Adaptively Secure Garbling with Near Optimal Online Complexity*

Sanjam Garg
University of California, Berkeley
sanjamg@berkeley.com

Akshayaram Srinivasan
University of California, Berkeley
akshayaram@berkeley.edu

Abstract

We construct an adaptively secure garbling scheme with an online communication complexity of $n + m + \mathsf{poly}(\log |C|, \lambda)$ where $C : \{0,1\}^n \to \{0,1\}^m$ is the circuit being garbled, and λ is the security parameter. The security of our scheme can be based on (polynomial hardness of) the Computational Diffie-Hellman (CDH) assumption, or the Factoring assumption or the Learning with Errors assumption. This is nearly the best achievable in the standard model (i.e., without random oracles) as the online communication complexity must be larger than both n and m. The online computational complexity of our scheme is $O(n + m) + \mathsf{poly}(\log |C|, \lambda)$. Previously known standard model adaptively secure garbling schemes had asymptotically worse online cost or relied on exponentially hard computational assumptions.

1 Introduction

Introduced in the seminal work of Yao [Yao86], garbling techniques are one of the main cornerstones of cryptography. Garbling schemes have found numerous applications in multiparty computation [Yao86, AF90, BMR90], parallel cryptography [AIK04, AIK05], one-time programs [GKR08], verifiable computation [GGP10, AIK10], functional encryption [SS10, GVW12, GKP+13], efficient zero-knowledge proofs [JKO13, FNO15] and program obfuscation [App14, LV16].

Garbling a circuit C and an input x yields a garbled circuit $\widetilde{C}$ and a garbled input $\widetilde{x}$ respectively. Next, using $\widetilde{C}$ and $\widetilde{x}$ anyone can efficiently compute $C(x)$ but security requires that $\widetilde{C}$ and $\widetilde{x}$ jointly reveal nothing about C or x beyond $C(x)$. Typical garbling schemes are only proved to satisfy the weaker notion of *selective security* where both the circuit C and the input x are chosen a priori. However, in certain applications, a stronger notion of *adaptive security* wherein the input x can be chosen adaptively based on the garbled circuit $\widetilde{C}$ is needed [BHR12a]. We refer to the size of $\widetilde{C}$ as the *offline* communication complexity and the size of $\widetilde{x}$ as the *online* communication complexity.

Constructing such adaptively secure garbling schemes with better online communication cost has been an active area of investigation [BHR12a, BGG+14, HJO+16, JW16, AS16, JKK+17, JSW17]. Despite tremendous effort, all standard model constructions of adaptively secure garbling which are based on polynomially hard assumptions have online communication cost that grows with the width of the circuit.

*Research supported in part from 2017 AFOSR YIP Award, DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

1.1 Our Contributions

We obtain a new adaptive garbling scheme with online communication complexity of $n + m + \mathsf{poly}(\log |C|, \lambda)$ where n is the input length of the circuit C, m is its output length and λ is the security parameter. This almost matches the lower bounds of n and m due to Applebaum et al. [AIKW13].[1] Moreover, this complexity is very close to the best known constructions for the selective security setting [AIKW13]. More formally, our main result is:

Theorem 1.1 *Assuming either the Computational Diffie-Hellman assumption or the Factoring assumption or the Learning with Errors assumption, there exists a construction of adaptive garbling scheme with online communication complexity of $n + m + \mathsf{poly}(\log |C|, \lambda)$ with simulation security.*

All prior constructions of adaptively secure garbling schemes in the standard model had online communication complexity that grew with either the circuit depth/width. Moreover, several of these schemes suffered from an exponential loss in security reduction. We summarize the known constructions and our new results in Table 1.

	Assumption	Online Communication Complexity	Security Loss	Model
[BHR12a] Const. 1	OWF	$n\lambda$	$\mathsf{poly}(\|C\|, \lambda)$	RO
[BHR12a] Const. 2	OWF	$\|C\| + n\lambda$	$\mathsf{poly}(\|C\|, \lambda)$	Std.
[BGG+14] Const.1	LWE	$(n+m)\mathsf{poly}(\lambda, d)$	$2^{O(d)}$	Std.
[BGG+14] Const.2	LWE + MDDH	$O(n+m) + \mathsf{poly}(\lambda, d)$	$2^{O(d)}$	Std.
[HJO+16] Const. 1	OWF	$(n+m+w)\mathsf{poly}(\lambda)$	$\mathsf{poly}(\|C\|, \lambda)$	Std.
[HJO+16] Const. 2	OWF	$(n+m+d)\mathsf{poly}(\lambda)$	$2^{O(d)}$	Std.
[JW16]	OWF	$(n+m+d)\mathsf{poly}(\lambda)$	$2^{O(d)}$	Std.
[JKK+17]	OWF	$(n+m+d)\mathsf{poly}(\lambda)$	$2^{O(d)}$	Std.
This work	CDH/Factoring/LWE	$n + m + \mathsf{poly}(\lambda, \log \|C\|)$	$\mathsf{poly}(\|C\|, \lambda)$	Std.

Table 1: Constructions of known and new adaptive garbling schemes (with simulation security).

Additionally, we note that as a special case, our result implies selectively secure garbling scheme with online cost $n + \mathsf{poly}(\lambda)$ from the same assumptions. Previously, this result was not known under CDH or Factoring. Specifically, constructions were known from DDH or RSA [AIKW13].

1.2 Applications

We now mention some of the applications of our result. These applications were already noted in the work of Hemenway et al. [HJO+16] and we improve their efficiency.

One-time Program and Verifiable Computation. Plugging our result in the one-time program construction of [GKR08], we get a construction of one-time program where the number of hardware tokens is $O(n + m + \mathsf{poly}(\lambda, \log |C|))$. Similarly, the running time of verification protocol in the work of [GGP10] can be improved to match our online complexity.

[1] In this work, we consider the standard simulation based security notion. Indeed, if one considers the weaker notion of indistinguishablity based security this lower bound can be bypassed as shown in [AS16, JSW17].

Compact Functional Encryption. Starting with a single-key, selective functional encryption scheme with weakly compact ciphertexts and using the transformations of [ABSV15, AS16, GS16, LM16] along with our construction of adaptively secure garbled circuits, we obtain a multi-key secure, adaptive functional encryption scheme whose ciphertext size grows only with the output size of the functions.

2 Our Techniques

In this section, we outline the main techniques and tools used in the construction of adaptively secure garbled circuits.

Adaptive Security Game. Before explaining our construction, let us first explain the adaptive security game in a bit more detail. In this game, the adversary provides the challenger with a circuit C and the challenger responds with a garbled circuit $\widetilde{C}$. The adversary later provides with an input x (that could potentially depend on $\widetilde{C}$) and the challenger responds with garbled input $\widetilde{x}$. In the real world, both the garbled circuit and the garbled input are generated honestly whereas in the ideal world, the garbled circuit $\widetilde{C}$ is generated by a simulator Sim_1 that is given the size of C as input and the garbled input $\widetilde{x}$ is generated by another simulator Sim_2 that is given $C(x)$ as input. The goal of the adversary is to distinguish between the real world and the ideal world distributions.

The reason why the proof of Yao's construction breaks down in the adaptive setting is because the distribution of the garbled circuit $\widetilde{C}$ in the intermediate hybrids depends on the value of the (adversarily chosen) input x. Naturally, Yao's approach is not feasible when the garbled circuit needs to be sent before the adversary gives its input x.

Prior Approaches. To solve the issue with Yao's construction, Bellare et al. [BHR12b] encrypted the garbled circuit by an (fully) equivocal encryption scheme and sent the ciphertext in the offline phase. Later, in the online phase, the key for decrypting this ciphertext was provided. Since an equivocal ciphertext can be opened to any value, the simulator in each intermediate hybrid opens the ciphertext sent in the offline phase to an appropriate simulated value (that depends on C and x). However, the key size for an equivocal encryption scheme in the standard model has to grow with the size of the message [Nie02] and in this case it grows with the size of the circuit. Thus, the online complexity of this approach has to grow with the size of the circuit.

The work of Hemenway, Jafargholi, Ostrovsky, Scafuro and Wichs [HJO$^+$16] improved the online complexity by replacing the fully equivocal encryption scheme with a *somewhere equivocal encryption*. Roughly speaking, a somewhere equivocal encryption allows to generate a ciphertext encrypting a vector of messages with "holes" in some positions. Later, these "holes" could be filled with arbitrary message values by deriving a suitable decryption key. Intuitively, in each intermediate hybrid, "holes" are created in the garbled circuit in those positions that depend on the input and the simulator fills these "holes" in the online phase based on the input x. The crucial aspect of a somewhere equivocal encryption is that its key size is only proportional to number of holes which could be much smaller than the total length of the message vector. Thus to minimize the online complexity, it is sufficient to come up with a sequence of hybrids where the number of holes in each intermediate hybrid is minimized. Hemenway et al. provide two sequences of hybrid arguments: the first sequence where the number of "holes" in each hybrid is at most the width

of the circuit and the second sequence of hybrids where the number of "holes" in each hybrid is at most the depth (with $2^{O(depth)}$ hybrids). However, even in this approach the online complexity could be as large as the circuit size as the circuit width or depth could be as large as the circuit itself.

Our approach. We follow the high level idea of Hemenway et al. [HJO$^+$16] in encrypting the garbled circuit using a somewhere equivocal encryption but employ a *crucial trick* to minimize the number of "holes" in each intermediate hybrid. At a very high level, we use the recent construction of updatable laconic oblivious transfer [CDG$^+$17, DG17, DGHM18, BLSV18] (which can be constructed based either on CDH/Factoring/LWE) to "linearize" the garbled circuit. Informally, a garbled circuit is "linearized" if the simulation of a garbled gate g depends *only* on simulating one additional gate. We note that all the prior approaches [HJO$^+$16, JW16] resulted in "non-linearized" garbled circuits. In particular, in all the prior works, simulating the garbled gate g depended on simulating *all* gates that provide inputs to g (which are at least two in number). With this "linearization" in place, we design a sequence of hybrids (based on the pebbling strategy of [Ben89]) where the number of "holes" in each intermediate hybrid is $O(\log(|C|))$. This allows us to achieve nearly optimal online complexity. We elaborate on our approach in the next subsection.

2.1 Our Approach: "Linearizing" the Garbled Circuit

We now explain our construction of "linearized" garbled circuits.

Step Circuits. To understand our construction, it is best to view the circuit C as a *sequence of step circuits*. In more details, we will consider C as a sequence of step circuits along with a database/memory D. For simplicity, we consider a circuit with a single output bit. The i-th step circuit implements the i-th gate (with some topological ordering of the gates) in the circuit C. The database D is initially loaded with the input x and contents of the database represent the state of the computation. That is, the snapshot of the database before the evaluation of the i-th step circuit contains the output of every gate $g < i$ in the execution of C on input x. The i-th step circuit reads contents from two pre-determined locations in the database and writes a bit to location i. The bits that are read correspond to the values in the input wires for the i-th gate. The output of the circuit is easily derived from the contents of the database at the end of the computation. To garble the circuit C, we must garble each of the step circuits and the database D.

Garbling Step Circuits. Our approach of garbling the step circuits involves a primitive called as updatable laconic oblivious transfer [CDG$^+$17]. To make the exposition easy, we first consider a simplistic setting where the database D is not protected i.e., it is revealed in the clear to the adversary. We will later explain how this restriction can be removed.

