Succinct Garbling Schemes from Functional Encryption through a Local Simulation Paradigm

Prabhanjan Ananth
MIT

Alex Lombardi
MIT

August 17, 2018

Abstract

We study a simulation paradigm, referred to as *local simulation*, in garbling schemes. This paradigm captures simulation proof strategies in which the simulator consists of many local simulators that generate different blocks of the garbled circuit. A useful property of such a simulation strategy is that only a few of these local simulators depend on the input, whereas the rest of the local simulators only depend on the circuit.

We formalize this notion by defining locally simulatable garbling schemes. By suitably realizing this notion, we give a new construction of succinct garbling schemes for Turing machines assuming the polynomial hardness of compact functional encryption and standard assumptions (such as either CDH or LWE). Prior constructions of succinct garbling schemes either assumed sub-exponential hardness of compact functional encryption or were designed only for small-space Turing machines.

We also show that a variant of locally simulatable garbling schemes can be used to generically obtain adaptively secure garbling schemes for circuits. All prior constructions of adaptively secure garbling that use somewhere equivocal encryption can be seen as instantiations of our construction.

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 4
- 1.2 Technical Overview . . . 6

2 Preliminaries **12**
- 2.1 Garbling Schemes for Circuits . . . 13
- 2.2 Decomposable Garbling Schemes for Circuits . . . 14
- 2.3 Succinct Garbling Schemes . . . 14
- 2.4 Indistinguishability Obfuscation . . . 15
- 2.5 Somewhere Equivocal Encryption . . . 16

3 Locally Simulatable Garbling Schemes **17**
- 3.1 Semi-Adaptive Local Simulation . . . 18
- 3.2 Strong Local Simulation . . . 19
- 3.3 Locally Simulatable Garbling Schemes . . . 22

4 Statements of Our Results **23**
- 4.1 Locally Simulatable Garbling . . . 23
- 4.2 Succinct Garbling . . . 23
- 4.3 Adaptively Secure Garbling . . . 24

5 Locally Simulatable Garbling from Laconic OT **24**
- 5.1 The LSGS Construction . . . 26
- 5.2 Strong Local Simulation . . . 29
- 5.3 Semi-Adaptive Local Simulation Security . . . 33

6 Succinct Garbling from IO through Locally Simulatable Garbling **33**
- 6.1 Bounded Runtime Case . . . 34
- 6.2 Proof of Security . . . 36
- 6.3 Removing the Bounded Runtime Restriction . . . 42
- 6.4 Succinct Garbling from Functional Encryption . . . 44

7 Adaptively Secure Garbling from Locally Simulatable Garbling **44**

A Selectively Secure Laconic OT from Indistinguishability Obfuscation **50**

1 Introduction

Garbling schemes are ubiquitous to cryptography. Their notable applications include secure computation on the web [GHV10, HLP11], constructions of functional encryption [SS10, GVW12, GKP+12], one-time programs [GKR08], delegation of computation [GGP10, AIK10], and garbled RAMs [GHL+14, GLOS15]. In fact, there are many more applications under the umbrella of randomized encodings, which are implied by garbling schemes. These applications include parallel cryptography [AIK04, AIK06], bootstrapping theorems in functional encryption and indistinguishability obfuscation [ABSV15, App14a], and key-dependent message security [BHHI10, App14b]. More recently, garbling schemes were also crucially used to solve two longstanding open problems in cryptography: achieving two-round passively secure MPC [GS17, BL18, GS18b] and identity-based encryption from weaker assumptions [DG17, BLSV18, DGHM18].

A garbling scheme allows for efficiently encoding a circuit C, represented by $\langle C \rangle$ (also referred to as *garbled circuit*), and separately encoding an input x, represented by $\langle x \rangle$. We require that given $\langle C \rangle$ and $\langle x \rangle$, it is possible to efficiently recover $C(x)$ and moreover, the encodings should not leak anything beyond $(C, C(x))$[1]. This notion was first introduced by Yao [Yao82, Yao86] as a technique to solve two-party secure computation (a full proof of this application was only given much later by Lindell and Pinkas [LP09]). More than three decades later, proposing new constructions of garbling schemes is still an active and fascinating research direction.

While the traditional notion of garbling schemes considers encoding circuits, this notion can be generalized for other models of computation. In particular, we consider garbling *Turing machines*; this notion is often referred to as succinct garbling schemes [BGL+15, CHJV15, KLW15]. The non-triviality in this setting is to encode both the Turing machine M and the input x in time independent of the runtime of M. In more detail, we require that the time to garble a Turing machine M should be polynomial in λ (security parameter) and $|M|$ while the time to encode an input x should be polynomial in λ and $|x|$. For decoding, we require that it should only take time polynomial in λ and t to recover $M(x)$, where t is the runtime of M on x.

Succinct garbling schemes have been used in many applications including time-lock puzzles [BGJ+16], concurrent zero-knowledge [CLP15], indistinguishability obfuscation for Turing machines [BGL+15, CHJV15, KLW15] and delegation for deterministic computations [BGL+15, CHJV15, KLW15]. In terms of constructions, the initial works of [BGL+15, CHJV15] proposed succinct garbling schemes with the caveat that the size of the garbled Turing machine grows with the maximum space taken by the Turing machine during its execution. Subsequently, Koppula et al. [KLW15] showed how to get rid of this caveat and presented a construction of succinct randomized encodings (a notion where M and x are encoded together) assuming indistinguishability obfuscation and one-way functions.

It is worth noting that the approach taken by [BGL+15] differs substantially from the approach taken by [CHJV15, KLW15] to obtain succinct randomized encodings. The construction of [BGL+15] is very simple to describe: they succinctly garble a Turing machine M (running in time at most T) by outputting an obfuscated program that on input $i \leq T$ outputs the ith garbled table of a Yao garbled circuit [Yao82, Yao86, LP09] associated to a circuit C representing M's computation. One might hope that this already yields a fully succinct garbling scheme, but the security proof of [BGL+15] requires hardwiring $O(s)$ bits of information in the obfuscated program when M requires space s, so this does not yield a fully succinct garbling scheme (which [KLW15] does achieve).

While the final result has an undesirable dependence on s, the [BGL+15] approach has the advantage of relying only on obfuscation for circuits of input length $\log(T) = O(\log(\lambda))$ and hence can be proved secure assuming the existence of polynomially secure functional encryption

[1] In this work, we only consider the case of hiding the input x. To hide the circuit C being garbled, we can garble an universal circuit with an encryption of C hardwired inside it and produce an input encoding of x along with the decryption key.

[AJ15, BV15, LZ17, LT17]. The approach of [CHJV15, KLW15] does not share this property, and indeed there is currently no known construction of fully succinct garbling from (poly-secure) FE. In general, there are a few primitives (such as trapdoor permutations and non-interactive key exchange) known to follow from FE [GPS16, GS16, GPSZ17, LZ17] while many others (such as NIZK [SW14, BP15], deniable encryption [SW14], and long output secure function evaluation [HW15]) we know only how to construct from IO (see [LZ17] for a more detailed discussion). One of our main goals is to understand whether constructing succinct garbling schemes requires the full power of IO in this sense.

Rather intriguingly, the progress on succinct randomized encodings followed a similar pattern to progress on the problem of constructing adaptively secure circuit garbling schemes. There is a simple transformation [BHR12] from selectively secure garbling schemes to adaptively secure garbling schemes in which the online complexity (that is, the size of the input encoding) grows with the circuit size. Subsequent to [BHR12], the work of [HJO+16] showed how to achieve adaptive schemes with online complexity that only depends on the width w of the circuit (from one-way functions) or depth d of the circuit (from $2^{-O(d)}$-secure one-way functions). Following [HJO+16], the works of [JW16, JSW17] present additional constructions of adaptive circuit garbling schemes. Finally, a beautiful work of Garg and Srinivasan [GS18a] showed how to achieve adaptive garbling schemes with online complexity $|x| + \mathrm{poly}(\log(|C|, \lambda)$ assuming either the computational Diffie-Hellman (CDH) or learning with errors (LWE) assumption.

We note that the measure of width complexity in the case of circuits is related to the measure of space complexity in the case of Turing machines. Indeed, we can transform a Turing machine M that requires space s on inputs of length n into a circuit of width $O(n+s)$; similarly, a circuit of width w can be simulated by a Turing machine which takes space at most $O(w)$. Moreover, there are actually major similarities between the *security proofs* of [HJO+16] (for their width-dependent adaptive garbling scheme) and [BGL+15] (for their space-dependent succinct garbling scheme). At a high level, both require opening up the [LP09] proof of security for Yao's garbling scheme and make use of the fact that security is argued by a gate-by-gate hybrid argument.

These similarities present the possibility of transporting some of the techniques from the adaptive garbling literature in order to construct new and improved succinct garbling schemes. In particular, we ask: can the ideas from [GS18a] be used to construct succinct garbling?

1.1 Our Contributions

We give a new construction of succinct garbling schemes using the ideas of [GS18a]. Unlike the work of [KLW15][2] based on sub-exponentially secure compact functional encryption, our construction is based on polynomially secure compact functional encryption and polynomially secure CDH/LWE. As an added advantage, our construction is conceptually simpler. Instead of using IO/FE to compress a Yao garbled circuit as in [BGL+15], we compress an appropriately modified [GS18a] garbled circuit.

To prove security, we identify a property, termed as *local simulation*, of selectively secure garbling schemes for circuits that when combined with other tools yields succinct garbling schemes. To describe this property, we first recall the security experiment of garbling schemes. To prove that a given garbling scheme is secure, one needs to exhibit a simulator with the following property: given just the circuit C and the output $C(x)$, it can output a simulated garbled circuit and input encoding that is indistinguishable from an honest garbled circuit and input encoding. Typically this indistinguishability is shown by a sequence of hybrids: in every step, a hybrid simulator is defined to take an input C and x produces the simulated garbling and input encoding. The first hybrid defines the honest garbling of C and the honest encoding of x, while the final hybrid defines the simulated distribution. At a bare minimum, our notion

[2] We note that [KLW15] construct succinct randomized encodings scheme and not garbling schemes. However, their construction can be adapted to get succinct garbling schemes

of local simulation captures a class of such hybrid arguments wherein the simulation of garbled circuit is divided into blocks and in every hybrid, only a *small* L_{sim}-sized subset of blocks are simulated using C and x while the rest are simulated only using C. We observe that this seemingly artificial property is already satisfied by current known schemes [Yao86, GS18a].

To make the local simulation notion useful for applications, we need to consider strengthenings of this notion. We formalize the above informal description of local simulation and call this **weak** local simulation; correspondingly the garbling scheme will be called a weak locally simulatable garbling scheme (weak LSGS). We consider two strengthenings: (i) **strong** locally simulatable garbling schemes (strong LSGS) and (ii) **semi-adaptive** locally simulatable garbling schemes (semi-adaptive LSGS). Both the notions of semi-adaptive LSGS and strong LSGS imply weak LSGS and will be parameterized by $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$, where L_{inp} refers to the online complexity of the garbling scheme.

We now state our results on succinct garbling.

Succinct Garbling. We prove the following theorem.

Theorem 1. *(Main Theorem) Assuming single-key compact*[3] *public-key functional encryption for circuits*[4] *and X, where $X \in \{$Computational Diffie-Hellman, Factoring, Learning with Errors$\}$, there exists a succinct garbling scheme for Turing machines.*

Previous constructions of succinct garbling schemes were based on indistinguishability obfuscation[5] (implied by *sub-exponentially* secure compact functional encryption) and one-way functions [KLW15]. This is the first work to show the feasibility of succinct garbling schemes from falsifiable assumptions. Moreover, [KLW15] is significantly more involved whereas our construction is conceptually simpler. We note that several works subsequent to [KLW15] use their construction to achieve various primitives including garbled RAM [CH16, CCC+16, CCHR16, ACC+16], constrained PRFs for Turing machines [DKW16], indistinguishability obfuscation for Turing machines with constant overhead [AJS17a], patchable indistinguishability obfuscation [AJS17b, GP17] and so on. We hope that our simpler construction will correspondingly yield simpler presentation of these applications as well.

One new consequence of the above theorem is that we obtain collusion-resistant functional encryption for Turing machines from collusion-resistant functional encryption for circuits and standard assumptions; this follows from [AS16].

We prove Theorem 1 in two steps. First, we prove the following proposition.

Proposition 1 (Informal)**.** *Assuming strong $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-LSGS and compact functional encryption for circuits, there exists a succinct garbling scheme in which the complexity of garbling a Turing machine M is $\mathrm{poly}(\lambda, |M|, L_{\mathsf{sim}})$ and the complexity of encoding x is $L_{\mathsf{inp}}(\lambda, |x|, m)$, where m is the output length of M.*

Once we prove the above proposition, we show how to instantiate strong LSGS from laconic oblivious transfer[6] to obtain our result.

[3] From prior works [BV15, AJS15], we can replace compact public-key FE with collusion-resistant FE in the theorem statement.

[4] A public-key functional encryption scheme is a public-key encryption scheme with the additional key generation procedure that takes as input circuit C and produces a functional key for C that can be used to decrypt an encryption of x to obtain $C(x)$. A **compact** functional encryption is a functional encryption scheme where the complexity to encrypt a message x is a fixed polynomial in $(\lambda, |x|)$ and in particular, the encryption complexity grows only with $\log(|C|)$. A functional encryption scheme is a **single-key scheme** if it satisfies $\{\mathsf{PK}, \mathsf{Enc}(\mathsf{PK}, x_0), sk_C\} \cong_c \{\mathsf{PK}, \mathsf{Enc}(\mathsf{PK}, x_1), sk_C\}$ for an adversarially chosen C and x and specifically, the adversary is only issued a signle key in the security experiment.

[5] See Section 2.4 for a formal definition.

[6] We actually use the existentially equivalent notion of *appendable laconic OT*, which we define later.

Proposition 2 (Informal). *Assuming laconic oblivious transfer, there exists a strong* $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$*-LSGS with* $L_{\mathsf{sim}} = \mathrm{poly}(\lambda)$.

Since laconic oblivious transfer can be instantiated from CDH, factoring, LWE and other assumptions [CDG+17, DG17, BLSV18, DGHM18], this proves Theorem 1. In addition, we note in Appendix A that laconic OT[7] can be constructed from IO and one-way functions; combined with the above propositions, this says that our succinct garbling scheme can also be instantiated from IO and OWFs alone (giving an alternative construction to [KLW15]).

We note that the garbling scheme of Yao [Yao86] also yields a strong $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-LSGS with L_{sim} proportional to the width of the circuit being garbled. Combining this with Proposition 1, we get a succinct garbling scheme for small space Turing machines; this is essentially the same scheme as that of [BGL+15].

Adaptive circuit garbling. Next, we show how to construct adaptive circuit garbling schemes using our notion of (semi-adaptive) LSGS. First, we recall the definition of adaptive circuit garbling schemes. In the adaptive security experiment, an adversary can submit the circuit C and the input x in any order; specifically, it can choose the input as a function of the garbled circuit or vice versa. We show,

Theorem 2 (Informal). *Assuming semi-adaptive* $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$*-LSGS and one-way functions, there exists an adaptively secure circuit garbling scheme with online complexity* $L_{\mathsf{inp}} + \mathrm{poly}(\lambda, L_{\mathsf{sim}})$.

This theorem can be seen as an abstraction of what the somewhere equivocal encryption-based technique of [HJO+16] can accomplish. For example, the semi-adaptive LSGS can be instantiated from laconic oblivious transfer, recovering the result of [GS18a]. The theorem below follows from a previous work [GS18a].

Theorem 3 ([GS18a]). *Assuming laconic oblivious transfer, there exists a semi-adaptive* $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$*-LSGS scheme with online complexity* $L_{\mathsf{inp}}(\lambda, n, m) = n + m + \mathrm{poly}(\lambda)$ *and* $L_{\mathsf{sim}} = \mathrm{poly}(\lambda)$*, where* n *and* m *denote the input and output lengths for the circuit.*

We note that Yao's garbling scheme is also a semi-adaptive $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-LSGS with L_{sim} being proportional to the width of the circuit and thus, combining the above two theorems we get an adaptively secure circuit garbling scheme with the online complexity proportional to the width of the circuit. This construction is essentially the same as the width-based construction of [HJO+16], with a more modular security proof.

We summarise the results in Figure 1.

1.2 Technical Overview

We first recall the garbling scheme of Yao [Yao86] and describe an overview of its security proof. Yao's scheme will serve as a starting point to understanding the definition of locally simulatable garbling schemes.

Yao's Garbling Scheme [Yao86]. Consider a boolean circuit $C : \{0,1\}^\ell \to \{0,1\}$ comprising only of NAND gates. For ease of presentation, we assume that C is layered such that all gates that are at the same distance from the output gate belong to the same layer. Moreover, every intermediate wire in the circuit connects two gates in adjacent layers.

The first step in the garbling of a circuit C is to generate two wire keys K_w^0 and K_w^1 for every wire w in the circuit. Next, associate with every gate G a garbled table consisting of four entries $(\mathsf{CT}_{00}, \mathsf{CT}_{01}, \mathsf{CT}_{10}, \mathsf{CT}_{11})$. For $b_0, b_1 \in \{0,1\}$, $\mathsf{CT}_{b_0 b_1}$ is an encryption of $K_{w_c}^{\mathrm{NAND}(b_0,b_1)}$ under the

[7] We can only achieve laconic OT satisfying selective security, which suffices for Proposition 2.

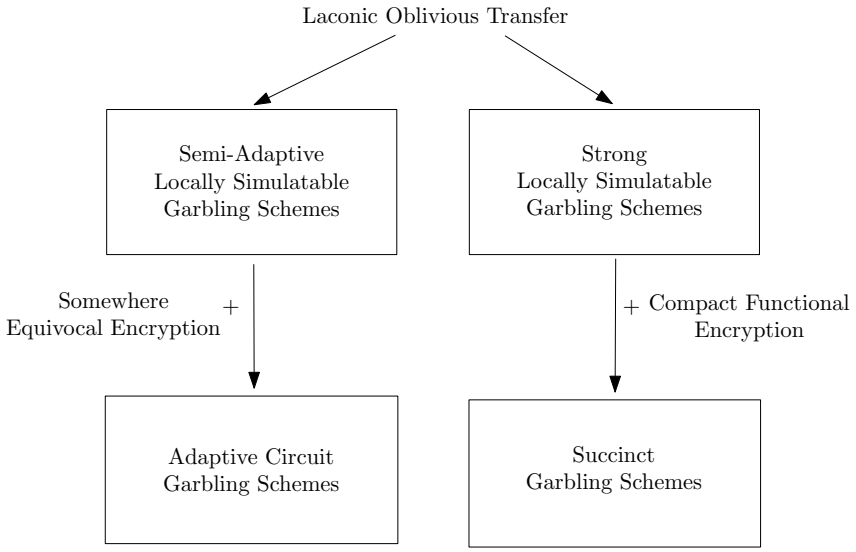

Figure 1: Summary of results.

two keys[8] $K_{w_a}^{b_0}$ and $K_{w_b}^{b_1}$. Wires w_a and w_b are input wires of G and w_c is the output wire of G. Finally, permute the garbled table $(\mathsf{CT}_{00}, \mathsf{CT}_{01}, \mathsf{CT}_{10}, \mathsf{CT}_{11})$. The garbling of C consists of permuted garbled tables associated with every gate in the circuit. The input encoding of x consists of keys $K_{w_i}^{x_i}$, where w_i is the i^{th} input wire of C and x_i is the i^{th} bit of x. Also part of the input encoding is a translation table that maps 0 to $K_{w_{\text{out}}}^0$ and 1 to $K_{w_{\text{out}}}^1$, where w_{out} is the output wire of C.

Selective Security of Yao's Garbling Scheme: To show that Yao's garbling scheme is secure we need to demonstrate a probabilistic polynomial time simulator Sim that given $(C, C(x))$ (and in particular, x is not given) outputs a simulated garbling of C and a simulated input encoding. Sim is defined as follows: every wire w is only associated with a single key K_w. Associated with every gate G is a garbled table consisting of $(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{CT}_3, \mathsf{CT}_4)$, where: for a randomly picked index $i^* \in [4]$, (i) CT_{i^*} is an encryption of K_{w_c} under keys K_{w_a} and K_{w_b}, (ii) for $i \neq i^*$, CT_i is an encryption of 0 under two randomly chosen secret keys (and in particular these two keys are not used anywhere). The simulated garbling of C consists of the simulated garbled tables associated with every gate in the circuit. The input encoding consists of the keys $\{K_w\}$ for every input wire w. In addition, it consists of the translation table that maps $C(x)$ to $K_{w_{\text{out}}}$ and maps $\overline{C(x)}$ to $K'_{w_{\text{out}}}$, where $K'_{w_{\text{out}}}$ is generated afresh.

The indistinguishability of the output of Sim from an honestly generated garbled circuit and input encoding can be argued by a hybrid argument explicitly described in [HJO+16]. This hybrid argument will be associated with a sequence of intermediate simulators $\mathsf{Sim}_1, \ldots, \mathsf{Sim}_q$. Except Sim_q, all the other simulators take as input circuit and the input; (C, x). The final simulator Sim_q takes as input $(C, C(x))$. Sim_1 computes the garbling of C and the input encoding of x as dictated by the scheme. The final intermediate simulator Sim_q is identical to Sim.

The i^{th} intermediate simulator Sim_i works as follows: for every wire w such that w is the output wire of a j^{th} layer for $j \geq i$, sample two keys K_w^0 and K_w^1. For any other wire w, sample a single wire key K_w. The simulator consists of two components:

- **Input-Dependent Simulation.** This component takes as input (C, x) and simulates all

[8] There are many ways of realizing an encryption scheme under two different secret keys. One convenient method is to secret share the message and encrypt the two shares using the two keys.

the garbled gates in the i^{th} layer of C. For every gate G (with input wires w_a, w_b and output wire w_c) in the i^{th} layer, generate a garbled table $(\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{CT}_3, \mathsf{CT}_4)$, where for a randomly picked index $i^* \in [4]$, (i) CT_{i^*} is an encryption of $K_{w_c}^{\mathsf{val}(w_c)}$ under keys K_{w_a} and K_{w_b}, (ii) for $i \neq i^*$, CT_i is an encryption of 0 under two randomly chosen secret keys (and in particular these two keys are not used anywhere). Here, $\mathsf{val}(w_c)$ denotes the value assigned to the wire w_c during the evaluation of C on x.

- **Input-Independent Simulation.** This component only takes as input C and simulates the garbled gates in all the layers except the i^{th} layer. There are two cases:
  - for a gate G in the j^{th} layer, for $j < i$, the simulation of the garbled gate for G is performed according to Sim.
  - for a gate G in the j^{th} layer, for $j > i$, the garbled gate for G is generated according to the scheme.

Once the computational indistinguishability of Sim_i and Sim_{i+1} is shown for every i, the security of the scheme follows.

Complexity of Input-Dependent Simulation. Observe that the output length of the input-dependent simulation component of every simulator Sim_i is only proportional to the width of the circuit (in other words, the maximum length of any layer in C). This observation has been crucially exploited in two lines of work:

- The work of [HJO+16] introduced the powerful tool of somewhere equivocal encryption (SEE) and showed how to combine it with the garbling scheme of Yao to obtain adaptive garbling schemes with online complexity that grows with the width of the circuit. Informally, somewhere equivocal encryption is used in conjunction with the above proof of security for Yao's scheme: in each step of a hybrid argument, the input-dependent simulated gates are equivocated in the online phase of the adaptive security game. Since the number of input-dependent simulated gates is bounded by the width w of the circuit, the online complexity of this garbling scheme is proportional to w. Alternative proof strategies for Yao's garbling scheme can be used instead of our sketch above to obtain, for example, the depth-based result of [HJO+16].
- The work of [BGL+15] showed how to combine indistinguishability obfuscation for circuits and the garbling scheme of Yao to obtain a succinct garbling scheme for small-space Turing machines. To garble a Turing machine M that has worst-case runtime T, they construct an obfuscation of a circuit that takes as input an index i and outputs the garbled table corresponding to the ith gate of C[9]. Security is argued by sequentially invoking the simulators $(\mathsf{Sim}_1, \ldots, \mathsf{Sim}_q)$ of Yao's garbling scheme. Hardwiring the entire simulator's output in the obfuscated circuit would ruin the encoding complexity of the succinct garbling scheme. However, it turns out that security can be argued when only the *input-dependent* simulation component is hardwired. This is exactly the reason why the encoding complexity of this succinct garbling scheme grows with the maximum space complexity of the Turing machines.

Locally Simulatable Garbling Schemes. We introduce the notion of a locally simulatable garbling scheme as an abstraction that connects the above proofs of adaptive security and succinctness for garbling schemes. We give a brief overview of the security property associated

[9]Their actual scheme instead outputs an entire layer of garbled tables at once, but this variant has the same efficiency and security proof.

with a locally simulatable garbling scheme. The security property is parameterized by an integer L_{sim} and a sequence of simulators $(\mathsf{Sim}_1, \ldots, \mathsf{Sim}_q)$ for some polynomial q. Every simulator Sim_i consists of an input-dependent component and an input-independent component.

- The input-dependent component of Sim_i takes as input circuit C and input x to be simulated. We require that this component of Sim_i is of size at most $L_{\mathsf{sim}} \cdot \mathrm{poly}(\lambda)$ for some fixed polynomial poly.
- The input-independent component of Sim_i takes as input only the circuit C.

We require that the output distribution of Sim_1 is computationally indistinguishable from an honest generated garbling of C and honestly generated encoding of x. The output distributions of Sim_i and Sim_{i+1} are required to be computationally indistinguishable. Finally, we require that the final simulator Sim_q does not have any input-dependent component and in particular, Sim_q can simulate the garbled circuit and input encoding on input $(C, C(x))$. We refer to this security property as *weak* local simulation security.

Note that Yao's garbling scheme, using the security proof given in our outline, is a particular instantiation of a locally simulatable garbling scheme with L_{sim} set to be the width of the circuit being garbled. The depth-based analysis of Yao's garbling scheme given in [HJO+16] can also be seen as an instantiation of weak local simulation, albeit with $q = 2^{O(d)}$ hybrids.

While the above security property captures the essence of local simulation, it does not suffice for either the application of adaptively secure garbling schemes or the application of succinct garbling schemes. To get around this, we strengthen the security definition in two ways, resulting in notions of **semi-adaptive** locally simulatable garbling schemes and **strong** locally simulatable garbling schemes.

Succinct Garbling from Strong LSGS. We define the notion of a strong locally simulatable garbling scheme and use it as an intermediate tool to construct a succinct garbling scheme. To motivate our definition, we will consider a candidate succinct garbling scheme (and proof strategy) from IO and a weak LSGS, and see what additional properties are required from the LSGS.

Generalizing the approach of [BGL+15], our candidate succinct garbling scheme is as follows: garbling a Turing machine M with a runtime bound T consists of computing an indistinguishability obfuscation of a circuit $H_{M,T,\mathsf{MSK}}$ with hardwired values M, T and a master secret key MSK. This circuit takes as input an index $i \leq T$, constructs the i^{th} gate of C, where C is the circuit representing T steps of M's computation on inputs of length n, and then outputs a garbling of this gate computed with respect to MSK. Encoding x consists of computing the input encoding of x with respect to the LSGS. Decoding proceeds by evaluating the obfuscated circuit on all indices ranging from 1 to T to obtain the different gate encodings. These encodings are then decoded to obtain the result.

