widetilde{\mathsf{pk}}, \widetilde{\mathsf{st}}\right)$.
  - We compute $\widetilde{\mathsf{E}}_j$ as follows.
    1. Consider the circuit $\mathsf{E}_j = \mathtt{E}\left[\mathsf{gsk}_{j+1}, r_{e,j}, t_2\right]$, where we hardcode, the garbling secret key gsk_{j+1}, and the random coins $r_{e,j}$.
    2. Compute a garbled circuit, $\widetilde{\mathsf{E}}_j \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}_j}\left(\mathsf{E}_j\right)$.
- In $\mathsf{Hyb}_{(j-1).5}$.
  - We sample $\mathsf{gsk}_{j+1}, \ldots, \mathsf{gsk}_{t_1}$ (or we no longer sample gsk_j).
  - We compute $\widetilde{\mathsf{E}}_j$ as follows.
    1. Compute $\mathsf{ct}_j \leftarrow \mathsf{LFE.Enc}_{\mathsf{pk}}\left(C^{t_2\cdot(j-1)}(x), \mathsf{gsk}_{j+1}, t_2; r_{e,j}\right)$, where $C^0(x) \leftarrow x$.
    2. Compute $\left(\widetilde{\mathsf{E}}_j, \widetilde{\mathsf{inp}}_j\right) \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^u, 1^{p+s}, \left(\mathsf{ct}_j, \widetilde{\mathsf{pk}}'\right)\right)$.
    Where $\widetilde{\mathsf{pk}}' = \mathsf{GC.Enc}_{\mathsf{gsk}_{j+1}}(\mathsf{pk}, [p])$.

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

□

Claim 3.11. *Suppose* LFE *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 $\mathsf{Hyb}_{(j-1).5}$.
  - We set, $\mathsf{ct}_j \leftarrow \mathsf{LFE.Enc}_{\mathsf{pk}}\left(C^{t_2\cdot(j-1)}(x), \mathsf{gsk}_{j+1}, t_2; r_{e,j}\right)$ (where $C^0(x) \leftarrow x$).
- In $\mathsf{Hyb}_{(j)}$.
  - We set $\mathsf{ct}_j \leftarrow \mathsf{LFE.Sim}_{\mathsf{pk}}\left(\widetilde{\mathsf{st}}', 0\ldots0\right)$, where $\widetilde{\mathsf{st}}' = \mathsf{GC.Enc}_{\mathsf{gsk}_{j+1}}\left(C^{t_2\cdot j}(x), p + [s]\right)$.

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

Running $\mathtt{C}^{t_2}$ 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$\left(\widetilde{\mathsf{st}}', 0\ldots0\right)$ 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 $\mathsf{Hyb}_{(j-1)} \approx_c \mathsf{Hyb}_j$. □

Claim 3.12. *Suppose* GC *is secure according to Definition 2.9.*

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

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

From inspection, note that Hyb_{-1} is the real algorithm, 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 Eurocrypt reviewers for helping simplify presentation in Appendix A and pointing out the use of laconic function evaluation instead of single-key FE for our main transformation. 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+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+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+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564, 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^0. 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 simulation-secure laconic function evaluation 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 λ, fix $|C_\lambda|, x_\lambda, t_\lambda \leq \ell(\lambda)$ 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 Sketch. 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 λ).

Let UEvalC_C be a circuit that runs UEval on input $\mathbf{A}_{\mathsf{attr}}, \mathbf{A}_{\mathsf{circ}}, C$, and computes $\mathbf{A}_C = (\mathbf{A}_{C:1}, \ldots, \mathbf{A}_{C:s})$ where $(C:1, \ldots, C:s)$ are the output wires of circuit C. UEvalC_C outputs $(\mathbf{A}_C, \mathbf{A}_{\mathsf{circ}})$. 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,

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

It is easy to see that $|\mathsf{UEvalC}_C| \leq |C| \cdot \mathsf{poly}(\lambda)$ and can be computed in time $O(|\mathsf{UEvalC}_C|)$ as the algorithm garbles gate-by-gate and performs $\mathsf{poly}(\lambda, n, m, q)$ operations on each gate. We show the functionality by induction 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 $\mathbf{A}_{\mathsf{attr}}$ for the $(t+1)$-th computation and the claim follows.