A laconic oblivious transfer is a protocol between two parties: sender and a receiver. The receiver holds a large database $D \in \{0,1\}^N$ and sends a short digest d (with length λ) of the database to the sender. The sender obtains as input a location $L \in [N]$ and two messages m_0, m_1. The sender computes a read-ciphertext c using his private inputs and the received digest d by running in time $\mathsf{poly}(\log N, |m_0|, |m_1|, \lambda)$ and sends c to the receiver. Note that the time required to compute the read-ciphertext c grows logarithmically with the size of the database. The receiver recovers the message $m_{D[L]}$ from the ciphertext c and the security requirement is that the message $m_{1-D[L]}$ is

computationally hidden. A laconic oblivious transfer is said to be *updatable* if it additionally allows updates on the database. In particular, the sender on input a location $L \in [N]$, a bit b, digest d and a sequence of λ messages $\{m_{j,0}, m_{j,1}\}_{j\in[\lambda]}$ creates a write-ciphertext c_w (by running in time that grows logarithmically with the size of the database). The receiver on input c_w can recover $\{m_{j,\mathsf{d}^*_j}\}_{j\in[\lambda]}$ where d^* is the digest of the updated database with bit b written in location L. As in the previous case, the security requires that the messages $\{m_{j,1-\mathsf{d}^*_j}\}_{j\in[\lambda]}$ are computationally hidden. An updatable laconic oblivious transfer was first constructed in [CDG+17] from the Decisional Diffie-Hellman (DDH) problem and the assumptions were later improved to CDH/Factoring in [DG17] and to LWE in [DGHM18, BLSV18].

Let us now give details on how to use updatable laconic OT to garble the circuit C. At a very high level, the garbled circuit consists of a sequence of garbled augmented step circuits $\widetilde{\mathsf{SC}}'_1, \ldots, \widetilde{\mathsf{SC}}'_N$ and the garbled input consists of the labels for executing the first garbled step circuit $\widetilde{\mathsf{SC}}'_1$. These garbled step circuits are constructed in special way such that the output of the garbled step circuit $\widetilde{\mathsf{SC}}'_i$ can be used to derive the labels for executing the next garbled step circuit $\widetilde{\mathsf{SC}}'_{i+1}$. Thus, starting from $\widetilde{\mathsf{SC}}'_1$, we can evaluate every garbled step circuit in the sequence. Let us now give details on the internals of the augmented step circuits.

The i-th augmented step circuit SC'_i takes as input the digest d of the snapshot of database D before the evaluation of i-th gate and two bits α_i and β_i. The bits α_i and β_i correspond to the inputs to gate i in the evaluation of C. The augmented step circuit SC'_i additionally has the set of both labels for each input wire of $\widetilde{\mathsf{SC}}'_{i+1}$ hardwired in its description. We denote these labels by $\{\mathsf{lab}^{\mathsf{d}}_{j,0}, \mathsf{lab}^{\mathsf{d}}_{j,1}\}_{j\in[\lambda]}$ that correspond to the digest and $\{\mathsf{lab}^{\alpha}_0, \mathsf{lab}^{\alpha}_1\}$ and $\{\mathsf{lab}^{\beta}_0, \mathsf{lab}^{\beta}_1\}$ that correspond to the input bits of gate $i+1$. SC'_i first computes the output of the i-th gate (denoted by γ) using α_i and β_i. This bit must be written to the database and the updated hash value must be fed to the next circuit SC'_{i+1}. Towards this goal, SC'_i computes a write-ciphertext c_w using the digest d, location i, bit γ and $\{\mathsf{lab}^{\mathsf{d}}_{j,0}, \mathsf{lab}^{\mathsf{d}}_{j,1}\}_{j\in[\lambda]}$. This write-ciphertext will be used to derive the labels corresponding to the updated value of the digest which is fed to SC'_{i+1}. Recall that SC'_{i+1} must also take in the input values to the $(i+1)^{th}$ gate of the circuit C. For this purpose, SC'_i also computes two read ciphertexts c_α, c_β using the value of the (updated) digest d^* and labels $\{\mathsf{lab}^{\alpha}_0, \mathsf{lab}^{\alpha}_1\}$ and $\{\mathsf{lab}^{\beta}_0, \mathsf{lab}^{\beta}_1\}$ respectively. These read ciphertexts will be used to derive the labels corresponding to the values of the input wires to the gate $i+1$. It finally outputs c_w, c_α, c_β. An evaluator for this garbled circuit can recover the set of labels for evaluating $\widetilde{\mathsf{SC}}'_{i+1}$ from these ciphertexts using the decryption functionality of updatable laconic OT.

Notice that in order to simulate the garbled step-circuit $\widetilde{\mathsf{SC}}'_i$, it is sufficient to simulate the garbled step-circuit $\widetilde{\mathsf{SC}}'_{i-1}$. This is because the labels for evaluating $\widetilde{\mathsf{SC}}_i$ are only hardwired in the step-circuit SC'_{i-1} and are not available anywhere else. Once the garbled step circuit $\widetilde{\mathsf{SC}}'_{i-1}$ is simulated, we can use the security of updatable laconic oblivious transfer and (plain) garbled circuits to simulate $\widetilde{\mathsf{SC}}'_i$. This helps us to achieve the right "linearized" structure for simulating the garbled step circuits.

Protecting the Database. In the above exposition, the database D is revealed in the clear which is clearly insecure as database holds the values of all the intermediate wires in the evaluation of the circuit. To protect the database, we mask the contents of the database with a random string. To be more precise, each step circuit additionally has the masking bits for the two input wires

and the masking bit for the output wire hardwired. When the step circuit is fed with the masked values of the input wires, it unmasks those values (using the hardwired masking bits) and computes the output of the gate. Finally, it uses the hardwired masking bit for the output wire to mask the output and uses this value to compute the updated digest. This trick of protecting the intermediate computation values using random masks is closely related to the "point and permute" construction of garbled circuits [BMR90, MNPS04].

Pebbling Game. As in the work of [HJO+16], we encrypt these garbled step circuits $\{\widetilde{\mathsf{SC}}'_i\}$ using a somewhere equivocal encryption scheme and send the ciphertext in the offline phase. Later in the online phase, we reveal the key for decrypting this ciphertext along with the labels for evaluating $\widetilde{\mathsf{SC}}'_1$. The task that remains is to come up with a sequence of hybrids such that the number of "holes" in each intermediate hybrid is minimized. Recall that a "hole" appears in a position that depends on the adaptively chosen input. To design a sequence of hybrids, we consider the following pebbling game.[2]

Consider a number line $1, 2, \ldots, N$. We are given some pebbles and we can place a pebble on the number line according the following rules:

- We can always place or remove a pebble from position 1.
- We can place or remove a pebble from position i if and only if there exists a pebble in position $i-1$.

The goal is to be place at position N by minimizing the number of pebbles (denoted as the *pebbling complexity*) present in the graph at any point of time. A trivial strategy would be to consecutively place pebbles starting from position 1 upto N. The maximum number of pebbles used is N and the hope is to have a strategy that uses far less pebbles.

Intuitively, a pebble in the above game corresponds to a "hole" in the somewhere equivocal ciphertext. Alternatively, we can view the process of placing a pebble in position i as simulating the i-th garbled circuit.[3] The above two rules naturally correspond to rules for simulating a garbled step circuit i.e., the first garbled step circuit can always be simulated and we can simulate the i-th garbled step circuit if the $(i-1)$-th garbled step circuit is simulated. Bennett [Ben89] showed an inductive pebbling strategy for the above game using $O(\log N)$ pebbles. This readily gives a sequence of hybrids to prove adaptive security where the number of "holes" in each intermediate hybrid is logarithmic in the size of the circuit. This helps us achieve nearly optimal online complexity.

Why is "linearization" important? The work of Hemenway et al. consider a pebbling game directly on the topology of the circuit rather than on a line graph. In more details, they interpreted the circuit C as a DAG with every gate in C being a node in the graph, the input gates represented as sources in the graph (nodes with in-degree 0) and output gate represented as sink (node with out-degree 0). The rules of the pebbling game are [4]:

[2]The pebble game we describe is a simplification of the actual pebbling game we design later. This simplification is sufficient to get the main intuition.

[3]These two views are equivalent since simulation of a garbled step circuit depends on the output of that step circuit which in turn depends on the adversarily chosen input x. Thus, if a garbled step circuit is simulated we must have a "hole" in the corresponding position of the somewhere equivocal encryption.

[4]For the sake of exposition, we give a simplified version of the pebbling game considered in the work of [HJO+16]. We refer the reader to their work for the full description.

1. A pebble can always be placed or removed from a source.

2. A pebble can be placed or removed from a node if all its predecessors have pebbles.

The goal is to place a pebble at the sink node by minimizing the number of pebbles placed in the graph at any point of time. Note that unlike our game, in order to place a pebble at a node it is required that pebbles are present on *all* the predecessors which are at least two in number. This makes the task of using logarithmic many pebbles extremely difficult and there are strong lower bounds [PTC76] concerning the pebbling complexity of the above game. In particular, the work of [PTC76] shows that existence of certain families of DAGs on n nodes (with in-degree 2 and out-degree more than 1) such that the pebbling complexity of those graphs is $\Omega(\frac{n}{\log n})$. This naturally corresponds to similar lower bounds on the pebbling complexity of circuits with fan-in 2 and fan-out greater than 1. Thus, to get around these lower bounds, the use of the "linearized" garbled circuit seems necessary.

Why Garbled RAM fails? To garble the step circuits and the database D, we could hope to use ideas from the garbled RAM literature [LO13, GHL$^+$14, GLOS15, GLO15].[5] This would have given us a garbling scheme based on just one-way functions instead of requiring public-key assumptions. However, all known approaches of constructing garbled RAM introduce additional dependencies in garbling step circuits. This implies that in order to garble a particular step circuit, at least two other step circuits must be garbled. Thus, the graph to be pebbled is no longer a straight line and the known lower bounds apply.

3 Preliminaries

Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$ there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$ we have $\mu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. For a probabilistic algorithm A, we denote $A(x; r)$ to be the output of A on input x with the content of the random tape being r. When r is omitted, $A(x)$ denotes a distribution. For a finite set S, we denote $x \leftarrow S$ as the process of sampling x uniformly from the set S. We will use PPT to denote Probabilistic Polynomial Time. We denote $[a]$ to be the set $\{1, \ldots, a\}$ and $[a, b]$ to be the set $\{a, a+1, \ldots, b\}$ for $a \leq b$ and $a, b \in \mathbb{Z}$. For a binary string $x \in \{0,1\}^n$, we will denote the i^{th} bit of x by x_i. We assume without loss of generality that the length of the random tape used by all cryptographic algorithms is λ. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial function.

3.1 Garbled Circuits

Below we recall the definition of garbling scheme for circuits [Yao82] with selective security (see Lindell and Pinkas [LP09] and Bellare et al. [BHR12b] for a detailed proof and further discussion). A garbling scheme for circuits is a tuple of PPT algorithms $(\mathsf{GarbleCkt}, \mathsf{EvalCkt})$. Very roughly, $\mathsf{GarbleCkt}$ is the circuit garbling procedure and $\mathsf{EvalCkt}$ the corresponding evaluation procedure. We use a formulation where input labels for a garbled circuit are provided as input to the garbling

[5] We in fact do not require the full power of garbled RAM as the locations that are accessed by each step circuit are fixed a priori.

procedure rather than generated as output. (This simplifies the presentation of our construction.) More formally:

- $\widetilde{\mathsf{C}} \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}\right)$: $\mathsf{GarbleCkt}$ takes as input a security parameter λ, a circuit C, and input labels $\mathsf{lab}_{w,b}$ where $w \in [n]$ ($[n]$ is the set of input wires to the circuit C) and $b \in \{0,1\}$. This procedure outputs a *garbled circuit* $\widetilde{\mathsf{C}}$. We assume that for each w, b, $\mathsf{lab}_{w,b}$ is chosen uniformly from $\{0,1\}^\lambda$.

- $y \leftarrow \mathsf{EvalCkt}\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in[n]}\right)$: Given a garbled circuit $\widetilde{\mathsf{C}}$ and a sequence of input labels $\{\mathsf{lab}_{w,x_w}\}_{w\in[n]}$ (referred to as the garbled input), $\mathsf{EvalCkt}$ outputs a string y.

Correctness. For correctness, we require that for any circuit C, input $x \in \{0,1\}^{|[n]|}$ and input labels $\{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}$ we have that:

$$\Pr\left[C(x) = \mathsf{EvalCkt}\left(\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in[n]}\right)\right] = 1$$

where $\widetilde{\mathsf{C}} \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}\right)$.