We are already implicitly assuming some properties of the underlying LSGS in order for the above construction to make any sense at all. Specifically,

- Our candidate implicitly assumes that a garbling of C is computed in a gate-by-gate fashion. To enable this, we introduce the notion of a local encoding of an LSGS, which guaratees that a garbling of C consists of components that are each computed in time independent of $|C|$; in particular, it must be computable from a small amount of information about C. In fact, we further require that this information about C is efficiently computable from M. In the case of Yao, this amounts to saying that an individual gate of C can be computed very efficiently from M.
- A priori, the master secret key MSK could be as large as $|C| = \mathrm{poly}(T)$. Strictly speaking, this means that the above candidate is not succinct. To overcome this, we think of $\mathsf{MSK} = (sk_1, \ldots, sk_N)$ and define a *local key generation* procedure that takes as input an index j

and only generates the local secret key sk_j. Then, the program H in our scheme takes as input an index i, determines the keys sk_j that are necessary to encode the i^{th} component of C, and then computes the i^{th} garbled component.

- To identify the subset of keys to be locally generated for the i^{th} component, we define a *key list generation* procedure that takes as input i and outputs a list L_i. This allows us to compress the potentially large MSK using a pseudorandom function key.
- The size of the input encoding of the succinct garbling candidate is exactly the same as the online complexity of the underlying strong LSGS scheme. Thus, in order for our scheme to be succinct, the online complexity of the underling LSGS scheme will have to be independent of T.

By carefully defining the above notions, we can guarantee that the program H is sufficiently small (polynomial in λ) so that the candidate garbling scheme is succinct. What remains is to prove security in a way such that programs H' that are obfuscated in the security proof are also small. This is the most subtle step; in particular, this is the step where [BGL+15] is limited to achieving succinctness that depends on the space of the Turing machine.

To prove the security of the above scheme, a naive approach would be to hardwire the entire simulated garbled circuit inside the obfuscation of $H_{M,T,\mathsf{MSK}}$; however, this would violate succinctness. Instead, we want to leverage *local simulation* in the following way: in each of a sequence of hybrid circuits $(H_1, \ldots, H_q)$, only hardwire the *input-dependent* components of Sim_i, and instead include the code of the input-independent components of Sim_i (which naively contains all of MSK) inside H_i. We would then hope to argue using some combination of IO security and LSGS security that adjacent hybrid programs in this sequence are indistinguishable.

If the size of the input-dependent portion is small, meaning polynomial in λ, then we can hope to achieve succinctness using this proof strategy. This approach again implicitly assumes properties of the LSGS; namely, that the input-independent local simulators each require only a small portion of the master secret key (just as in the honest garbling case). This is required so that the hybrid circuit H_i is still small.

Unfortunately, the security argument above is flawed. The problem is that information about the master secret key MSK is contained within the obfuscated program $\tilde{H}$, so it is unclear how to argue that the input-dependent components of Sim_i and Sim_{i+1} (i.e. the components that are hardwired) are indistinguishable. Indeed, if the above strategy is not carefully implemented (e.g. if the program H_i actually reveals the entire MSK), they will be distinguishable.

To circumvent these issues, we require that the input-dependent portion of the garbled circuit output by Sim_i is indistinguishable from the corresponding input-dependent portion of the garbled circuit output by Sim_{i+1} *even in the presence of* $\{sk_j\}_{j \in S}$*, where* S *consists of all indices accessed by the input-independent portion of the garbled circuit.* In fact, we define a stronger property that allows the adversary to choose the keys $\{sk_j\}_{j \in S}$.

In order to complete the hybrid argument, our proof strategy then works in two steps: first switch the input-dependent components of the simulated circuit from Sim_i to Sim_{i+1} (using the above strong LSGS security), and then switch the input-independent components from Sim_i to Sim_{i+1}. Since we are actually including the code of these input-independent simulators within the obfuscated circuit, we must require that the input-independent components of Sim_i and Sim_{i+1} are functionally equivalent to invoke IO security.

To summarize, a strong LSGS must satisfy two main properties in order for the security of our succinct garbling scheme to be proved:

- The input-dependent components of Sim_i and Sim_{i+1} must be indistinguishable *even given* all of the local secret keys necessary to compute the input-independent components of Sim_i.
- The *algorithms* computing the input-independent components of Sim_i and Sim_{i+1} must be functionally equivalent.

Indeed, the security proof of [BGL+15] can be retroactively seen as invoking the above properties of Yao's garbling scheme. For completeness, we sketch a proof (see Theorem 6) that Yao's garbling scheme satisfies this definition with L_{sim} proportional to the width of the circuit.

Constructing Strong LSGS from Laconic OT. In order to complete the proof of Theorem 1, we show that the garbling scheme of [GS18a] can be adapted to satisfy our strong LSGS notion with $L_{\mathsf{sim}} = \mathrm{poly}(\lambda)$. We begin by giving a high level description of the [GS18a] garbling scheme:

- An encoding of an input x consists of (1) a somewhere equivocal encryption secret key, (2) a one-time pad encryption $r \oplus x$ of x, (3) a hash value $h_0 = H(r \oplus x || 0^{|C|-n})$ of an initial memory state for the computation, (4) a signature on h_0, and (5) the one-time pads corresponding to each output gate. The hash function H is associated to a laconic OT scheme (we omit a discussion of laconic OT from this overview).
- An encoding of a circuit C consists of $s = |C|$ "garbled programs" maintaining the following invariant: after executing i such programs, the evaluator will have obtained a one-time pad encryption of the first $n + i$ gates of C evaluated on the input x along with a hash of this one-time padded state and a signature on this hash value. The garbled programs are then jointly encrypted using a somewhere equivocal encryption scheme.
- Simulation security is argued by a sequence of hybrid simulators; in a hybrid simulator, each garbled program is either computed via an input-independent simulator or an input-dependent simulator, and moreover only $\mathrm{poly}(\lambda)$ garbled programs require input-dependent simulation. To prove adaptive security, the input-dependent simulated gates are equivocated as part of the input encoding.

We interpret the scheme of [GS18a] – after removing the somewhere equivocal encryption layer – as a LSGS by thinking of each garbled program above as one component of the LSGS. Indeed, we show that each garbled program in the [GS18a] scheme only requires a small amount of the garbling secret key and that the input-dependent components of Sim_i and Sim_{i+1} are indistinguishable even in the presence of adversarially chosen secret keys used for the other components. In fact, all but one of the properties of a strong LSGS as defined earlier can be demonstrated to hold for the [GS18a] scheme without modification.

The only problem with using the [GS18a] scheme as a strong LSGS is that computation of the initial hash value $H(r \oplus x || 0^{|C|-n})$ requires $O(|C|)$ time. Naively, this means that computing even the input encoding would take $O(|C|)$ time, but [GS18a] note that if H is computed via a Merkle tree, the computation of $H(0^{|C|-n})$ can be delegated to the garbled circuit and only $H(r \oplus x)$ need be computed during the input encoding. However, computing $H(0^{|C|-n})$ *cannot* be done locally (i.e. distributed in pieces to local components of the garbled circuit), which violates the local encoding property of a strong LSGS.

To circumvent this problem, we modify the [GS18a] scheme so that the initial hash value $h_0 = H(r \oplus x)$ is a hash of only an n-bit string, and we redesign the garbled programs so that each step updates the one-time padded computation state by *appending* the next value. Instantiating this corresponds to a new notion of *appendable laconic OT*, which we define and construct generically from laconic OT. The local simulators for our new scheme remain essentially the same, and our previous security proof carries over to this modified version. We note that the same modification could be made to the [GS18a] adaptive garbled circuit construction, with the advantage that the more complicated notion of *updatable laconic OT* is not required, and hence the [GS18a] scheme can be somewhat simplified.

Combining this construction of strong LSGS from laconic OT with our construction of succinct garbling from FE and strong LSGS, we obtain Theorem 1.

Adaptive Garbling from Semi-Adaptive LSGS. In order to construct adaptive garbling schemes, it turns out that the notion of strong LSGS does not capture the essence of the adaptive security proof. We define a notion of semi-adaptive LSGS and show that a semi-adaptive LSGS can be used to construct adaptive circuit garbling schemes. We define the notion below.

The semi-adaptive security property is associated with a sequence of simulators $(\mathsf{Sim}_1, \ldots, \mathsf{Sim}_q)$ for some polynomial $q = q(\lambda)$. As before, the output of Sim_i consists of an input-dependent component and an input-independent component. However, in this security definition, we allow the adversary to choose the input *after* he receives the input-*independent* component of the garbled circuit from the challenger. In particular, the adversary can choose the instance as a function of the input-independent component.

Our transformation from semi-adaptive LSGS to adaptive garbling is inspired by the work of [HJO+16]. In particular, our transformation abstracts out the usage of somewhere equivocal encryption in this and other prior works. In this transformation, the size of the input-dependent component (i.e. L_{sim}) determines the size of the secret key in a somewhere equivocal encryption scheme, and hence plays a role in determining the online complexity of the adaptive garbling scheme. The online complexity of the resulting adaptively secure garbling scheme is the sum of $\mathrm{poly}(\lambda, L_{\mathsf{sim}})$ and the online complexity L_{inp} of the semi-adaptive LSGS. This can be used to recover the result of [GS18a] (as well as that of [HJO+16]).

2 Preliminaries

We denote the security parameter by λ. We assume the familiarity of the reader with standard cryptographic notions. The output x of a probabilistic algorithm A on input y will be denoted by $x \leftarrow A(y)$. Consider two PPT distributions $\mathcal{D}_0$ and $\mathcal{D}_1$. If $\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally indistinguishable then we denote it by $\mathcal{D}_0 \approx_c \mathcal{D}_1$.

We consider boolean circuits (over arbitrary boolean basis) in this work. By the size of a circuit, we mean the number of gates in the circuit.

Turing Machines. A single-tape Turing machine is described by a 7-tuple $M = (Q, \Sigma, \delta, q_0, q_{acc}, q_{rej})$, where Q is the set of states, Σ is the set of symbols on the tape (including input symbols), $\delta : Q \times \Sigma \to Q \times \Sigma \times \{+1, -1\}$ is the transition function, q_0 is the start state, q_{acc} is the accept state and finally, q_{rej} is the reject state. We assume that the tape of the Turing machine is initialized with the input and the head position points to the leftmost end of the tape.

We define $\mathsf{CktTransform}$ to take as input Turing machine M running in time at most T on inputs of length at most n and outputs an equivalent circuit C. That is $C(x) = M(x)$, where x is of length n and $M(x)$ takes time at most T.

We define the algorithm $\mathsf{FindGate}$ that takes as input (M, n, T, i) and outputs a list of indices such that j is in the list if and only if j^{th} wire is incident to the i^{th} gate of C, where $C = \mathsf{CktTransform}(M, n, T)$. This algorithm proceeds as follows: first, the Turing machine is made oblivious. In particular, the tape head moves as follows: it starts with the leftmost end of the tape, moves to the right end of the tape just before the end symbol then moves to the leftmost end of the tape and so on. Note that such a "strongly oblivious" TM can be achieved with quadratic overhead.

Given a strongly oblivious TM, the position of the tape head after j steps can be computed (as a function of j) by a circuit of size $\mathrm{poly}(\log(T))$. Finally, observe that a single movement of tape head can be represented by a constant-sized circuit.

Note that the running time of $\mathsf{FindGate}(M, n, T, i)$ is polynomial in $(|M|, n, \log(T))$.

2.1 Garbling Schemes for Circuits

We recall the definition of garbling schemes [Yao86, BHR12]. A garbling scheme $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ for boolean circuits is defined by the following algorithms.

- $\mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$: On input security parameter λ, size $s(\lambda)$ of the circuit to be garbled, input length n, output length m, output the master secret key gsk.
- $\mathsf{Gb}(\mathsf{gsk}, C)$: On input garbling key gsk, a boolean circuit C of size at most $s(\lambda)$, input length n and output length m, output the garbled circuit $\langle C \rangle$.
- $\mathsf{IEnc}(\mathsf{gsk}, x)$: On input garbling key gsk, input x, output the input encoding $\langle x \rangle$.
- $\mathsf{Eval}(\langle C \rangle, \langle x \rangle)$: On input garbled circuit $\langle C \rangle$ and input encoding $\langle x \rangle$, output $C(x)$.

We define the properties below.

Correctness. For every $s, n, m > 0$, for every s-sized circuit C of input length n and output length m, input $x \in \{0,1\}^n$, we have:

$$\Pr\left[\mathsf{Eval}\left(\langle C \rangle, \langle x \rangle\right) = C(x) \;:\; \begin{array}{c} \mathsf{gsk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m); \\ \langle C \rangle \leftarrow \mathsf{Gb}(\mathsf{gsk}, C); \\ \langle x \rangle \leftarrow \mathsf{IEnc}(\mathsf{gsk}, x) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

for some negligible function negl.

Efficiency. We require the following efficiency conditions to hold.

- $\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval}$ are computable in time polynomial in its inputs.
- **Online Complexity:** Π has **online (communication) complexity** L if $|\langle x \rangle| \leq L$ for all inputs x. Π has **online computational complexity** L if $\Pi.\mathsf{IEnc}(\mathsf{MSK}, x)$ runs in time L.

Selective Security. Intuitively, we require that the garbling of a circuit C and the encoding of an input x does not reveal information about x beyond $C(x)$. We capture this in the following definition.

Definition 1 (Selective Security)**.** *We say that a garbling scheme* $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ *satisfies (selective) simulation security if there exists a PPT simulator* Sim *such that for every large security parameter* λ*, for every* $s, n, m > 0$*, every circuit* C *of size at most* s *with input length* n *and output length* m*, every* $x \in \{0,1\}^n$*, the following holds:*

$$\left\{\mathsf{Sim}(1^\lambda, C, C(x))\right\} \approx_c \left\{\mathsf{Gb}(\mathsf{gsk}, C), \mathsf{IEnc}(\mathsf{gsk}, x)\right\},$$

where $\mathsf{gsk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, 1^m)$.

Adaptive Security. In the selective security definition, we required that the adversary declare the circuit C and input x of his choice in the beginning of the experiment. We can consider a stronger notion where the adversary can submit x after receiving the simulated garbled circuit or vice versa. Such a notion is termed as adaptive security and we define this formally below.

We first define the following experiment parameterized by PPT adversary $\mathcal{A}$ (with boolean output), PPT stateful simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ and PPT challenger Ch.

$\mathsf{Expt}_{\mathcal{A},\mathsf{Sim},\mathsf{Ch}}(1^\lambda, b)$: The challenger Ch generates $\mathsf{gsk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$, where $s(\lambda)$ is an upper bound on the size of the circuit being garbled. Adversary $\mathcal{A}$ makes the following two queries in any order:

- **Circuit Query**: $\mathcal{A}$ submits circuit C.
  - If $b = 0$, generate $\langle C \rangle \leftarrow \mathsf{Sim}_1(1^\lambda, C)$. If the adversary makes the circuit query after making the input query x then the simulator also additionally receives as input $C(x)$.
  - If $b = 1$, generate $\langle C \rangle \leftarrow \mathsf{Gb}(\mathsf{gsk}, C)$.

  Send $\langle C \rangle$ to $\mathcal{A}$.

- **Input Query**: $\mathcal{A}$ submits input x.
  - If $b = 0$, generate $\langle x \rangle \leftarrow \mathsf{Sim}_2(1^\lambda, C(x))$. If the adversary makes the input query before making the circuit query, then Sim_2 does not receive $C(x)$ as input.
  - If $b = 1$, generate $\langle x \rangle \leftarrow \mathsf{IEnc}(\mathsf{gsk}, x)$.

  Send $\langle x \rangle$ to $\mathcal{A}$.

The output of the experiment is the output of $\mathcal{A}$.

We define adaptive security below.

Definition 2 (Adaptive Security)**.** *Consider a garbling scheme* Π *for circuits.* Π *is said to be* **adaptively secure** *if for a sufficiently large* $\lambda \in \mathbb{N}$*, for any PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* Sim *such that:*

$$\left| \Pr\left[0 \leftarrow \mathsf{Expt}_{\mathcal{A},\mathsf{Sim},\mathsf{Ch}}(1^\lambda, 0)\right] - \Pr\left[0 \leftarrow \mathsf{Expt}_{\mathcal{A},\mathsf{Sim},\mathsf{Ch}}(1^\lambda, 1)\right] \right| \leq \mathsf{negl}(\lambda),$$

for some negligible function negl*.*

2.2 Decomposable Garbling Schemes for Circuits

A garbling scheme $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ is said to be **decomposable** if it has the following specific form:

- $\Pi.\mathsf{Gen}(1^\lambda, 1^n)$ independently samples $2n$ labels $\overline{\mathsf{lab}} = (\mathsf{lab}_{i,b})_{i\in[n],b\in\{0,1\}}$, where n is the input length of the circuit C to be garbled, and outputs $\mathsf{MSK} = \overline{\mathsf{lab}}$.
- $\Pi.\mathsf{IEnc}(\mathsf{MSK}, x)$ outputs $(\mathsf{lab}_{i,x_i})_{i\in[n]}$.
- $\Pi.\mathsf{Sim}(1^{|C|}, \overrightarrow{\mathsf{lab}}, C(x))$ is rewritten as an algorithm that takes as additional input n labels $\overrightarrow{\mathsf{lab}}$ and outputs a simulated circuit.

In such schemes, we will use the notation $\mathsf{lab}[x]$ to denote the tuple $(\mathsf{lab}_{i,x_i})_{i\in n}$.

2.3 Succinct Garbling Schemes

We define the notion of succinct garbling schemes. A succinct garbling scheme SuccGC for a class of Turing machines $\mathcal{M}$ consists of the following algorithms.

- $\mathsf{Gen}(1^\lambda, 1^n, 1^m)$: On input security parameter, input length n, output length m, output the master secret key MSK.
- $\mathsf{TMEncode}(\mathsf{MSK}, M)$: On input master secret key MSK, Turing machine M, output a TM encoding $\langle M \rangle$.
- $\mathsf{InpEncode}(\mathsf{MSK}, x)$: On input master secret key MSK, input x, output an input encoding $\langle x \rangle$.
- $\mathsf{Eval}(\langle M \rangle, \langle x \rangle)$: On input TM encoding $\langle M \rangle$, input encoding $\langle x \rangle$, output a value y.

Correctness. For every $n, m > 0$, for every input $x \in \{0,1\}^n$, Turing machine M running in time at most 2^λ on x and $M(x) \in \{0,1\}^m$, we have:

$$\mathsf{Pr}\left[\mathsf{Eval}\left(\langle M\rangle, \langle x\rangle\right) = M(x) : \begin{array}{c}\mathsf{MSK}\leftarrow\mathsf{Gen}(1^\lambda,1^n,1^m);\\ \langle M\rangle\leftarrow\mathsf{TMEncode}(\mathsf{MSK},M);\\ \langle x\rangle\leftarrow\mathsf{InpEncode}(\mathsf{MSK},x)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

for some negligible function negl.

Efficiency. The following properties holds:

- $\mathsf{Gen}(1^\lambda, 1^n, 1^m)$ runs in time polynomial in (λ, n, m).
- $\mathsf{TMEncode}(\mathsf{MSK}, M)$ runs in time polynomial in $(|\mathsf{MSK}|, |M|)$.
- $\mathsf{InpEncode}(\mathsf{MSK}, x)$ runs in time polynomial in $(|\mathsf{MSK}|, n)$.
- $\mathsf{Eval}(\langle M\rangle, \langle x\rangle)$ runs in time polynomial in $(\lambda, |M|, n, m, t)$, where t is the time to compute M on x.

Definition 3. *A scheme* $\mathsf{SuccGC} = (\mathsf{Gen}, \mathsf{TMEncode}, \mathsf{InpEncode}, \mathsf{Eval})$ *for a class of Turing machines is said to be a secure succinct garbling scheme if it satisfies the correctness, efficiency properties (defined above) and additionally satisfies the following security property: for large enough security parameter* λ*, for every* $n, m > 0$*, every PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* Sim *such that for every TM* M*, input* $x \in \{0,1\}^n$*,*

$$\left\{\mathsf{Sim}\left(1^\lambda, 1^n, 1^t, M, M(x)\right)\right\} \approx_c \left\{(\mathsf{TMEncode}(\mathsf{MSK}, M), \mathsf{InpEncode}(\mathsf{MSK}, x))\right\},$$

where $\mathsf{MSK} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^m)$ *and* t *is the time to compute* M *on* x*.*

We also consider a weaker notion of succinct garbling schemes, called bounded runtime succinct garbling schemes. We give the formal definition below.

Definition 4 (Bounded Runtime SGC)**.** *A scheme* $\mathsf{SuccGC} = (\mathsf{Gen}, \mathsf{TMEncode}, \mathsf{InpEncode}, \mathsf{Eval})$ *for a class of Turing machines with* m*-bit outputs associated with a worst case time bound* T *is said to be a* ***bounded runtime*** *succinct garbling scheme if it satisfies the correctness property (defined above) and additionally satisfies the following:*

- $\mathsf{Gen}, \mathsf{TMEncode}$ *and* $\mathsf{InpEncode}$ *additionally take as input a worst case time bound* T *(in binary representation). The runtime of* $\mathsf{Gen}, \mathsf{TMEncode}$ *and* $\mathsf{InpEncode}$ *grows only poly-logarithmically in* T*. We allow the runtime of* Eval *to grow polynomially in* T*.*
- *(Security) For every large security parameter* λ*, every PPT adversary* $\mathcal{A}$*, there exists a PPT simulator* Sim *such that for every TM* M*, input* $x \in \{0,1\}^n$*,*

  $$\left\{\mathsf{Sim}(1^\lambda, 1^n, 1^T, M, M(x))\right\} \approx_c \left\{(\mathsf{TMEncode}(\mathsf{MSK}, T, M), \mathsf{InpEncode}(\mathsf{MSK}, T, x))\right\},$$

  where $\mathsf{MSK} \leftarrow \mathsf{Gen}(1^\lambda, T, 1^n, 1^m)$*. The simulator runs in time polynomial in* $(\lambda, T, |M|, n, m)$ *and* $M(x)$ *is the output of* M *on* x *after* T *steps.*

2.4 Indistinguishability Obfuscation

We define the notion of indistinguishability obfuscation (iO) for circuits [BGI+01, GGH+13] below.

Definition 5 (Indistinguishability Obfuscator (iO) for Circuits)**.** *A uniform PPT algorithm* iO *is called an* ε*-secure indistinguishability obfuscator for a circuit family* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$*, where* $\mathcal{C}_\lambda$ *consists of circuits* C *of the form* $C : \{0,1\}^{n(\lambda)} \to \{0,1\}$*, if the following holds:*

- **Completeness:** *For every* $\lambda \in \mathbb{N}$*, every* $C \in \mathcal{C}_\lambda$*, every input* $x \in \{0,1\}^n$*, where* n *is the input length of* C*, we have that*

  $$\mathsf{Pr}\left[C'(x) = C(x) \ : \ C' \leftarrow \mathsf{iO}(\lambda, C)\right] = 1$$

- **ε-Indistinguishability:** *For any PPT distinguisher D, the following holds: for all sufficiently large $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$ such that $C_0(x) = C_1(x)$ for all inputs $x \in \{0,1\}^n$, we have:*

$$\left|\mathsf{Pr}\left[D(\lambda, \mathsf{iO}(\lambda, C_0)) = 1\right] - \mathsf{Pr}[D(\lambda, \mathsf{iO}(\lambda, C_1)) = 1]\right| \leq \varepsilon$$

If ε is negligible in λ then we refer to iO as a secure indistinguishability obfuscator.

iO from compact FE. The works of [AJ15, BV15] showed how to achieve indistinguishability obfuscation for circuits of input length n from $2^{n^2} \cdot \mathsf{negl}(\lambda)$-secure compact functional encryption scheme. Even indistinguishability obfuscator for circuits with logarithmic-sized inputs can only be based on quasi-polynomially secure functional encryption. In subsequent works [LZ17, LT17], this bound was improved and in particular, we have the following theorem.

Theorem 4. *[LZ17, LT17] For every large enough security parameter λ, assuming $2^n \cdot \varepsilon$-secure compact functional encryption, there exists an ε-secure indistinguishability obfuscator for circuits with input length n.*

In particular, when $n = \log(\lambda)$ and ε is negligible in security parameter, iO for n-length circuits can be based on polynomially secure compact functional encryption.

2.5 Somewhere Equivocal Encryption

We present the definition of somewhere equivocal encryption from [HJO+16].

Syntax. A somewhere equivocal encryption scheme with block-length B, message-length n (in blocks), and equivocation-parameter t (all polynomials in the security parameter) is a tuple of probabilistic polynomial algorithms $\mathsf{sEQ} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ such that:

- **Key Generation**, $\mathsf{KeyGen}(1^\lambda)$: On input λ, it outputs a key eqk. The size of the equivocation key is polynomial in (λ, B, t).
- **Encryption,** $\mathsf{Enc}(\mathsf{eqk}, \overrightarrow{m} = (m_1, \ldots, m_n))$: On input equivocation key eqk, messages $m_1, \ldots, m_n$ with $m_i \in \{0,1\}^B$, and outputs a ciphertext CT.
- **Decryption,** $\mathsf{Dec}(\mathsf{eqk}, \mathsf{CT})$: On input key eqk, ciphertext CT, it outputs a vector of messages $m_1, \ldots, m_n$.
- **Simulated Encryption,** $\mathsf{SimEnc}(1^\lambda, I, \{m_i\}_{i \notin I})$: On input security parameter λ, index set $I \subseteq [n]$, set of messages $\{m_i\}_{i \notin I}$, it outputs a ciphertext CT and state st.
- **Simulated Key Generation,** $\mathsf{SimKey}(\mathsf{st}, \{m_i\}_{i \in I})$: On input state st, set of messages $\{m_i\}_{i \in I}$, it outputs a key eqk'.

The above algorithms are required to satisfy the following properties:

Correctness. For every $\overrightarrow{m} = (m_1, \ldots, m_n) \in \{0,1\}^{B \times n}$, it should hold that $\mathsf{Dec}(\mathsf{eqk}, \mathsf{Enc}(\mathsf{eqk}, \overrightarrow{m})) = \overrightarrow{m}$, where $\mathsf{eqk} \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Simulation with no holes. We require that the distribution of $(\mathsf{CT}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(1^\lambda, \emptyset, \overrightarrow{m})$ and $\mathsf{eqk} \leftarrow \mathsf{SimKey}(\mathsf{st}, \emptyset)$ to be identical to $\mathsf{eqk} \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{eqk}, \overrightarrow{m})$. In other words, simulation when there are no holes (i.e., $I = \emptyset$) is identical to honest key generation and key generation.

Security. We define the security property next.

