# Cut-and-Choose for Garbled RAM

Peihan Miao*
University of California, Berkeley

**Abstract**

Garbled RAM, introduced by Lu and Ostrovsky (Eurocrypt 2013), provides a novel method to garble RAM (Random Access Machine) programs directly. It can be seen as a RAM analogue of Yao's garbled circuits such that, the size of the garbled program and the time it takes to create and evaluate it, is proportional only to the running time of the RAM program, avoiding the inefficient process of first converting it into a circuit. Secure RAM computation for two parties is a key application of garbled RAM. However, this construction is secure only against semi-honest adversaries.

In this paper we provide a cut-and-choose technique for garbled RAM. This gives the first constant round two-party secure computation protocol for RAM programs secure against malicious adversaries that makes only black-box use of the underlying cryptographic primitives. Our protocol allows for garbling multiple RAM programs being executed on a persistent database. Security of our construction is argued in the random oracle model.

*Research supported in part from a DARPA/ARL SAFEWARE Award, AFOSR Award FA9550-15-1-0274, NSF CRII Award 1464397 and a research grant from the Okawa Foundation. The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

# Contents


**1 Introduction** **3**
- 1.1 Our Results . . . 4
- 1.2 Concurrent and Independent Work . . . 5

**2 Our Techniques** **5**
- 2.1 Roadmap . . . 9

**3 Preliminaries** **9**
- 3.1 Garbled RAM Scheme . . . 9
- 3.2 Garbled RAM Against Malicious Adversaries . . . 10

**4 Black-Box Garbled RAM** **12**
- 4.1 Uniform Circuits . . . 12
- 4.2 Garbled RAM Scheme . . . 12

**5 Adaptive Secure Garbling Schemes** **13**
- 5.1 LEGO Bricks . . . 14
- 5.2 Static Secure LEGO Garbling Scheme . . . 17
- 5.3 Adaptive Secure LEGO Garbling Scheme . . . 20
- 5.4 Adaptive Secure Garbled RAM . . . 21

**6 Generating an Adaptive LEGO Garbled Circuit** **22**
- 6.1 The New LEGO Protocol . . . 22
- 6.2 Analysis . . . 27

**7 Our Construction** **27**
- 7.1 Generating Partial Labels . . . 28
- 7.2 Our Protocol . . . 29
- 7.3 Security . . . 29

**A Building Blocks** **37**
- A.1 RAM Model . . . 37
- A.2 Circuit Garbling Scheme . . . 37
- A.3 Commitment Scheme . . . 38

**B Zero-Knowledge Proof of Knowledge via OT** **38**

**C Postponed Proofs** **39**
- C.1 Proof of Theorem 3 . . . 39
- C.2 Proof of Theorem 5 . . . 40
- C.3 Proof of Theorem 7 . . . 41
- C.4 Proof of Theorem 8 . . . 43

# 1 Introduction

Alice owns a large private database $D$ and wants to store it on the cloud (Bob) in an encrypted form. Subsequently Alice and Bob want to compute and learn the output of arbitrary dynamically chosen programs $P_1, P_2, \cdots$ on their private inputs $x_1 = (x_1^{\mathsf{A}}, x_1^{\mathsf{B}}), x_2 = (x_2^{\mathsf{A}}, x_2^{\mathsf{B}}), \cdots$ and the previously stored database, which gets updated as these programs are executed. During the computation the two parties do not want to leak their private inputs to each other. Can we do this?

Starting with seminal works of Yao [Yao82] and Goldreich, Micali and Wigderson [GMW87], in the past few decades, both theoretical and practical improvements have been pushing the limits of the overall efficiency of such schemes. However most of these constructions are devised only for circuits and securely computing a RAM program involves the inefficient process of first converting it into a circuit. For example, Yao's approach requires that the program be first converted to a circuit — the size of which will need to grow at least with the size of the input. Hence, in the example above, for each program that they want to compute, Alice will need to send a message that grows with the size of the database. Using fully homomorphic encryption [Gen09] we can reduce the size of Alice's message, but the cloud still needs to compute on the entire encrypted database. Consequently the work of the cloud still grows with the size of the database. These solutions can be prohibitive for various applications. For example, in the case of binary search the size of the database can be exponentially larger than execution path of the insecure solution. In other words security comes at the cost of an exponential overhead. We note that additionally even in settings where the size of the database is small, generic transformations from RAM programs with running time $T$ result in a circuit of size $\mathcal{O}\left(T^3 \log T\right)$ [CR73, PF79], which can be prohibitively inefficient.

**Secure computation for RAM programs.** Motivated by the above considerations, various secure computation techniques that work directly for RAM programs have been developed. For instance, Ostrovsky and Shoup [OS97] show how general secure RAM computation can be done using oblivious RAM techniques [Gol87, Ost90, GO96]. Subsequently, Gordon et al. [GKK+12] demonstrated an efficient realization based on specialized number-theoretic protocols. In follow up works [LO13a, WHC+14, AHMR15], significant asymptotic and practical efficiency improvements have been obtained. However, all these works require *round complexity* on the order of the running time of the program.