Selective Security. For security, we require that there exists a PPT simulator $\mathsf{Sim}_{\mathsf{Ckt}}$ such that for any circuit C and input $x \in \{0,1\}^{|[n]|}$, we have that

$$\left\{\widetilde{\mathsf{C}}, \{\mathsf{lab}_{w,x_w}\}_{w\in[n]}\right\} \overset{c}{\approx} \left\{\mathsf{Sim}_{\mathsf{Ckt}}\left(1^\lambda, 1^{|C|}, C(x), \{\mathsf{lab}_{w,x_w}\}_{w\in[n]}\right), \{\mathsf{lab}_{w,x_w}\}_{w\in[n]}\right\}$$

where $\widetilde{\mathsf{C}} \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}\right)$ and for each $w \in [n]$ and $b \in \{0,1\}$ we have $\mathsf{lab}_{w,b} \leftarrow \{0,1\}^\lambda$. Here $\overset{c}{\approx}$ denotes that the two distributions are computationally indistinguishable.

Theorem 3.1 ([Yao86, LP09]) *Assuming the existence of one-way functions, there exists a construction of garbling scheme for circuits.*

3.2 Updatable Laconic Oblivious Transfer

In this subsection, we recall the definition of updatable laconic oblivious transfer from [CDG+17] .

Definition 3.2 ([CDG+17]) *An updatable laconic oblivious transfer consists of the following algorithms:*

- $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$: *It takes as input the security parameter 1^λ (encoded in unary) and outputs a common reference string* crs.

- $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$: *It takes as input the common reference string* crs *and database $D \in \{0,1\}^*$ as input and outputs a digest* d *and a state $\widehat{D}$. We assume that the state $\widehat{D}$ also includes the database D.*

- $e \leftarrow \mathsf{Send}(\mathsf{crs}, \mathsf{d}, L, m_0, m_1)$: *It takes as input the common reference string* crs*, a digest* d*, a location $L \in \mathbb{N}$ and two messages $m_0, m_1 \in \{0,1\}^{p(\lambda)}$ and outputs a ciphertext e.*

- $m \leftarrow \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, e, L)$: *This is a RAM algorithm with random read access to* $\widehat{D}$*. It takes as input a common reference string* crs*, a ciphertext* e*, and a database location* $L \in \mathbb{N}$ *and outputs a message* m*.*

- $e_w \leftarrow \mathsf{SendWrite}(\mathsf{crs}, \mathsf{d}, L, b, \{m_{j,0}, m_{j,1}\}_{j=1}^{|\mathsf{d}|})$: *It takes as input the common reference string* crs*, a digest* d*, a location* $L \in \mathbb{N}$*, a bit* $b \in \{0,1\}$ *to be written, and* $|\mathsf{d}|$ *pairs of messages* $\{m_{j,0}, m_{j,1}\}_{j=1}^{|\mathsf{d}|}$*, where each* $m_{j,c}$ *is of length* $p(\lambda)$ *and outputs a ciphertext* e_w*.*

- $\{m_j\}_{j=1}^{|\mathsf{d}|} \leftarrow \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, L, b, e_w)$: *This is a RAM algorithm with random read/write access to* $\widehat{D}$*. It takes as input the common reference string* crs*, a location* L*, a bit* $b \in \{0,1\}$ *and a ciphertext* e_w*. It updates the state* $\widehat{D}$ *(such that* $D[L] = b$*) and outputs messages* $\{m_j\}_{j=1}^{|\mathsf{d}|}$*.*

We require an updatable laconic oblivious transfer to satisfy the following properties.

Correctness: *We require that for any database* D *of size at most* $M = \mathsf{poly}(\lambda)$*, any memory location* $L \in [M]$*, any pair of messages* $(m_0, m_1) \in \{0,1\}^{p(\lambda)}$ *where* $p(\cdot)$ *is a polynomial that*

$$\Pr\left[m = m_{D[L]} \;\middle|\; \begin{array}{cl} \mathsf{crs} & \leftarrow \mathsf{crsGen}(1^\lambda) \\ (\mathsf{d}, \widehat{D}) & \leftarrow \mathsf{Hash}(\mathsf{crs}, D) \\ e & \leftarrow \mathsf{Send}(\mathsf{crs}, \mathsf{d}, L, m_0, m_1) \\ m & \leftarrow \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, e, L) \end{array} \right] = 1,$$

Correctness of Writes: *Let database* D *be of size at most* $M = \mathsf{poly}(\lambda)$ *and let* $L \in [M]$ *be any memory location. Let* D^* *be a database that is identical to* D *except that* $D^*[L] = b$*. For any sequence of messages* $\{m_{j,0}, m_{j,1}\}_{j \in [\lambda]} \in \{0,1\}^{p(\lambda)}$ *we require that*

$$\Pr\left[\begin{array}{l} m'_j = m_{j,\mathsf{d}^*_j} \\ \forall j \in [|\mathsf{d}|] \end{array} \;\middle|\; \begin{array}{cl} \mathsf{crs} & \leftarrow \mathsf{crsGen}(1^\lambda) \\ (\mathsf{d}, \widehat{D}) & \leftarrow \mathsf{Hash}(\mathsf{crs}, D) \\ (\mathsf{d}^*, \widehat{D}^*) & \leftarrow \mathsf{Hash}(\mathsf{crs}, D^*) \\ e_w & \leftarrow \mathsf{SendWrite}\left(\mathsf{crs}, \mathsf{d}, L, b, \{m_{j,0}, m_{j,1}\}_{j=1}^{|\mathsf{d}|}\right) \\ \{m'_j\}_{j=1}^{|\mathsf{d}|} & \leftarrow \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, L, b, e_w) \end{array} \right] = 1,$$

Sender Privacy: *There exists a PPT simulator* $\mathsf{Sim}_{\ell\mathsf{OT}}$ *such that the for any non-uniform PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.,*

$$\left| \Pr[\mathsf{SenPrivExpt}^{\mathsf{real}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{SenPrivExpt}^{\mathsf{ideal}}(1^\lambda, \mathcal{A}) = 1] \right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{SenPrivExpt}^{\mathsf{real}}$ *and* $\mathsf{SenPrivExpt}^{\mathsf{ideal}}$ *are described in Figure 1.*

Sender Privacy for Writes: *There exists a PPT simulator* $\mathsf{Sim}_{\ell\mathsf{OTW}}$ *such that the for any non-uniform PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *s.t.,*

$$\left| \Pr[\mathsf{WriSenPrivExpt}^{\mathsf{real}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{WriSenPrivExpt}^{\mathsf{ideal}}(1^\lambda, \mathcal{A}) = 1] \right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{WriSenPrivExpt}^{\mathsf{real}}$ *and* $\mathsf{WriSenPrivExpt}^{\mathsf{ideal}}$ *are described in Figure 2.*

$\mathsf{SenPrivExpt}^{\mathsf{real}}[1^\lambda, \mathcal{A}]$	$\mathsf{SenPrivExpt}^{\mathsf{ideal}}[1^\lambda, \mathcal{A}]$
1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$.	1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$.
2. $(D, L, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$.	2. $(D, L, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$.
3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$.	3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$.
4. Output $\mathcal{A}_2(\mathsf{st}, \mathsf{Send}(\mathsf{crs}, \mathsf{d}, L, m_0, m_1))$.	4. Output $\mathcal{A}_2(\mathsf{st}, \mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D, L, m_{D[L]}))$.

Figure 1: Sender Privacy Security Game

$\mathsf{WriSenPrivExpt}^{\mathsf{real}}[1^\lambda, \mathcal{A}]$	$\mathsf{WriSenPrivExpt}^{\mathsf{ideal}}[1^\lambda, \mathcal{A}]$
1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$.	1. $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$.
2. $(D, L, b, \{m_{j,0}, m_{j,1}\}_{j\in[\lambda]}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$.	2. $(D, L, b, \{m_{j,0}, m_{j,1}\}_{j\in[\lambda]}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{crs})$.
3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$.	3. $(\mathsf{d}, \widehat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$.
	4. $(\mathsf{d}^*, \widehat{D}^*) \leftarrow \mathsf{Hash}(\mathsf{crs}, D^*)$ where D^* be a database that is identical to D except that $D^*[L] = b$.
4. $e_w \leftarrow \mathsf{SendWrite}(\mathsf{crs}, \mathsf{d}, L, b, \{m_{j,0}, m_{j,1}\}_{j=1}^{\lvert\mathsf{d}\rvert})$	5. $e_w \leftarrow \mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D, L, b, \{m_{j,\mathsf{d}^*_j}\}_{j\in[\lambda]})$
5. Output $\mathcal{A}_2(\mathsf{st}, e_w)$.	6. Output $\mathcal{A}_2(\mathsf{st}, e_w)$.

Figure 2: Sender Privacy for Writes Security Game

Efficiency: *The algorithm Hash runs in time* $|D|\mathsf{poly}(\log|D|, \lambda)$*. The algorithms* Send*,* SendWrite*,* Receive*,* ReceiveWrite *run in time* $\mathsf{poly}(\log|D|, \lambda)$*.*

Theorem 3.3 ([CDG$^+$17, DG17, BLSV18, DGHM18]) *Assuming either the Computational Diffie-Hellman assumption or the Factoring assumption or the Learning with Errors assumption, there exists a construction of updatable laconic oblivious transfer.*

Remark 3.4 *We note that the security requirements given in Definition 3.2 is stronger than the one in [CDG$^+$17] as we require the* crs *to be generated before the adversary provides the database D and the location L. However, the construction in [CDG$^+$17] already satisfies this definition since in the proof, we can guess the location by incurring a $1/D$ loss in the security reduction.*

3.3 Somewhere Equivocal Encryption

We now recall the definition of Somewhere Equivocal Encryption from the work of [HJO$^+$16].

Definition 3.5 ([HJO$^+$16]) *A somewhere equivocal encryption scheme with block-length s, message length n (in blocks) and equivocation parameter t (all polynomials in the security parameter) is a tuple of probabilistic polynomial algorithms* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ *such that:*

- $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$: *It is a PPT algorithm that takes as input the security parameter (encoded in unary) and outputs a key* key.
- $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, m_1 \dots m_n)$: *It is a PPT algorithm that takes as input a key* key *and a vector of messages* $\overline{m} = m_1 \dots m_n$ *with each* $m_i \in \{0,1\}^s$ *and outputs a ciphertext* $\overline{c}$.
- $\overline{m} \leftarrow \mathsf{Dec}(\mathsf{key}, \overline{c})$: *It is a deterministic algorithm that takes as input a key* key *and a ciphertext* $\overline{c}$ *and outputs a vector of messages* $\overline{m} = m_1 \dots m_n$.
- $(\mathsf{st}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$: *It is a PPT algorithm that takes as input a set of indices* $I \subseteq [n]$ *and a vector of messages* $(m_i)_{i \notin I}$ *and outputs a ciphertext* $\overline{c}$ *and a state* st.
- $\mathsf{key}' \leftarrow \mathsf{SimKey}(\mathsf{st}, (m_i)_{i \in I})$: *It is a PPT algorithm that takes as input the state information* st *and a vector of messages* $(m_i)_{i \in I}$ *and outputs a key* key'.

and satisfies the following properties:

Correctness. *For every* $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$, *for every* $\overline{m} \in \{0,1\}^{s \times n}$ *it holds that:*

$$\mathsf{Dec}(\mathsf{key}, \mathsf{Enc}(\mathsf{key}, \overline{m})) = \overline{m}$$

Simulation with No Holes. *We require that the distribution of* $(\overline{c}, \mathsf{key})$ *computed via* $(\mathsf{st}, \overline{c}) \leftarrow \mathsf{SimEnc}(\overline{m}, \emptyset)$ *and* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{st}, \emptyset)$ *to be identical to* $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $\overline{c} \leftarrow \mathsf{Enc}(\mathsf{key}, m_1 \dots m_n)$. *In other words, simulation when there are no holes (i.e.,* $I = \emptyset$*) is identical to honest key generation and encryption.*

Security. *For any PPT adversary* $\mathcal{A}$, *there exists a negligible function* $\nu = \nu(\lambda)$ *such that:*

$$\left| \Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 0) = 1] - \Pr[\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}(1^\lambda, 1) = 1] \right| \leq \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}$ *is defined as follows:*

Experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{A},\Pi}$

1. *The adversary* $\mathcal{A}$ *on input* 1^λ *outputs a set* $I \subseteq [n]$ *s.t.* $|I| < t$, *a vector* $(m_i)_{i \notin I}$, *and a challenge* $j \in [n] \setminus I$. *Let* $I' = I \cup \{j\}$.
2. - *If* $b = 0$, *compute* $\overline{c}$ *as follows:* $(\mathsf{st}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I}, I)$.
   - *If* $b = 1$, *compute* $\overline{c}$ *as follows:* $(\mathsf{st}, \overline{c}) \leftarrow \mathsf{SimEnc}((m_i)_{i \notin I'}, I')$.
3. *Send* $\overline{c}$ *to the adversary* $\mathcal{A}$.
4. *The adversary* $\mathcal{A}$ *outputs the set of remaining messages* $(m_i)_{i \in I}$.
   - *If* $b = 0$, *compute* key *as follows:* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{st}, (m_i)_{i \in I})$.
   - *If* $b = 1$, *compute* key *as follows:* $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{st}, (m_i)_{i \in I'})$
5. *Send* key *to the adversary.*

6. $\mathcal{A}$ outputs b' which is the output of the experiment.