Definition 6. *We say that a somewhere equivocal encryption scheme* $\mathsf{sEQ} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ *with block-length* B*, message-length* n *(in blocks), and equivocation-parameter* t *(all polynomials in the security parameter) is said to be* **secure** *if the following holds for any PPT adversary* $\mathcal{A}$*, sufficiently large security parameter* $\lambda \in \mathbb{N}$*,*

$$|\mathsf{Pr}[\mathsf{Expt}_{\mathsf{sEQ},\mathcal{A}}(1^\lambda, 0) = 1] - \mathsf{Pr}[\mathsf{Expt}_{\mathsf{sEQ},\mathcal{A}}(1^\lambda, 1) = 1]| \leftarrow \mathsf{negl}(\lambda),$$

for some negligible function λ*. The experiment* $\mathsf{Expt}_{\mathsf{sEQ},\mathcal{A}}(1^\lambda, b)$ *is defined below:*

$\underline{\mathsf{Expt}_{\mathsf{sEQ},\mathcal{A}}(1^\lambda, b)}$*:*

1. *The adversary* $\mathcal{A}$ *on input security parameter* λ*, it sends two index sets* $I \subseteq [n]$ *and* $J \subseteq [n]$ *such that* $|I \cup J| \leq t$*, messages* $\{m_i \in \{0,1\}^B\}_{i \notin I}$*. Denote* $K = I \cup J$*.*
2. *The challenger sends* CT*, where* CT *is computed as follows:*
   - *If* $b = 0$*, it computes* $(\mathsf{CT}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(1^\lambda, I, \{m_i\}_{i \notin I})$*.*
   - *If* $b = 1$*, it computes* $\mathsf{CT} \leftarrow \mathsf{SimEnc}(1^\lambda, I, \{m_i\}_{i \notin K})$*.*
3. *The adversary then sends the rest of the messages* $\{m_i\}_{i \in I}$*.*
4. *The challenger sends* eqk *which is computed as follows:*
   - *If* $b = 0$*, it computes* $\mathsf{eqk} \leftarrow \mathsf{SimKey}(\mathsf{st}, \{m_i\}_{i \in I})$*.*
   - *If* $b = 1$*, it computes* $\mathsf{eqk} \leftarrow \mathsf{SimKey}(\mathsf{st}, \{m_i\}_{i \in K})$*.*
5. *The output of* $\mathcal{A}$ *is the output of the experiment.*

The definition as stated in Hemenway et al. considers the case when $|J| = 1$. By a standard hybrid argument, any definition that satisfies their definition also satisfies the above definition.

Theorem 5 ([HJO+16])**.** *Assuming the existence of one-way functions, there exists a somewhere equivocal encryption scheme for any polynomial message-length* n*, block-length* B*, and equivocation parameter* t*, having key size* $t \cdot B \cdot \mathrm{poly}(\lambda)$ *and ciphertext of size* $n \cdot B$ *bits.*

3 Locally Simulatable Garbling Schemes

In this section, we describe our main conceptual contribution: the notion of a *locally simulatable garbling scheme*. We refer the reader to the technical overview for an informal explanation of this security property.

We define three notions of local simulatability, beginning with the weakest definition.

Definition 7 (Weak L_{sim}-Local Simulation Security)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ *denote a garbling scheme for a class of circuits* $\mathcal{C}$*. We say that* Π *satisfies* **weak** L_{sim}**-local simulation security** *with* N_{comps} *local components if there exist PPT algorithms* $(\{\mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}}\}_{\mathbf{k} \in [q]})$ *and a vector of sets* $(T_1, \ldots, T_q) \subseteq [N_{\mathrm{comps}}]^q$ *for some polynomial* $q = q(\lambda)$ *such that no PPT adversary* $\mathcal{A}$ *wins the following game with non-negligible advantage over* $\frac{1}{2}$*.*

- $\mathcal{A}$ *outputs a circuit* C *of size* s*, an input* $x \in \{0,1\}^n$*, and a hybrid index* $0 \leq \mathbf{k} \leq q - 1$*.* $\mathcal{A}$ *sends* $(C, x, \mathbf{k})$ *to the challenger.*
- *The challenger samples* $\mathsf{MSK} \leftarrow \mathsf{Gen}(1^\lambda)$*, and a bit* $b \xleftarrow{\$} \{0,1\}$*.*
- *For every* $i \in [N_{\mathrm{comps}}]$*, the challenger computes* $\langle g_i \rangle$ *in one of two ways:*
  - *If* $i \in T_{\mathbf{k}+b}$*, compute* $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{MSK}, i, C, x)$*.*
  - *If* $i \notin T_{\mathbf{k}+b}$*, compute* $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{MSK}, i, C, C(x))$*.*

The notation $\mathsf{SimGtEnc}_0$ *is interpreted to denote a component of the honest garbling algorithm.*

- *The challenger computes the simulated input encoding* $\langle x \rangle \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}+b}(\mathsf{MSK}, C, x)$ *and sends* $\left((\langle g_i \rangle)_{i \in [N_{\mathrm{comps}}]}, \langle x \rangle\right)$ *to* $\mathcal{A}$.
- $\mathcal{A}$ *outputs a bit* b' *and wins if* $b' = b$.

Finally, we have the following syntactic requirements:

- **The final simulator is input-independent.** *That is,* $T_q = \emptyset$ *and* $\mathsf{SimInpEnc}_q$ *is an efficient function of* $(C, C(x))$ *rather than* (C, x).
- **Gate encodings are small.** *That is, the output length of* $\mathsf{SimGtEnc}$ *is at most* $\mathrm{poly}(\lambda)$.
- **Input-dependent simulation is local.** *That is,* $|T_{\mathbf{k}}| \leq L_{\mathsf{sim}}$ *for every* $\mathbf{k}$.

Remark 1. *Note that weak* L_{sim}*-local simulation security implies the selective simulation security (Definition 1) of garbling schemes. In particular, the simulator* Sim *is defined as follows: it takes as input* $(1^\lambda, C, C(x))$*, generates the master secret key* $\mathsf{MSK} \leftarrow \mathsf{Gen}(1^\lambda)$*, computes the garbled circuit* $(\langle g_1 \rangle, \ldots, \langle g_{N_{\mathrm{comps}}} \rangle)$ *by applying* $\mathsf{SimGtEnc}_q(\mathsf{MSK}, \cdot, C, C(x))$ *on every gate in the circuit, and finally computes the input encoding using* $\mathsf{SimInpEnc}_q(\mathsf{MSK}, C, C(x))$*. Note that we are crucially using the fact that the input-dependent simulation component of* $(\mathsf{SimGtEnc}_q, \mathsf{SimInpEnc}_q)$ *is empty and that* $\mathsf{SimInpEnc}_q$ *can be computed from* $(C, C(x))$.

Discussion. We note that Yao's garbling scheme [Yao86] already satisfies weak L_{sim}-local simulation security, where L_{sim} is $O(w)$ with w being the width of the circuit being garbled. We refer the reader to Section 1.2 for an informal description of Yao's garbling scheme. The gate encodings correspond to the garbled tables associated with every gate in the circuit. As noted in Section 1.2, the i^{th} hybrid in the proof of security is associated with the intermediate simulator Sim_i that performs the input-dependent simulation of the i^{th} layer of the garbled circuit and performs the input-**in**dependent simulation of the garbled gates in the other layers.

While Definition 7 captures some notion of local simulation, for applications such as adaptive garbling and succinct garbling, it turns out that we require *strengthenings* of this definition.

3.1 Semi-Adaptive Local Simulation

We first give a definition that suffices to construct adaptive garbling schemes. The main difference between this scheme and the weak local simulation is that the adversary can choose the input x *after* seeing the input-**in**dependent components of the garbled circuit.

In the following definition, we highlight in red the changes from the weak local simulation security definition.

Definition 8 (Semi-Adaptive L_{sim}-Local Simulation Security)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ *denote a garbling scheme for a class of circuits* $\mathcal{C}$*. We say that* Π *satisfies* **semi-adaptive** L_{sim}**-local simulation security** *with* N_{comps} *local components if there exist PPT algorithms* $\left(\{\mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}}\}_{\mathbf{k} \in [N_{\mathrm{comps}}]}\right)$ *and a vector of sets* $(T_1, \ldots, T_q) \subseteq [N_{\mathrm{comps}}]^q$ *for some polynomial* $q = q(\lambda)$ *such that no PPT adversary* $\mathcal{A}$ *wins the following game with non-negligible advantage.*

- $\mathcal{A}$ *outputs a circuit* C *and hybrid index* $0 \leq \mathbf{k} \leq q - 1$ *and sends* $(C, \mathbf{k})$ *to the challenger.*
- *The challenger samples* $\mathsf{MSK} \leftarrow \mathsf{Gen}(1^\lambda)$*, and a bit* $b \xleftarrow{\$} \{0, 1\}$.

- *For every $i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$, the challenger computes $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{MSK}, i, C)$.*[10] *The challenger sends these garbled gates to $\mathcal{A}$. When $\mathbf{k} = 0$, this is interpreted to mean that the challenger computes an honestly generated garbled circuit and sends the components corresponding to $[N_{\text{comps}}] \backslash T_1$.*
- *$\mathcal{A}$ selects an input x and sends it to the challenger.*
- *For every $i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$, the challenger computes $\langle g_i \rangle$ in one of two ways (and then sends it to $\mathcal{A}$):*
  - *If $i \in T_{\mathbf{k}+b}$, compute $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{MSK}, i, C, x)$.*
  - *If $i \notin T_{\mathbf{k}+b}$, compute $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{MSK}, i, C, C(x))$.*
- *The challenger computes the simulated input encoding $\langle x \rangle \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}+b}(\mathsf{MSK}, C, x)$ and sends $\langle x \rangle$ to $\mathcal{A}$.*
- *$\mathcal{A}$ outputs a bit b' and wins if $b' = b$.*

Finally, we have the following syntactic requirements, which are identical to the syntactic requirements of Definition 7:

- **The final simulator is input-independent.** *That is, $T_q = \emptyset$ and $\mathsf{SimInpEnc}_q$ is an efficient function of $(C, C(x))$ rather than (C, x).*
- **Gate encodings are small.** *That is, the output length of $\mathsf{SimGtEnc}$ is at most $\text{poly}(\lambda)$.*
- **Input-dependent simulation is local.** *That is, $|T_{\mathbf{k}}| \leq L_{\mathsf{sim}}$ for every $\mathbf{k}$.*

3.2 Strong Local Simulation

We now define the notion of locally simulatable garbling scheme that suffices for the succinct garbling application. Intuitively, we need our garbling scheme (and corresponding local simulation strategy) to be decomposable into components each of whose security depends only on a small part of the secret key. We begin by defining a "local encoding property" of a garbling scheme Π. Roughly, speaking this property says that the garbling of a circuit C consists of many gate encodings, each of which are "small" (polynomial in the security parameter). Moreover, the master secret key consists of many independently generated keys such that every gate encoding is generated as a function of a small subset of these keys.

Definition 9 (Local Encoding Property)**.** *Let $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ denote a garbling scheme for a class of circuits $\mathcal{C}$. We define probabilistic polynomial time algorithms $(\mathsf{KeyListGen}, \mathsf{LocalKG}, \mathsf{GateEnc})$ as follows.*

- **Key List Generation:** *$\mathsf{KeyListGen}(C, i)$ takes as input a circuit C of size s, an index $i \in [N_{\text{comps}} = \text{poly}(s)]$ and outputs a list $\mathbf{L}_i \subseteq [N_{\text{keys}} = \text{poly}(s)]$ along with a state $\mathsf{st}_i \in \{0,1\}^{\text{poly}(\log(s))}$.*
- **Local Key Generation:** *$\mathsf{LocalKG}(1^\lambda, j)$ takes as input the security parameter and an index $j \in [N_{\text{keys}}]$, and it outputs a local secret key sk_j.*
- **Gate Encoding:** *$\mathsf{GateEnc}(\mathsf{st}, (sk_j)_{j \in \mathbf{L}})$ takes as input a state st and a tuple of local secret keys $(sk_j)_{j \in \mathbf{L}}$. It outputs an encoding $\langle g \rangle$.*

We say that Π has **local encoding** *$(\mathsf{KeyListGen}, \mathsf{LocalKG}, \mathsf{GateEnc})$ with N_{comps} components if the algorithms Gen, Gb, and IEnc have the following specific form:*

(i) $\mathsf{Gen}(1^\lambda, 1^s)$, on input security parameter λ and an upper bound on the circuit size s, outputs the master secret key $\mathsf{MSK} = (sk_1, \ldots, sk_{N_{\text{keys}}})$, where $sk_j \leftarrow \mathsf{LocalKG}(1^\lambda, j)$ for $j \in [N_{\text{keys}}]$

[10] Note that this simulator does not receive the output $C(x)$, unlike the weak local simulatability definition.

(ii) $\mathsf{Gb}(\mathsf{MSK}, C)$, *on input master secret key* MSK *and a circuit* C, *first computes* $\mathsf{KeyListGen}(C, i)$ *for every* $i \in [N_{\text{comps}}]$ *to obtain lists* $\mathbf{L}_i$ *and states* st_i. *It then computes* $\mathsf{GateEnc}(\mathsf{st}_i, (sk_j)_{j \in \mathbf{L}_i})$ *to obtain* $\langle g_i \rangle$ *for every* $i \in [s]$ *and outputs* $\langle C \rangle = (\langle g_1 \rangle, \ldots, \langle g_N \rangle)$.

(iii) $\mathsf{IEnc}(\mathsf{MSK}, x)$ *only depends on* $\mathsf{SK}_{\text{Inp}} := (sk_j)_{j \in \mathbf{I}}$ *for some fixed set* $\mathbf{I} \subset [N_{\text{keys}}]$ *of size* $\text{poly}(n, \lambda)$, *rather than the entire master secret key. This set* $\mathbf{I}$ *should be an efficiently computable function of* (n, s).

Moreover, we require that $\mathsf{KeyListGen}(C, i)$ *runs in polynomial time* $\text{poly}(|M|, \log(s))$ *if* C *can be represented by a Turing machine* M. *In this case, we re-define* $\mathsf{KeyListGen}$ *to take as input* (M, T, n, i).

In the following definition, we highlight in red the changes from the weak simulation security definition.

Definition 10 (Strong L_{sim}-Local Simulation Security)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ *denote a garbling scheme for a class of circuits* $\mathcal{C}$ *with local encoding* $(\mathsf{KeyListGen}, \mathsf{LocalKG}, \mathsf{GateEnc})$. *We say that it satisfies* **strong** L_{sim}**-local simulation security** *if there exist PPT algorithms* $\left(\{\mathsf{SimKeyListGen}_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}}\}_{\mathbf{k} \in [q]}\right)$ *and a vector of sets* $(T_1, \ldots, T_q) \subseteq [s]^q$ *for some polynomial* $q = q(\lambda)$ *such that no PPT adversary* $\mathcal{A}$ *wins the following game with non-negligible advantage over* $\frac{1}{2}$.

- $\mathcal{A}$ *chooses a circuit* C, *input* x, *and hybrid index* $0 \leq \mathbf{k} \leq q - 1$. $\mathcal{A}$ *sends* $(C, x, \mathbf{k})$ *to the challenger.*
- *For every* $i \in [N_{\text{comps}}]$, *the challenger evaluates* $\mathsf{SimKeyListGen}_{\mathbf{k}}(C, i)$ *to obtain* $\mathbf{L}_i^{(\mathbf{k})} \subseteq [N_{\text{keys}}]$ *and* $\mathsf{st}_i^{(\mathbf{k})} \in \{0,1\}^{\text{poly}(\log(s))}$.
- *Define* $\mathsf{list}_{\mathbf{k}}^{\mathsf{rand}} = \bigcup_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{(\mathbf{k})}$ *and* $\mathsf{list}_{\mathbf{k}}^{\mathsf{public}} = \bigcup_{i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{(\mathbf{k})}$.
- $\mathcal{A}$ *selects secret keys* $\{sk_j\}_{j \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}}$ *and sends them to the challenger.*
- *For every* $j \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$, *the challenger samples* $sk_j \leftarrow \mathsf{LocalKG}(1^\lambda, j)$ *and sets* $\mathsf{MSK} = (sk_j)_{j \in [N_{\text{comps}}]}$.
- *The challenger now samples a uniformly random bit* b, *samples from the distribution* $D_b(\mathsf{MSK})$, *and sends the output to* $\mathcal{A}$. *Sampling from the distribution* $D_b(\mathsf{MSK})$ *is defined as follows:*
  - *For every* $i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$, *sample* $\langle g_i \rangle$ *in one of two ways:*
    - *If* $i \in T_{\mathbf{k}+b}$, *compute* $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{MSK}, i, C, x)$.
    - *If* $i \notin T_{\mathbf{k}+b}$, *compute* $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{st}_i^{(\mathbf{k}+b)}, (sk_j)_{j \in \mathbf{L}_i^{(\mathbf{k}+b)}}, C(x))$, *where* $(\mathsf{st}_i^{(\mathbf{k}+b)}, \mathbf{L}_i^{(\mathbf{k}+b)}) = \mathsf{SimKeyListGen}_{\mathbf{k}+b}(C, i)$.
  - *Sample* $\langle x \rangle \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}+b}(\mathsf{MSK}, C, x)$.
  - *Output* $\left(\left\{\langle g_i \rangle\right\}_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}, \langle x \rangle, \left\{sk_j\right\}_{j \in \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}}\right)$.
- *The adversary outputs a bit* b' *and wins if* $b' = b$.

Finally, we have the following syntactic requirements, which subsume the syntactic requirements of Definition 7:

- **The final simulator is input-independent.** *That is,* $T_q = \emptyset$ *and* $\mathsf{SimInpEnc}_q$ *is an efficient function of* $(C, C(x))$ *rather than* (C, x).
- **Neighbor lists and states are small and efficiently computable.**[11] *That is,* $\left|\mathbf{L}_i^{(\mathbf{k})}\right| = \text{poly}(\log(s))$ *and* $\left|\mathsf{st}_i^{(\mathbf{k})}\right| = \text{poly}(\log(s))$ *for every* i *and every* $\mathbf{k}$. *As in the case of* $\mathsf{KeyListGen}$,

[11] Note that this implies that gate encodings are small and efficiently computable.

we require that $\mathsf{SimKeyListGen}(C, i)$ *runs in time polynomial in* $(|M|, \log(s))$ *if* C *can be represented by a Turing machine* M *running in time at most* T *on inputs of length* n*; we then use the notation* $\mathsf{SimKeyListGen}(M, T, n, i)$.

- **Input-dependent simulation is local.** *That is,* $|T_{\mathbf{k}}| \leq L_{\mathsf{sim}}$ *for every* $\mathbf{k}$.
- **Input-independent hybrid simulators are functionally equivalent.** *That is, for every* $\mathbf{k} \in [q-1]$*, we require that* $\mathsf{SimGtEnc}_{\mathbf{k}}(\mathsf{st}_i, \cdot)$ *and* $\mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{st}_i, \cdot)$ *are functionally equivalent and* $\mathsf{SimKeyListGen}_{\mathbf{k}}(\cdot, i)$ *and* $\mathsf{SimKeyListGen}_{\mathbf{k}+1}(\cdot, i)$ *are functionally equivalent for every* $i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$.

The main differences to note about strong local simulation security are that the input-independent simulator only takes the *local secret keys* as input rather than the master secret key, and that local indistinguishability is required to hold even when the "non-local keys" (i.e. the keys used to garble any gate not in $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$) are given to the adversary. Finally, note that the input-independent simulator does not receive the full circuit C as input, so all dependence on C for the (simulated and real) gate encoding must be controlled by the $\mathsf{KeyListGen}$ algorithm.

We claim that Yao's garbling scheme (see Section 1.2 for an informal description of the scheme) is an instantiation of a garbling scheme satisfying strong local simulation security.

Theorem 6. *Yao's garbling scheme [Yao86] for circuits satisfies strong* L_{sim}*-local simulation security, where* $L_{\mathsf{sim}} = w \cdot \mathrm{poly}(\lambda)$*,* λ *is the security parameter and* w *is the width of the circuit garbled.*

Proof Sketch. We first observe that Yao's garbling scheme satisfies the local encoding property. Without loss of generality, we assume that the class of circuits associated with the Yao's garbling scheme consists only of layered circuits where all the wires in one layer input wires to the next layer. Moreover, we assume that every layer has the same number of gates. To prove the local encoding property, we define the following algorithms:

- $\mathsf{KeyListGen}(C, i)$: It takes as input circuit C of size s, index $i \in [s]$ (that is, N_{comps} is set to be s), outputs a list $\mathbf{L}_i \subseteq [N_{\mathrm{keys}}]$, where N_{keys} is set to be the number of wires in C, such that index j belongs to $\mathbf{L}_i$ if and only if j^{th} wire is incident to the i^{th} gate and $\mathsf{st}_i = (i, \mathbf{L}_i)$. Recall that local encoding (Definition 9) must satisfy the property that $\mathsf{KeyListGen}(M, T, n, i)$ can be computed in time $\mathrm{poly}(|M|, \log(T), \log(n))$ when M is a Turing machine. To achieve this, we compute $\mathsf{FindGate}(M, n, T, i)$ (defined in Section 2) to get $\mathbf{L}_i$. Then output $(i, \mathbf{L}_i)$, as before.
- $\mathsf{LocalKG}(1^\lambda, j)$: It takes as input security parameter λ, index $j \in [N_{\mathrm{keys}}]$ and outputs the local key $sk_j = (K_0^j, K_1^j)$ corresponding to the j^{th} wire. The label K_0^j is associated with bit 0 and the label K_1^j is associated with bit 1.
- $\mathsf{GateEnc}(\mathsf{st}, (sk_j)_{j \in \mathbf{L}_i})$: It takes as input $\mathsf{st} = (i, \mathbf{L}_i)$, the set of wire labels $(sk_j)_{j \in \mathbf{L}_i}$ and computes a garbled table of the i^{th} gate in C. To compute this garbled table, the labels of the wires incident to the i^{th} gate are required and this is exactly the list $(sk_j)_{j \in \mathbf{L}_i}$.

To prove the strong local simulation property, we describe all the intermediate simulators. The computational indistinguishability of the output distributions of these simulators follow essentially from the proof of security of Yao's garbling scheme and thus we omit the details. Formally, we define $\left(\{\mathsf{SimKeyListGen}_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}}\}_{\mathbf{k} \in [q]}\right)$ and a vector of sets $(T_1, \ldots, T_q)$. We set $q = 2d - 1$, where d is the number of layers in the circuit being garbled.

- $T_{\mathbf{k}}$ consists of the indices corresponding to all the gates in the $\frac{\mathbf{k}}{2}^{th}$ and $(\frac{\mathbf{k}}{2}+1)^{th}$ layers if $\mathbf{k}$ is even, otherwise it consists of the indices corresponding to all the gates in the $(\lfloor \frac{\mathbf{k}}{2} \rfloor + 1)^{th}$ layer if $\mathbf{k}$ is odd.
- $\mathsf{SimKeyListGen}_{\mathbf{k}}(C, i)$: This is the same as $\mathsf{KeyListGen}$.

- $\mathsf{SimGtEnc}_{\mathbf{k}}(\mathsf{Arg})$: We consider the following two cases depending on Arg. Before that, we define some rules that dictate how the garbled gate is generated. We say that a garbled table of an i^{th} gate is generated according to *white rule* if the garbled table of the i^{th} gate is computed honestly. We say that a garbled table of an i^{th} gate is generated according to *grey rule* if every entry in the garbled table of the i^{th} gate encrypts only K_b^j, where j^{th} wire is the output wire of the i^{th} gate and b is the value carried by the i^{th} gate during the computation of C on x. We say that a garbled table of an i^{th} gate is generated according to *black rule* if the j^{th} wire is assigned two lables K_0^j and K_1^j, where j^{th} wire is the output wire of the i^{th} gate, and every entry in the garbled table of the i^{th} gate encrypts only K_b^j with bit b randomly sampled.

  We now describe the following two cases.

  - (Input-dependent Simulation) If $\mathsf{Arg} = (\mathsf{MSK}, i, C, x)$: the garbled table of the i^{th} gate is generated according to the grey rule.
  - (Input-independent Simulation) If $\mathsf{Arg} = (\mathsf{st}, (sk_j)_{j \in \mathbf{L}_i^{\mathbf{k}}}, C(x))$: we consider the following cases.
    * If $\mathbf{k}$ is odd and the i^{th} gate is in a layer below the $(\lfloor \frac{\mathbf{k}}{2} + 1 \rfloor)^{th}$ layer of C: the garbled table of the i^{th} gate is generated according to the black rule.
    * If $\mathbf{k}$ is odd and the i^{th} gate is in a layer above the $(\lfloor \frac{\mathbf{k}}{2} \rfloor + 1)^{th}$ layer of C: in this case, the garbled table of the i^{th} gate is generated according to the white rule.
    * If $\mathbf{k}$ is even and the i^{th} gate is in a layer below $(\frac{\mathbf{k}}{2})^{th}$ layer of C: the garbled table of the i^{th} gate is generated according to the black rule.
    * If $\mathbf{k}$ is even and the i^{th} gate is in a layer above $(\frac{\mathbf{k}}{2} + 1)^{th}$ layer of C: in this case, the garbled table of the i^{th} gate is generated according to the white rule.
- $\mathsf{SimInpEnc}_{\mathbf{k}}(\mathsf{MSK}^{(\mathbf{k})}, C, x)$: Suppose the labels of the input wires are $(K_0^{j_1}, K_1^{j_1}, \ldots, K_0^{j_n}, K_1^{j_n})$. Generate the input wire labels to be $\mathbf{K_I} = (K_{b_1}^{j_1}, \ldots, K_{b_n}^{j_n})$, where the bits $b_1, \ldots, b_n$ are randomly sampled. To generate the translation table, we consider two cases. If $\mathbf{k} < q$ then the translation table is generated honestly, otherwise the translation table is generated as follows: suppose the garbled table of the i^{th} output gate encrypts the label K then include in the translation table $\left(i; K \to (C(x))_i; K' \to \overline{(C(x))_i}\right)$ with K' uniformly sampled independent of all the labels sampled in the garbled circuit. The notation $K \to b$ means the label K represents bit b and $(C(x))_i$ denotes the i^{th} output bit of $C(x)$.

We omit the proof of indistinguishability of the consecutive simulated distributions since it essentially follows from the proof of security of Yao's garbling scheme (see [HJO+16] for a more modular proof). □

3.3 Locally Simulatable Garbling Schemes

We conclude the section by giving a full definition of locally simulatable garbling.

Definition 11 ((Weak; Semi-Adaptive; Strong) $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-Locally Simulatable Garbling Schemes)**.** *A scheme consisting of algorithms* $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval}, \mathsf{KeyListGen}, \mathsf{LocalKG}, \mathsf{GateEnc})$ *is said to be a* $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$**-locally simulatable garbling scheme** *with* N_{comps} *components if* $(\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$ *is a garbling scheme (Section 2.1) and additionally, the following holds:*

- **Local encoding***: In the "strong" case,* $(\mathsf{KeyListGen}, \mathsf{LocalKG}, \mathsf{GateEnc})$ *is a local encoding for* Π *with* N_{comps} *components.*
- **Online computational complexity***: The evaluation of* $\Pi.\mathsf{IEnc}(\mathsf{SK}_{\mathrm{Inp}}, x)$ *runs in time* $L_{\mathsf{inp}} = L_{\mathsf{inp}}(\lambda, |x|, m)$.
- L_{sim}**-simulation security***:* Π *satisfies (weak; semi-adaptive; strong)* L_{sim}*-simulation security with* N_{comps} *components.*

4 Statements of Our Results

In this section, we summarize our main results for reference throughout the paper.

4.1 Locally Simulatable Garbling

Our results on the existence of locally simulatable garbling schemes, proved in Section 5, are as follows.