Claim A.3. *AB-LFE scheme (construction 3 in [HLL23]) is efficiency preserving in its setup.*

Proof Sketch. Recall that the setup in construction 3 consists of sampling random matrices to generate crs and running procedure UEvalC on the crs and circuit C to output crs, digest_C.

Similar to Lemma 3.1, if the repeated circuit computation is performed after some non-repeated computation, i.e. we compute $X^t(Y(y), x)$, for some repeated computation X and non-repeated computation Y - the resulting computation can also be represented as a repeated circuit. The claim follows by a straightforward modification of Lemma 3.1.

Claim A.4. *If the AB-LFE scheme is efficiency preserving, then the transformation in section 4.4. [QWW18] to a two-outcome AB-LFE scheme is efficiency-preserving in its setup.*

Proof Sketch. In the transformation, we repeat the circuit into a copy of the circuit and it's negation. Since both circuit C and negation circuit $\tilde{C}$ when run on the original AB-LFE scheme are efficiency-preserving, the resulting setup is also efficiency-preserving.

The LFE scheme in [HLL23] is finally constructed by a generic transformation from a (two-outcome) attribute-based LFE and a fully-homomorphic encryption scheme [GKP+13, QWW18]. We sketch that their template preservers efficiency.

Claim A.5. *The evaluation function of a fully-homomorphic encryption, such as [GSW13], is efficiency-preserving.*

Proof Sketch. Similar to the proof of Claim A.2, most FHE evaluation algorithms (including [GSW13]) in literature perform a gate-by-gate computation on the ciphertext. Since each gate can be replaced by a meta-gate, the evaluation algorithm is efficiency-preserving.

Claim A.6. *If a (two-outcome) AB-LFE scheme has an efficiency preserving setup, then the transformation in section 4.4. [QWW18] to a LFE scheme is efficiency preserving in its setup.*

Proof Sketch. The setup procedure can be summarized as running the (two-outcome) AB-LFE on the function $\mathsf{FHE.Eval}(C^t, \cdot)$. Since the FHE.Eval function and the (two-outcome) AB-LFE function are efficiency preserving, then so is their composition.

$$\mathsf{ABLFE.Setup}\left(1^\lambda, \mathsf{FHE.Eval}(C^t, \cdot)\right) = \mathsf{ABLFE.Setup}\left(1^\lambda, S^t_{\mathsf{FHE.Eval}}(\cdot)\right) = S^t_{\mathsf{ABLFE.Setup}}(\cdot)$$

Combining all the proofs, we have that their scheme satisfies the efficiency-preserving property. □

B Linear Space Dependence of Evaluation Time

The runtime of our evaluation algorithm (Proposition 3.5), depends on the evaluation of the base SRE scheme, garbled circuit evaluation and LFE decryption. This includes how long it takes to evaluate a repeated circuit $\mathtt{K}[S_{\mathsf{C}}]$ on the SRE algorithm and the circuit C on the LFE decryption.

We observe that when evaluating a circuit on a Turing Machine, one critical factor in the evaluation time is the time required to run the circuit. For a general layered circuit with width s and depth d, this could take $s^2 \cdot d$, rather than $s \cdot d$. This inefficiency arises because at each depth of the circuit, the wires connecting gates can be arbitrary. Thus, the Turing machine might have additional head movement overhead to evaluate each gate i.e. the TM head might need to move to the appropriate position, potentially taking s time for every gate. These efficiencies can be mitigated in two ways, (i) use the RAM model - by considering the computation in the RAM model, we can access the memory in $O(1)$ time, (ii) if the circuit is designed such that within each layer, gates depend only on inputs that are spatially close (local connections), then the evaluation time also reduces to $s \cdot d$.

For simplicity, we just state our evaluation efficiency in the RAM model. But since we consider step circuits for a Turing machine (the step circuits only write on the tape in the current head position and move the head left or right on one invocation), it's possible to evaluate this structured step circuit in time $s \cdot d$. Here, we present the simplified proposition only focusing on the RAM model.