Theorem 3.6 ([HJO+16]) *Assuming the existence of one-way functions, there exists a somewhere equivocal encryption scheme for any polynomial message-length n, black-length s and equivocation parameter t, having key size $t \cdot s \cdot \mathsf{poly}(\lambda)$ and ciphertext of size $n \cdot s \cdot \mathsf{poly}(\lambda)$ bits.*

3.4 Adaptive Garbled Circuits

We provide the definition of adaptive garbled circuits from [HJO+16].

Definition 3.7 *An adaptive garbling scheme for circuits is a tuple of PPT algorithms* (AdaGarbleCkt, AdaGarbleInp, *such that:*

- $(\widetilde{C}, \mathsf{st}) \leftarrow \mathsf{AdaGarbleCkt}(1^\lambda, C)$: *It is a PPT algorithm that takes as input the security parameter 1^λ (encoded in unary) and a circuit $C : \{0,1\}^n \to \{0,1\}^m$ as input and outputs a garbled circuit $\widetilde{C}$ and state information* st.
- $\widetilde{x} \leftarrow \mathsf{AdaGarbleInp}(\mathsf{st}, x)$: *It is a PPT algorithm that takes as input the state information* st *and an input $x \in \{0,1\}^n$ and outputs the garbled input $\widetilde{x}$.*
- $y = \mathsf{AdpEvalCkt}(\widetilde{C}, \widetilde{x})$: *Given a garbled circuit $\widetilde{C}$ and a garbled input $\widetilde{x}$, it outputs a value $y \in \{0,1\}^m$.*

Correctness. *For every $\lambda \in \mathbb{N}$, $C : \{0,1\}^n \to \{0,1\}^m$ and $x \in \{0,1\}^n$ it holds that:*

$$\Pr\left[(\widetilde{C}, \mathsf{st}) \leftarrow \mathsf{AdaGarbleCkt}(1^\lambda, C); \widetilde{x} \leftarrow \mathsf{AdaGarbleInp}(\mathsf{st}, x) : C(x) = \mathsf{AdpEvalCkt}(\widetilde{C}, \widetilde{x})\right] = 1$$

Adaptive Security. *There exists a PPT simulator $\mathsf{Sim} = (\mathsf{SimC}, \mathsf{SimIn})$ such that, for any non-uniform PPT adversary $\mathcal{A}$ there exists a negligible function ν such that:*

$$\left|Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 0) = 1] - Pr[\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}(1^\lambda, 1) = 1]\right| \leq \nu(\lambda)$$

where the experiment $\mathsf{Exp}^{\mathsf{adaptive}}_{\mathcal{A},\mathsf{GC},\mathsf{Sim}}$ is defined as follows:

1. *The adversary specifies the circuit C and obtains $\widetilde{C}$ where $\widetilde{C}$ is created as follows:*
   - *If $b = 0$: $(\widetilde{C}, \mathsf{st}) \leftarrow \mathsf{AdaGarbleCkt}(1^\lambda, C)$.*
   - *If $b = 1$: $(\widetilde{C}, \mathsf{st}) \leftarrow \mathsf{SimC}(1^\lambda, 1^{|C|})$.*
2. *The adversary $\mathcal{A}$ specifies the input x and gets $\widetilde{x}$ created as follows:*
   - *If $b = 0$, $\widetilde{x} \leftarrow \mathsf{AdaGarbleInp}(\mathsf{st}, x)$.*
   - *If $b = 1$, $\widetilde{x} \leftarrow \mathsf{SimIn}(\mathsf{st}, C(x))$*
3. *Finally, the adversary outputs a bit b', which is the output of the experiment.*

Online Complexity. *The running time of* AdaGarbleInp *is called as the online computational complexity and $|\widetilde{x}|$ is called as the online communication complexity.*

4 Our Construction

In this section, we provide our construction of adaptive garbled circuits. The main theorem is:

Theorem 4.1 *Assuming the existence of updatable laconic oblivious transfer, somewhere equivocal encryption and garbling scheme for circuits with selective security, there exists a construction of adaptive garbling scheme for circuits. The online communication complexity of our scheme is* $n + m + \mathsf{poly}(\lambda, \log |C|)$ *and the online computational complexity is* $O(n + m + \mathsf{poly}(\lambda, \log |C|))$.

From Theorems 3.1, 3.3, 3.6 we obtain the following corollary:

Corollary 4.2 *Assuming either the Computational Diffie-Hellman assumption or the Factoring assumption or the Learning with Errors assumption, there exists a construction of adaptive garbling scheme for circuits with online communication complexity of* $n + m + \mathsf{poly}(\lambda, \log |C|)$ *and online computational complexity of* $O(n + m + \mathsf{poly}(\lambda, \log |C|))$.

We start with some notation on how we denote circuits. We choose this notation to simplify the description of our construction. In the rest of the paper, whenever we mention a circuit C, we implicitly mean the universal circuit $U[C]$ with the circuit C hardwired in it. This is done so that the topology of the circuit $U[C]$ does not reveal anything about C except its size.

Notation. We model a circuit $C : \{0,1\}^n \to \{0,1\}^m$ as a set of $N - n$ NAND gates, each having fan-in 2. We number the gates of the circuit as follows. The input gates are given the numbers $\{1, \ldots, n\}$. The intermediate gates are numbered $\{n+1, n+2, \ldots, N-m\}$ such that a gate that receives its input from gates i and j is given a number greater than i and j. The output gates are numbered $\{N - m + 1, \ldots, N\}$. Each gate $g \in [n+1, N]$ is *described by a tuple* $(i, j) \in [g-1]^2$ where outputs of gates i and j serves as inputs to gate g.

Construction. Let $(\mathsf{crsGen}, \mathsf{Hash}, \mathsf{Send}, \mathsf{Receive}, \mathsf{SendWrite}, \mathsf{ReceiveWrite})$ be an updatable laconic oblivious transfer scheme, and $(\mathsf{GarbleCkt}, \mathsf{EvalCkt})$ be a garbling scheme for circuits. Moreover, let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKey})$ be a somewhere equivocal encryption with the block-length $s = |\widetilde{\mathsf{SC}}|$ (where $\widetilde{\mathsf{SC}}$ denotes a garbled version of the step-circuit SC defined in Figure 4), the message-length equal to $N - n$ and the equivocation parameter $t = \log N$ (the choice of t comes from the security proof).

The formal description of our adaptive garbling scheme appears in Figure 3. In this construction a selective secure garbling scheme is used to garble a step circuit SC repeatedly. This step circuit is described in Figure 4. We now provide an informal overview of this construction. At a high level, the adaptive garbling of a circuit C simply consists of garbling of the $N-n$ intermediate step circuits using the standard selectively secure (Yao's) garbling scheme. The entire N bit state of the computation is stored in an external memory using laconic OT and each step circuit accesses two bits in this memory. These garbled step circuits are encrypted using a somewhere equivocal encryption scheme and the resulting ciphertext is sent in the offline phase. Later in the online phase, the key for decrypting this ciphertext is revealed. Note that in our description we use the string $r \in \{0,1\}^N$ as a one time pad to hide the state of the computation.

$\mathsf{AdaGarbleCkt}(1^\lambda, C)$**:** On input a circuit $C : \{0,1\}^n \to \{0,1\}^m$ do:

1. Sample $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$, $\mathsf{key} \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $r \leftarrow \{0,1\}^N$.
2. For each $g \in [n+1, N+1]$, $k \in [\lambda]$ and $b \in \{0,1\}$ sample $\mathsf{lab}^g_{k,b} \leftarrow \{0,1\}^\lambda$. (We use $\{\mathsf{lab}^g_{k,b}\}$ to denote $\{\mathsf{lab}^g_{k,b}\}_{k\in[\lambda],b\in\{0,1\}}$.)
3. **for** each g from N down to $n+1$ **do**:
   (a) Let (i,j) be the description of the gate g.
   (b) Compute (where the step-circuit SC is described in Figure 4)
   $$\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, \mathsf{SC}[\mathsf{crs}, (r_i, r_j, r_g), (i,j), \{\mathsf{lab}^{g+1}_{k,b}\}, 0], \{\mathsf{lab}^g_{k,b}\}\right).$$
4. Compute $c \leftarrow \mathsf{Enc}(\mathsf{key}, \{\widetilde{\mathsf{SC}}_g\}_{g\in[n+1,N]})$.
5. Output $\widetilde{C} := (\mathsf{crs}, c)$ and $\mathsf{st} := (r, \mathsf{key}, \{\mathsf{lab}^{n+1}_{k,b}\})$.

$\mathsf{AdaGarbleInp}(\mathsf{st}, x)$: On input the state st and a string $x \in \{0,1\}^n$ do:

1. Parse st as $(r, \mathsf{key}, \{\mathsf{lab}^{n+1}_{k,b}\})$
2. Set $D := r_1 \oplus x_1 \| \ldots \| r_n \oplus x_n \| 0^{N-n}$ and compute $(\mathsf{d}, \widehat{D}) := \mathsf{Hash}(\mathsf{crs}, D)$.
3. Output $\widetilde{x} := \left(\{\mathsf{lab}^{n+1}_{k,\mathsf{d}_k}\}_{k\in[\lambda]}, r_1 \oplus x_1 \| \ldots \| r_n \oplus x_n, \mathsf{key}, r_{N-m+1}, \ldots, r_N\right)$.

$\mathsf{AdpEvalCkt}(\widetilde{C}, \widetilde{x})$: On input garbled circuit $\widetilde{C}$, and garbled input $\widetilde{x}$ do:

1. Parse $\widetilde{C}$ as (crs, c) and $\widetilde{x}$ as $(\{\mathsf{lab}_k\}_{k\in[\lambda]}, s_1, \ldots, s_n, \mathsf{key}, r_{N-m+1}, \ldots r_N)$.
2. Set $D := s_1 \| \ldots \| s_n \| 0^{N-n}$ and compute $(\mathsf{d}, \widehat{D}) := \mathsf{Hash}(\mathsf{crs}, D)$.
3. Compute $\{\widetilde{\mathsf{SC}}_g\}_{g\in[n+1,N]} := \mathsf{Dec}(\mathsf{key}, c)$.
4. Set $\overline{\mathsf{lab}} := \{\mathsf{lab}_k\}_{k\in[\lambda]}$.
5. **for** each g from $n+1$ to N **do**:
   (a) Let (i,j) be the description of gate g.
   (b) Compute $(\gamma, e) := \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{EvalCkt}(\widetilde{\mathsf{SC}}_g, \overline{\mathsf{lab}}), i), j)$.
   (c) Set $\overline{\mathsf{lab}} := \mathsf{ReceiveWrite}^{\widehat{D}}(\mathsf{crs}, g, \gamma, e)$.
6. Recover the contents of the memory D from the final state $\widehat{D}$.
7. Output $D_{N-m+1} \oplus r_{N-m+1} \| \ldots \| D_m \oplus r_N$.