Theorem 7 (Construction of Strong Locally Simulatable Garbling)**.** *Assuming the existence of selectively secure laconic oblivious transfer (Definition 12), there exists a strong locally simulatable garbling scheme (Definition 10) for s-sized circuits with input length n and output length m with the following parameters:*

- *The online computational complexity is given by $L_{\mathsf{inp}}(\lambda, |x|, m) = n \cdot \mathrm{poly}(\lambda, \log(s)) + m$.*
- *The online communication complexity is given by $n + m + \mathrm{poly}(\lambda, \log(s))$.*
- *For every s-sized C, the simulation locality is given by $L_{\mathsf{sim}}(\lambda, C) = \mathrm{poly}(\lambda, \log(s))$.*

Corollary 1. *The conclusion of Theorem 7 holds under either the CDH assumption or the LWE assumption.*

Remark 2. *The online computational complexity of the CDH-based scheme in Theorem 7 is $O(n) + m + \mathrm{poly}(\lambda, \log(s))$.*

Theorem 8 (Construction of Semi-Adaptive Locally Simulatable Garbling (informal))**.** *Assuming the existence of adaptively secure laconic oblivious transfer (Definition 12), there exists a semi-adaptive locally simulatable garbling scheme (Definition 8) with the parameters of Theorem 7.*

4.2 Succinct Garbling

Our results on the existence of succinct garbling schemes, proved in Section 6, are as follows.

Theorem 9 (Construction of Succinct Garbling)**.** *Suppose that the following objects exist:*

- *A strong $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-locally simulatable garbling scheme Π_{LGC} for all circuits.*
- *Indistinguishability obfuscation for $\mathcal{C}$ where the runtime of the obfuscation is a fixed polynomial in s and λ. The circuit class $\mathcal{C} = \bigcup_{k \in \mathbb{N}} \mathcal{C}_k$, where $\mathcal{C}_k$ consists of all s-sized circuits with input length $k \cdot \log(\lambda)$.*
- *A family of one-way functions.*

Then, there exists a bounded-runtime succinct garbling scheme (Definition 4) SuccGC for Turing machines with polynomial runtime $T(|x|)$, input length n and output length m, where:

- *The size of an encoded Turing machine $\langle M \rangle$ (and runtime of $\mathsf{TMEncode}(\mathsf{MSK}, M)$) is $\mathrm{poly}(\lambda, L_{\mathsf{sim}}(\lambda, C), |M|, \log(T))$, where $C = \mathsf{CktTransform}(M, n, T)$.*
- *The runtime of $\mathsf{InpEncode}(\mathsf{MSK}, x)$ is $L_{\mathsf{inp}}(\lambda, n, m)$, and the online communication complexity of SuccGC matches that of Π_{LGC}.*

Moreover, there exists a succinct garbling scheme for Turing machines with unbounded *polynomial runtime, bounded input length n and bounded output length m, where:*

- *The size of an encoded Turing machine $\langle M \rangle$ (and runtime of $\mathsf{TMEncode}(\mathsf{MSK}, M)$) is $\mathrm{poly}(\lambda, L_{\mathsf{sim}}(\lambda, C), |M|)$, where $C = \mathsf{CktTransform}(M, n, 2^\lambda)$.*

- *The runtime of* $\mathsf{InpEncode}(\mathsf{MSK}, x)$ *is* $\mathrm{poly}(\lambda) \cdot L_{\mathsf{inp}}(\lambda, n, m)$ *and the online communication complexity matches that of* Π_{LGC} *up to a factor of* λ*.*

An immediate consequence of the above theorem is the following.

Theorem 10. *A succinct garbling scheme exists under either of the following assumptions (all with polynomial security):*

- *Compact functional encryption for* $\mathsf{P/poly}$ *and one of* $\{CDH, LWE\}$*.*
- *Indistinguishability obfuscation for* $\mathsf{P/poly}$ *and one-way functions.*

The second of these two statements follows from Appendix A.

4.3 Adaptively Secure Garbling

Our result on adaptively secure garbling schemes, proved in Section 7, is as follows.

Theorem 11 (Construction of Adaptively Secure Garbling)**.** *Suppose that the following objects exist:*

- *A semi-adaptive* $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$*-locally simulatable garbling scheme* Π_{LGC} *for* $\mathsf{P/poly}$*.*
- *A family of one-way functions.*

Then, there exists an adaptively secure garbling scheme Π_{Ada} *for* $\mathsf{P/poly}$ *(with bounded output length* m*) with online communication complexity* $L_{\mathsf{inp}}(\lambda, |x|, m) + L_{\mathsf{sim}} \cdot \mathrm{poly}(\lambda)$*.*

We now proceed to prove the above results.

5 Locally Simulatable Garbling from Laconic OT

In this section, we prove Theorem 7; that is, we give a construction of a garbling scheme satisfying *strong local simulation security* with $L_{\mathsf{sim}} = \mathrm{poly}(\lambda, \log(s))$, $L_{\mathsf{inp}}(\lambda, |x|, m) = m + n \cdot \mathrm{poly}(\lambda)$, and $N_{\mathrm{comps}} = s$. This construction assumes the existence of *appendable laconic oblivious transfer* (lacOT), which is a modification of laconic OT [CDG+17] that we define below. Our construction follows the [GS18a] construction of adaptively secure garbled circuits from lacOT, with the following differences:

- We do not make use of somewhere equivocal encryption.
- The natural adaptation of [GS18a] requires a long preprocessing step[12]; we make use of appendable laconic OT rather than updatable laconic OT to avoid this problem (and simplify the construction).
- We make some minor additional changes for ease of presentation.

The [GS18a] result is easily recovered by combining this construction with the generic transformation of Section 7.

We begin by recalling the definition of laconic oblivious transfer, which is taken from [CDG+17, GS18a].

Definition 12 (Laconic Oblivious Transfer)**.** *A laconic oblivious transfer scheme* lacOT *is specified by four algorithms* $(\mathsf{Gen}, \mathsf{H}, \mathsf{Send}, \mathsf{Receive})$ *with the following syntax.*

- $\mathsf{Gen}(1^\lambda)$ *takes as input the security parameter and outputs a common reference string* crs*.*

[12] For those familiar with [GS18a], the preprocessing corresponds to computing the initial hash value $H(\mathsf{crs}, 0^s)$.

- $\mathsf{H}(\mathsf{crs}, D)$ *is a deterministic algorithm that takes as input the CRS as well as a database* $D \in \{0,1\}^*$ *and ouputs a hash value* h *and a state* $\hat{D}$*.*
- $\mathsf{Send}(\mathsf{crs}, h, i, m_0, m_1)$ *takes as input the CRS, hash value* h*, a pair of messages* (m_0, m_1) *and an index* $i \in \mathbb{N}$*. It outputs a ciphertext* c*.*
- $\mathsf{Receive}^{\hat{D}}(\mathsf{crs}, c, i)$ *is an algorithm with random access to* $\hat{D}$ *that takes as input the CRS, a ciphertext* c*, and an index* $i \in \mathbb{N}$*. It outputs a message* m*.*

The scheme lacOT *must satisfy the following correctness and security properties:*

- ***Correctness:*** *For all* $D \in \{0,1\}^*$*, all* $i \in \mathbb{N}$*, and all* $(m_0, m_1) \in \{0,1\}^2$*, we have that*

$$\mathsf{Receive}^{\hat{D}}(\mathsf{crs}, \mathsf{Send}(\mathsf{crs}, h, i, m_0, m_1), i) = m_{D_i}$$

  with probability 1*, where* $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $(h, \hat{D}) \leftarrow \mathsf{H}(\mathsf{crs}, D)$*.*
- ***Sender Privacy:*** *For all* $D \in \{0,1\}^*$*, all* $i \in \mathbb{N}$*, and all* $(m_0, m_1) \in \{0,1\}^2$*, we have that*

$$\Big(\mathsf{crs}, h, \hat{D}, \mathsf{Send}(\mathsf{crs}, h, i, m_0, m_1)\Big) \cong_c \Big(\mathsf{crs}, h, \hat{D}, \mathsf{Send}(\mathsf{crs}, h, i, m_{D_i}, m_{D_i})\Big)$$

  where $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $(h, \hat{D}) = \mathsf{H}(\mathsf{crs}, D)$*.*

Adaptive Security of Laconic OT. The above security notion is *selective* in the sense that the database D must be specified independently of the CRS. In order to obtain a locally simulatable garbling scheme satsifying semi-adaptive security, we require that the underlying laconic OT scheme satisfies *adaptive security*, meaning that the computational indistinguishability holds even for (D, i) chosen by an adversary given the CRS in advance. However, strong local simulation security of our garbling scheme will follow from the selective security of the underlying laconic OT scheme.

On the Existence of Laconic OT. By [CDG+17, DG17, BLSV18, DGHM18], we know that laconic OT can be constructed from either the CDH assumption, the LWE assumption, or the LPN assumption with noise rate $\log^2(n)/n$.

Appendability In the construction below, we will make use of a laconic OT scheme with an additional functionality: the ability of the sender and receiver to *update* their hash value h and state $\hat{D}$, respectively, in a way that corresponds to appending a bit b to the database. We require that this update procedure is entirely non-interactive once they have agreed on the bit b.

Definition 13 (Appendable Laconic Oblivious Transfer)**.** *A laconic OT scheme* $\mathsf{lacOT} = (\mathsf{Gen}, \mathsf{H}, \mathsf{Send}, \mathsf{Receive})$ *is called* ***appendable*** *if it supports the following two additional (deterministic) algorithms.*

- $\mathsf{HashAppend}(\mathsf{crs}, h, b)$ *takes as input the CRS, hash value* h*, and a bit* b*. It outputs an updated hash value* h'*.*
- $\mathsf{Append}^{\hat{D}}(\mathsf{crs}, b)$ *is an algorithm with random access to* $\hat{D}$ *that takes as input the CRS and a bit* b*. It updates the state* $\hat{D} \to \hat{D}'$*.*

The correctness property that we require is that for all $N \in \mathbb{N}$ *and all* $D \in \{0,1\}^N$*, we have that*

$$(\mathsf{HashAppend}(\mathsf{crs}, h, b), \mathsf{Append}^{\hat{D}}(\mathsf{crs}, b)) = \mathsf{H}(\mathsf{crs}, D')$$

with probability 1*, where* $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda)$*,* $(h, \hat{D}) = \mathsf{H}(\mathsf{crs}, D)$*, and* $D' \in \{0,1\}^{N+1}$ *is the database satisfying* $D'_{N+1} = b$ *and* $D'_i = D_i$ *for all* $i \le N$*. We require* no additional security property.

Obtaining Appendable Laconic OT We first note that appendable laconic OT can be constructed generically from any laconic OT. The construction is as follows. We start with a laconic OT scheme $\mathsf{laconic} = (\mathsf{laconic.Gen}, \mathsf{laconic.H}, \mathsf{laconic.Send}, \mathsf{laconic.Receive})$, and we assume that $\mathsf{laconic.H}(\mathsf{crs}, D)$ is computed via a Merkle tree that iteratively computes $H(\mathsf{crs}, \cdot)$ for some hash function H mapping 2λ bits to λ bits (and that the resulting state $\hat{D}$ is simply the Merkle tree). By [CDG+17], this assumption is without loss of generality.[13]

Given such a laconic OT scheme, one can define an appendable laconic OT scheme as follows.

- $\mathsf{Gen}(1^\lambda)$ calls and outputs $\mathsf{crs} \leftarrow \mathsf{laconic.Gen}(1^\lambda)$.
- $\mathsf{H}(\mathsf{crs}, D)$ is defined as follows: for $D \in \{0,1\}^{\lambda N+n}$, let $N = \sum_{i=0}^{\lambda} N_i 2^i$ be the binary representation of N.[14] Then, let $N^{(j)} = \sum_{\ell=\log(N)-j}^{\log(N)} N_\ell 2^\ell$ be the jth (reverse) truncation of N, let $D^{(j)} = D[\lambda N^{(j-1)} : \lambda N^{(j)} - 1]$ (this will be the empty string by definition when $N_{\log(N)-j} = 0$), and let $(h^{(j)}, \hat{D}^{(j)}) = \mathsf{laconic.H}(\mathsf{crs}, D^{(j)})$ (again interpreted to be the empty string if $N_{\log(N)-j} = 0$). Finally, let $d = D[\lambda N, \lambda N + n - 1]$ and output $h = \lambda N + n || h^{(0)} || h^{(1)} || \ldots || h^{(\log(N))} || d$ and $\hat{D} = (\lambda N + n, \hat{D}^{(0)}, \hat{D}^{(1)}, \ldots, \hat{D}^{(\log(N))}, d)$.
- $\mathsf{Send}(\mathsf{crs}, h, i, m_0, m_1)$ interprets $h = \lambda N + n || h^{(0)} || h^{(1)} || \ldots || h^{(\log(N))} || d$ and does the following: if $i \geq \lambda N$, output $m_{d_{i-\lambda N}}$. Otherwise, output $\mathsf{laconic.Send}(\mathsf{crs}, h^{(j)}, i')$, where $j \in [\log(N)]$ is the unique index such that $i \in [\lambda N^{(j-1)}, \lambda N^{(j)} - 1]$ and $i' = i - \lambda N^{(j-1)}$.
- $\mathsf{Receive}^{\hat{D}}(\mathsf{crs}, c, i)$ outputs c if $i \geq \lambda N$, and otherwise calls $\mathsf{laconic.Receive}^{\hat{D}^{(j)}}(\mathsf{crs}, c, i')$ for (j, i') as above.
- $\mathsf{HashAppend}(\mathsf{crs}, h, b)$ interprets $h = \lambda N + n || h^{(0)} || \ldots || h^{(\log(N))} || d$ and does the following. If $n \neq \lambda - 1$, simply append b to the end of h. Otherwise, let j^* be such that $N_j = 1$ for all $j < j^*$ and $N_{j^*} = 0$. Then, let $h'_{\log(N)} = d || b$ and repeatedly compute $h'_{\ell-1} = \mathsf{laconic.H}(\mathsf{crs}, h_\ell || h'_\ell)$ for $\log(N) \geq \ell \geq \log(N) - j^* + 1$. Finally, output $\lambda(N+1) || h^{(0)} || h^{(1)} || \ldots || h^{(\log(N)-j^*-1)} || h'^{(\log(N)-j^*)}$.
- $\mathsf{Append}^{\hat{D}}(\mathsf{crs}, b)$ computes the hash value h associated to the database D, executes $\mathsf{HashAppend}(\mathsf{crs}, h, b)$, and updates (i.e. merges) the corresponding Merkle trees $\hat{D}^{(0)}, \ldots, \hat{D}^{(\log(N))}$ as dictated by the new hashing.

The correctness and security properties of our new laconic OT scheme are immediately inherited from the correctness and security of $\mathsf{laconic}$. The only other property that needs to be verified is correctness of appending, which follows immediately from the fact that $\mathsf{laconic.H}$ is computed via a Merkle tree.

5.1 The LSGS Construction

Let $\mathsf{laconic} = (\mathsf{laconic.Gen}, \mathsf{laconic.H}, \mathsf{laconic.Send}, \mathsf{laconic.Receive}, \mathsf{laconic.HashAppend}, \mathsf{laconic.Append})$ be an appendable laconic OT scheme. Let $\Pi = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{Eval})$ be a decomposable garbling scheme (Section 2.2) with simulation security hiding both the circuit and input.[15]

We construct a locally simulatable garbling scheme Π_{LGC} for circuits of size s, input length n, and output length m by first specifying a **local encoding** (Definition 9) with $N_{\text{comps}} = s$

[13] Technically, this is only possible when we start with laconic OT satisfying *adaptive security*. The [CDG+17] construction can be modified to the setting of selective security using a Merkle tree with a new, independently sampled crs_j for each depth j of the tree, and the generic transformation of [CDG+17] can easily be adapted to this setting as well.

[14] We assume that $N \leq 2^\lambda$.

[15] By this, we mean that the simulator takes as input only $1^{|C|}$, n input labels $\overrightarrow{\mathsf{lab}}$, and the output $C(x)$. This definition can be achieved generically from the version stated in Section 2.2 by garbling a universal circuit with the particular circuit C hard-coded as input.

components and then proving that the associated garbling scheme satisfies **strong L_{sim}-local simulation security** (Definition 10) for $L_{\mathsf{sim}} = O(\log(s))$.

To give intuition for our local encoding, we first give an informal description of the local secret keys in the scheme. The local secret keys are as follows:

- The common key $sk_\perp$, which will be the common reference string of a laconic OT scheme.
- Local labels $sk_{i,\text{labels}}$ for every $i \in [s]$, which are input labels for instances of Π.
- Local one-time pads $sk_{i,\text{pad}}$ for every $i \in [s]$.

We accordingly identify the set $[N_{\text{keys}}]$ with the set $\{\perp\} \cup \{(i,\text{labels})\}_{i\in[s]} \cup \{(i,\text{pad})\}_{i\in[s]}$. Moreover, we define $\mathbf{I} := \{\perp, (1,\text{labels})\} \cup \{(i,\text{pad})\}_{i=1}^{n} \cup \{(i,\text{pad})\}_{i=s-m+1}^{s}$

We now formally define the local encoding for Π_{LGC}.

$\Pi_{\mathsf{LGC}}.\mathsf{KeyListGen}(C, i)$: On input a circuit C and index $i \in [N_{\text{comps}} = s]$, compute the gates $a, b \in [s]$ that are the children of the ith gate of C under some canonical ordering. Output $\mathbf{L}_i = \{\perp, (i,\text{labels}), (i,\text{pad}), (i+1,\text{labels}), (a,\text{pad}), (b,\text{pad})\}$ and $\mathsf{st}_i = (i, a, b)$.

Implementing $\Pi_{\mathsf{LGC}}.\mathsf{KeyListGen}$ for Turing Machines Recall that a local encoding (Definition 9) must satisfy the property that $\mathsf{KeyListGen}(M, T, n, i)$ can be computed in time $\text{poly}(|M|, \log(T), \log(n))$ when M is a Turing machine. Indeed, it is easy to see that $\mathsf{KeyListGen}(M, T, n, i)$ can be implemented via a single call to $\mathsf{FindGate}(M, n, T, i)$ (as defined in Section 2), which gives the result.

$\Pi_{\mathsf{LGC}}.\mathsf{LocalKG}(1^\lambda, j)$: There are three cases describing local key generation.

- If $j = \perp$, compute $\mathsf{crs} \leftarrow \mathsf{lacOT.Gen}(1^\lambda)$ and output $sk_\perp = \mathsf{crs}$.
- If $j = (i, \text{pad})$, sample a uniformly random bit r_i and output $sk_{i,\text{pad}} = r_i$.
- If $j = (i, \text{labels})$, sample two sets of input labels: $\overline{\mathsf{lab}}^{R,i} \leftarrow \Pi.\mathsf{Gen}(1^\lambda, 1^\lambda)$[16], and $\overline{\mathsf{lab}}^{W,i} := \left(\overline{\mathsf{hlab}}^{W,i}, \overline{\mathsf{alab}}^{W,i}, \overline{\mathsf{blab}}^{W,i}\right) \leftarrow \Pi.\mathsf{Gen}(1^\lambda, 1^{\lambda+2})$. Then output $sk_{i,\text{labels}} = \left(\overline{\mathsf{lab}}^{R,i}, \overline{\mathsf{lab}}^{W,i}\right)$.

$\Pi_{\mathsf{LGC}}.\mathsf{GateEnc}(\mathsf{st}, (sk_j)_{j\in\mathbf{L}})$:

- Parse $\mathsf{st} = (i, a, b)$.
- Parse the local secret keys $(sk_j)_{j\in\mathbf{L}}$ as follows: $sk_\perp = \mathsf{crs}$, $sk_{i,\text{labels}} = \left(\overline{\mathsf{lab}}^{R,i}, \overline{\mathsf{lab}}^{Wi}\right)$, $sk_{i,\text{pad}} = r_i$, $sk_{i+1,\text{labels}} = \left(\overline{\mathsf{lab}}^{R,i+1}, \overline{\mathsf{lab}}^{Wi+1}\right)$, $sk_{a,\text{pad}} = r_a$, $sk_{b,\text{pad}} = r_b$.
- Define circuits R and W as in Figure 2 and Figure 3, respectively.
- Compute $\widetilde{R}_i \leftarrow \Pi.\mathsf{Gb}\left(\Pi.\mathsf{MSK} := \overline{\mathsf{lab}}^{R,i}, R[\mathsf{crs}, \overline{\mathsf{lab}}^{W,i}, a, b]\right)$.
- Compute $\widetilde{W}_i \leftarrow \Pi.\mathsf{Gb}\left(\Pi.\mathsf{MSK} := \overline{\mathsf{lab}}^{W,i}, W[\mathsf{crs}, r_a, r_b, r_i, \overline{\mathsf{lab}}^{R,i+1}, i]\right)$.
- Output $\left(\widetilde{R}_i, \widetilde{W}_i\right)$.

$\Pi_{\mathsf{LGC}}.\mathsf{IEnc}(\mathsf{MSK}, x)$: Output $\left(\mathsf{crs}, \mathsf{lab}^{R,1}[h_0], D_0, (r_\ell)_{\ell=s-m+1}^{s}\right)$, where $(h_0, \hat{D}_0) = \mathsf{lacOT.H}(\mathsf{crs}, D_0)$ and $D_0 = (x_1 \oplus r_1)||\ldots||(x_n \oplus r_n)$. Note that this is a function of $\mathsf{SK}_{\text{Inp}} := (sk_j)_{j\in\mathbf{I}}$.

$\Pi_{\mathsf{LGC}}.\mathsf{Eval}(\langle C\rangle, \langle x\rangle)$: First, compute $(h_0, \hat{D}_0) = \mathsf{lacOT.H}(\mathsf{crs}, D_0)$ (in order to compute $\hat{D}_0$). Then, for each $i = (a, b) \in [s]$, do the following:

[16]Recall that $\Pi.\mathsf{Gen}$ is defined as follows: it takes as input $(1^\lambda, 1^n)$ and outputs $2n$ input labels, each of length λ.

Input: hash value h
Hardwired Values: keys $\mathsf{crs}, \overline{\mathsf{lab}}^{W,i}$; indices a, b

- Compute $\mathsf{CT}_a \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h, a, \mathsf{alab}_0^{W,i}, \mathsf{alab}_1^{W,i})$
- Compute $\mathsf{CT}_b \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h, b, \mathsf{blab}_0^{W,i}, \mathsf{blab}_1^{W,i})$.
- Output $\mathsf{hlab}^{W,i}[h], \mathsf{CT}_a, \mathsf{CT}_b$.

Figure 2: Description of the Read Circuit R.

Input: hash value h, bits v_a, v_b
Hardwired Values: keys $\mathsf{crs}, r_a, r_b, r_i, \overline{\mathsf{lab}}^{R,i+1}$, index i

- Compute $v_i := r_i \oplus ((v_a \oplus r_a)\ \mathsf{NAND}\ (v_b \oplus r_b))$.
- Compute $h' = \mathsf{lacOT.HashAppend}(\mathsf{crs}, h, v_i)$.
- Output $v_i, \mathsf{lab}^{R,i+1}[h']$.

Figure 3: Description of the Write Circuit W.

- Compute $\left(\mathsf{hlab}^{W,i}[h_{i-1}], \mathsf{CT}_a, \mathsf{CT}_b\right) = \Pi.\mathsf{Eval}(\widetilde{R}_i, \mathsf{lab}^{R,i}[h_{i-1}])$.
- Compute $\mathsf{alab}_{v_a}^{W,i} = \mathsf{lacOT.Receive}^{\hat{D}_{i-1}}(\mathsf{crs}, \mathsf{CT}_a, a)$ and $\mathsf{blab}_{v_b}^{W,i} = \mathsf{lacOT.Receive}^{\hat{D}_{i-1}}(\mathsf{crs}, \mathsf{CT}_b, b)$.
- Compute $v_i || \mathsf{lab}^{R,i+1}[h_i] = \Pi.\mathsf{Eval}(\widetilde{W}_i, \mathsf{lab}^{W,i}[h_{i-1}||v_a||v_b])$.
- Execute $\mathsf{lacOT.Append}^{\hat{D}_{i-1}}(\mathsf{crs}, v_i)$ and obtain updated database state $\hat{D}_i$.

Finally, output $(r_j \oplus D_s[j])_{j=s-|y|}^{s}$.

Correctness. This garbling scheme is proven correct by an inductive argument that for each $i \in [s-n]$, $(h_i, \hat{D}_i) = \mathsf{lacOT.H}(\mathsf{crs}, D_i)$, where D_i is the database of length $i+n$ satisfying $D_i[\ell] = r_\ell \oplus \mathrm{val}_\ell(C, x)$ for all $\ell \in [i+n]$, and $\mathrm{val}_\ell(C, x)$ is the value of the ℓth gate of C when evaluated on input x.

We argue as follows. By definition $(h_0, \hat{D}_0) = \mathsf{lacOT.H}(\mathsf{crs}, D_0)$. For the inductive step, suppose that $(h_{i-1}, \hat{D}_{i-1}) = \mathsf{lacOT.H}(\mathsf{crs}, D_{i-1})$. Then, by the correctness of Π, we have that the $(2i-1)$th execution of $\Pi.\mathsf{Eval}$ outputs $\left(\mathsf{hlab}^{W,i}[h_{i-1}], \mathsf{CT}_a, \mathsf{CT}_b\right)$ (where CT_a and CT_b are lacOT ciphertexts using hash value h_{i-1}). By the correctness of lacOT, we then have that the ith pair of executions of $\mathsf{lacOT.Receive}$ output $\mathsf{alab}_{v_a}^{W,i}$ and $\mathsf{blab}_{v_b}^{W,i}$. Thus, by the correctness of Π, the $2i$th execution of $\Pi.\mathsf{Eval}$ outputs (h_i, v_i), where $(h_i, \hat{D}_i) = \mathsf{lacOT.H}(\mathsf{crs}, D_i)$ by the correctness of $\mathsf{lacOT.HashAppend}$. Finally, we conclude that the ith execution of $\mathsf{lacOT.Append}$ correctly outputs $\hat{D}_i$ by the correctness of $\mathsf{lacOT.Append}$. This completes the induction.

Online Complexity. We note that by construction, the size of an input encoding of $x \in \{0,1\}^n$ is $n + m + \text{poly}(\lambda)$.[17] In addition, the online computational complexity of this scheme is $n \cdot \text{poly}(\lambda) + m$; the only work done in the execution of $\Pi_{\mathsf{LGC}}.\mathsf{IEnc}$ is hashing the input x. More precisely, the online computational complexity is $\frac{n}{\lambda} \cdot \mathsf{Time}(\mathsf{lacOT.H}) + m + \text{poly}(\lambda)$, so if $\mathsf{lacOT.H}$ runs in *linear* time (as is the case for the CDH-based laconic OT of [DG17]), then the online computational complexity is $O(n) + m + \text{poly}(\lambda)$.

5.2 Strong Local Simulation

We now write down the associated $\mathsf{SimKeyListGen}_{\mathbf{k}}$, $\mathsf{SimGtEnc}_{\mathbf{k}}$ and $\mathsf{SimInpEnc}_{\mathbf{k}}$ algorithms for $\mathbf{k} \in [q = \text{poly}(\lambda, s)]$ that satisfy strong L_{sim}-local simulation security for $L_{\mathsf{sim}} = \log(s)$. Our proof of security follows that of [GS18a]. In order to prove *local simulatability*, we will define three local simulator algorithms – $\mathsf{SimGtEnc}_{\text{White}}$, $\mathsf{SimGtEnc}_{\text{Grey}}$, and $\mathsf{SimGtEnc}_{\text{Black}}$ – and then define $(\mathsf{SimKeyListGen}_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}})$ where each hybrid $\mathbf{k}$ corresponds to a "pebble configuration" $\mathsf{mode}^{(\mathbf{k})} \in \{\text{White}, \text{Grey}, \text{Black}\}^s$. Our simulators are defined as follows.