Proposition B.1. *Assuming* SRE *has semi-effient evaluation and the* LFE *scheme has efficient-decryption where the running time of* $\mathsf{SRE.Eval}, \mathsf{Dec}_{\mathsf{pk}}(\cdot)$ *is at most* $t \cdot |C| \cdot \mathsf{poly}(\lambda)$ *(namely, linear dependence on* $|C|$*) in the RAM model then the running time of the resulting* $\mathsf{SRE}'.\mathsf{Eval}$ *is at most* $t \cdot |C| \cdot \mathsf{poly}(\lambda)$ *in the RAM model.*

Proof Sktech. Similar to the proof above, because the LFE has the required efficiency properties, the Setup circuit has depth t_2, width $|C| \cdot \mathsf{poly}(\lambda)$, the GC satisfies evaluation efficiency, and the SRE is assumed to have efficient evaluation, then the complete scheme has linear evaluation efficiency in the RAM model.

C Fully Succinct RE from Sub-Exponential Hardness

In this section, we observe that there is a efficiency vs security tradeoff of our succinct-randomized encodings which we can leverage to gain more efficiency at the cost of a higher security loss.

Our scheme satisfies the parameters detailed in Corollary 1.2. We begin by constructing a base scheme that is 1-succinct, which can then be amplified using the transformation in Section 3. Notably, there exists a trivial method to construct 1-succinct randomized encodings, without the use of cryptography - the encoding algorithm simply runs the circuit $C^{(t)}$ on input x and outputs $C^{(t)}(x)$. However, this approach does not guarantee that the efficiency of the encoding algorithm is linear in $|C|$(see discussion in Appendix B). We instead use garbled circuits to make the run-time of encoding efficient.

Let's assume that there exists a garbled circuit scheme that is secure against adversaries running in time 2^{λ^a}, where $a \in (0,1)$ is some constant and the security parameter is λ. We can construct a 1-succinct randomized encoding for circuit $C : \{0,1\}^s \to \{0,1\}^s$ with the following encode algorithm $\mathsf{SRE.Encode}(1^\lambda, C, x, t)$,

- We fix the security parameter to be $\widetilde{\lambda} = (\log \lambda)^\alpha$ where α is a constant such that $\alpha = 2/a$.
- Sample $\mathsf{gsk} \leftarrow \{0,1\}^{\widetilde{\lambda}}$ and compute $\widetilde{C^{(t)}} \leftarrow \mathsf{GC.Garble}_{\mathsf{gsk}}(1^{\widetilde{\lambda}}, C^{(t)})$, and $\forall i \in [s], \widetilde{x_i} \leftarrow \mathsf{GC.Enc}_{\mathsf{gsk}}(x_i, i)$.
- Output $\mathsf{SRE}.\widetilde{x} \leftarrow \left(\widetilde{C^{(t)}}, \{\widetilde{x}_i\}_{i \in [s]}\right)$.

The efficiency of SRE.Encode is $t \cdot |C| \cdot \mathsf{poly}(\log \lambda)$, and efficiency of SRE.Eval is $t \cdot |C| \cdot \mathsf{poly}(\log \lambda)$ in the RAM model.

We now discuss security, let $\mathcal{A}$ be an adversary that runs in time $\mathsf{poly}(\lambda)$ and breaks security of the 1-succinct SRE aboove with non-negligible advantage. We use $\mathcal{A}(1^\lambda)$ to construct an efficient adversary

$\mathcal{B}(1^{\widetilde{\lambda}})$ which runs in time $2^{\widetilde{\lambda}^a}$ and contradicts the security of GC with non-negligible advantage. $\mathcal{B}(1^{\widetilde{\lambda}})$ calls the adversary $\mathcal{A}(1^{\lambda})$ on input $\mathsf{SRE}.\widetilde{x}$ and outputs whatever $\mathcal{A}$ outputs. Since, $2^{\widetilde{\lambda}^a} = 2^{\log^2 \lambda}$, adversary $\mathcal{A}$ runs in polynomial in λ i.e. $2^{O(\log \lambda)}$ and runs within time $2^{\widetilde{\lambda}^a}$. By contradiction, our resulting scheme is secure.

We can amplify this scheme by repeatedly applying the transformation in Section 3. Let the i-th iteration of the transofrmation, be such that,

- Let the primitives LFE, GC be sub-exponentially secure such that security holds against adversaries running in time 2^{λ^a} where $a \in (0, 1)$ is some constant.
- Let SRE be a $1/i$-succinct randomized encoding scheme such that the runtime of encode is $t^{1/i} \cdot |C| \cdot \mathsf{poly}^i(\log \lambda)$, and the runtime to evaluate is $t \cdot |C| \cdot \mathsf{poly}^i(\log \lambda)$ in the RAM model. Additionally, security holds against adversaries running in time $\mathsf{poly}(\lambda)$.