Figure 3: Adaptive Garbling Scheme for Circuits

Step Circuit SC

Input: A digest d.
Hardcoded: The common reference string crs, a triplet of masking bits (r_i, r_j, r_g), a description (i, j) of gate g, a set of labels $\{\mathsf{lab}_{k,b}\}$ and a bit τ ($\tau = 1$ case is only relevant for the proof).

1. Compute $e_b \leftarrow \mathsf{SendWrite}(\mathsf{crs}, \mathsf{d}, g, b, \{\mathsf{lab}_{k,0}, \mathsf{lab}_{k,1}\}_{k\in[\lambda]})$ for $b \in \{0,1\}$.
2. Define for all $\alpha, \beta \in \{0,1\}$, $\gamma(\alpha,\beta) := \begin{cases} \mathsf{NAND}(\alpha \oplus r_i, \beta \oplus r_j) \oplus r_g & \text{if } \tau = 0 \\ r_g & \text{if } \tau = 1 \end{cases}$
3. Generate
$$f_0 \leftarrow \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, (\gamma(0,0), e_{\gamma(0,0)}), (\gamma(0,1), e_{\gamma(0,1)})\right),$$
$$f_1 \leftarrow \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, (\gamma(1,0), e_{\gamma(1,0)}), (\gamma(1,1), e_{\gamma(1,1)})\right).$$
4. Output
$$\mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, i, f_0, f_1\right)$$

Figure 4: Description of the Step Circuit

Communication Complexity of $\mathsf{AdaGarbleInp}$. It follows from the construction that the communication complexity of $\mathsf{AdaGarbleInp}$ is $\lambda^2 + n + m + |\mathsf{key}|$. From the parameters used in the somewhere equivocal encryption, we note that $|\mathsf{key}| = |\widetilde{\mathsf{SC}}|\mathsf{poly}(\log N, \lambda)$. It follows from the efficiency properties of updatable laconic oblivious transfer that $|\mathsf{SC}|$ is $\mathsf{poly}(\log N, \lambda)$. Thus, $|\mathsf{key}|$ is $\mathsf{poly}(\log N, \lambda)$.

Computational Complexity of $\mathsf{AdaGarbleInp}$. The running time of $\mathsf{AdaGarbleInp}$ described in Figure 3 grows with the circuit size N. We note that the running time can be made independent of the circuit size N by analyzing the specific laconic OT construction of Cho et al. [CDG+17]. The construction of Cho et al. uses a Merkle tree to hash a large database into a short digest. Recall that Merkle hash is efficiently updatable. Specifically, let y and y' be two strings given as a sequence of blocks of λ bits and y, y' differ in only the first k blocks. Given the Merkle hash on y and a set of $\log |y|$ hash values, there is an efficient procedure running in time $O(\lambda(k + \log |y|))$ that computes the Merkle hash on y'. We use this property to reduce the online computational complexity of our construction.

Recall that in our construction, the contents of the database at the very beginning of our computation needs to be set to $((r_{[1,n]} \oplus x)||0^{N-n})$ where x is the n-bit input. However, note that the input x is specified in the online phase. So the goal is to compute the hash of $((r_{[1,n]} \oplus x)||0^{N-n})$ in the online phase efficiently. To do this, we compute the hash of 0^N in the offline phase and store the value of this hash along with the specific hash values for updating the first $\lceil n/\lambda \rceil$ blocks. Once x is specified in the online phase, we use the stored value to compute the hash on $((r_{[1,n]} \oplus x)||0^{N-n})$ by performing the Merkle hash update. The crucial point is that this update is efficient (i.e. grows only with $|x| + \log N$).

Also, the algorithm $\mathsf{AdaGarbleInp}$ does not need the entire input r as input. It suffices to provide the first n and the last m bits of r (i.e., $\{r_1 \dots r_n, r_{N-m+1} \dots r_N\}$ as input to $\mathsf{AdaGarbleInp}$.

Correctness. Let D^{g^*} be the contents of the database at the end of Step 5.(c) of $\mathsf{AdpEvalCkt}$ in the g^*-th iteration of the **for** loop. We first argue via an inductive argument that for each gate $g^* \in [1, N]$, $D_g^{g^*}$ is the output of gate g masked with r_g for every $g \in [1, g^*]$. Given this, the correctness follows by setting $g^* := N$ and observing that the $\{D_k^N\}_{k\in[N-m+1,N]}$ is unmasked using $r_{[N-m+1,N]}$ in Step 7 of $\mathsf{AdpEvalCkt}$.

The base case is $g^* = n$ which is clearly true since in the beginning D^n is set as $(r_{[1,n]} \oplus x || 0^{N-n})$. In order to prove the inductive step for a gate g^* (with description (i, j)), we now argue that that the γ recovered in Step 4.(b) of $\mathsf{AdpEvalCkt}$ corresponds to $\mathsf{NAND}(D_i^{g^*-1} \oplus r_i, D_j^{g^*-1} \oplus r_j) \oplus r_{g^*}$ which by inductive hypothesis corresponds to output of the gate g^* masked with r_{g^*}. This is shown as follows.

$$
\begin{aligned}
(\gamma, e) &:= \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{EvalCkt}(\widetilde{\mathsf{SC}}_g, \overline{\mathsf{lab}}), i), j) \\
&= \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, i, f_0, f_1\right), i), j) \\
&= \mathsf{Receive}^{\widehat{D}}(\mathsf{crs}, f_{D_i^{g^*-1}}, j) \\
&= \mathsf{Receive}^{\widehat{D}}\left(\mathsf{crs}, \mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, (\gamma(D_i^{g^*-1}, 0), e_{\gamma(D_i^{g^*-1},0)}), (\gamma(D_i^{g^*-1}, 1), e_{\gamma(D_i^{g^*-1},1)})\right), j\right) \\
&= \left(\gamma(D_i^{g^*-1}, D_j^{g^*-1}), e_{\gamma(D_i^{g^*-1}, D_j^{g^*-1})}\right) \\
&= \left(\mathsf{NAND}(D_i^{g^*-1} \oplus r_i, D_j^{g^*-1} \oplus r_j) \oplus r_{g^*}, e_{\mathsf{NAND}(D_i^{g^*-1} \oplus r_i, D_j^{g^*-1} \oplus r_j) \oplus r_{g^*}}\right)
\end{aligned}
$$

5 Proof of Security

In this section, we prove that the construction presented in the previous section satisfies adaptive security. In Subsection 5.1, we start by defining circuit configurations. Next, in Subsection 5.2 we show that both the real and ideal world executions are special cases of this circuit configuration (this will also provide a formal description of our simulator). Finally, in the rest of the subsection we show that the real and ideal world executions are indistinguishable. The indistinguishability argument proceeds by a sequence of hybrids over different configurations.

5.1 Circuit Configuration

Our proof of security proceeds via a hybrid argument over different *circuit configurations* which we describe in this section. A circuit configuration denoted by $\mathsf{conf} = (I, \{(g, \mathsf{mode}_g)\}_{g\in[n+1,N]})$ consists of a set $I \subseteq [n+1, N]$ and a set of tuples (g, mode_g) where for each gate $g \in [n+1, N]$ $\mathsf{mode}_g \in \{\mathsf{White}, \mathsf{Gray}, \mathsf{Black}\}$ describes the mode of operation of gate g.

The subset I denotes the set of indices in which there is a "hole" in the outer encryption layer. At an intuitive level, the White mode corresponds to the Real Garbling (as is done in the honest execution), the Gray mode corresponds to the Input Dependent Simulation (where the step circuit for this gate is in simulation but depends on the input), and the Black mode corresponds to the Input Independent Simulation (where simulation is done independent of the input). In other words, White mode matches the real execution, Black mode matches the ideal execution and Gray mode is an intermediate execution mode. Looking ahead, initially all the step circuit will be in White mode

and the goal will be to convert all of them to Black in the simulation. Note that we refer to modes with color names as these modes will coincide with the pebbling game that we later describe.

Valid configurations. We say that a configuration $\mathsf{conf} = (I, \{(g, \mathsf{mode}_g)\}_{g\in[n+1,N]})$ is *valid* if and only if:

1. If $\mathsf{mode}_g = \mathsf{Black}$ then for every $k > g$, $\mathsf{mode}_k = \mathsf{Black}$.
2. If $\mathsf{mode}_g = \mathsf{Gray}$ then $g \in I$.

In other words, if gate g is in Black mode then we require that all the subsequent gates are also in Black mode. Moreover, if a gate g is in the Gray mode then there is a hole in positions g in the outer encryption layer.

Simulation in a valid configuration. In Figure 5 we describe the simulated circuit garbling SimC and the simulated input garbling SimIn functions for any given valid configuration conf. Note that these simulated garbling functions take as input the circuit C and x respective as inputs which the ideal world simulation does not. We describe our simulator functions with these additional inputs so that it captures simulation in all of our intermediate hybrids. We note that final ideal world simulation does not uses these values.

5.2 Our Hybrids

For every valid circuit configuration $\mathsf{conf} = (I, \{(g, \mathsf{mode}_g)\}_{g\in[n+1,N]})$, we define $\mathsf{Hybrid}_{\mathsf{conf}}$ to be a distribution of $\widetilde{C}$ and $\widetilde{x}$ as given in Figure 5. We start by observing that both real world and ideal distribution from Definition 3.7 can be seen as instance of $\mathsf{Hybrid}_{\mathsf{conf}}$ where $\mathsf{conf} = (\emptyset, \{(g, \mathsf{White})\}_{g\in[n+1,N]})$ and $\mathsf{conf} = (\emptyset, \{(g, \mathsf{Black})\}_{g\in[n+1,N]})$, respectively. In other words, the real world distribution corresponds to having all gates in White mode and the ideal world distribution corresponds to having all gates in Black mode. The goal is to move from the real world distribution to the ideal world distribution while minimizing the maximum number of gates in the Gray mode in any intermediate hybrid.

5.2.1 Rules of Indistinguishability

We will now describe the two rules (we call these rule A and rule B) to move from one valid circuit configuration conf to another valid configuration conf' such that $\mathsf{Hybrid}_{\mathsf{conf}}$ is computationally indistinguishable from $\mathsf{Hybrid}_{\mathsf{conf}'}$.

Rule A: Very roughly, rule A says that for any valid configuration conf we can indistinguishably change gate g^* in White mode to Gray mode if it is the first gate or if its predecessor is also in Gray mode. More formally, let $\mathsf{conf} = (I, \{(g, \mathsf{mode}_g)\}_{g\in[n+1,N]})$ and $\mathsf{conf}' = (I', \{(g, \mathsf{mode}'_g)\}_{g\in[n+1,N]})$ be two valid circuit configurations and $g^* \in [n+1, N]$ be a gate such that:

- For all $g \in [n+1, N] \setminus g^*$ we have that $\mathsf{mode}_g = \mathsf{mode}'_g$.
- $g^* \notin I$, $I' = I \cup \{g^*\}$, and $|I'| \leq t$ (where t is the equivocation parameter).
- Either $g^* = n+1$ or $(g^* - 1, \mathsf{Gray}) \in \mathsf{conf}$.