$\mathsf{SimGtEnc}_{\text{White}}(i, a, b, \mathsf{crs}, \overline{\mathsf{lab}}^{R,i}, \overline{\mathsf{lab}}^{W,i}, r_i, \overline{\mathsf{lab}}^{R,i+1}, r_a, r_b)$: This algorithm is identical to the honest $\mathsf{GateEnc}$ algorithm (with unused inputs removed for convenience).

$\mathsf{SimGtEnc}_{\text{Grey}}(\mathsf{MSK}, i, a, b, h_{i-1}, h_i, v_a, v_b, v_i)$: This algorithm controls our input-dependent simulation; it takes as additional input hash values (h_{i-1}, h_i) and bits (v_a, v_b, v_i). It operates as follows.

- Compute ciphertexts $\mathsf{CT}_a \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h_{i-1}, a, \left(\mathsf{alab}_{v_a}^{W,i}, \mathsf{alab}_{v_a}^{W,i}\right))$, $\mathsf{CT}_b \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h_{i-1}, b, \left(\mathsf{blab}_{v_b}^{W,i}, \mathsf{blab}_{v_b}^{W,i}\right))$.
- Compute simulated garbled circuits $\widetilde{R}_i \leftarrow \Pi.\mathsf{Sim}\left(1^{|R|}, \mathsf{lab}^{R,i}[h_{i-1}], \left(\mathsf{hlab}^{W,i}[h_{i-1}], \mathsf{CT}_a, \mathsf{CT}_b\right)\right)$ as well as $\widetilde{W}_i \leftarrow \Pi.\mathsf{Sim}\left(1^{|W|}, \mathsf{lab}^{W,i}[h_{i-1}||v_a||v_b], v_i||\mathsf{lab}^{R,i+1}[h_i]\right)$.
- Output $\left(\widetilde{R}_i, \widetilde{W}_i\right)$.

$\mathsf{SimGtEnc}_{\text{Black}}(i, a, b, \mathsf{crs}, \overline{\mathsf{lab}}^{R,i}, \overline{\mathsf{lab}}^{W,i}, r_i, \overline{\mathsf{lab}}^{R,i+1})$: This algorithm crucially **does not require r_a or r_b as input**. It computes $\widetilde{R}_i$ exactly as in $\mathsf{GateEnc}$, and computes $\widetilde{W}'_i \leftarrow \Pi.\mathsf{Gb}(\mathsf{MSK} := \overline{\mathsf{lab}}^{W,i}, W'[\mathsf{crs}, r_i, \mathsf{lab}^{R,i+1}, i])$, where the **fake write circuit** W' is defined in Fig. 4. Finally, it outputs $\left(\widetilde{R}_i, \widetilde{W}'_i\right)$.

We now define $(\mathsf{SimKeyListGen}_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}})$, where each $\mathbf{k}$ is associated to a configuration $\mathsf{mode}^{(\mathbf{k})} \in \{\text{white}, \text{grey}, \text{black}\}^s$. The configuration $\mathsf{mode}^{(\mathbf{k})}$ is specified by the pebbling game defined in [GS18a], which we will describe later.

The set $T_{\mathbf{k}}$ is defined to be the set of all i such that $\mathsf{mode}_i^{(\mathbf{k})} = \text{Grey}$.

$\mathsf{SimKeyListGen}_{\mathbf{k}}(C, i)$ outputs $\mathbf{L}_i^{(\mathbf{k})} = \{\perp, (i, \text{labels}), (i, \text{pad}), (i+1, \text{labels}), (a, \text{pad}), (b, \text{pad})\}$ – as computed by $\mathsf{KeyListGen}(C, i)$ – if $\mathsf{mode}_i^{(\mathbf{k})} \in \{\text{White}, \text{Grey}\}$, and outputs $\mathbf{L}_i^{(\mathbf{k})} = \{\perp, (i, \text{labels}), (i, \text{pad}), (i+1, \text{labels})\}$ if $\mathsf{mode}^{(\mathbf{k})} = \text{Black}$. It also always outputs $\mathsf{st}_i^{(\mathbf{k})} = (i, a, b)$.

[17] If we only care about obtaining strong local simulatability and not semi-adaptive security, the output dependence can be moved to the circuit/TM encoding instead of the input encoding.

Input: hash value h, bits v_a, v_b
Hardwired Values: index i, keys $\mathsf{crs}, r_i, \overline{\mathsf{lab}}^{R,i+1}$

- Set $v_i := r_i$.
- Compute $h' = \mathsf{lacOT.HashAppend}(\mathsf{crs}, h, v_i)$.
- Output $v_i, \mathsf{lab}^{R,i+1}[h']$.

Figure 4: Description of the Fake Write Circuit W'.

$\underline{\mathsf{SimGtEnc}_{\mathbf{k}}}$: To define this algorithm, we must describe both the input-dependent and input-independent simulation.

The input-dependent simulation algorithm receives as input (MSK, i, C, x) and does the following:

- Compute the children (a, b) of i in C (as in $\mathsf{KeyListGen}$).
- For every gate g, set v_g to be $r_g \oplus \mathrm{val}_g(C, x)$, where $\mathrm{val}_g(C, x)$ is the value of the gate g of C on input x.
- Compute $h_{i-1} = \mathsf{lacOT.H}(\mathsf{crs}, D_{i-1})$, where $D_{i-1} = v_1 || \ldots || v_{i-1}$, and similarly compute $h_i = \mathsf{lacOT.H}(\mathsf{crs}, D_i)$
- Call $\mathsf{SimGtEnc}_{\mathrm{Grey}}(\mathsf{MSK}, i, a, b, h_{i-1}, h_i, v_a, v_b, v_i)$ for $i = (a, b)$.

The input-independent simulation algorithm receives as input $(\mathsf{st}, (sk_j)_{j \in \mathbf{L}}, C(x))$ and calls $\mathsf{SimGtEnc}_{\mathsf{mode}_i^{(\mathbf{k})}}(\mathsf{st}, (sk_j))$.[18]

$\underline{\mathsf{SimInpEnc}_{\mathbf{k}}(\mathsf{MSK}, C, x)}$: For all output gates $\ell > s - m$, set $r'_\ell = r_\ell \oplus C(x)_{\ell - s + m}$ if $\mathsf{mode}_\ell^{(\mathbf{k})} =$ black; otherwise, set $r'_\ell = r_\ell$. If $\mathsf{mode}_g^{(\mathbf{k})} = \mathrm{Black}$ for every $g > n$, then set $v_i = r_i$ for all $i \in [n]$; otherwise, set $v_i = r_i \oplus x_i$ for all $i \in [n]$. Output $\left(\mathsf{crs}, \mathsf{lab}^{R,1}[h_0], (v_i)_{i=1}^{|x|}, (r'_j)_{j=s-|y|+1}^{s}\right)$, where $h_0 = \mathsf{lacOT.H}(\mathsf{crs}, v_1 || \ldots || v_n)$.

To finish our description of the local simulation strategy, we must describe the configurations $\mathsf{mode}^{(\mathbf{k})} \in \{\mathrm{White}, \mathrm{Grey}, \mathrm{Black}\}^s$. We say that a pair of modes $(\mathsf{mode}, \mathsf{mode}')$ is a **valid transition** if mode and mode' differ on a single index $i \in [s]$, and one of the following two conditions holds.

- **Grey Rule:** $\{\mathsf{mode}_i, \mathsf{mode}'_i\} = \{\mathrm{Grey}, \mathrm{White}\}$ and $\mathsf{mode}_{i-1} = \mathrm{Grey}$ (or $i = 1$).
- **Black Rule:** $\{\mathsf{mode}_i, \mathsf{mode}'_i\} = \{\mathrm{Grey}, \mathrm{Black}\}$, $\mathsf{mode}_{i-1} = \mathrm{Grey}$ (or $i = 1$), and $\mathsf{mode}_\ell = \mathrm{Black}$ for all $\ell > i$.

Our sequence of modes $\mathsf{mode}^{(1)}, \ldots, \mathsf{mode}^{(q)}$ is then defined by a *pebbling strategy.*

Lemma 1 ([GS18a] Lemma 5.2)**.** *There exists a sequence of modes* $(\mathsf{mode}^{(1)}, \ldots, \mathsf{mode}^{(q)})$ *with* $q = \mathrm{poly}(s)$ *satisfying the following properties.*

- $\mathsf{mode}^{(1)}$ *is the "all* White*" mode.*
- $\mathsf{mode}^{(q)}$ *is the "all* Black*" mode.*

[18] The gate number i is always part of st, so this is well-defined and efficient.

- $(\mathsf{mode}^{(\mathbf{k})}, \mathsf{mode}^{(\mathbf{k}+1)})$ *is a valid transition for each* $\mathbf{k}$.
- *For every* $\mathbf{k}$*, at most* $\log(s)$ *indices* i *satisfy* $\mathsf{mode}_i^{(\mathbf{k})} = \text{Grey}$.

This completes our description of the locally simulatable garbling scheme and its simulation strategy. We first observe that the syntactic properties of strong local simulation are satisfied:

- **The final simulator is input-independent.** Namely, $T_q = \emptyset$ and $\mathsf{SimInpEnc}_q$ is an efficient function of $(C, C(x))$ rather than (C, x) because $\mathsf{mode}_i^{(q)} = \text{Black}$ for all i.
- **Neighbor lists and states are small and efficiently computable.** In particular, $\left|\mathbf{L}_i^{(\mathbf{k})}\right| \leq 6$ and $\left|\mathsf{st}_i^{(\mathbf{k})}\right| = 3\log(s)$ for all $(i, \mathbf{k})$.
- **Input-dependent simulation is local.** By Lemma 1, $|T_{\mathbf{k}}| \leq \log(s)$ for every $\mathbf{k}$.
- **Input-independent hybrid simulators are functionally equivalent.** Indeed, the input-independent simulation algorithms and simulated neighbor algorithms in hybrid $\mathbf{k}$ and hybrid $\mathbf{k}+1$ are functionally equivalent, as on input i they are a fixed function of $\mathsf{mode}_i^{(\mathbf{k})} = \mathsf{mode}_i^{(\mathbf{k}+1)}$.

Thus, all of the auxiliary requirements (as per Definition 10) of our simulation strategy are satisfied, so all that remains to be verified is the indistinguishability condition. Recall the indistinguishability condition: for every C, x, $\mathbf{k}$, and adversarially chosen local secret keys $\{sk_j\}_{j \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}}$, we want to show that $D_0(\mathsf{MSK}) \approx_c D_1(\mathsf{MSK})$, where $D_b(\mathsf{MSK})$ is sampled in the following way:

- For every $i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$, sample $\langle g_i \rangle$ in one of two ways:
  - If $i \in T_{\mathbf{k}+b}$, compute $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{MSK}, i, C, x)$.
  - If $i \notin T_{\mathbf{k}+b}$, compute $\langle g_i \rangle \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+b}(\mathsf{st}_i^{(\mathbf{k}+b)}, (sk_j)_{j \in \mathbf{L}_i^{(\mathbf{k}+b)}}, C(x))$, where $(\mathsf{st}_i^{(\mathbf{k}+b)}, \mathbf{L}_i^{(\mathbf{k}+b)}) = \mathsf{SimKeyListGen}_{\mathbf{k}+b}(C, i)$.
- Sample $\langle x \rangle \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}+b}(\mathsf{MSK}, C, x)$.
- Output $\left(\left\{\langle g_i \rangle\right\}_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}, \langle x \rangle, \left\{sk_j\right\}_{j \in \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}}\right)$.

We are guaranteed by Lemma 1 that $(\mathsf{mode}^{(\mathbf{k})}, \mathsf{mode}^{(\mathbf{k}+1)})$ is a valid transition for every $\mathbf{k}$. Let i^* be the unique index for which $\mathsf{mode}_{i^*}^{(\mathbf{k})} \neq \mathsf{mode}_{i^*}^{(\mathbf{k}+1)}$. We will prove the indistinguishability condition by considering two cases.

Case 1: $(\mathsf{mode}^{(\mathbf{k})}, \mathsf{mode}^{(\mathbf{k}+1)})$ **satisfies the "Grey Rule."** This case follows from the argument [GS18a] Lemma 5.3, with the caveat of keeping track of which secret keys are random vs. adversarially generated, and which keys are given to the adversary. In addition, because we are only proving a form of selective security, we can rely on the selective security of the underlying Laconic OT scheme (along with the simulation security of the underlying decomposable garbling scheme).

We sketch a sequence of (sub)hybrids proving indistinguishability in this case. We assume that $i^* \neq 1$ (the $i^* = 1$ case follows similarly), and we assume that $\mathsf{mode}_{i^*}^{(\mathbf{k})} = \text{White}$ (the other case is symmetric). In each hybrid, we only change how the i^*th gate encoding is sampled.

- Hyb_0 : This is the original distribution $D_0(\mathsf{MSK}) = \left\{\left(\left\{\langle g_i \rangle^{(\mathbf{k})}\right\}_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}, \langle x \rangle^{(\mathbf{k})}, \{sk_j\}_{j \in \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}}\right)\right\}$.
- Hyb_1 : This matches the original distribution, except that the first half of $\langle g_{i^*} \rangle$ (the garbled "Read" circuit) is instead generated by computing ciphertexts $\mathsf{CT}_a \leftarrow \mathsf{lacOT.Send}\left(\mathsf{crs}, h_{i^*-1}, a, \overline{\mathsf{alab}}^{W,i^*}\right)$,

$\mathsf{CT}_b \leftarrow \mathsf{lacOT.Send}\left(\mathsf{crs}, h_{i^*-1}, b, \overline{\mathsf{blab}}^{W,i^*}\right)$, and sampling

$$\widetilde{R}_{i^*} \leftarrow \Pi.\mathsf{Sim}\Big(1^{|R|}, \mathsf{lab}^{R,i^*}[h_{i^*-1}], \Big(\mathsf{hlab}^{W,i^*}[h_{i^*-1}], \mathsf{CT}_a, \mathsf{CT}_b\Big)\Big).$$

Indistinguishability between Hyb_0 and Hyb_1 follows from the simulation security of Π, where we make use of the fact that $(i^*, \text{labels}) \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$ and $(i^*, \text{labels}) \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}$ (the latter is true because the previous gate is in "Grey" mode), so the labels $\overline{\mathsf{lab}}^{R,i^*}[1^\lambda \oplus h_{i^*-1}]$ do not appear elsewhere in the hybrid random variables (again because the previous gate is in "Grey" mode).

- Hyb_2 : This matches Hyb_1, except that the ciphertexts CT_a and CT_b are instead sampled as $\mathsf{CT}_a \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h_{i^*-1}, a, \left(\mathsf{alab}_{v_a}^{W,i^*}, \mathsf{alab}_{v_a}^{W,i^*}\right))$, $\mathsf{CT}_b \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h_{i^*-1}, b, \left(\mathsf{blab}_{v_b}^{W,i^*}, \mathsf{blab}_{v_b}^{W,i^*}\right))$.
  Indistinguishability between Hyb_1 and Hyb_2 follows from the sender privacy of lacOT. This follows from the fact that crs is sampled uniformly at random (i.e., $\perp \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$) – all other parameters may be chosen adversarially (and publicly) in the reduction to $\mathsf{lacOT.Send}$ security (as long as the database D is chosen independently of crs, which is the case here).
- Hyb_3: This is the distribution $D_1(\mathsf{MSK}) = \left\{\left(\left\{\langle g_i\rangle^{(\mathbf{k}+1)}\right\}_{i\in T_{\mathbf{k}}\cup T_{\mathbf{k}+1}}, \langle x\rangle^{(\mathbf{k}+1)}, \{sk_j\}_{j\in\mathsf{list}_{\mathbf{k}}^{\mathsf{public}}}\right)\right\}$.
  More specifically, this matches Hyb_2, except that the second half of $\langle g_{i^*}\rangle$ (the garbled "Write" circuit) is instead generated by sampling $\widetilde{W}_{i^*} \leftarrow \Pi.\mathsf{Sim}\left(1^{|W|}, \mathsf{lab}^{W,i^*}[h_{i^*-1}], h_{i^*}||\mathsf{lab}^{R,i^*+1}[h_{i^*}]\right)$.
  Indistinguishability between Hyb_2 and Hyb_3 follows from the simulation security of Π, where we make use of the fact that that $(i^*, \text{labels}) \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$ and $(i^*, \text{labels}) \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}$ (the latter is true because the previous gate is in "Grey" mode), so the labels $\overline{\mathsf{lab}}^{W,i^*}[1^\lambda \oplus (h_{i^*-1}||v_a||v_b)]$ do not appear elsewhere in the hybrid random variables (this additionally uses the fact that $\widetilde{R}_{i^*}$ is already simulated).

Case 2: $(\mathsf{mode}^{(\mathbf{k})}, \mathsf{mode}^{(\mathbf{k}+1)})$ satisfies the "Black Rule." This case follows from the argument [GS18a] Lemma 5.4, with the caveat of keeping track of which secret keys are random vs. adversarially generated, and which keys are given to the adversary.

We describe a sequence of (sub)hybrids proving indistinguishability in this case. We assume that $i^* \neq 1$ (the $i^* = 1$ case follows similarly), and we assume that $\mathsf{mode}_{i^*}^{(\mathbf{k})} = \text{Black}$ (the other case is symmetric). In each hybrid, we only change how the i^*th gate encoding is sampled (and possibly the input encoding).

- Hyb_0 : This is the original distribution $D_0(\mathsf{MSK}) = \left\{\left(\left\{\langle g_i\rangle^{(\mathbf{k})}\right\}_{i\in T_{\mathbf{k}}\cup T_{\mathbf{k}+1}}, \langle x\rangle^{(\mathbf{k})}, \{sk_j\}_{j\in\mathsf{list}_{\mathbf{k}}^{\mathsf{public}}}\right)\right\}$.
- Hyb_1 : This matches the original distribution, except that the first half of $\langle g_{i^*}\rangle$ is instead generated by computing ciphertexts $\mathsf{CT}_a \leftarrow \mathsf{lacOT.Send}\left(\mathsf{crs}, h_{i^*-1}, a, \overline{\mathsf{alab}}^{W,i^*}\right)$, $\mathsf{CT}_b \leftarrow \mathsf{lacOT.Send}\left(\mathsf{crs}, h_{i^*-1}, b, \overline{\mathsf{blab}}^{W,i^*}\right)$, and sampling

$$\widetilde{R}_{i^*} \leftarrow \Pi.\mathsf{Sim}\Big(1^{|R|}, \mathsf{lab}^{R,i^*}[h_{i^*-1}], \Big(\mathsf{hlab}^{W,i^*}[h_{i^*-1}], \mathsf{CT}_a, \mathsf{CT}_b\Big)\Big).$$

Indistinguishability between Hyb_0 and Hyb_1 follows from the simulation security of Π, where we make use of the fact that $(i^*, \text{labels}) \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$ and $(i^*, \text{labels}) \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}$ (the latter is true because the previous gate is in "Grey" mode), so the labels $\overline{\mathsf{lab}}^{R,i^*}[1^\lambda \oplus h_{i^*-1}]$ do not appear elsewhere in the hybrid random variables (again because the previous gate is in "Grey" mode).

- Hyb_2 : This matches Hyb_1, except that the ciphertexts CT_a and CT_b are instead sampled as $\mathsf{CT}_a \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h_{i^*-1}, a, \left(\mathsf{alab}_{v_a}^{W,i^*}, \mathsf{alab}_{v_a}^{W,i^*}\right))$, $\mathsf{CT}_b \leftarrow \mathsf{lacOT.Send}(\mathsf{crs}, h_{i^*-1}, b, \left(\mathsf{blab}_{v_b}^{W,i^*}, \mathsf{blab}_{v_b}^{W,i^*}\right))$.
  Indistinguishability between Hyb_1 and Hyb_2 follows from the sender privacy of lacOT. This follows from the fact that crs is sampled uniformly at random (i.e., $\perp \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$) – all other parameters may be chosen adversarially (and publicly) in the reduction to $\mathsf{lacOT.Send}$ security.
- Hyb_3: This matches Hyb_2, except that the second half of $\langle g_{i^*} \rangle$ is instead generated by sampling $\widetilde{W}_{i^*} \leftarrow \Pi.\mathsf{Sim}\left(1^{|W|}, \mathsf{lab}^{W,i^*}[h_{i^*-1}], v_{i^*} || \mathsf{lab}^{R,i^*+1}[h_{i^*}]\right)$[19].
  Indistinguishability between Hyb_2 and Hyb_3 follows from the simulation security of Π, where we make use of the fact that that $(i^*, \text{labels}) \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$ and $(i^*, \text{labels}) \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}$ (the latter is true because the previous gate is in "Grey" mode), so the labels $\overline{\mathsf{lab}}^{W,i^*}[1^\lambda \oplus (h_{i^*-1}||v_a||v_b)]$ do not appear elsewhere in the hybrid random variables (this additionally uses the fact that $\widetilde{R}_{i^*}$ is already simulated).
- Hyb_4: This is the distribution $D_1(\mathsf{MSK}) = \left\{\left(\left\{\langle g_i \rangle^{(\mathbf{k}+1)}\right\}_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}, \langle x \rangle^{(\mathbf{k}+1)}, \{sk_j\}_{j \in \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}}\right)\right\}$.
  More specifically, the difference between Hyb_3 and Hyb_4 is that in Hyb_3, we define $v_{i^*} = r_{i^*}$ while in Hyb_4, we define $v_{i^*} = r_{i^*} \oplus \mathrm{val}_{i^*}(C, x)$, and if i^* is an output gate, then one bit of $\langle x \rangle$ changes from $r'_{i^*} = r_{i^*} \oplus \mathrm{val}_{i^*}(C, x)$ to $r'_{i^*} = r_{i^*}$.
  We claim that Hyb_3 and Hyb_4 produce *identical* distributions. This follows from the fact that $(i^*, \text{pad}) \in \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$ and $(i^*, \text{pad}) \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}$; the latter fact crucially requires that all gates $g > i^*$ are already in Black mode, so that (i^*, pad) is no longer in the neighbor list for any other gate.

Having analyzed the two cases, we have completed the proof of strong local simulation security.

5.3 Semi-Adaptive Local Simulation Security

In the previous section, we proved that the scheme Π_{LGC} satisfies strong local simulation security. It also satisfies semi-adaptive local simulation security (with the same L_{inp} and L_{sim}) assuming that the underlying laconic OT scheme is adaptively secure. This follows by the same hybrid arguments desribed in the previous section, with the following minor differences:

- We make use of the fact that $\mathsf{SimGtEnc}_{\mathrm{White}}$ and $\mathsf{SimGtEnc}_{\mathrm{Black}}$ *do not* require the output $C(x)$ as input (so that input-independent simulated gates can be provided in the semi-adaptive security game).
- The indistinguishability of Hyb_1 and Hyb_2 (in both of the cases above) follows by a reduction from the adaptive security of the laconic OT scheme; this is necessary because in the semi-adaptive security game, the input x (and hence the database D) can depend on the input-independent simulated gates (and hence on the crs of the laconic OT scheme).

6 Succinct Garbling from IO through Locally Simulatable Garbling

In this section, we prove Theorem 9; that is, we construct a succinct garbling scheme (Definition 3) from indistinguishability obfuscation and a strong locally simulatable garbling scheme (Definition 10).

[19] Recall that $v_{i^*} = r_{i^*}$ is not sampled honestly at this point (since we started in Black mode), but v_g is sampled honestly for every $g < i^*$.

Input: index $i \leq T$
Hardwired Values: TM M, input length n, time bound T, puncturable PRF keys K_1, K_2.

- Compute $\mathsf{KeyListGen}(M, T, n, i)$ to obtain $(\mathbf{L}_i, \mathsf{st}_i)$.
- For every $j \in \mathbf{L}_i$, compute $sk_j \leftarrow \mathsf{LocalKG}(1^\lambda, j; \mathsf{PRF}(K_1, j))$.
- Compute $\langle g_i \rangle \leftarrow \mathsf{GateEnc}\left(\mathsf{st}_i, \{sk_j\}_{j \in \mathbf{L}_i}; \mathsf{PRF}(K_2, i)\right)$.
- Output $\langle g_i \rangle$.

Figure 5: Description of H.

To construct a succinct garbling scheme, we first construct the weaker notion of bounded runtime succinct garbling scheme (Definition 4) and then show how to generically transform it into a succinct garbling scheme without any restriction on the upper bound on the runtime of the Turing machines.

6.1 Bounded Runtime Case

We show how to construct a bounded runtime succinct garbling scheme from a $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-strong locally simulatable garbling scheme $\Pi_{\mathsf{LGC}} = (\Pi_{\mathsf{LGC}}.\mathsf{Gen}, \Pi_{\mathsf{LGC}}.\mathsf{Gb}, \Pi_{\mathsf{LGC}}.\mathsf{IEnc}, \Pi_{\mathsf{LGC}}.\mathsf{Eval})$ with local encoding algorithms $(\mathsf{KeyListGen}, \mathsf{LocalKG}, \mathsf{GateEnc})$. The size of the succinct garbling depends on the parameters L_{sim} and L_{inp}.

We use the following two additional tools in this transformation.

- A puncturable pseudorandom function family PRF.
- An indistinguishability obfuscator iO for polynomial-sized circuits of input length $\log(T)$ and size $\mathrm{poly}(\lambda, \log(T))$, where T is a time bound on the running time of the Turing machines.

We denote the bounded runtime succinct garbling scheme by $\mathsf{SuccGC} = (\mathsf{Gen}, \mathsf{TMEncode}, \mathsf{InpEncode}, \mathsf{Eval})$ for a set $\mathcal{M}$.

$\underline{\mathsf{Setup}(1^\lambda, T, 1^n, 1^m)}$: On input security parameter λ, time bound T, input n and output length m, sample a pseudorandom function key K_1. Output $\mathsf{MSK} = (K_1, m, n)$.

$\underline{\mathsf{TMEncode}(\mathsf{MSK}, T, M)}$: On input master secret key MSK, time bound T, and Turing machine M, sample a PRF key K_2. Compute $\widetilde{H} \leftarrow \mathsf{iO}(1^\lambda, H)$, where H is defined in Figure 5. The circuit H will be suitably padded, as is standard in iO constructions, such that the circuits H, $H_{\mathbf{k}}$ (Figure 6) and $\mathsf{Hyb}.H_{\mathbf{k}}$ (Figure 7) have the same sizes, for every $\mathbf{k} \in [q]$. Set the TM encoding $\langle M \rangle$ to be $(T, \widetilde{H})$.

$\underline{\mathsf{InpEncode}(\mathsf{MSK}, T, x)}$: On input master secret key MSK, time bound T and input x, compute $sk_j \leftarrow \mathsf{LocalKG}(1^\lambda, j; \mathsf{PRF}(K_1, j))$ for every $j \in [\mathbf{I}]$, where $\mathbf{I}$ is the set associated with $\Pi_{\mathsf{LGC}}.\mathsf{IEnc}$ and T, m (see Definition 9). Set $\mathsf{SK}_{\mathrm{Inp}} = (sk_j)_{j \in [\mathbf{I}]}$. Output $\langle x \rangle \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{IEnc}\left(\mathsf{SK}_{\mathrm{Inp}}, x\right)$.

$\underline{\mathsf{Eval}(\langle M\rangle, \langle x\rangle)}$: It takes as input TM encoding $\langle M\rangle$ and input encoding $\langle x\rangle$. Parse $\langle M\rangle = (T, \widetilde{H})$. For every $i \in [T]$, compute $\langle g_i\rangle = \widetilde{H}(i)$. Output $y := \Pi_{\mathsf{LGC}}.\mathsf{Eval}(\langle C\rangle, \langle x\rangle)$, where $\langle C\rangle = (\langle g_1\rangle, \ldots, \langle g_T\rangle)$.