Construction sketch - We construct SRE', a $1/(i+1)$-succinct randomized encoding scheme by running the tranformation in Section 3 where primitives LFE, GC are run on security parameter $\widetilde{\lambda} = (\log \lambda)^{\alpha}$ where $\alpha = 2/a$, and the base SRE is run on security parameter λ.
The runtime of encode is $t^{1/(i+1)} \cdot |C| \cdot \mathsf{poly}^{i+1}(\log \lambda)$ and the runtime to evaluate is $t \cdot |C| \cdot \mathsf{poly}^{i+1}(\log \lambda)$ in the RAM model. Additionally, by similar reductions to the 1-succinct case, the security can be shown against any adversaries running in polynomial time.

Let $\mathsf{poly}(\log \lambda) = (\log \lambda)^k$ for some constant k. After $i = \log \lambda / \log \log \lambda - 1$ iterations of the transformation, we have,

- Efficiency of the resulting encode algorithm is,

$$t^{1/(i+1)} \cdot |C| \cdot (\log \lambda)^{k \cdot i + 1} = t^{\log \log(\lambda)/\log(\lambda)} \cdot |C| \cdot (\log \lambda)^{k \cdot \log \lambda / \log \log \lambda}.$$

  Since t is polynomial, we have that $t = 2^{O(\log \lambda)}$, and hence, $t^{\log \log(\lambda)/\log(\lambda)} \leq 2^{O(\log \log \lambda)} \in \mathsf{poly}(\log \lambda)$. Additionally, $\log \lambda^{k \cdot \log \lambda / \log \log \lambda} = 2^{k \cdot \log \lambda} = \lambda^k$.

  Thus, the time to encode is

$$\in \mathsf{poly}(\log \lambda) \cdot |C| \cdot \lambda^k \in |C| \cdot \mathsf{poly}(\lambda).$$

- The efficiency of evaluate is similarly, $t \cdot |C| \cdot \mathsf{poly}(\lambda)$ in the RAM model.
- The resulting randomized encoding is secure against adversaries running in time $\mathsf{poly}(\lambda)$.

Thus, we can construct a fully-succinct SRE from sub-exponential circular-secure LWE assumption as in Corollary 1.2.

D Time lock puzzle from SREs

In this section, we sketch the construction of time-lock puzzles from semi ε-succinct SRE scheme and worst-case non-parallelizable language. We use the notation and theorem statement from [BGJ+16] almost verbatim.

Assumption D.1. (Worst-Case Non-parallelizable Languages) A language $\mathcal{L}$ that is decidable by a turing machine in time $t(\cdot)$ and space $s(\cdot)$ is non-parallelizable in the worst-case with gap $\varepsilon < 1$ if for every family of non-uniform circuits $\mathcal{B} \in \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ where $\mathsf{depth}(\mathcal{B}_\lambda) \leq t^\epsilon(\lambda)$ and every large enough λ, $\mathcal{B}_\lambda$ fails to decide $\mathcal{L} \cap \{0,1\}^\lambda$.

In the following construction, we use the machine-hiding version of SREs (see Remark 2.3). Accordingly, the simulator does not take the machine as input. (Also, the evaluation algorithm does not take the machine as input.)

Construction D.2 (Construction 3.5 [BGJ+16] (adapted))**.** Let $s(\lambda)$ be a fixed polynomial in λ. Let SRE be a semi $\varepsilon_{\mathsf{sre}}$-succinct randomized encoding scheme. For $\mu \in \{0,1\}^\lambda, t \leq 2^\lambda$, let M^t_μ be the machine that on input $x \in \{0,1\}^\lambda$ outputs μ after t steps (assume that $t \geq \lambda + \omega(1)$ and that $|M^t_\mu|$ can be described by 3λ bits and is padded to use up space $3\lambda + s(\lambda)$).

- Time-lock puzzle generation on message $\mu \in \{0,1\}^\lambda$ - $\widetilde{x} \leftarrow \mathsf{SRE.Encode}(1^\lambda, M^t_\mu, 0^\lambda, t, 1^{3\lambda+s(\lambda)})$.
- Time-lock puzzle evaluation - $\mu \leftarrow \mathsf{SRE.Eval}(\widetilde{x}, t, 1^\lambda)$.