$\mathsf{SimC}(1^\lambda, C)$**:** On input a circuit $C : \{0,1\}^n \to \{0,1\}^m$ do:

1. Sample $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda)$ and $r \xleftarrow{\$} \{0,1\}^N$.
2. For each $g \in [n+1, N+1]$, $k \in [\lambda]$ and $b \in \{0,1\}$ sample $\mathsf{lab}^g_{k,b} \leftarrow \{0,1\}^\lambda$. (We use $\{\mathsf{lab}^g_{k,b}\}$ to denote $\{\mathsf{lab}^g_{k,b}\}_{k\in[\lambda],b\in\{0,1\}}$.)
3. **for** each g from N down to $n+1$ such that $g \notin I$ **do**:
   (a) Let (i,j) be the description of the gate g.
   (b) If $\mathsf{mode}_g = \mathsf{White}$ then compute (where the step-circuit SC is described in Figure 4)
   $$\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, \mathsf{SC}[\mathsf{crs}, (r_i, r_j, r_g), (i,j), \{\mathsf{lab}^{g+1}_{k,b}\}, 0], \{\mathsf{lab}^g_{k,b}\}\right).$$
   (c) If $\mathsf{mode}_g = \mathsf{Black}$ then compute
   $$\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, \mathsf{SC}[\mathsf{crs}, (0,0,r_g), (i,j), \{\mathsf{lab}^{g+1}_{k,b}\}, 1], \{\mathsf{lab}^g_{k,b}\}\right).$$
4. Compute $(\mathsf{st}_1, c) \leftarrow \mathsf{SimEnc}(I, \{\widetilde{\mathsf{SC}}_g\}_{g\notin I})$.
5. Output $\widetilde{C} := (\mathsf{crs}, c)$ and $\mathsf{st} := (r, \mathsf{st}_1, \{\mathsf{lab}^g_{k,b}\}_{k,b,g})$.

$\mathsf{SimIn}(\mathsf{st}, x, y)$**:** On input state $\mathsf{st} = (r, \mathsf{st}_1, \{\mathsf{lab}^g_{k,b}\}_{k,b,g})$, a string $x \in \{0,1\}^n$, and $y \in \{0,1\}^m$ do:

Notation: For $g \in [n+1, N+1]$ we let D_g be such that

$$D_{g,w} = \begin{cases} x_w \oplus r_w & w \le n, \\ E_w \oplus r_w & n+1 \le w < g, \\ 0 & \text{otherwise}, \end{cases}$$

where E_w is the bit assigned to wire w of the circuit C computed on input x. Finally, we let d_g be the digest of D_g (i.e., $(\mathsf{d}_g, \cdot) := \mathsf{Hash}(\mathsf{crs}, D_g)$) and $\mathsf{d}_{g,k}$ be the k^{th} bit of d_g.

1. **for** each g from N down to $n+1$ such that $g \in I$:
   (a) Set $e \leftarrow \mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D_g, g, D_{g+1,g}, \{\mathsf{lab}^{g+1}_{k,\mathsf{d}_{g+1,k}}\}_{k\in[\lambda]})$.
   (b) Set $\mathsf{out} \leftarrow \mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D_g, i, \mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D_g, j, e))$
   (c) Compute $\widetilde{\mathsf{SC}}_g \leftarrow \mathsf{Sim}_{\mathsf{Ckt}}\left(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}, \{\mathsf{lab}^g_{k,\mathsf{d}_{g,k}}\}_{k\in[\lambda]}\right)$.
2. Compute $\mathsf{key} \leftarrow \mathsf{SimKey}(\mathsf{st}_1, \{\widetilde{\mathsf{SC}}_g\}_{g\in I})$.
3. **for** each $g \in [N-m+1, N]$ do:
   (a) If $\mathsf{mode}_g \in \mathsf{Black}$ then set $r'_g = r_g \oplus y_{w-N+m}$.
   (b) Else, $r'_g = r_g$.
4. Output $\widetilde{x} := \left(\{\mathsf{lab}^{n+1}_{k,\mathsf{d}_{n+1,k}}\}_{k\in[\lambda]}, r_1 \oplus x_1 \| \ldots \| r_n \oplus x_n, \mathsf{key}, r'_{N-m+1,N} \ldots r'_N\right)$.

Figure 5: Garbling in configuration $\mathsf{conf} = (I, \{(h, \mathsf{mode}_h)\}_{h\in[n+1,N]})$.

- $(g^*, \mathsf{White}) \in \mathsf{conf}$ and $(g^*, \mathsf{Gray}) \in \mathsf{conf}'$.

In Lemma 5.3 we show that for an valid configurations $\mathsf{conf}, \mathsf{conf}'$ satisfying the above constraints we have that $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$. Note that we can also use this rule to move a gate g^* from Gray mode to White mode. We refer to those invocations of the rule as *inverse A rule*.

Rule B: Very roughly, rule B says that for any configuration for any valid configuration conf we can indistinguishably change gate g^* in Gray mode to Black mode if all gates subsequent to g^* is in Black mode and the predecessor is in Gray mode. More formally, let $\mathsf{conf} = (I, \{(g, \mathsf{mode}_g)\}_{g\in[n+1,N]})$ and $\mathsf{conf}' = (I', \{(g, \mathsf{mode}'_g)\}_{g\in[n+1,N]})$ be two valid circuit configurations and $g^* \in [n+1, N]$ be a gate such that:

- For all $g \in [n+1, N] \setminus g^*$ we have that $\mathsf{mode}_g = \mathsf{mode}'_g$.
- $g^* \in I$, $I' = I \setminus \{g^*\}$, and $|I| \leq t$ (where t is the equivocation parameter).
- For all $g \in [g^*+1, N]$ we have that $(g, \mathsf{Black}) \in \mathsf{conf}$.
- Either $g^* = n+1$ or $(g^*-1, \mathsf{Gray}) \in \mathsf{conf}$.
- $(g^*, \mathsf{Gray}) \in \mathsf{conf}$ and $(g^*, \mathsf{Black}) \in \mathsf{conf}'$.

In Lemma 5.4 we show that for an valid configurations $\mathsf{conf}, \mathsf{conf}'$ satisfying the above constraints we have that $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$.

5.2.2 Interpreting the rules of indistinguishability as a pebbling game

Our sequence of hybrids from the real to the ideal world follow an optimal strategy for the following pebbling game. The two rules described above correspond to the rules of our pebbling game below.

Consider the positive integer line $n+1, n+2, \ldots N$. We are given pebbles of two colors: gray and black . A black pebble corresponds to a gate in the Black (i.e., input independent simulation) mode and a gray pebble corresponds to a gate in the Gray (i.e., input dependent simulation) mode. A position without any pebble corresponds to real garbling or in the White mode. We can place the pebbles on this positive integer line according to the following two rules:

Rule A: We can place or remove a gray pebble in position i if and only if there is a gray pebble in position $i-1$. This restriction does not apply to position $n+1$: we can always place or remove a gray pebble at position $n+1$.

Rule B: We can replace a gray pebble in position i with a black pebble as long as all the positions $> i$ have black pebbles and there is a gray pebble in position $i-1$ or if $i = n+1$.

Optimization goal of the pebbling game. The goal is to pebble the line $[n+1, N]$ such that every position has a black pebble while minimizing the number of gray pebbles that are present on the line at any point in time.

5.2.3 Optimal Pebbling Strategy

To provide some intuition, we start with the naïve pebbling strategy. The naïve pebbling strategy involves starting from position $n+1$ and placing a gray pebble at every position in $[n+1, N]$ and then replacing them with black pebbles from N to $n+1$. However, this strategy uses a total of $N-n$ gray pebbles. Using a more clever strategy it is actually possible to do the same using only $\log(N-n)$ gray pebbles. This is argued in the following lemmas.

Lemma 5.1 *For any integer $n+1 \leq p \leq n+2^k-1$, it is possible to make $O((p-n)^{\log_2 3}) \approx O((p-n)^{1.585})$ moves and get a gray pebble at position p using k gray pebbles.*

Proof This proof is taken verbatim from [GPSZ17]. First we observe to get a gray pebble placed at p, for each $i \in [n+1, p-1]$ there must have been at some point a gray pebble placed at location i.

Next, we observe that it suffices to show we can get a gray pebble at position $p = n + 2^k - 1$ for every k using $O(3^k) = O((p-n)^{\log_2 3})$ steps. Indeed, for more general p, we run the protocol for $p' = n + 2^k - 1$ where $k = \lceil \log_2(p-n-1) \rceil$, but stop the first time we get a gray pebble at position p. Since $p'/p \leq 3$, the running time is at most $O((p-n)^{\log_2 3})$.

Now for the algorithm. The sequence of steps will create a fractal pattern, and we describe the steps recursively. We assume an algorithm A_{k-1} using $k-1$ gray pebbles that can get a gray pebble at position $n + 2^{k-1} - 1$. The steps are as follows:

- Run A_{k-1}. There is now a gray pebble at position $n + 2^{k-1} - 1$ on the line.
- Place the remaining gray pebble at position $n + 2^{k-1}$, which is allowed since there is a gray pebble at position $n + 2^{k-1} - 1$.
- Run A_{k-1} in reverse, recovering all of the $k-1$ gray pebbles used by A. The result is that there is a single gray pebble on the line at position $n + 2^{k-1}$.
- Now associate the portion of the number line starting at $n + 2^{k-1} + 1$ with a new number line. That is, associate $n + 2^{k-1} + a$ on the original number line with $n' + a$ (where $n' = n + 2^{k-1}$) on the new number line. We now have $k-1$ gray pebbles, and on this new number line, all of the same rules apply. In particular, we can always add or remove a gray pebble from the first position $n' + 1 = n + 2^{k-1} + 1$ since we have left a gray pebble at $n + 2^{k-1}$. Therefore, we can run A_{k+1} once more on the new number line starting at $n' + 1$. The end result is a pebble at position $n' + 2^{k-1} - 1 = n + 2^{k-1} + (2^{k-1} - 1) = n + 2^k - 1$.

It remains to analyze the running time. The algorithm makes 3 recursive calls to A_{k-1}, so by induction the overall running time is $O(3^k)$, as desired. ■

Using the above lemma, we now give an optimal strategy for our pebbling game.

Lemma 5.2 *For any $N \in \mathbb{N}$, there exists a strategy for pebbling the line graph $[n+1, N]$ according to rules A and B by using at most $\log N$ gray pebbles and making $\mathsf{poly}(N)$ moves.*

Proof The strategy is given below. For each g from N down to $n+1$ **do**:

1. Use the strategy in Lemma 5.1 to place a gray pebble in position g. Note that there exists a gray pebble in position $g-1$ as well.
2. Replace the gray pebble in position g with a black pebble. This replacement is allowed since all positions $> g$ have black pebbles and there is a gray pebble in position $g-1$.
3. Recover all the gray pebbles by reversing the moves.

The correctness of this strategy follows by inspection and the number of moves is polynomial in N. ■

5.2.4 Completing the Hybrids

Using the strategy given in Lemma 5.2 yields a sequence of configurations $\mathsf{conf}_0 \dots \mathsf{conf}_\ell$ for an appropriate polynomial ℓ with conf_0 and conf_ℓ being the real and the ideal world distributions and for each $i \in [\ell]$ we have that $\mathsf{Hybrid}_{\mathsf{conf}_{i-1}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}_i}$ either using rule A (i.e., Lemma 5.3) or using rule B (i.e., Lemma 5.4). Finally note that the number of holes in the garbled circuit needed is the maximum size of I over the sequence of hybrids (i.e. the maximum number of gray pebbles used). Thus, it suffices to set the equivocation parameter t for somewhere equivocal encryption scheme to $\log N$. This completes the proof of security.

5.3 Proof of Indistinguishability for the Rules

In this subsection, we will use the security of underlying primitives to implement the two rules.

5.3.1 Implementing Rule A

Lemma 5.3 (Rule A) *Let* conf *and* conf' *be two valid circuit configurations satisfying the constraints of rule A, then assuming the security of somewhere equivocal encryption, garbling scheme for circuits and updatable laconic oblivious transfer, we have that* $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$.

Proof We prove this via a hybrid argument.

- $\underline{\mathsf{Hybrid}_{\mathsf{conf}}}$: This is our starting hybrid and is distributed as $\mathsf{Hybrid}_{(I,\{(g,\mathsf{mode}_g)\}_{g\in[n+1,N]})}$.
- $\underline{\mathsf{Hybrid}_1}$: In this hybrid, we change the configuration to $(I', \{(g, \mathsf{mode}_g)\}_{g\in[n+1,N]})$. This hybrid is distributed as $\mathsf{Hybrid}_{(I',\{(g,\mathsf{mode}_g)\}_{g\in[n+1,N]})}$.