Correctness. Consider input x and $M \in \mathcal{M}$. Let C be the output of $\mathsf{CktTransform}(M, |x|, T)$ (defined in Section 2). Let $\langle M\rangle \leftarrow \mathsf{TMEncode}(\mathsf{MSK}, T, M)$ and $\langle x\rangle \leftarrow \mathsf{InpEncode}(\mathsf{MSK}, T, x)$, where $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda)$. As in Eval, parse $\langle M\rangle$ as $(T, \widetilde{H})$.

Observe that $(\widetilde{H}(1), \ldots, \widetilde{H}(T))$ is computationally indistinguishable from $(\langle g_1\rangle, \ldots, \langle g_T\rangle) \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gb}(\mathsf{MSK}, C)$ by the security of pseudorandom functions. This means that the output distributions of $\mathsf{Eval}(\langle M\rangle, \langle x\rangle)$ and $\Pi_{\mathsf{LGC}}.\mathsf{Eval}((\langle g_1\rangle, \ldots, \langle g_T\rangle), \langle x\rangle)$ are computationally indistinguishable. By the correctness and local encoding property of Π_{LGC}, we have that $\Pi_{\mathsf{LGC}}.\mathsf{Eval}((\langle g_1\rangle, \ldots, \langle g_T\rangle), \langle x\rangle) = C(x)$ (which is nothing but $M(x)$) with overwhelming probability. Thus, with overwhelming probability, we have $\mathsf{Eval}(\langle M\rangle, \langle x\rangle) = M(x)$.

If Π_{LGC} satisfied perfect correctness then SuccGC will (unconditionally) satisfy perfect correctness.

Efficiency. Suppose that Π_{LGC} is a $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-strong locally simulatable garbling scheme, and in particular has a $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-local encoding algorithm. Then, our scheme SuccGC has online computational complexity $L_{\mathsf{inp}}(\lambda, |x|, m) \cdot \mathrm{poly}(\lambda)$[20].

The time to compute $\widetilde{H}$ is poly $(\lambda, \max\{|H|, |H_{\mathbf{k}}|, |\mathsf{Hyb}.H_{\mathbf{k}}|\})$. We calculate $|H|, |H_{\mathbf{k}}|, |\mathsf{Hyb}.H_{\mathbf{k}}|$ below.

- From Definition 9, $\mathsf{KeyListGen}$ runs in time at most $\mathrm{poly}(|M|, \log(T))$. $\mathsf{LocalKG}$ runs in time polynomial in the size of its inputs, i.e., $\mathrm{poly}(\lambda, \log(\mathrm{poly}(T)))$. The input to $\mathsf{GateEnc}$ is $|\mathbf{L}|$ number of secret keys, each of size $\mathrm{poly}(\lambda)$, where $|\mathbf{L}|$ is the length of output of $\mathsf{KeyListGen}(M, i)$, for any $i \le T$. From Definition 10, we have that $|\mathbf{L}|$ is upper bounded by $\mathrm{poly}(\lambda, \log(T))$. Thus, $\mathsf{GateEnc}$ can be computed in time at most $\mathrm{poly}(\lambda, |M|, \log(T))$. Thus, the size of H is at most $\mathrm{poly}(\lambda, |M|, \log(T))$.
- From Definition 10, we have $|T_{\mathbf{k}}| \le L_{\mathsf{sim}}$ and thus, membership in $T_{\mathbf{k}}$ can be decided by a circuit of size at most $\mathrm{poly}(\lambda, L_{\mathsf{sim}})$. We have that $\mathsf{SimKeyListGen}_{\mathbf{k}}(\cdot)$ runs in time at most polynomial in $(|M|, \log(T))$. As before, $\mathsf{LocalKG}$ can be computed in time at most $\mathrm{poly}(\lambda, \log(T))$. Note that the input-independent simulator $\mathsf{SimGateEnc}_{\mathbf{k}}$ runs in time polynomial in $\lambda, |\mathsf{st}|, |\mathbf{L}|$. From Definition 10, we have both $|\mathsf{st}|$ and $|\mathbf{L}|$ upper bounded by $\mathrm{poly}(\lambda, \log(T))$. Hence, $\mathsf{SimGateEnc}$ runs in time at most $\mathrm{poly}(\lambda, \log(T))$. Using these calculations, we can upper bound $|H_{\mathbf{k}}|$ to be $\mathrm{poly}(\lambda, L_{\mathsf{sim}}, |M|, \log(T))$.
- From Definition 10, we have $|T_{\mathbf{k}}|, |T_{\mathbf{k}+1}| \le L_{\mathsf{sim}}$ and thus the membership in $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$ can be determined by a circuit of size $\mathrm{poly}(\lambda, L_{\mathsf{sim}})$. As before, $\mathsf{KeyListGen}$ and $\mathsf{LocalKG}$ can be computed in time at most $\mathrm{poly}(\lambda, |M|, \log(T))$. Also, as before, input-independent simulator $\mathsf{SimGateEnc}$ can be computed in time at most $\mathrm{poly}(\lambda, \log(|C|))$. Thus, we can upper bound $|\mathsf{Hyb}.H_{\mathbf{k}}|$ to be $\mathrm{poly}(\lambda, L_{\mathsf{sim}}, |M|, \log(T))$.

Using the above bounds, we conclude that $|\widetilde{H}| = \mathrm{poly}(\lambda, L_{\mathsf{sim}}, |M|, \log(T))$.

In Section 5, we show the existence of a strong locally simulatable garbling scheme with L_{sim} upper bounded by polynomial in λ. Using this, we obtain a succinct garbling scheme with succinctness $\mathrm{poly}(\lambda, |M|, \log(T))$. If we instead implement the strong locally simulatable garbing schemes using Yao's garbling scheme, we obtain a succinct garbling scheme with succinctness $\mathrm{poly}(\lambda, |M|, S)$, where S denotes a bound on the space of the Turing machine to be garbled; this scheme is exactly the scheme of [BGL+15].

[20] Note that the computation of $\mathsf{SK}_{\mathrm{Inp}}$ can be performed in the setup phase and in this case, the online computational complexity of SuccGC is $L_{\mathsf{inp}}(\lambda, |x|, m)$.

6.2 Proof of Security

To prove the security of SuccGC, we invoke the strong local simulation security of Π_{LGC}. Π_{LGC} is associated with a set of local simulators $(\{\mathsf{SimKeyListGen}_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}}\}_{\mathbf{k}\in[q]})$ and a vector of sets $\{T_{\mathbf{k}}\}_{\mathbf{k}\in[q]}$ that satisfies strong local simulation security. We prove the following lemma.

Lemma 2. SuccGC *satisfies the security property of a succinct garbling scheme (Definition 4).*

We crucially employ the local simulation strategy of Π_{LGC} to prove the above lemma. Namely, for a fixed (adversarially chosen) TM, input and time bound (M, x, T), we define the hybrid simulators $(\mathsf{SimTMEncode}_{\mathbf{k}}, \mathsf{SimInpEncode}_{\mathbf{k}})$, for $\mathbf{k} \in [q]$.

Informally, $\mathsf{SimTMEncode}_{\mathbf{k}}$ outputs an obfuscated circuit, which has hardwired inside it the input-dependent simulated gate encodings along with a small subset of secret keys and the input-**in**dependent gate encodings and the other secret keys are generated "on-the-fly" (i.e. within the circuit) using a PRF key.

$\underline{\mathsf{SimTMEncode}_{\mathbf{k}}(\mathsf{MSK}, 1^T, M, x, y)}$: On input master secret key MSK, time bound T, Turing machine M, input x, output $y = M(x)$, do the following:

- Compute $C = \mathsf{CktTransform}(M, |x|, T)$.
- For every $i^* \in [N_{\text{comps}}]$, compute $\mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i^*)$ to obtain $\mathbf{L}^{\mathbf{k}}_{i^*}$ and $\mathsf{st}^{\mathbf{k}}_{i^*}$.
- Let $\widehat{\mathbf{L}} = \left(\bigcup_{i^*\in T_{\mathbf{k}}} \mathbf{L}^{\mathbf{k}}_{i^*}\right) \bigcap \left(\bigcup_{i^*\notin T_{\mathbf{k}}} \mathbf{L}^{\mathbf{k}}_{i^*}\right)$.
- For every $j^* \in \bigcup_{i^*\in T_{\mathbf{k}}} \mathbf{L}^{\mathbf{k}}_{i^*}$, sample $sk^{(\mathbf{k})}_{j^*} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*)$. For every $j^* \notin \bigcup_{i^*\in T_{\mathbf{k}}} \mathbf{L}^{\mathbf{k}}_{i^*}$, sample $sk^{(\mathbf{k})}_{j^*} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*; \mathsf{PRF}(K_1, j^*))$. That is, all the keys indexed by $\bigcup_{i^*\in T_{\mathbf{k}}} \mathbf{L}^{\mathbf{k}}_{i^*}$ are sampled using fresh randomness and the other keys are sampled using randomness drawn from a PRF. Set $\mathsf{MSK}^{(\mathbf{k})} = \left(sk^{(\mathbf{k})}_1, \ldots, sk^{(\mathbf{k})}_{N_{\text{keys}}}\right)$.
- For every $i^* \in T_{\mathbf{k}}$, sample $\langle \tilde{g}_{i^*}\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}\left(\mathsf{MSK}^{(\mathbf{k})}, i^*, C, x\right)$.
- Compute $\widetilde{H}_{\mathbf{k}} \leftarrow \mathsf{iO}(1^\lambda, H_{\mathbf{k}})$, where $H_{\mathbf{k}}$ is defined in Figure 6. Set the TM encoding $\langle M\rangle$ to be $(T, \widetilde{H}_{\mathbf{k}})$.

Note: The running time of $\mathsf{SimTMEncode}_{\mathbf{k}}$ *is polynomial in* $(\lambda, T, |M|, |x|, |M(x)|)$.

$\underline{\mathsf{SimInpEncode}_{\mathbf{k}}(\mathsf{MSK}, 1^T, M, x)}$: Output $\langle x\rangle = \Pi_{\mathsf{LGC}}.\mathsf{SimInpEnc}_{\mathbf{k}}(\mathsf{MSK}, C, x)$, where $C = \mathsf{CktTransform}(M, |x|, T)$.

We then define the simulator $\mathsf{SuccGC.Sim}$ that on input $(1^\lambda, 1^n, 1^T, M, M(x))$ it first generates $\mathsf{MSK} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gen}(1^\lambda, T, 1^n, 1^m)$, then computes $\mathsf{SimTMEncode}_q(\mathsf{MSK}, 1^n, 1^T, M, M(x))$ and $\mathsf{SimInpEncode}_q(\mathsf{MSK}, 1^n, 1^T, M, M(x)))$. Note that $\mathsf{SimTMEncode}_q$ and $\mathsf{SimInpEncode}_q$ only require $(\mathsf{MSK}, 1^n, 1^T, M, M(x))$ as input rather than $(\mathsf{MSK}, 1^T, M, x)$ since the algorithms $(\mathsf{SimKeyListGen}_q, \mathsf{SimGtEnc}_q, \mathsf{SimInpEnc}_q)$ associated with Π_{LGC} only require $(M, M(x))$ rather than (M, x) (we crucially use the fact that $T_q = \emptyset$).

We now prove that for every Turing machine M and input x, the pairs of distributions $D_{\mathbf{k}}$ and $D_{\mathbf{k}+1}$ are $\mathsf{negl}(\lambda)$-computationally indistinguishable, where

$$D_{\mathbf{k}} := \left\{\left(\mathsf{SimTMEncode}_{\mathbf{k}}(\mathsf{MSK}, 1^T, M, x), \mathsf{SimInpEncode}_{\mathbf{k}}(\mathsf{MSK}, 1^T, M, x)\right)\right\}.$$

Once we have done so, we will have proved Lemma 2.

Lemma 3. *Assuming that* Π_{LGC} *is a* ε*-secure strong locally simulatable garbling scheme,* PRF *is a* ε'*-secure puncturable PRF and* iO *is a* ε''*-secure indistinguishability obfuscator, we have* $D_{\mathbf{k}}$ *and* $D_{\mathbf{k}+1}$ *are* $7(\varepsilon + \varepsilon' + \varepsilon'')$*-computationally indistinguishable, for every* $\mathbf{k} \in [q]$.

Input: index $i \leq T$

Hardwired Values: TM M, input length n, time bound T, value y, set $T_{\mathbf{k}}$, set $\widehat{\mathbf{L}}$, secret keys $\{sk_{j^*}\}_{j^* \in \mathbf{L}^{(\mathbf{k})}}$, simulated gates $\{\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k})}\}_{i^* \in T_{\mathbf{k}}}$, PRF keys K_1, K_2.

- If $i \in T_{\mathbf{k}}$, output $\langle \tilde{g}_i \rangle^{(\mathbf{k})}$.
- Otherwise:
  - Compute $\mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i)$ to obtain $(\mathbf{L}_i^{\mathbf{k}}, \mathsf{st}_i^{\mathbf{k}})$.
  - For every $j \in \mathbf{L}_i^{\mathbf{k}} \backslash \widehat{\mathbf{L}}$, compute $sk_j = \mathsf{LocalKG}(1^\lambda, j; \mathsf{PRF}(K_1, j))$.
  - Compute $\langle g_i \rangle^{(\mathbf{k})} = \mathsf{SimGtEnc}_{\mathbf{k}}\left(\mathsf{st}_i, \{sk_j\}_{j \in \mathbf{L}_i^{\mathbf{k}}}, y; \mathsf{PRF}(K_2, i)\right)$.
  - Output $\langle g_i \rangle^{(\mathbf{k})}$.

Figure 6: Description of $H_{\mathbf{k}}$.

Proof. We state the following hybrids.

Hyb_1**:** $D_{\mathbf{k}}$**.** The output distribution of this hybrid is identical to $D_{\mathbf{k}}$.

Hyb_2**: Hardwire Gate Encodings and Keys with respect to** $T_{\mathbf{k}+1}$**.** In this hybrid, the gate encodings with respect to $T_{\mathbf{k}+1}$ are hardwired. In addition, the keys indexed by the set $(\cup_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{\mathbf{k}}) \cap (\cup_{i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{\mathbf{k}})$ are hardwired inside the obfuscated circuit. Recall that in the previous hybrid, the keys indexed by the set $\cup_{i \in T_{\mathbf{k}}} \mathbf{L}_i^{\mathbf{k}}$ are hardwired in the obfuscated circuit. We describe the formal details below.

The garbling of M is computed as follows: On input master secret key MSK, time bound T, Turing machine M, and input x, perform the following operations.

- Compute $C = \mathsf{CktTransform}(M, |x|, T)$.
- For every $i^* \in [N_{\text{comps}}]$, compute $\mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i^*)$ to obtain $(\mathbf{L}_{i^*}^{\mathbf{k}}, \mathsf{st}_{i^*}^{\mathbf{k}})$.
- Set $\widehat{\mathbf{L}} = \left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{\mathbf{k}}\right) \bigcap \left(\bigcup_{i^* \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{\mathbf{k}}\right)$.
- For every $j^* \in \bigcup_{i^* \in T_{\mathbf{k}}} \mathbf{L}_{i^*}^{\mathbf{k}}$, sample $sk_{j^*}^{(\mathbf{k})} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*)$. For every $j^* \notin \bigcup_{i^* \in T_{\mathbf{k}}} \mathbf{L}_{i^*}^{\mathbf{k}}$, sample $sk_{j^*}^{(\mathbf{k})} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*; \mathsf{PRF}(K_1, j^*))$. That is, all the keys indexed by $\bigcup_{i^* \in T_{\mathbf{k}}} \mathbf{L}_{i^*}^{\mathbf{k}}$ are sampled using fresh randomness and the other keys are sampled using randomness drawn from a PRF. Set $\mathsf{MSK}^{(\mathbf{k})} = (sk_1^{(\mathbf{k})}, \ldots, sk_q^{(\mathbf{k})})$.
- For every $i^* \in T_{\mathbf{k}}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\mathsf{MSK}^{(\mathbf{k})}, i^*, C, x)$.
- For every $i^* \in T_{\mathbf{k}+1} \backslash T_{\mathbf{k}}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\mathsf{st}_{i^*}, (sk_j^{(\mathbf{k})})_{j \in \mathbf{L}_{i^*}^{\mathbf{k}}}, y; \mathsf{PRF}(K_2, i^*))$. Note that the simulated gate encodings are still computed according to $(\Pi_{\mathsf{LGC}}.\mathsf{SimKeyListGen}_{\mathbf{k}}, \Pi_{\mathsf{LGC}}.\mathsf{SimGtEnc}_{\mathbf{k}}, \Pi_{\mathsf{LGC}}.\mathsf{SimInpEnc}_{\mathbf{k}})$.
- Puncture K_1 at $\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{\mathbf{k}}$ to obtain the key K_1^*. Puncture K_2 at $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$ to obtain the key K_2^*.
- Compute $\widetilde{\mathsf{Hyb}.H_{\mathbf{k}}} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{Hyb}.H_{\mathbf{k}})$, where $\mathsf{Hyb}.H_{\mathbf{k}}$ is defined in Figure 7. Set the TM encoding $\langle M \rangle$ to be $(T, \widetilde{\mathsf{Hyb}.H_{\mathbf{k}}})$.

Input: index $i \leq T$
Hardwired Values: TM M, value y, set $T_{\mathbf{k}}$, set $\widehat{\mathbf{L}}$, secret keys $\{sk_{j^*}\}_{j^* \in \widehat{\mathbf{L}}}$, simulated gates $\{\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k})}\}_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}$, punctured PRF keys K_1^*, K_2^*.

- If $i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$, output $\langle \tilde{g}_i \rangle^{(\mathbf{k})}$.
- Otherwise:
  - Compute $\mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i)$ to obtain $(\mathbf{L}_i^{\mathbf{k}}, \mathsf{st}_i^{\mathbf{k}})$.
  - For every $j \in \mathbf{L}_i^{\mathbf{k}} \backslash \widehat{\mathbf{L}}$, compute $sk_j = \mathsf{LocalKG}(1^\lambda, j; \mathsf{PRF}(K_1^*, j))$.
  - Compute $\langle g_i \rangle^{(\mathbf{k})} = \mathsf{SimGtEnc}_{\mathbf{k}}(\mathsf{st}_i^{\mathbf{k}}, \{sk_j\}_{j \in \mathbf{L}_i^{\mathbf{k}}}, y; \mathsf{PRF}(K_2^*, i))$.
  - Output $\langle g_i \rangle^{(\mathbf{k})}$.

Figure 7: Description of $\mathsf{Hyb}.H_{\mathbf{k}}$.

The encoding of x is still computed using $\mathsf{SimInpEnc}_{\mathbf{k}}$ as in the previous hybrid. The output of this hybrid is the garbling of M (as computed above) and the input encoding of x.

Note that the circuits $H_{\mathbf{k}}$ and $\mathsf{Hyb}.H_{\mathbf{k}}$ are equivalent; the only difference between the descriptions of $H_{\mathbf{k}}$ and $\mathsf{Hyb}.H_{\mathbf{k}}$ is that for the indices corresponding to $T_{\mathbf{k}+1}$, the outputs are pre-computed (according to $H_{\mathbf{k}}$) and hardwired in $\mathsf{Hyb}.H_{\mathbf{k}}$. By the security of iO, it follows that the output distributions of Hyb_1 and Hyb_2 are computationally indistinguishable.

Hyb_3: Fresh randomness used for $T_{\mathbf{k}+1}$. The secret keys $\{sk_j\}_{j \in \bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}}$ are generated using fresh randomness (instead of $\mathsf{PRF}(K_1, \cdot)$); that is, generate $sk_j \leftarrow \mathsf{LocalKG}(1^\lambda, j)$ for every $j \in \bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}$. Recall that in the previous hybrid only the secret keys indexed by $\bigcup_{i^* \in T_{\mathbf{k}}} \mathbf{L}_{i^*}$ are generated using fresh randomness. Also, for every $i^* \in T_{\mathbf{k}+1} \backslash T_{\mathbf{k}}$, compute $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\mathsf{st}_{i^*}, (sk_j^{(\mathbf{k})})_{j \in \mathbf{L}_{i^*}}, y)$ (freshly sampled randomness is used). The rest of the steps of the garbling procedure is as described in the previous hybrid.

By the security of puncturable PRFs (recall that the PRF key K_1 is punctured at $\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i$ and PRF key K_2 is punctured at $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$), the output distributions of Hyb_2 and Hyb_3 are computationally indistinguishable[21].

Hyb_4: Switch Hardwired Gate Encodings from $\mathsf{SimGtEnc}_{\mathbf{k}}$ to $\mathsf{SimGtEnc}_{\mathbf{k}+1}$. In this hybrid, the simulated gates hardwired in Figure 7 are instead generated using ($\mathsf{SimKeyListGen}_{\mathbf{k}+1}$, $\mathsf{SimGtEnc}_{\mathbf{k}+1}$) and the simulated input encoding is computed according to $\mathsf{SimInpEnc}_{\mathbf{k}+1}$. That is,

- Compute $C = \mathsf{CktTransform}(M, |x|, T)$.
- For every $i^* \in [N_{\text{comps}}]$, compute $\mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i^*)$ to obtain $(\mathbf{L}_{i^*}^{\mathbf{k}}, \mathsf{st}_{i^*}^{\mathbf{k}})$.
- For every $i^* \in [N_{\text{comps}}]$, also compute $\mathsf{SimKeyListGen}_{\mathbf{k}+1}(M, T, n, i^*)$ to obtain $(\mathbf{L}_{i^*}^{\mathbf{k}+1}, \mathsf{st}_{i^*}^{\mathbf{k}+1})$.
- Set $\widehat{\mathbf{L}} = \left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{\mathbf{k}}\right) \bigcap \left(\bigcup_{i^* \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{\mathbf{k}}\right)$.

[21] Note that the security of puncturable PRFs is invoked twice (for K_1^* and K_2^*) to argue indistinguishability of Hyb_2 and Hyb_3.

- For every $j^* \in \bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}$, sample $sk_{j^*}^{(\mathbf{k})} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*)$. For every $j^* \notin \bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}$, sample $sk_{j^*}^{(\mathbf{k})} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*; \mathsf{PRF}(K_1, j^*))$. That is, all the keys indexed by $\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}$ are sampled using fresh randomness and the other keys are sampled using randomness drawn from a PRF. Set $\mathsf{MSK}^{(\mathbf{k})} = (sk_1^{(\mathbf{k})}, \ldots, sk_q^{(\mathbf{k})})$.
- For every $i^* \in T_{\mathbf{k}+1}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{MSK}^{(\mathbf{k})}, i^*, C, x)$.
- For every $i^* \in T_{\mathbf{k}} \backslash T_{\mathbf{k}+1}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{st}_{i^*}^{(\mathbf{k}+1)}, (sk_j^{(\mathbf{k}+1)})_{j \in \mathbf{L}_{i^*}^{\mathbf{k}+1}}, y)$.
- Puncture K_1 at $\left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}\right)$ to obtain the key K_1^*. Puncture K_2 at $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$ to obtain the key K_2^*.
- Compute $\widetilde{\mathsf{Hyb}.H_{\mathbf{k}}} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{Hyb}.H_{\mathbf{k}})$, where $\mathsf{Hyb}.H_{\mathbf{k}}$ is defined in Figure 7. Note that the gate encodings $\{\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k}+1)}\}_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}$ are hardwired inside the circuit $\mathsf{Hyb}.H_{\mathbf{k}}$ (instead of $\{\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k})}\}_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}$). We want to emphasize that the secret keys indexed by the list $\widehat{\mathbf{L}}$ (computed with respect to $\mathsf{SimKeyListGen}_{\mathbf{k}}(\cdot)$) are hardwired in the circuit, just like the previous hybrids. Moreover, $\mathsf{SimKeyListGen}_{\mathbf{k}}(\cdot)$ and $\mathsf{SimGtEnc}_{\mathbf{k}}(\cdot)$ are still used in the circuit to generate the i^{th} gate encodings for $i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$. Set the TM encoding $\langle M \rangle$ to be $(T, \widetilde{\mathsf{Hyb}.H_{\mathbf{k}}})$.

The input encoding is generated as follows: compute $\langle x \rangle \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}+1}(\mathsf{MSK}, 1^T, M, x)$. We prove the following claim.

Claim 1. *The strong local simulation security property of Π_{LGC} implies that the output distributions of hybrids Hyb_3 and Hyb_4 are computationally indistinguishable.*

Proof. Suppose the output distributions of Hyb_3 and Hyb_4 are distinguishable, we violate the security of Π_{LGC}. Let $\mathcal{A}$ distinguish the output distributions of Hyb_3 and Hyb_4.

We construct adversary $\mathcal{B}$ as follows: let $\mathsf{list}_{\mathbf{k}}^{\mathsf{public}} = \bigcup_{i^* \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{(\mathbf{k})}$ and $\mathsf{list}_{\mathbf{k}}^{\mathsf{rand}} = \bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{(\mathbf{k})}$, where $(\mathbf{L}_{i^*}^{(\mathbf{k})}, \mathsf{st}_{i^*}) = \mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i^*)$. For every $j \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}$, $\mathcal{B}$ computes $sk_j^{(\mathbf{k})} = \mathsf{PRF}(K_1^*, j)$, where K_1^* is the PRF key K_1 punctured at $\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}$. It sends $\{sk_j^{(\mathbf{k})}\}_{j \notin \mathsf{list}_{\mathbf{k}}^{\mathsf{rand}}}$ to the challenger of Π_{LGC}. $\mathcal{B}$ receives gate encodings $\{\langle g_i \rangle\}_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}}$, input encoding $\langle x \rangle$, secret keys $\{sk_j\}_{j \in \mathsf{list}_{\mathbf{k}}^{\mathsf{public}}}$ and then constructs the circuit $\mathsf{Hyb}.H_{\mathbf{k}}$ (Figure 7); denote $\widetilde{\mathsf{Hyb}.H_{\mathbf{k}}}$ to be the output of $\mathsf{iO}(1^\lambda, \mathsf{Hyb}.H_{\mathbf{k}})$. $\mathcal{B}$ sets $\widehat{M} = (T, \widetilde{\mathsf{Hyb}.H_{\mathbf{k}}})$. It then sends $(\widehat{M}, \langle x \rangle)$ to $\mathcal{A}$.

If the challenger of Π_{LGC} computed the gate encodings according to $(\mathsf{SimKeyListGen}_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}})$ then the distribution of $(\widehat{M}, \langle x \rangle)$ sent to $\mathcal{A}$ corresponds to Hyb_3, otherwise we are in Hyb_4. Hence, if $\mathcal{A}$ can distinguish Hyb_3 and Hyb_4 then $\mathcal{B}$ can break the strong local simulation security of Π_{LGC} with the same advantage. This proves the claim. □

Hyb_5: Hardwire lists $\mathbf{L}_i^{\mathbf{k}+1}$ for $i \in T_{k+1}$. In this hybrid, the local keys corresponding to $\mathbf{L}_i^{\mathbf{k}+1}$ are additionally hardwired. Note that the list $\mathbf{L}_i^{\mathbf{k}+1}$ is computed by $\mathsf{SimKeyListGen}_{\mathbf{k}+1}$. Also, K_1 is punctured at the list $\left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}\right) \cup \left(\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}\right)$. The rest of the hybrid remains the same. In particular, the input encoding is still encoded with respect to $\mathsf{SimInpEncode}_{\mathbf{k}+1}$.

We elaborate on the TM encoding below.