Proposition D.3. *Assuming there exists a worst-case parallelizable language with gap $\varepsilon_{\mathsf{wc}}$ that is decidable in bounded-space i.e. time $t(\lambda)$, space $s(\lambda)$ where $s(\cdot)$ is a fixed polynomial independent of $t(\cdot)$. Then the construction above using $\varepsilon_{\mathsf{sre}}$-succinct randomized encoding is secure for any gap $\varepsilon_{\mathsf{tlp}}$ where $\varepsilon_{\mathsf{sre}} < \varepsilon_{\mathsf{wc}}$ and $\varepsilon_{\mathsf{tlp}} < \varepsilon_{\mathsf{wc}}$. The time-lock puzzle can be constructed in time $t(\lambda)^{\varepsilon_{\mathsf{sre}}} \cdot s(\lambda) \cdot \mathsf{poly}(\lambda)$ and can be evaluated in time $t(\lambda) \cdot s(\lambda) \cdot \mathsf{poly}(\lambda)$ in the RAM model.*

Proof. The proof follows identically to Theorem 3.7 in [BGJ+16]. We repeat the proof almost verbatim and sketch the major difference in our proof which exists as the time to encode SRE.Encode is no-longer independent of t.

Assume towards contradiction that there exists a polynomial size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, and a polynomially bounded function $t(\cdot) \geq \underline{t}(\cdot)$ such that $\mathsf{depth}(\mathcal{A}_\lambda) < t^{\varepsilon_{\mathsf{tlp}}}(\lambda)$ and for some polynomial $p(\cdot)$ and infinitely many $\lambda \in \mathbb{N}$, there exists a pair of solutions $\mu_0, \mu_1 \in \{0,1\}^\lambda$ such that,

$$\Pr\left[b \leftarrow \mathcal{A}_\lambda(Z) : \begin{array}{c} b \leftarrow \{0,1\} \\ Z \leftarrow \mathsf{Puzzle.Gen}(t(\lambda), \mu_b) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p(\lambda)}.$$

Let $\mathcal{L}$ be the worst-case non-parallelizable language that is decidable in time $t(\lambda)$ and space $s(\lambda)$. We construct an adversary $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ that decides $\mathcal{L} \cap \{0,1\}^\lambda$. For any λ as above with corresponding $\mu_0, \mu_1 \in \{0,1\}^\lambda$, let $M^{\mathcal{L},t}_{\mu_0,\mu_1}$ be a machine that on input $x \in \{0,1\}^\lambda$, outputs μ_1 if $x \in \mathcal{L}$ and μ_0 if $x \notin \mathcal{L}$, after exactly $t(\lambda)$ steps. Such a machine exists, takes space at most $3\lambda + s(\lambda)$ and we further assume that it can be described in 3λ bits (which is possible for large enough λ), and has the same description as $M^t_{\mu_b}$. Given input $x \in \{0,1\}^\lambda$, to decide if $x \in \mathcal{L}$, a randomized $\mathcal{B}'_\lambda$ acts as follows:

- Sample $Z \leftarrow \widehat{M}^{\mathcal{L},t}_{\mu_0,\mu_1}(x) \leftarrow \mathsf{SRE.Encode}(1^\lambda, M^{\mathcal{L},t}_{\mu_0,\mu_1}, x, t(\lambda), 1^{3\lambda+s(\lambda)})$.
- Obtain $b \leftarrow \mathcal{A}_\lambda(Z)$ and output b.

Note that $\mathcal{B}'_\lambda$ is of polynomial size and depth $t^{\varepsilon_{\mathsf{sre}}} \cdot s(\lambda) \cdot \mathsf{poly}(\lambda) + \mathsf{depth}(\mathcal{A}_\lambda) \leq t^{\varepsilon_{\mathsf{sre}}} \cdot s(\lambda) \cdot \mathsf{poly}(\lambda) + t^{\varepsilon_{\mathsf{tlp}}}(\lambda) \in o(t^{\varepsilon_{\mathsf{wc}}}(\lambda))$, where the last membership holds since $s(\cdot)$ and $\mathsf{poly}(\cdot)$ are fixed polynomials and thus holds for large enough $t(\lambda)$.

The rest of the proof follows identically where we first show that $\mathcal{B}'_\lambda$ decides the language $x \in \mathcal{L}$ with noticable advantage, and then do the parallel repition argument to construct $\mathcal{B}$ that contradicts the fact that $\mathcal{L}$ is non-parallelizing.

□