  Computational indistinguishability between hybrid $\mathsf{Hybrid}_{\mathsf{conf}}$ and Hybrid_1 reduces directly to the security of somewhere equivocal encryption scheme. We give a formal reduction in Appendix A
- $\underline{\mathsf{Hybrid}_2}$: By conditions of Rule A we have that $\mathsf{mode}_{g^*-1} = \mathsf{Gray}$. Thus, we have that $g^* - 1 \in I'$. Therefore, we note that the input labels $\{\mathsf{lab}^{g^*}_{k,b}\}$ are not used in SimC but only in SimIn where it is used to generate $\widetilde{\mathsf{SC}}_{g^*-1}$ and $\widetilde{\mathsf{SC}}_{g^*}$. In this hybrid, we postpone the sampling of $\{\mathsf{lab}^{g^*}_{k,b}\}$ and the generation of $\widetilde{\mathsf{SC}}_{g^*}$ from SimC to SimIn.

  The change in hybrid Hybrid_2 from Hybrid_1 is just syntactic and the distributions are identical.
- $\underline{\mathsf{Hybrid}_3}$: In this hybrid, we change the sampling of $\{\mathsf{lab}^{g^*}_{k,b}\}$ and the generation of $\widetilde{\mathsf{SC}}_{g^*}$. Specifically, we do not sample the entire set of labels $\{\mathsf{lab}^{g^*}_{k,b}\}$ but a subset namely $\{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_k$ and we generate $\widetilde{\mathsf{SC}}_{g^*}$ from the simulated distribution. (Note that since $g^* - 1$ is also in Gray mode. Thus we have that $\widetilde{\mathsf{SC}}_{g^*-1}$ is also simulated and only $\{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_k$ are needed for its generation.) More formally, we generate

$$\widetilde{\mathsf{SC}}_{g^*} \leftarrow \mathsf{Sim}_{ckt}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}, \{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_{k\in[\lambda]})$$

  where $\mathsf{out} \leftarrow \mathsf{SC}[\mathsf{crs}, (r_i, r_j, r_g), (i, j), \{\mathsf{lab}^{g^*+1}_{k,b}\}, 0](\mathsf{d}_{g^*})$.

The only change in hybrid Hybrid_3 from Hybrid_2 is in the generation of the garbled circuit $\widetilde{\mathsf{SC}}_{g^*}$ and the security follows directly from the selective security of the garbling scheme. We show this reduction in Appendix A.

- $\underline{\mathsf{Hybrid}_4}$: In this hybrid, we change how the output value out hardwired in $\widetilde{\mathsf{SC}}_{g^*}$ is generated. Recall that in hybrid Hybrid_3 this value is generated by first computing f_0 and f_1 as in Figure 4 and then generating out as $\mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, i, f_0, f_1\right)$. In this hybrid, we just generate $f_{D_{g^*,i}}$ and use the laconic OT simulator to generate out. More formally, out is generated as

$$\mathsf{out} \leftarrow \mathsf{Sim}_{\ell\mathsf{OT}}\left(\mathsf{crs}, D_{g^*}, i, f_{D_{g^*,i}}\right).$$

  Computational indistinguishability between hybrids Hybrid_3 and Hybrid_4 follows directly from the sender privacy of the laconic OT scheme. The reduction is given in Appendix A.

- $\underline{\mathsf{Hybrid}_5}$: In this hybrid, we change how the value $f_{D_{g^*,i}}$ is generated. Recall from Figure 4 that $f_{D_{g^*,i}}$ is set as $\mathsf{Send}(\mathsf{crs}, \mathsf{d}, j, (\gamma(D_{g^*,i}, 0), e_{\gamma(D_{g^*,i},0)}),$ $(\gamma(D_{g^*,i}, 1), e_{\gamma(D_{g^*,i},1)}))$. We change the distribution of $f_{D_{g^*,i}}$ to $\mathsf{Sim}_{\ell\mathsf{OT}}(\mathsf{crs}, D_{g^*},$ $j, e_{D_{g^*+1,g^*}})$, where $e_{D_{g^*+1,g^*}}$ is sampled as in Figure 4.

  Computational indistinguishability between hybrids Hybrid_4 and Hybrid_5 follows directly from the sender privacy of the laconic OT scheme. The argument is analogous to the argument of indistinguishability between Hybrid_3 and Hybrid_4.

- $\underline{\mathsf{Hybrid}_6}$: In this hybrid, we change how $e_{D_{g^*+1,g^*}}$ is generated. More specifically, we generate it using the simulator $\mathsf{Sim}_{\ell\mathsf{OTW}}$. In other words, $e_{D_{g^*+1,g}}$ is generated as

$$\mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D_{g^*}, g^*, D_{g^*+1,g^*}, \{\mathsf{lab}^{g^*+1}_{k,\mathsf{d}_{g^*+1,k}}\}_{k\in[\lambda]}).$$

  Computational indistinguishability between hybrids Hybrid_4 and Hybrid_5 follows directly from the sender privacy for writes of the laconic OT scheme.

- $\underline{\mathsf{Hybrid}_7}$: In this hybrid, we reverse the changes made earlier with respect to sampling of $\{\mathsf{lab}^{g^*}_{k,b}\}$. Specifically, we sample all values $\{\mathsf{lab}^{g^*}_{k,b}\}$ and not just $\{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_k$. Additionally, this is done in SimC rather than SimIn.

  Note that this change is syntactic and the hybrids Hybrid_6 to Hybrid_7 are identical. Finally, observe that hybrid Hybrid_7 is the same as $\mathsf{Hybrid}_{\mathsf{conf}'}$.

This completes the proof of the lemma. We additionally note that the above sequence of hybrids is reversible. This implies the inverse rule A. ■

5.3.2 Implementing Rule B

Lemma 5.4 (Rule B) *Let* conf *and* conf' *be two valid circuit configurations satisfying the constraints of rule B, then assuming the security of somewhere equivocal encryption, garbling scheme for circuits and updatable laconic oblivious transfer, we have that* $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_{\mathsf{conf}'}$.

Proof We prove this via a hybrid argument starting with $\mathsf{Hybrid}_{\mathsf{conf}'}$ and ending in hybrid $\mathsf{Hybrid}_{\mathsf{conf}}$. We follow this ordering of the hybrids as this keeps the proof very close to the proof of Lemma 5.3. In particular, we start with hybrid $\mathsf{Hybrid}_{\mathsf{conf}'}$ and make changes to get a hybrid Hybrid_7 which are almost the same as the hybrids in Lemma 5.3. One key difference is that we set the value $D_{g^*+1,g}$ differently than how it is set in the execution of SimIn in Figure 5. Specifically, we set D_{g^*+1,g^*} as r_{g^*} rather than $E_{g^*} \oplus r_{g^*}$. Note that this also corresponding changes the value of d_{g^*+1} which is the hash of D_{g^*+1}. Finally we provide argument that hybrids Hybrid_7 and $\mathsf{Hybrid}_{\mathsf{conf}}$ are identical.

- $\underline{\mathsf{Hybrid}_{\mathsf{conf}'}}$: This is our starting hybrid and is distributed as $\mathsf{Hybrid}_{(I',\{(g,\mathsf{mode}'_g)\}_{g\in[n+1,N]})}$.

- $\underline{\mathsf{Hybrid}_1}$: In this hybrid, we change the configuration to $(I, \{(g, \mathsf{mode}'_g)\}_{g\in[n+1,N]})$. This hybrid is distributed as $\mathsf{Hybrid}_{(I,\{(g,\mathsf{mode}'_g)\}_{g\in[n+1,N]})}$.

  Computational indistinguishability between hybrid $\mathsf{Hybrid}_{\mathsf{conf}'}$ and Hybrid_1 reduces directly to the security of somewhere equivocal encryption.

- $\underline{\mathsf{Hybrid}_2}$: By conditions of Rule B we have that $\mathsf{mode}_{g^*-1} = \mathsf{Gray}$. Thus, we have that $g^* - 1 \in I'$. Therefore, we note that the input labels $\{\mathsf{lab}^{g^*}_{k,b}\}$ are not used in SimC but only in SimIn where it is used to generate $\widetilde{\mathsf{SC}}_{g^*-1}$ and $\widetilde{\mathsf{SC}}_{g^*}$. In this hybrid, we postpone the sampling of $\{\mathsf{lab}^{g^*}_{k,b}\}$ and the generation of $\widetilde{\mathsf{SC}}_{g^*}$ from SimC to SimIn.

  The change in hybrid Hybrid_2 from Hybrid_1 is just syntactic and the distributions are identical.

- $\underline{\mathsf{Hybrid}_3}$: In this hybrid, we change the sampling of $\{\mathsf{lab}^{g^*}_{k,b}\}$ and the generation of $\widetilde{\mathsf{SC}}_{g^*}$. Specifically, we do not sample the entire set of labels $\{\mathsf{lab}^{g^*}_{k,b}\}$ but a subset namely $\{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_k$ and we generate $\widetilde{\mathsf{SC}}_{g^*}$ from the simulated distribution. (Note that since $g^* - 1$ is also in Gray mode. Thus we have that $\widetilde{\mathsf{SC}}_{g^*-1}$ is also simulated and only $\{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_k$ are needed for its generation.) More formally, we generate

  $$\widetilde{\mathsf{SC}}_{g^*} \leftarrow \mathsf{Sim}_{ckt}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}, \{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_{k\in[\lambda]})$$

  where $\mathsf{out} \leftarrow \mathsf{SC}[\mathsf{crs}, (0,0,r_g), (i,j), \{\mathsf{lab}^{g^*+1}_{k,b}\}, 1](\mathsf{d}_{g^*})$.

  The only change in hybrid Hybrid_3 from Hybrid_2 is in the generation of the garbled circuit $\widetilde{\mathsf{SC}}_{g^*}$ and the security follows directly from the selective security of the garbling scheme.

- $\underline{\mathsf{Hybrid}_4}$: In this hybrid, we set change how the output value out hardwired in $\widetilde{\mathsf{SC}}_{g^*}$ is generated. Recall that in hybrid Hybrid_3 this value is generated by first computing f_0 and f_1 as in Figure 4 and then generating out as $\mathsf{Send}(\mathsf{crs}, \mathsf{d}, i, f_0, f_1)$. In this hybrid, we just generate $f_{D_{g^*,i}}$ and use the laconic OT simulator to generate out. More formally, out is generated as

  $$\mathsf{out} \leftarrow \mathsf{Sim}_{\ell\mathsf{OT}}\left(\mathsf{crs}, D_{g^*}, i, f_{D_{g^*,i}}\right).$$

  Computational indistinguishability between hybrids Hybrid_3 and Hybrid_4 follows directly from the sender privacy of the laconic OT scheme.

- $\underline{\mathsf{Hybrid}_5}$: In this hybrid, we change how the how the value $f_{D_{g^*,i}}$ is generated in hybrid Hybrid_4. Recall from Figure 4 that $f_{D_{g^*,i}}$ is set as $\mathsf{Send}\left(\mathsf{crs}, \mathsf{d}, j, e_{r_{g^*}}, e_{r_{g^*}}\right)$. We change the distribution of $f_{D_{g^*,i}}$ to $\mathsf{Sim}_{\ell\mathsf{OT}}\left(\mathsf{crs}, D_g, j, e_{r_{g^*}}\right)$, where $e_{r_{g^*}}$ is sampled as in Figure 4.

  Computational indistinguishability between hybrids Hybrid_3 and Hybrid_4 follows directly from the sender privacy of the laconic OT scheme. The argument is analogous to the argument of indistinguishability between Hybrid_2 and Hybrid_3.

- $\underline{\mathsf{Hybrid}_6}$: In this hybrid, we change how $e_{r_{g^*}}$ is generated. More specifically, we generate it using the simulator $\mathsf{Sim}_{\ell\mathsf{OTW}}$. In other words, $e_{r_{g^*}}$ is generated as

$$\mathsf{Sim}_{\ell\mathsf{OTW}}(\mathsf{crs}, D_{g^*}, g^*, r_{g^*}, \{\mathsf{lab}^{g^*+1}_{k,\mathsf{d}_{g^*+1,k}}\}_{k\in[\lambda]}).$$

  Computational indistinguishability between hybrids Hybrid_4 and Hybrid_5 follows directly from the sender privacy for writes of the laconic OT scheme.