- Compute $C = \mathsf{CktTransform}(M, |x|, T)$.
- For every $i^* \in [N_{\mathrm{comps}}]$, compute $\mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i^*)$ to obtain $(\mathbf{L}_{i^*}^{\mathbf{k}}, \mathsf{st}_{i^*}^{\mathbf{k}})$.
- For every $i^* \in [N_{\mathrm{comps}}]$, also compute $\mathsf{SimKeyListGen}_{\mathbf{k}+1}(M, T, n, i^*)$ to obtain $(\mathbf{L}_{i^*}^{\mathbf{k}+1}, \mathsf{st}_{i^*}^{\mathbf{k}+1})$.
- Set $\widehat{\mathbf{L}} = \left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}\right) \cup \left(\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}\right)$.

- For every $j^* \in \bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}$, sample $sk_{j^*}^{(\mathbf{k})} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*)$. For every $j^* \notin \bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}$, sample $sk_{j^*}^{(\mathbf{k})} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*; \mathsf{PRF}(K_1, j^*))$. That is, all the keys indexed by $\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}$ are sampled using fresh randomness and the other keys are sampled using randomness drawn from a PRF. Set $\mathsf{MSK}^{(\mathbf{k})} = (sk_1^{(\mathbf{k})}, \ldots, sk_q^{(\mathbf{k})})$.
- For every $i^* \in T_{\mathbf{k}+1}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{MSK}^{(\mathbf{k})}, i^*, C, x)$.
- For every $i^* \in T_{\mathbf{k}} \backslash T_{\mathbf{k}+1}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{st}_{i^*}^{(\mathbf{k}+1)}, (sk_j^{(\mathbf{k}+1)})_{j \in \mathbf{L}_{i^*}^{\mathbf{k}+1}}, y)$.
- Puncture K_1 at $\left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}\right) \cup \left(\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}\right)$ to obtain the key K_1^*. Puncture K_2 at $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$ to obtain the key K_2^*.
- Compute $\widetilde{\mathsf{Hyb}.H_{\mathbf{k}}} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{Hyb}.H_{\mathbf{k}})$, where $\mathsf{Hyb}.H_{\mathbf{k}}$ is defined in Figure 7. Set the TM encoding $\langle M \rangle$ to be $(T, \widetilde{\mathsf{Hyb}.H_{\mathbf{k}}})$.

The indistinguishability of Hyb_4 and Hyb_5 follows from the security of iO.

Hyb_6: Pseudorandomness used for $T_{\mathbf{k}}$ and Randomness for $T_{\mathbf{k}+1}$. Recall that in the previous hybrid, the secret keys corresponding to indices in the set $\bigcup_{i \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_i^{(\mathbf{k})}$ are generated using uniformly sampled randomness. Similarly, the simulated gates corresponding to the indices in the set $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$ are also generated using uniformly sampled randomness. In this hybrid, we instead generate the TM garbling as follows.

- Compute $C = \mathsf{CktTransform}(M, |x|, T)$.
- For every $i^* \in [N_{\text{comps}}]$, compute $\mathsf{SimKeyListGen}_{\mathbf{k}}(M, T, n, i^*)$ to obtain $(\mathbf{L}_{i^*}^{\mathbf{k}}, \mathsf{st}_{i^*}^{\mathbf{k}})$.
- For every $i^* \in [N_{\text{comps}}]$, also compute $\mathsf{SimKeyListGen}_{\mathbf{k}+1}(M, T, n, i^*)$ to obtain $(\mathbf{L}_{i^*}^{\mathbf{k}+1}, \mathsf{st}_{i^*}^{\mathbf{k}+1})$.
- Set $\widehat{\mathbf{L}} = \left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}\right) \cup \left(\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}\right)$.
- For every $j^* \in \bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}$, sample $sk_{j^*}^{(\mathbf{k}+1)} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*)$. For every $j^* \notin \bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}$, sample $sk_{j^*}^{(\mathbf{k}+1)} \leftarrow \mathsf{LocalKG}(1^\lambda, j^*; \mathsf{PRF}(K_1, j^*))$. That is, all the keys indexed by $\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}$ are sampled using fresh randomness and the other keys are sampled using randomness drawn from a PRF. Set $\mathsf{MSK}^{(\mathbf{k}+1)} = (sk_1^{(\mathbf{k}+1)}, \ldots, sk_q^{(\mathbf{k}+1)})$.
- For every $i^* \in T_{\mathbf{k}+1}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{MSK}^{(\mathbf{k}+1)}, i^*, C, x)$.
- For every $i^* \in T_{\mathbf{k}} \backslash T_{\mathbf{k}+1}$, sample $\langle \tilde{g}_{i^*} \rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{st}_{i^*}^{(\mathbf{k}+1)}, (sk_j^{(\mathbf{k}+1)})_{j \in \mathbf{L}_{i^*}^{\mathbf{k}+1}}, y; \mathsf{PRF}(K_2, i^*))$.
- Puncture K_1 at $\left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}\right) \cup \left(\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}\right)$ to obtain the key K_1^*. Puncture K_2 at $T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$ to obtain the key K_2^*.
- Compute $\widetilde{\mathsf{Hyb}.H_{\mathbf{k}}} \leftarrow \mathsf{iO}(1^\lambda, \mathsf{Hyb}.H_{\mathbf{k}})$, where $\mathsf{Hyb}.H_{\mathbf{k}}$ is defined in Figure 7. Set the TM encoding $\langle M \rangle$ to be $(T, \widetilde{\mathsf{Hyb}.H_{\mathbf{k}}})$.

From the security of puncturable pseudorandom functions, the output distributions of Hyb_5 and Hyb_6 are computationally indistinguishable.

Hyb_7: $D_{\mathbf{k}+1}$. The output distribution of this hybrid is identical to the distribution $D_{\mathbf{k}+1}$.

The circuits obfuscated in Hyb_6, namely $\mathsf{Hyb}.H_{\mathbf{k}}$, and Hyb_7, namely $H_{\mathbf{k}+1}$, have the following differences:

- The hardwired keys in Hyb_6 are indexed by the set $\left(\bigcup_{i^* \in T_{\mathbf{k}} \cup T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}}\right) \cup \left(\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}\right)$, while the hardwired keys in Hyb_7 are indexed by the set $\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}$.

- In Hyb_6, $\mathsf{SimKeyListGen}_{\mathbf{k}}(\cdot)$ is used in $\mathsf{Hyb}.H_{\mathbf{k}}$. However, in Hyb_7, $\mathsf{SimKeyListGen}_{\mathbf{k}+1}(\cdot)$ is used in $\mathsf{Hyb}.H_{\mathbf{k}}$.
- In Hyb_6, $\mathsf{SimGtEnc}_{\mathbf{k}}(\cdot)$ is used in $\mathsf{Hyb}.H_{\mathbf{k}}$. However, in Hyb_7, $\mathsf{SimGtEnc}_{\mathbf{k}+1}(\cdot)$ is used in $\mathsf{Hyb}.H_{\mathbf{k}}$.

We argue that $\mathsf{Hyb}.H_{\mathbf{k}}$ and $H_{\mathbf{k}+1}$ are functionally equivalent. To argue this, we prove that the outputs of the two circuits are the same on all inputs. We consider the following cases.

- **Case** $i \in T_{\mathbf{k}+1}$**.** The output of $\mathsf{Hyb}.H_{\mathbf{k}}$ is a gate encoding $\langle g_i \rangle^{(\mathbf{k})}$ that is hardwired in the circuit. It is computed as $\langle g_i \rangle^{(\mathbf{k}+1)} = \mathsf{SimGtEnc}_{\mathbf{k}+1}\left(\mathsf{MSK}^{(\mathbf{k}+1)}, i^*, C, x\right)$ and in particular, the gate encoding is computed using fresh randomness. Observe that this is identical to the output of $H_{\mathbf{k}+1}$ on i.
- **Case** $i \in T_{\mathbf{k}} \backslash T_{\mathbf{k}+1}$**.** The output of $\mathsf{Hyb}.H_{\mathbf{k}}$ is a gate encoding $\langle g_i \rangle^{(\mathbf{k}+1)} = \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{st}_{i^*}^{(\mathbf{k}+1)}, (sk_j^{(\mathbf{k}+1)})_{j \in \mathbf{L}_{i^*}}, y; \mathsf{PRF}(K_2, i^*))$. Observe that this is identical to the output of $H_{\mathbf{k}+1}$ on i and in particular, even in $H_{\mathbf{k}+1}$, the gate encoding is computed using a PRF. Note that unlike $\mathsf{Hyb}.H_{\mathbf{k}}$, the gate encodings indexed by $T_{\mathbf{k}} \backslash T_{\mathbf{k}+1}$ are not hardwired in $H_{\mathbf{k}+1}$.
- **Case** $i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$**.** In $\mathsf{Hyb}.H_{\mathbf{k}}$, the gate encoding $\langle g_i \rangle^{(\mathbf{k})}$ is generated as $\langle g_i \rangle^{(\mathbf{k})} = \mathsf{SimGtEnc}_{\mathbf{k}}(\mathsf{st}_i^{\mathbf{k}}, \{sk_j\}_{j \in \mathbf{L}_i^{\mathbf{k}}}, y; \mathsf{PRF}(K_2^*, i))$; that is, the $\mathbf{k}^{th}$ simulator $(\mathsf{SimKeyListGen}_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}})$ is still used to generate the gate encodings. However, in $H_{\mathbf{k}+1}$, $\langle g_i \rangle^{(\mathbf{k}+1)}$ is generated as $\langle g_i \rangle^{(\mathbf{k}+1)} = \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{st}_i^{\mathbf{k}+1}, \{sk_j\}_{j \in \mathbf{L}_i^{\mathbf{k}+1}}, y; \mathsf{PRF}(K_2^*, i))$ and in particular, $(\mathsf{SimKeyListGen}_{\mathbf{k}+1}, \mathsf{SimGtEnc}_{\mathbf{k}+1})$ is used to generate the gate encodings. The local secret keys are generated identically in both Hyb_6 and Hyb_7. That is, all the secret keys indexed by $\bigcup_{i^* \in T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}$ are generated using uniform randomness, while all the secret keys indexed by $\bigcup_{i^* \notin T_{\mathbf{k}+1}} \mathbf{L}_{i^*}^{\mathbf{k}+1}$ are generated using PRF key K_1.

  From Definition 10, it follows that for every $i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$, $\mathsf{SimKeyListGen}_{\mathbf{k}}(\cdot)$ and $\mathsf{SimKeyListGen}_{\mathbf{k}+1}(\cdot)$ are functionally equivalent and also, $\mathsf{SimGtEnc}_{\mathbf{k}}(\cdot)$ and $\mathsf{SimGtEnc}_{\mathbf{k}+1}(\cdot)$ are functionally equivalent. Using this, it follows that the output of $H_{\mathbf{k}+1}$ and $\mathsf{Hyb}.H_{\mathbf{k}}$ are identical on every $i \notin T_{\mathbf{k}} \cup T_{\mathbf{k}+1}$.

This proves that the circuits $\mathsf{Hyb}.H_{\mathbf{k}}$ and $H_{\mathbf{k}+1}$ are functionally equivalent. Assuming the security of iO, the output distribution of Hyb_6 is computationally indistinguishable from the output distribution of Hyb_7. □

From the above lemma, we have the following.

Lemma 4. *Assuming that there exists an ε-secure strong locally simulatable garbling scheme, an ε'-secure puncturable PRF and an ε''-secure indistinguishability obfuscation scheme, then there exists an $7T(\varepsilon + \varepsilon' + \varepsilon'')$-secure bounded runtime succinct garbling scheme with T being an upper bound on the runtime of the Turing machines.*

Combining the above lemma with Theorem 4, we have the following lemma.

Lemma 5. *Let λ be the security parameter. Assuming that there exists an ε-secure locally simulatable garbling scheme, an ε'-secure puncturable PRF and an ε''-secure compact functional encryption scheme, then there exists an $7T\left(\varepsilon + \varepsilon' + 2^{\log(T)}\varepsilon''\right)$-secure bounded runtime succinct garbling scheme with T being an upper bound on the runtime of the Turing machines.*

In particular, if ε and ε'' are any negligible functions in the security parameter and T is any polynomial in security parameter then the bounded runtime succinct garbling scheme is polynomially secure assuming any polynomially secure compact functional encryption scheme and any polynomially secure strong locally simulatable garbling scheme.

6.3 Removing the Bounded Runtime Restriction

We now show how to transform a bounded runtime succinct garbling scheme (Definition 4) into a (standard) succinct garbling scheme (Definition 3). We denote the bounded runtime succinct garbling scheme to be $\mathsf{SuccGC}_{BR} = (\mathsf{Setup}, \mathsf{TMEncode}, \mathsf{InpEncode}, \mathsf{Eval})$ and the (standard) succinct garbling scheme to be $\mathsf{SuccGC}^* = (\mathsf{Setup}, \mathsf{TMEncode}, \mathsf{InpEncode}, \mathsf{Eval})$. As an intermediate tool, we use a secret key encryption scheme $(Setup, E, D)$.

We describe SuccGC^* below.

- $\mathsf{Gen}(1^\lambda, 1^n, 1^m)$: On input security parameter λ, input length n and output length m,
  - For $i \in [\lambda]$, compute $K_i \leftarrow Setup(1^\lambda)$.
  - For $i \in [\lambda]$, compute $\mathsf{SuccGC}_{BR}.\mathsf{MSK}_i \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{Gen}(1^\lambda, 2^i, 1^n, 1^m)$.

  Output $\mathsf{MSK} = \left(\{K_i\}_{i\in[\lambda]}, \{\mathsf{SuccGC}_{BR}.\mathsf{MSK}_i\}_{i\in[\lambda]}\right)$.
- $\mathsf{TMEncode}(\mathsf{MSK}, M)$: On input master secret key MSK, Turing machine M,
  - For $i \in [\lambda]$, compute $\mathsf{SuccGC}_{BR}.\langle M_i\rangle \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{TMEncode}\left(\mathsf{SuccGC}_{BR}.\mathsf{MSK}_i, M_i\right)$, where M_i is defined as follows: on input x, it executes M for 2^i steps and if the computation terminates then it outputs $(1||M(x))$, otherwise it outputs $(0||K_{i+1}||0^{m-|K_{i+1}|-1})$ (let $K_{\lambda+1} = \perp$).
  - For $i \in [\lambda]$, compute $\mathsf{CT}_i^{tm} \leftarrow E\left(K_i, \mathsf{SuccGC}_{BR}.\langle M_i\rangle\right)$.

  Output $\langle M\rangle = \left(K_1, \left\{\mathsf{CT}_i^{tm}\right\}_{i\in[\lambda]}\right)$.
- $\mathsf{InpEncode}(\mathsf{MSK}, x)$
  - For $i \in [\lambda]$, $\mathsf{SuccGC}_{BR}.\langle x\rangle_i \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{InpEncode}(\mathsf{SuccGC}_{BR}.\mathsf{MSK}_i, x)$.
  - For $i \in [\lambda]$, compute $\mathsf{CT}_i^{inp} \leftarrow E\left(K_i, \mathsf{SuccGC}_{BR}.\langle x\rangle_i\right)$.

  Output $\langle x\rangle = \left(\left\{\mathsf{CT}_i^{inp}\right\}_{i\in[\lambda]}\right)$.
- $\mathsf{Eval}(\langle M\rangle, \langle x\rangle)$: On input TM encoding $\langle M\rangle$, input encoding $\langle x\rangle$, do the following for $i = 1, \ldots, \lambda$:
  - Compute $\mathsf{SuccGC}_{BR}.\langle M_i\rangle \leftarrow D(K_i, \mathsf{CT}_i^{tm})$.
  - Compute $\mathsf{SuccGC}_{BR}.\langle x\rangle_i \leftarrow D(K_i, \mathsf{CT}_i^{inp})$.
  - Compute $y \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{Eval}(\mathsf{SuccGC}_{BR}.\langle M_i\rangle, \mathsf{SuccGC}_{BR}.\langle x\rangle_i)$
  - If y is of the form $(1||y')$ then output y'. Else, if y is of the form $(0||y'||0^{m-|K_{i+1}|-1})$ then set $K_{i+1} = y'$ and continue the process.

We now prove the properties of SuccGC.

Correctness. Consider an input x and TM M running in time at most $t \leq 2^\lambda$ on x. For simplicity, let t be a power of two; the analysis goes through even if that is not the case. For every $i < \log(t)$, from the correctness of SuccGC_{BR} and the correctness of secret key encryption scheme, it follows that the output of $\mathsf{SuccGC}_{BR}.\mathsf{Eval}(\langle M_i\rangle, \langle x\rangle_i)$ is $(0||K_{i+1}||0^{m-|K_{i+1}|-1})$. Finally for $i = \log(t)$ it follows that the output of $\mathsf{SuccGC}_{BR}.\mathsf{Eval}(\langle M_i\rangle, \langle x\rangle_i)$ is $M(x)$.

Efficiency. Consider an input x and TM M running in time at most $t \leq 2^\lambda$ on x and as before, we assume that t is a power of two. The running time of $\mathsf{Gen}(1^\lambda, 1^n, 1^m)$ is polynomial in λ. The running time of $\mathsf{TMEncode}(\mathsf{MSK}, M)$ is polynomial in $(\lambda, |M|)$. The running time of $\mathsf{InpEncode}(\mathsf{MSK}, x)$ is polynomial in (λ, n). We now focus on the running time of $\mathsf{Eval}(\langle M\rangle, \langle x\rangle)$: note that the for loop is executed only for at most $\log(t)$ number of iterations. The running time of every iteration is upper bounded by $\mathrm{poly}(\lambda, t, |M|, n, m)$, by the efficiency property of SuccGC_{BR}. This proves that the running time of Eval is upper bounded by $\mathrm{poly}(\lambda, t, |M|, n, m)$.

Security. We prove the following lemma.

Lemma 6. *Assuming that* SuccGC_{BR} *is an* ε*-secure bounded runtime succinct garbling scheme and* $(Setup, E, D)$ *is an* ε'*-secure secret key encryption scheme,* SuccGC *is a* $\lambda \cdot (\varepsilon + \varepsilon')$*-secure succinct garbling scheme.*

Proof. Let $\mathsf{SuccGC}_{BR}.\mathsf{Sim}$ be the simulator of $\mathsf{SuccGC}_{BR}.\mathsf{Sim}$. We describe the simulator Sim of SuccGC. It takes as input $(1^\lambda, 1^n, t, M, y)$ and performs the following operations.

- For every $i \in [\lambda]$, execute $K_i \leftarrow Setup(1^\lambda)$.
- For every positive integer $i < \log(t)$, do the following: generate $(\langle M_i \rangle, \langle \bot \rangle_i) \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^i, M_i, K_{i+1})$. Compute $\mathsf{CT}_i^{tm} \leftarrow E(K_i, \mathsf{SuccGC}_{BR}.\langle M_i \rangle)$. Compute $\mathsf{CT}_i^{inp} \leftarrow E(K_i, \mathsf{SuccGC}_{BR}.\langle \bot \rangle_i)$.
- For $i = \lceil \log(t) \rceil$, do the following: generate $(\mathsf{SuccGC}_{BR}.\langle M_i \rangle, \mathsf{SuccGC}_{BR}.\langle \bot \rangle_i) \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^i, M_i, y)$. Compute $\mathsf{CT}_i^{tm} \leftarrow E(K_i, \mathsf{SuccGC}_{BR}.\langle M_i \rangle)$. Compute $\mathsf{CT}_i^{inp} \leftarrow E(K_i, \mathsf{SuccGC}_{BR}.\langle \bot \rangle_i)$.
- For $\lambda \geq i > \lceil \log(t) \rceil$, do the following: generate $\mathsf{CT}_i^{tm} \leftarrow E(K_i, 0^{L_i})$, where L_i is the size of the simulated TM encoding output by $\mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^i, M, y)$. Generate $\mathsf{CT}_i^{inp} \leftarrow E(K_i, 0^{L'_i})$, where L'_i is the size of the simulated input encoding output by $\mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^i, M, y)$.

Output $\left(\langle M \rangle = \left(K_1, \left\{ \mathsf{CT}_i^{tm} \right\}_{i \in [\lambda]} \right), \langle x \rangle = \left(\left\{ \mathsf{CT}_i^{inp} \right\}_{i \in [\lambda]} \right) \right)$.

Observe that the running time of Sim is $\mathrm{poly}(\lambda, t, n, |M|, |y|)$. We argue this step by step. In the first bullet, the runtime is polynomial in λ. In the second bullet, the runtime of $\mathsf{SuccGC}_{BR}.\mathsf{Sim}$ is $\mathrm{poly}(\lambda, 2^i, n, |M|, y) \leq \mathrm{poly}(\lambda, t, n, |M|, y)$ and moreover, $\mathsf{SuccGC}_{BR}.\mathsf{Sim}$ is executed at most $\lceil \log(t) \rceil$ number of times. In the third bullet, runtime of $\mathsf{SuccGC}_{BR}.\mathsf{Sim}$ is $\mathrm{poly}(\lambda, t, n, |M|, y)$. To argue about the runtime of the simulator in the last bullet, it suffices to argue about the length of L_i and L'_i, for every $\lambda \geq i > \lceil \log(t) \rceil$. This is because the runtime of the simulator in the final bullet is polynomial in $(\lambda, \sum_{\lambda \geq i > \lceil \log(t) \rceil} |L_i| + |L'_i|)$. Since the output length of $\mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^i, M, y)$ is $\mathrm{poly}(\lambda, n, i, |M|, y)$, it follows that $|L_i| + |L'_i| = \mathrm{poly}(\lambda, n, i, |M|, y)$. Combining all these observations, it follows that the runtime of Sim is $\mathrm{poly}(\lambda, n, t, |M|, |y|)$.

Consider an input $x \in \{0,1\}^n$, TM M running in time at most 2^λ and $|M(x)| = m$. We describe the hybrids below.

Hyb_1: the output of this hybrid is $(\langle M \rangle, \langle x \rangle)$, where: (i) $\mathsf{MSK} \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^m)$, (ii) $\langle M \rangle \leftarrow \mathsf{TMEncode}(\mathsf{MSK}, M)$ and, (iii) $\langle x \rangle \leftarrow \mathsf{InpEncode}(\mathsf{MSK}, x)$.

$\mathsf{Hyb}_{2.i}$, for $i < \lfloor \log(t) \rfloor$: for every $i' \leq i$, generate $(\langle M_{i'} \rangle, \langle \bot \rangle_{i'}) \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^{i'}, M_{i'}, K_{i'+1})$. For every $i < i' < \lambda$, generate $\mathsf{SuccGC}_{BR}.\langle M_{i'} \rangle \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{TMEncode}(\mathsf{SuccGC}_{BR}.\mathsf{MSK}_{i'}, M_{i'})$ and $\mathsf{SuccGC}_{BR}.\langle x \rangle_{i'} \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{InpEncode}(\mathsf{SuccGC}_{BR}.\mathsf{MSK}_{i'}, x)$. The rest of the steps is as in the previous hybrid.

From the security of SuccGC_{BR}, it follows that the output distributions of Hyb_1 and $\mathsf{Hyb}_{2.1}$ and the output distributions of $\mathsf{Hyb}_{2.i}$ and $\mathsf{Hyb}_{2.i+1}$ are computationally indistinguishable.

$\mathsf{Hyb}_{2.\lceil \log(t) \rceil}$: for every $i' < \lceil \log(t) \rceil$, generate $(\langle M_{i'} \rangle, \langle \bot \rangle_{i'}) \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^{i'}, M_{i'}, K_{i'+1})$. Generate $(\langle M_{\lceil \log(t) \rceil} \rangle, \langle x \rangle_{\lceil \log(t) \rceil}) \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{Sim}(1^\lambda, 1^n, 2^{\lceil \log(t) \rceil}, M_{\lceil \log(t) \rceil}, M(x))$. For every $\lceil \log(t) \rceil < i' < \lambda$, generate $\mathsf{SuccGC}_{BR}.\langle M_{i'} \rangle \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{TMEncode}(\mathsf{SuccGC}_{BR}.\mathsf{MSK}_{i'}, M_{i'})$ and $\mathsf{SuccGC}_{BR}.\langle x \rangle_{i'} \leftarrow \mathsf{SuccGC}_{BR}.\mathsf{InpEncode}(\mathsf{SuccGC}_{BR}.\mathsf{MSK}_{i'}, x)$. The rest of the steps is as in the previous hybrid.

From the security of SuccGC_{BR}, it follows that the output distributions of the previous hybrid and $\mathsf{Hyb}_{2.\lceil \log(t) \rceil}$ are computationally indistinguishable.

$\mathsf{Hyb}_{3.i}$, for $\lceil \log(t) \rceil < i \leq \lambda$: for every $\lceil \log(t) \rceil < i' \leq i$, generate $\mathsf{CT}^{tm}_{i'}$ and CT^{inp}_{i} as encryptions of zeroes. The rest of the steps are as in the previous hybrid.

From the security of $(Setup, E, D)$, it follows that the output distributions of $\mathsf{Hyb}_{2.\lceil \log(t) \rceil}$ and $\mathsf{Hyb}_{3.\lceil \log(t) \rceil + 1}$ and the output distributions of $\mathsf{Hyb}_{3.i}$ and $\mathsf{Hyb}_{3.i+1}$ are computationally indistinguishable. We crucially use the fact that the key K_{i+1} is not used in $\mathsf{Hyb}_{3.i}$.

The output distributions of $\mathsf{Hyb}_{3.\lambda}$ and Sim are identical. □

6.4 Succinct Garbling from Functional Encryption

Combining Lemmas 5 and 6, we have the following lemma.

Lemma 7. *Let λ be the security parameter. Assuming that exists an ε_1-secure strong locally simulatable garbling scheme, ε_2-secure puncturable PRF scheme, ε_3-secure symmetric encryption scheme and an ε_4-secure compact functional encryption scheme, then there exists an $\lambda(7t(\varepsilon_1 + \varepsilon_2 + 2^{\log(t)}\varepsilon_4) + \varepsilon_3)$-secure bounded runtime succinct garbling scheme for Turing machines running in time at most t.*

In particular, if ε_4 is any negligible function in the security parameter and t is any polynomial in the security parameter then there exists a polynomially secure succinct garbling scheme assuming any polynomially secure compact functional encryption scheme.

By combining Theorem 4 and the above lemma, we get the following result.

Theorem 12. *Assuming polynomial hardness of compact functional encryption and polynomial hardness of computational DH (or factoring or LWE), there exists a succinct garbling scheme for Turing machines that run in time (arbitrary) polynomial in its input.*

7 Adaptively Secure Garbling from Locally Simulatable Garbling

In this section, we prove Theorem 11. That is, we show how to construct an adaptive garbling scheme starting from a semi-adaptive $(L_{\mathsf{sim}}, L_{\mathsf{inp}})$-locally simulatable garbling scheme, denoted by $\Pi_{\mathsf{LGC}} = (\Pi_{\mathsf{LGC}}.\mathsf{Gen}, \Pi_{\mathsf{LGC}}.\mathsf{Gb}, \Pi_{\mathsf{LGC}}.\mathsf{IEnc}, \Pi_{\mathsf{LGC}}.\mathsf{Eval})$, equipped with simulation strategy $(\{T_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}}\})_{\mathbf{k} \in [q]}$. Combining our transformation with Yao's garbled circuits yields the [HJO+16] construction of adaptive garbled circuits with online complexity $\mathrm{poly}(\lambda, w)$ (where w denotes the width of the circuit), while combining our transformation with Section 5 yields the [GS18a] construction of adaptive garbled circuits with optimal online complexity.

In our transformation, we use the tool of somewhere equivocal encryption (Section 2.5), denoted by $\mathsf{sEQ} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ with equivocation parameter $t = 2L_{\mathsf{sim}}$ and block-length $B = \mathrm{poly}(\lambda)$[22]. We denote the resulting adaptive garbling scheme to be $\Pi_{\mathsf{Ada}} = (\mathsf{Gen}, \mathsf{Gb}, \mathsf{IEnc}, \mathsf{Eval})$.

$\mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$: On input security parameter λ and maximum circuit size s, input length n, output length m, it computes:

- $\Pi_{\mathsf{LGC}}.\mathsf{gsk} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$.
- $\mathsf{eqk} \leftarrow \mathsf{KeyGen}(1^\lambda)$

[22] We set B to be the size of a single (simulated) gate encoding, which is guaranteed to have size $\mathrm{poly}(\lambda)$ by the efficiency requirements of a locally simulatable garbling scheme.

Output the garbling key of adaptively secure garbling scheme as $\mathsf{gsk} = (\mathsf{eqk}, \Pi_{\mathsf{LGC}}.\mathsf{gsk})$.

$\underline{\mathsf{Gb}(\mathsf{gsk}, C)}$: On input garbling key $\mathsf{gsk} = (\mathsf{eqk}, \Pi_{\mathsf{LGC}}.\mathsf{gsk})$ and circuit $C \in \mathcal{C}$, it computes $\underline{\Pi_{\mathsf{LGC}}.\langle C \rangle \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gb}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, C)}$. Parse $\Pi_{\mathsf{LGC}}.\langle C \rangle$ as a sequence of blocks $(\langle g_1 \rangle, \ldots, \langle g_s \rangle)$ with block-length B. Compute $\mathsf{CT} \leftarrow \mathsf{sEQ.Enc}\left(\mathsf{eqk}, \Pi_{\mathsf{LGC}}.\langle C \rangle\right)$ for $i \in [N_{\text{keys}}]$, and output $\langle C \rangle = \mathsf{CT}$.

$\underline{\mathsf{IEnc}(\mathsf{gsk}, x)}$: On input garbling key $\mathsf{gsk} = (\mathsf{eqk}, \Pi_{\mathsf{LGC}}.\mathsf{gsk})$ and input x, it computes $\Pi_{\mathsf{LGC}}.\langle x \rangle \leftarrow \underline{\Pi_{\mathsf{LGC}}.\mathsf{IEnc}(\mathsf{gsk}, x)}$. It outputs this encoding along with the decryption key of sEQ. That is, it outputs $\langle x \rangle = \left(\Pi_{\mathsf{LGC}}.\langle x \rangle, \mathsf{eqk}\right)$.

$\underline{\mathsf{Eval}(\langle C \rangle, \langle x \rangle)}$: On input garbled circuit $\langle C \rangle = \mathsf{CT}$ and input encoding $\langle x \rangle = \left(\Pi_{\mathsf{LGC}}.\langle x \rangle, \mathsf{eqk}\right)$, it computes:

- $\Pi_{\mathsf{LGC}}.\langle C \rangle \leftarrow \mathsf{sEQ.Dec}(\mathsf{eqk}, \mathsf{CT})$
- $\mathsf{out} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Eval}(\Pi_{\mathsf{LGC}}.\langle C \rangle, \Pi_{\mathsf{LGC}}.\langle x \rangle)$.

Output out.

The correctness follows from the correctness of Π_{LGC} and the somewhere equivocal encryption scheme.

Efficiency. The size of $\langle x \rangle$, generated by $\mathsf{IEnc}(\mathsf{gsk}, x)$, is calculated as follows:

$$\begin{aligned} |\langle x \rangle| &= |\Pi_{\mathsf{LGC}}.\langle x \rangle| + |\mathsf{eqk}| \\ &= L_{\mathsf{inp}} + B \cdot L_{\mathsf{sim}} \cdot \mathrm{poly}(\lambda) \\ &= L_{\mathsf{inp}}(\lambda, |x|, |C(x)|) + L_{\mathsf{sim}} \cdot \mathrm{poly}(\lambda), \end{aligned}$$

In particular, we have that $|\mathsf{eqk}| \leq B \cdot L_{\mathsf{sim}} \cdot \mathrm{poly}(\lambda)$ by Theorem 5.

The Simulator. We prove the following theorem.

Theorem 13. *Assuming that* sEQ *is secure and that* Π_{LGC} *satisfies semi-adaptive* L_{sim}*-local simulation security, the scheme* Π_{Ada} *is an adaptively secure garbling scheme.*

Let $(\{T_{\mathbf{k}}, \mathsf{SimGtEnc}_{\mathbf{k}}, \mathsf{SimInpEnc}_{\mathbf{k}}\})_{\mathbf{k} \in [q]}$ denote the simulation strategy associated to Π_{LGC}. We first present the simulator of Π_{Ada}, denoted by $\mathsf{Sim}_{\mathsf{AdGC}}$ and then prove that the indistinguishability of simulated distribution and the real distribution.

$\mathsf{Sim}_{\mathsf{AdGC}}$: Upon receiving the security parameter λ and circuit C[23], it samples $\Pi_{\mathsf{LGC}}.\mathsf{gsk} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$ and first computes $\Pi_{\mathsf{LGC}}.\langle C \rangle^{(q)} = (\langle g_1 \rangle^{(q)}, \ldots, \langle g_s \rangle^{(q)})$ by sampling $\langle g_i \rangle^{(q)} \leftarrow \mathsf{SimGtEnc}_q(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C)$. In addition, it samples $\mathsf{eqk} \leftarrow \mathsf{KeyGen}(1^\lambda)$. It then computes a ciphertext $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{eqk}, \Pi_{\mathsf{LGC}}.\langle C \rangle^{(q)})$. Finally, it outputs $(\langle C \rangle = \mathsf{CT}, \mathsf{st} = (\Pi_{\mathsf{LGC}}.\mathsf{gsk}, \mathsf{eqk}))$.

Upon receiving the input x, the simulator computes $\langle x \rangle^{(q)} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{SimInpEnc}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, C, y = C(x))$ and outputs $(\mathsf{eqk}, \langle x \rangle^{(q)})$.

Proof of Security. We now present the hybrids that prove that Π_{LGC} is adaptively secure with simulator $\mathsf{Sim}_{\mathsf{AdGC}}$. We consider the following hybrid security games.

[23] Recall that without loss of generality, we only attempt to hide the input in this construction.

- Hyb_0: This corresponds to the real experiment. The adversary adaptively queries circuit C and input x and in turn receives $\langle C\rangle$ and $\langle x\rangle$. Both the garbled circuit and the input encoding are generated honestly.
- $\mathsf{Hyb}_{\mathbf{k}}$: for each $\mathbf{k} \in [q]$ within the simulation strategy of Π_{LGC}, we define a corresponding hybrid security game. In this game, the circuit encoding $\langle C\rangle$ is computed in the following way:
  - Sample $\Pi_{\mathsf{LGC}}.\mathsf{gsk} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$ and first compute $(\langle g_i\rangle^{(\mathbf{k})})_{i\notin T_{\mathbf{k}}}$ by sampling $\langle g_i\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C)$.
  - Sample $(\mathsf{CT}, \mathsf{st}_0) \leftarrow \mathsf{SimEnc}(1^\lambda, T_{\mathbf{k}}, (\langle g_i\rangle^{(\mathbf{k})})_{i\notin T_{\mathbf{k}}})$.
  - Output $(\langle C\rangle = \mathsf{CT}, \mathsf{st} = (\Pi_{\mathsf{LGC}}.\mathsf{gsk}, \mathsf{st}_0))$.

  Upon receiving the input x, the input encoding is then computed in the following way
  - Compute $\langle x\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, C, x)$.
  - For each $i \in T_{\mathbf{k}}$, sample $\langle g_i\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C, x)$.
  - Compute $\mathsf{eqk}' \leftarrow \mathsf{SimKey}(\mathsf{st}_0, \{\langle g_i\rangle^{(\mathbf{k})}\}_{i\in T_{\mathbf{k}}})$.
  - Output $(\mathsf{eqk}', \langle x\rangle^{(\mathbf{k})})$.

By construction, Hyb_q is exactly the simulated distribution, so it suffices to show that $\mathsf{Hyb}_{\mathbf{k}}$ is computationally indistinguishable from $\mathsf{Hyb}_{\mathbf{k}+1}$ for every $\mathbf{k} \in [q]$. We do this using two additional intermediate hybrids.

- $\mathsf{Hyb}_{\mathbf{k},1}$: In this game, the circuit encoding $\langle C\rangle$ is computed in the following way:
  - Sample $\Pi_{\mathsf{LGC}}.\mathsf{gsk} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$ and first compute $(\langle g_i\rangle^{(\mathbf{k})})_{i\notin T_{\mathbf{k}}\cup T_{\mathbf{k}+1}}$ by sampling $\langle g_i\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C)$.
  - Sample $(\mathsf{CT}, \mathsf{st}_0) \leftarrow \mathsf{SimEnc}(1^\lambda, T_{\mathbf{k}} \cup T_{\mathbf{k}+1}, (\langle g_i\rangle^{(\mathbf{k})})_{i\notin T_{\mathbf{k}}\cup T_{\mathbf{k}+1}})$.
  - Output $(\langle C\rangle = \mathsf{CT}, \mathsf{st} = (\Pi_{\mathsf{LGC}}.\mathsf{gsk}, \mathsf{st}_0))$.

  Upon receiving the input x, the input encoding is then computed in the following way
  - Compute $\langle x\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, C, x)$.
  - For each $i \in T_{\mathbf{k}}$, sample $\langle g_i\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C, x)$.
  - For each $i \in T_{\mathbf{k}+1}\backslash T_{\mathbf{k}}$, sample $\langle g_i\rangle^{(\mathbf{k})} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C)$.
  - Compute $\mathsf{eqk}' \leftarrow \mathsf{SimKey}(\mathsf{st}_0, \{\langle g_i\rangle^{(\mathbf{k})}\}_{i\in T_{\mathbf{k}}\cup T_{\mathbf{k}+1}})$.
  - Output $(\mathsf{eqk}', \langle x\rangle^{(\mathbf{k})})$.
- $\mathsf{Hyb}_{\mathbf{k},2}$: In this game, the circuit encoding $\langle C\rangle$ is computed in the following way:
  - Sample $\Pi_{\mathsf{LGC}}.\mathsf{gsk} \leftarrow \Pi_{\mathsf{LGC}}.\mathsf{Gen}(1^\lambda, 1^s, 1^n, 1^m)$ and first compute $(\langle g_i\rangle^{(\mathbf{k}+1)})_{i\notin T_{\mathbf{k}}\cup T_{\mathbf{k}+1}}$ by sampling $\langle g_i\rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C)$.
  - Sample $(\mathsf{CT}, \mathsf{st}_0) \leftarrow \mathsf{SimEnc}(1^\lambda, T_{\mathbf{k}} \cup T_{\mathbf{k}+1}, (\langle g_i\rangle^{(\mathbf{k}+1)})_{i\notin T_{\mathbf{k}}\cup T_{\mathbf{k}+1}})$.
  - Output $(\langle C\rangle = \mathsf{CT}, \mathsf{st} = (\Pi_{\mathsf{LGC}}.\mathsf{gsk}, \mathsf{st}_0))$.

  Upon receiving the input x, the input encoding is then computed in the following way
  - Compute $\langle x\rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimInpEnc}_{\mathbf{k}+1}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, C, x)$.
  - For each $i \in T_{\mathbf{k}+1}$, sample $\langle g_i\rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\Pi_{\mathsf{LGC}}.\mathsf{gsk}, i, C, x)$.
  - For each $i \in T_{\mathbf{k}}\backslash T_{\mathbf{k}+1}$, sample $\langle g_i\rangle^{(\mathbf{k}+1)} \leftarrow \mathsf{SimGtEnc}_{\mathbf{k}+1}(\mathsf{gsk}, i, C)$.
  - Compute $\mathsf{eqk}' \leftarrow \mathsf{SimKey}(\mathsf{st}_0, \{\langle g_i\rangle^{(\mathbf{k}+1)}\}_{i\in T_{\mathbf{k}}\cup T_{\mathbf{k}+1}})$.
  - Output $(\mathsf{eqk}', \langle x\rangle^{(\mathbf{k})})$.

The indistinguishability of $\mathsf{Hyb}_{\mathbf{k}} \cong_c \mathsf{Hyb}_{\mathbf{k},1}$ follows directly from the security of sEQ. The indistinguishability of $\mathsf{Hyb}_{\mathbf{k},1} \cong_c \mathsf{Hyb}_{\mathbf{k},2}$ follows directly from the semi-adaptive security of Π_{LGC}. Finally, the indistinguiahbility of $\mathsf{Hyb}_{\mathbf{k},2} \cong_c \mathsf{Hyb}_{\mathbf{k}+1}$ follows directly from the security of sEQ. This completes the proof of security.

Acknowledgements

We thank Vinod Vaikuntanathan for useful discussions.

References

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *Annual Cryptology Conference*, pages 657–677. Springer, 2015.

[ACC+16] Prabhanjan Ananth, Yu-Chi Chen, Kai-Min Chung, Huijia Lin, and Wei-Kai Lin. Delegating ram computations with adaptive soundness and privacy. In *Theory of Cryptography Conference*, pages 3–30. Springer, 2016.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 166–175, 2004.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Computational Complexity*, 15(2):115–162, 2006.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *International Colloquium on Automata, Languages, and Programming*, pages 152–163. Springer, 2010.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Annual Cryptology Conference*, pages 308–326. Springer, 2015.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. *Eprint*, 730:2015, 2015.

[AJS17a] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for turing machines: Constant overhead and amortization. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 252–279, 2017.

[AJS17b] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Patchable indistinguishability obfuscation: io for evolving software. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–155. Springer, 2017.

[App14a] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 162–172. Springer, 2014.

[App14b] Benny Applebaum. Key-dependent message security: Generic amplification and completeness. *Journal of Cryptology*, 27(3):429–451, 2014.

[AS16] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *Theory of Cryptography Conference*, pages 125–153. Springer, 2016.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 345–356. ACM, 2016.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Siddartha Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 423–444. Springer, 2010.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 500–532. Springer, 2018.

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. In *Advances in Cryptology - EUROCRYPT 2018*, 2018.

[BP15] Nir Bitansky and Omer Paneth. Zaps and non-interactive witness indistinguishability from indistinguishability obfuscation. In *Theory of Cryptography Conference*, pages 401–427. Springer, 2015.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 171–190. IEEE, 2015.

[CCC+16] Yu-Chi Chen, Sherman SM Chow, Kai-Min Chung, Russell WF Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel ram from indistinguishability obfuscation. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 179–190. ACM, 2016.

[CCHR16] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Adaptive succinct garbled ram or: How to delegate your database. In *Theory of Cryptography Conference*, pages 61–90. Springer, 2016.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In *Annual International Cryptology Conference*, pages 33–65. Springer, 2017.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled ram. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 169–178. ACM, 2016.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and RAM programs. In *STOC*, 2015.

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In *Annual Cryptology Conference*, pages 287–307. Springer, 2015.

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *Annual International Cryptology Conference*, pages 537–569. Springer, 2017.

[DGHM18] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In *IACR International Workshop on Public Key Cryptography*. Springer, 2018.

[DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 124–153. Springer, 2016.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled ram revisited. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 405–422. Springer, 2014.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Annual Cryptology Conference*, pages 155–172. Springer, 2010.

[GKP+12] Shafi Goldwasser, Yael Tauman Kalai, Raluca A Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. *IACR Cryptology ePrint Archive*, 2012:733, 2012.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. One-time programs. In *Annual International Cryptology Conference*, pages 39–56. Springer, 2008.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 449–458. ACM, 2015.

[GP17] Sanjam Garg and Omkant Pandey. Incremental program obfuscation. In *Annual International Cryptology Conference*, pages 193–223. Springer, 2017.

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In *Annual Cryptology Conference*, pages 579–604. Springer, 2016.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 156–181. Springer, 2017.

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In *Theory of Cryptography Conference*, pages 419–442. Springer, 2016.

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round mpc from bilinear maps. *FOCS 2017*, 2017.

[GS18a] Sanjam Garg and Akshayaram Srinivasan. Adaptively secure garbling with near optimal online complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 535–565. Springer, 2018.

[GS18b] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 468–499. Springer, 2018.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 162–179, 2012.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In *CRYPTO*, 2016.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Annual Cryptology Conference*, pages 132–150. Springer, 2011.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 163–172. ACM, 2015.

[JSW17] Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. In *Theory of Cryptography Conference*, pages 40–71. Springer, 2017.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of yao's garbled circuits. In *TCC*, 2016.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, 2015.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[LT17] Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local prgs. In *Annual International Cryptology Conference*, pages 630–660. Springer, 2017.

[LZ17] Qipeng Liu and Mark Zhandry. Exploding obfuscation: A framework for building applications of obfuscation from polynomial hardness. *IACR Cryptology ePrint Archive*, 2017:209, 2017.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 463–472. ACM, 2010.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

[Yao82] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A Selectively Secure Laconic OT from Indistinguishability Obfuscation

In this section, we give a construction of (selectively secure) laconic OT from IO and one-way functions. This provides, for the sake of completeness, a construction of succinct garbling

schemes from IO and OWFs without additional assumptions. Before the construction, we make two remarks on barriers to improving the results.

- The construction in this section cannot obviously be modified to rely on the polynomial security of compact functional encryption. Indeed, we are not even aware of any construction of oblivious transfer (a primitive that is implied by laconic OT using the techniques of [BLSV18]) from poly-secure FE.
- We are inherently limited to constructing *selectively secure* laconic OT; in particular, in an adaptively secure scheme, the hash function $\mathsf{lacOT.H}(\mathsf{crs}, \cdot)$ is a collision-resistant hash function, and there are substantial barriers [AS16] to constructing CRHFs from IO and OWFs.

We now proceed with the construction, which is a modification of the [CDG+17] construction of laconic OT from witness encryption and somewhere statistically binding (SSB) hash functions. SSB hash functions are in particular collision-resistant, but we note that to obtain selective security it suffices to start with a weaker notion of *input-dependent SSB hash functions*, which we define below.

Definition 14 (Input-dependent SSB Hash Functions)**.** *A family of input-dependent somewhere statistically binding hash functions* $\mathsf{SSB} = (\mathsf{Gen}, \mathsf{H}, \mathsf{BindingGen})$ *(with factor* 2 *compression) is specified by three algorithms.*

- $\mathsf{Gen}(1^\lambda)$ *is a randomized algorithm that samples a hash key* k*.*
- $\mathsf{H}(k, x)$ *is a deterministic algorithm that takes as input a hash key* k *and an input* $x \in \{0,1\}^{2\lambda}$*. It outputs a hash value* $h \in \{0,1\}^\lambda$*.*
- $\mathsf{BindingGen}(1^\lambda, x^*, i^*)$ *is a randomized algorithm that takes as input a string* $x^* \in \{0,1\}^{2\lambda}$ *and an index* $i^* \in [2\lambda]$*. It samples a hash key* k*.*

Such a scheme must satisfy the following correctness and security properties.

- ***Correctness (Binding)****: For all* $\lambda \in \mathbb{N}$*, all* $x^* \in \{0,1\}^{2\lambda}$*, and all* $i^* \in [2\lambda]$*, with all but negligible probabaility over the randomness of* $k \leftarrow \mathsf{BindingGen}(1^\lambda, x^*, i^*)$*, there exists no* $x \in \{0,1\}^{2\lambda}$ *such that* $\mathsf{H}(k, x) = \mathsf{H}(k, x^*)$ *and* $x_i \neq x^*_i$*.*
- ***Security****: There is a negligible function* $\mu(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*, all* $x^* \in \{0,1\}^{2\lambda}$*, and all* $i^* \in [2\lambda]$*, we have that the distributions* $\{k \leftarrow \mathsf{Gen}(1^\lambda)\}$ *and* $\{k \leftarrow \mathsf{BindingGen}(1^\lambda, x^*, i^*)\}$ *are* $\mu(\lambda)$*-indistinguishable.*

We first note that input-dependent SSB hash functions can be constructed from IO and one-way functions via a standard puncturing arugment.

Lemma 8. *If indistinguishability obfuscation and one-way functions exist, then so do input-dependent SSB hash functions.*

Proof. Let iO denote an IO scheme, let $G : \{0,1\}^{\lambda/2} \to \{0,1\}^\lambda$ denote a PRG, and let PRF denote a puncturable PRF family with domain $\{0,1\}^{2\lambda}$ and range $\{0,1\}^{\lambda/2}$. We define our hash family $\mathsf{SSB} = (\mathsf{SSB.Gen}, \mathsf{SSB.H}, \mathsf{SSB.BindingGen})$ as follows.

- $\mathsf{SSB.Gen}(1^\lambda)$ samples a PRF key K and outputs an obfuscation $\tilde{P} \leftarrow \mathsf{iO}(P_K)$ of the program P_K defined in Fig. 8.
- $\mathsf{SSB.H}(k, x)$ interprets k as an obfuscated program and evaluates k on input x.
- $\mathsf{SSB.BindingGen}(1^\lambda, x^*, i^*)$ samples a PRF key K and punctured key $K\{x^*\}$, samples a string $r \leftarrow \{0,1\}^\lambda$ uniformly at random, and outputs an obfuscation $\tilde{P}_{x^*,r} \leftarrow \mathsf{iO}(P_{K\{x^*\},x^*,r})$ of the program $P_{K\{x^*\},x^*,r}$ defined in Fig. 9.

Input: string $x \in \{0,1\}^{2\lambda}$
Hardwired: PRF key K

- Output $G(\mathsf{PRF}(K, x))$.

Figure 8: Description of P_K.

Input: string $x \in \{0,1\}^{2\lambda}$
Hardwired: Punctured key $K\{x^*\}$, input x^*, output r

- If $x = x^*$, output r.
- If $x \neq x^*$, output $G(\mathsf{PRF}(K\{x^*\}, x))$.

Figure 9: Description of $P_{K\{x^*\},x^*,r}$.

The scheme satisfies binding because with all but negligible probability over the choice of $r \leftarrow \{0,1\}^\lambda$, r is not in the range of the PRG G, and hence *no input* x other than x^* will satisfy $\mathsf{H}(k, x) = \mathsf{H}(k, x^*)$ for $k = \mathsf{iO}(P_{K\{x^*\},x^*,r})$.

The scheme is secure by a standard puncturing argument; namely, an obfuscation of P_K is computationally indistinguishable from an obfuscation of the program $P_{K\{x^*\},x^*,G(\mathsf{PRF}(K,x^*))}$ (by the security of iO), which is in turn computationally indistinguishable from an obfuscation of $P_{K\{x^*\},x^*,r}$ for uniformly random r (by the security of the puncturable PRF family and the PRG). □

Next, we see that the [CDG+17] construction carries over to our setting, yielding a construction of selectively secure laconic OT with factor 2 compression from witness encryption and input-dependent SSB hash functions.

Lemma 9. *If witness encryption and input-dependent SSB hash functions exist, then so does laconic OT with factor* 2 *compression.*

Proof. Given witness encryption scheme $\mathsf{WE} = (\mathsf{WE.Enc}, \mathsf{WE.Dec})$ and input-dependent SSB hash family $\mathsf{SSB} = (\mathsf{SSB.Gen}, \mathsf{SSB.H})$, we construct laconic OT with factor 2 compression as follows.

- $\mathsf{lacOT.Gen}(1^\lambda)$ samples $\mathsf{crs} \leftarrow \mathsf{SSB.Gen}(1^\lambda)$ and outputs crs.
- $\mathsf{lacOT.H}(\mathsf{crs}, D)$ computes $h = \mathsf{SSB.H}(\mathsf{crs}, D)$ and outputs (h, D).
- $\mathsf{lacOT.Send}(\mathsf{crs}, h, i, m_0, m_1)$ outputs two ciphertexts: $\mathsf{CT}_0 \leftarrow \mathsf{WE.Enc}(\mathcal{R}, (\mathsf{crs}, h, i, 0), m_0)$ and $\mathsf{CT}_1 \leftarrow \mathsf{WE.Enc}(\mathcal{R}, (\mathsf{crs}, h, i, 1), m_1)$, where $\mathcal{R}$ is the relation with instance $x = (\mathsf{crs}, h, i, b)$, witness w, and satisfies $R(x, w) = 1$ if and only if $\mathsf{SSB.H}(\mathsf{crs}, w) = h$ and $w_i = b$.
- $\mathsf{lacOT.Receive}(\mathsf{crs}, D, \mathsf{CT}_0, \mathsf{CT}_1)$ calls $m = \mathsf{WE.Dec}(\mathcal{R}, D, \mathsf{CT}_{D[i]})$ and outputs m.

Correctness of the above scheme follows from the correctness of the underlying witness encryption scheme.

The proof of sender privacy is as follows: by the security of SSB, we know that for every $D \in \{0,1\}^{2\lambda}$, $i \in [\lambda]$, and pair of messages (m_0, m_1),

$$(\mathsf{crs}, h, D, \mathsf{lacOT.Send}(\mathsf{crs}, h, i, m_0, m_1)) \cong_c \left(\widetilde{\mathsf{crs}}, \tilde{h}, D, \mathsf{lacOT.Send}(\widetilde{\mathsf{crs}}, \tilde{h}, i, m_0, m_1)\right),$$

where $\mathsf{crs} \leftarrow \mathsf{SSB.Gen}(1^\lambda)$, $h = \mathsf{lacOT.H}(\mathsf{crs}, D)$, $\widetilde{\mathsf{crs}} \leftarrow \mathsf{SSB.BindingGen}(1^\lambda, D, i)$, and $\tilde{h} = \mathsf{lacOT.H}(\widetilde{\mathsf{crs}}, D)$, by the security of SSB. Moreover, we have that

$$\left(\widetilde{\mathsf{crs}}, \tilde{h}, D, \mathsf{lacOT.Send}(\widetilde{\mathsf{crs}}, \tilde{h}, i, m_0, m_1)\right) \cong_c \left(\widetilde{\mathsf{crs}}, \tilde{h}, D, \mathsf{lacOT.Send}(\widetilde{\mathsf{crs}}, \tilde{h}, i, m_{D_i}, m_{D_i})\right)$$

by the security of the witness encryption scheme. This is because by the binding property of SSB, we know that with probability $1 - \mathsf{negl}(\lambda)$, there exists no D' such that $\mathsf{SSB.H}(\widetilde{\mathsf{crs}}, D') = \mathsf{SSB.H}(\widetilde{\mathsf{crs}}, D)$ and $D'_i = D_i \oplus 1$, so the WE ciphertext corresponding to $m_{D_i \oplus 1}$ hides $m_{D_i \oplus 1}$. Finally, we have that

$$\left(\widetilde{\mathsf{crs}}, \tilde{h}, D, \mathsf{lacOT.Send}(\widetilde{\mathsf{crs}}, \tilde{h}, i, m_{D_i}, m_{D_i})\right) \cong_c (\mathsf{crs}, h, D, \mathsf{lacOT.Send}(\mathsf{crs}, h, i, m_{D_i}, m_{D_i}))$$

by the security of SSB, completing the proof of sender privacy for lacOT. □

Finally, we note that the boostrapping theorem of [CDG+17] applies to the setting of selectively secure laconic OT as well, if an independently sampled laconic OT CRS is sampled for each level of the Merkle tree (rather than re-using the same hash key over and over again). This yields a selectively secure laconic OT scheme with arbitrary compression, completing the construction of laconic OT from IO and one-way functions.