- $\underline{\mathsf{Hybrid}_7}$: In this hybrid, we reverse the changes made earlier with respect to sampling of $\{\mathsf{lab}^{g^*}_{k,b}\}$. Specifically, we sample all values $\{\mathsf{lab}^{g^*}_{k,b}\}$ and not just $\{\mathsf{lab}^{g^*}_{k,\mathsf{d}_{g^*,k}}\}_k$. Additionally, this is done in SimC rather than SimIn.

  Note that this change is syntactic and the hybrids Hybrid_6 to Hybrid_7 are identical.

- $\underline{\mathsf{Hybrid}_{\mathsf{conf}}}$: This corresponds to the hybrid $\mathsf{Hybrid}_{(I,\{(g,\mathsf{mode}_g)\}_{g\in[n+1,N]}}$.

  Observe that the only difference between Hybrid_7 and $\mathsf{Hybrid}_{\mathsf{conf}}$ is how D_{g^*+1,g^*} is set. Namely, in Hybrid_7 this value is set to be r_{g^*} while in $\mathsf{Hybrid}_{\mathsf{conf}}$ this value is set as $r_{g^*} \oplus \mathsf{NAND}(D_{g^*,i} \oplus r_i, D_{g^*,j} \oplus r_j,)$. We argue that the distributions Hybrid_7 and $\mathsf{Hybrid}_{\mathsf{conf}}$ are identical. Two cases arise:

  - $g^* \leq N - m$: In this case, note that since r_{g^*} is not anywhere else we have that the distribution r_{g^*} and $r_{g^*} \oplus \mathsf{NAND}(D_{g^*,i} \oplus r_i, D_{g^*,j} \oplus r_j)$ are both uniform and identical.
  - $g^* > N - m$: In this case, we have that $r_{g^*} = y_{g^*-N+m} \oplus r'_{g^*}$ which is again identical to the distribution of r_{g^*} in $\mathsf{Hybrid}_{\mathsf{conf}}$.

This completes the proof of the lemma. ■

References

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 657–677, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[AF90] Martín Abadi and Joan Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th Annual Symposium on Foundations of Computer Science*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, CCC '05, pages 260–274, Washington, DC, USA, 2005. IEEE Computer Society.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP 2010: 37th International Colloquium on Automata, Languages and Programming, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 152–163, Bordeaux, France, July 6–10, 2010. Springer, Heidelberg, Germany.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 166–184, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 162–172, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[AS16] Prabhanjan Vijendra Ananth and Amit Sahai. Functional encryption for turing machines. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 125–153, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 134–153, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. To appear in Eurocrypt, 2018. `https://eprint.iacr.org/2017/967`.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic receiver oblivious transfer and applications. *To appear in Crypto*, 2017.

[DG17] Nico Döttling and Sanjam Garg. Identity based encryption from diffie-hellman assumptions. *To appear in Crypto*, 2017.

[DGHM18] Nico Dttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. To appear in PKC, 2018. `https://eprint.iacr.org/2017/978`.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 191–219, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 405–422, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 210–229, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 449–458, Portland, OR, USA, June 14–17, 2015. ACM Press.

[GPSZ17] Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 156–181, 2017.

[GS16] Sanjam Garg and Akshayaram Srinivasan. Single-key to multi-key functional encryption with polynomial loss. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 419–442, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 149–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[JKK+17] Zahra Jafargholi, Chethan Kamath, Karen Klein, Ilan Komargodski, Krzysztof Pietrzak, and Daniel Wichs. Be adaptive, avoid overcommitting. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 133–163, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference*

on Computer and Communications Security, pages 955–966, Berlin, Germany, November 4–8, 2013. ACM Press.

[JSW17] Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. In *TCC 2017: 15th Theory of Cryptography Conference, Part II*, Lecture Notes in Computer Science, pages 40–71. Springer, Heidelberg, Germany, March 2017.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, volume 9985 of *Lecture Notes in Computer Science*, pages 433–458, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[LM16] Baiyu Li and Daniele Micciancio. Compactness vs collusion resistance in functional encryption. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 443–468, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 719–734, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[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, April 2009.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, *57th Annual Symposium on Foundations of Computer Science*, pages 11–20, New Brunswick, NJ, USA, October 9–11, 2016. IEEE Computer Society Press.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 287–302, 2004.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[PTC76] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical systems theory*, 10(1):239–251, Dec 1976.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 463–472, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

A Completing Proofs of Lemma 5.3

Claim A.1 *Assuming the security of somewhere equivocal encryption scheme, we have* $\mathsf{Hybrid}_{\mathsf{conf}} \overset{c}{\approx} \mathsf{Hybrid}_1$.

Proof We give a reduction to the security of somewhere equivocal encryption.

Generating the Garbled Circuit. To generate the garbled circuit $\widetilde{C}$:

1. Execute the steps 1,2,3 as described in Figure 5.
2. Interact with the external challenger by giving $\{\widetilde{\mathsf{SC}}_g\}_{g \notin I}$ as the challenge messages, I as the challenge subset and g^* as the challenge index. Obtain c as the challenge ciphertext.
3. Output (crs, c) as the garbled circuit $\widetilde{C}$ and $(r, \mathsf{st}_1, \{\mathsf{lab}^g_{k,b}\}_{k,b,g})$ as the state st.

Generating the Garbled Input. To generate the garbled input $\widetilde{x}$:

1. Execute the steps 1,2 as described in Figure 5.
2. Interact with the external challenger by providing $\{\widetilde{\mathsf{SC}}_g\}_{g \in I}$ as the remaining messages and obtain key'.
3. Execute the rest of the steps as described in Figure 5 and output $\widetilde{x}$ using the key key' obtained from the external challenger.

Notice that if the reduction is playing in the experiment $\mathsf{Exp}^{\mathsf{simenc}}_{\mathcal{B},\Pi}(1^\lambda, 0)$ then the distribution of inputs to the adversary is identical to $\mathsf{Hybrid}_{\mathsf{conf}}$. Else, it is distributed identically to Hybrid_1. Thus, the reduction breaks the security of somewhere equivocal encryption. ■

Claim A.2 *Assuming the selective security of garbling scheme for circuits,* $\mathsf{Hybrid}_2 \overset{c}{\approx} \mathsf{Hybrid}_3$

Proof We give a reduction to the security of garbling scheme for circuits.

Generating the Garbled Circuit. For each $g \in [n+1, N+1] \setminus \{g^*\}$, $k \in [\lambda]$ and $b \in \{0,1\}$ sample $\mathsf{lab}^g_{k,b} \leftarrow \{0,1\}^\lambda$. Generate the garbled circuit $\widetilde{C}$ as in Hybrid_2. Note that there is a "hole" in position g^* in the outer encryption layer and hence $\{\mathsf{lab}^{g^*}_{k,b}\}$ is not needed in the generation of $\widetilde{C}$. Also, recall that by assumption the gate $g^* - 1$ is Gray or $g^* = n + 1$.

Generating the Garbled Input: To generate the garbled input $\widetilde{x}$ do:

1. Interact with the garbled circuits challenger and give $\mathsf{SC}[\mathsf{crs}, (r_i, r_j, r_g), (i,j), \{\mathsf{lab}_{k,b}^{g^*+1}\}, 0]$ (where the description of g^* is (i,j)) as the challenge circuit and d_{g^*} as the challenge input. Obtain $\widetilde{\mathsf{SC}}_{g^*}$ and $\{\mathsf{lab}_{k,\mathsf{d}_{g^*,k}}^{g^*}\}_{k\in[\lambda]}$.
2. For each $g \in I' \setminus \{g^*\}$, generate $\widetilde{\mathsf{SC}}_g$ as described in Figure 5. Note that for generating $\widetilde{\mathsf{SC}}_{g^*-1}$ it is sufficient to only know $\{\mathsf{lab}_{k,\mathsf{d}_{g^*,k}}^{g^*}\}_{k\in[\lambda]}$.
3. Execute the rest of the steps as described in Figure 5 and output $\widetilde{x}$.

Notice that if the garbling $\widetilde{\mathsf{SC}}_{g^*}$ is generated using the honest procedure then the inputs to the adversary are distributed identically to Hybrid_2. Else, they are distributed identically to Hybrid_3. Thus, the reduction breaks the selective security of garbling scheme for circuits which is a contradiction.

■

Claim A.3 *Assuming the sender privacy of updatable laconic oblivious transfer,* $\mathsf{Hybrid}_3 \overset{c}{\approx} \mathsf{Hybrid}_4$

Proof

We show that $\mathsf{Hybrid}_3 \overset{c}{\approx} \mathsf{Hybrid}_4$ by giving a reduction to the sender privacy of updatable laconic oblivious transfer.

Generating the Garbled Circuit. Obtain crs from the external challenger and generate the garbled circuit $\widetilde{C}$ as in Hybrid_3.

Generating the Garbled Input: 1. To generate $\widetilde{\mathsf{SC}}_{g^*}$ with the description of g^* equal to (i,j):

(a) Compute e_0, e_1, f_0, f_1 as described in Figure 4.

(b) Interact with the laconic OT challenger by giving D_{g^*} as the challenge database, i as the challenge locations and give f_0, f_1 as the challenge messages. Obtain the challenge ciphertext out.

(c) Compute $\widetilde{\mathsf{SC}}_{g^*} \leftarrow \mathsf{Sim}_{ckt}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}, \{\mathsf{lab}_{k,\mathsf{d}_{g^*,k}}^{g^*}\}_{k\in[\lambda]})$

2. Generate $\widetilde{\mathsf{SC}}_g$ for all $g \in I' \setminus \{g^*\}$ as in Hybrid_3.
3. Execute the rest of the steps as described in Figure 5 to generate the garbled input $\widetilde{x}$.

Note that if out is generated using the honest procedure then the distribution of inputs to the adversary is identical to Hybrid_3. Else, it is identical to Hybrid_4. Thus, the above reduction breaks the sender privacy of updatable laconic oblivious transfer. ■

Claim A.4 *Assuming the sender privacy for writes of updatable laconic oblivious transfer,* $\mathsf{Hybrid}_5 \overset{c}{\approx} \mathsf{Hybrid}_6$.

Proof We give a reduction to the sender privacy for writes of updatable laconic oblivious transfer.

Generating the Garbled Circuit. Obtain crs from the external challenger and generate the garbled circuit $\widetilde{C}$ as in Hybrid_5.

Generating the Garbled Input: 1. To generate $\widetilde{\mathsf{SC}}_{g^*}$ with the description of g^* equal to (i,j):

(a) Interact with the laconic OT challenger by giving D_{g^*} as the challenge database, g^* as the challenge location, D_{g^*+1,g^*} as the challenge bit and $\{\mathsf{lab}_{k,b}^{g^*+1}\}$ as the sequence of challenge messages. It obtains $e_{D_{g^*+1,g^*}}$ as the challenge ciphertext.

(b) Generate $f_{D_{g^*+1,g^*}}$ and out as in Hybrid_5.

(c) Compute $\widetilde{\mathsf{SC}}_{g^*} \leftarrow \mathsf{Sim}_{ckt}(1^\lambda, 1^{|\mathsf{SC}|}, \mathsf{out}, \{\mathsf{lab}_{k,\mathsf{d}_{g^*},k}^{g^*}\}_{k\in[\lambda]})$

2. Generate $\widetilde{\mathsf{SC}}_g$ for all $g \in I' \setminus \{g^*\}$ as in Hybrid_5.
3. Execute the rest of the steps as described in Figure 5 to generate the garbled input $\widetilde{x}$.

Note that if $e_{D_{g^*+1,g^*}}$ is generated using the honest procedure then the distribution of inputs to $\mathcal{A}$ is identical to Hybrid_5. Else, it is identical to Hybrid_6. Thus, the reduction breaks the sender privacy for writes of updatable laconic oblivious transfer. ■