In a recent line of work [LO13b, GHL+14, GLOS15], positive results on round efficient secure computation for RAM programs have been achieved, and more recently constructions that only make *black-box* use of cryptographic primitive [GLO15] have been constructed. These improvements are obtained by realizing the notion of garbled random-access machines (garbled RAMs) [LO13b] as a method to garble RAM programs directly, a RAM analogue of Yao's garbled circuits [Yao82].

**Semi-honest vs. malicious adversaries.** All two-party secure RAM computation protocols obtained using garbled RAM are secure only against semi-honest adversaries. Hence, an important question is how to "convert" the protocol into one that is secure in the presence of malicious adversaries, while preserving the efficiency, round complexity, and black-box use of underlying cryptographic primitives of the original protocol to the greatest extent possible. Of course, one possibility is to use the compiler of Goldreich, Micali and Wigderson [GMW87]. This compiler converts any protocol that is secure for semi-honest adversaries into one that is secure for malicious adversaries, and as such is a powerful tool for demonstrating feasibility. However, it is based on reducing the statement that needs to be proved (in our case, the honesty of the parties' behavior) to an NP-complete problem, and using generic zero-knowledge proofs to prove this statement. The

resulting secure protocol is therefore rather inefficient. In particular, this procedure will lead to non-black-box use of the underlying cryptographic primitives.

**Persistent vs. non-persistent database.** In the setting of RAM programs, the ability to store a *persistent* private database that can be computed on multiple times can be very powerful. Traditionally, secure computation on RAM programs is thus studied in two settings. In the *non-persistent* database setting, one considers only a single program execution. While in the *persistent* database setting, one considers execution of many programs on the same database; the database can be modified by these programs during the execution and these changes persist for future program executions. This feature is very important as it allows to execute a sequence of programs without requiring to initialize the data for every execution, implying that the database can be huge and the execution time of each program does not need to depend on the size of the database. Previous garbled RAM schemes [GHL$^+$14, GLOS15, GLO15] allow to garble any sequence of programs on a persistent database. We show that our secure RAM computation protocol preserves this property in the presence of malicious adversaries.

**Black-box vs. non-black-box.** Starting with Impagliazzo-Rudich [IR90, IR89], researchers have been very interested in realizing cryptographic goals making only black-box use of underlying primitives. It has been the topic of many important recent works in cryptography [IKLP06, PW09, Wee10, GLOV12, GOSV14, GLO15]. On the other hand, the problem of realizing black-box construction for various primitive is still open, e.g. multi-statement non-interactive zero-knowledge [BFM88, FLS99, GOS06] and oblivious transfer extension [Bea96].[1] From a complexity perspective, black-box constructions are very appealing as they often lead to conceptually simpler and qualitatively more efficient constructions.[2]

Motivated by stronger security guarantee and black-box constructions in the persistent database setting, in this work, we ask the following question:

*Can we securely compute RAM programs on a persistent database against malicious adversaries making only black-box use of cryptographic primitives?*

### 1.1 Our Results

In this paper, we provide the first constant round two-party secure RAM computation protocol making only black-box use of underlying cryptographic primitives with security guarantee against malicious adversaries.

**Main Theorem (Informal).** *There exists a black-box secure RAM computation protocol with constant round complexity which is secure against malicious adversaries, where the size of the database stored by one of the parties is $\tilde{\mathcal{O}}(|D|)$, and the communication and computational complexity of one protocol execution is $\tilde{\mathcal{O}}(t)$ where $t$ is the running time of program $P$ executed in one protocol execution. Here $\tilde{\mathcal{O}}(\cdot)$ ignores $\mathsf{poly}(\log t, \log |D|, \kappa, s)$ factors where $\kappa, s$ are the security parameters. The protocol allows for maintaining a persistent garbled database across execution of multiple programs. Our construction is proved secure in the random oracle model.*

[1] Interestingly for oblivious transfer extension we do know black-box construction based on stronger assumptions [IKNP03].

[2] Additionally, black-box constructions enable implementations agnostic to the implementation of the underlying primitives. This offers greater flexibility allowing for many optimizations, scalability, and choice of implementation.

Our result technically can be seen as an adaptation of LEGO cut-and-choose technique [NO09, FJN+13] for the setting of garbled RAM.

### 1.2 Concurrent and Independent Work

Concurrently with our work, Hazay and Yanai [HY16] also consider the questions of secure 2-party RAM computation against malicious adversaries. They present a constant-round protocol building on the the semi-honest protocols [GHL+14, GLOS15] and cut-and-choose techniques [LP07], but make a non-black-box use of one-way functions. Moreover, they only allow for a weaker notion of persistent database, which we refer to as *weak persistent* setting in Table 1, where all the programs as well as the inputs are known beforehand to the parties.[3] Also concurrently, Garg et al. [GGMP16] study this question in the multiparty setting. They demonstrate a constant-round multi-party computation protocol that makes black-box use of one-way functions. Their work is based on the black-box garbled RAM construction [GLOS15] (as we do) and the constant-round MPC construction of [BMR90]. Their semi-honest secure protocol allows for a persistent database, whereas their maliciously secure protocol achieves the weak persistent notion. See Table 1 for a comparison of our work with prior and concurrent work.

| | **parties** | **security** | **persistence** | **model** | **assumption** | **rounds** |
|---|---|---|---|---|---|---|
| [GHL+14] | 2-party | semi-honest | persistent | OT-hybrid | IBE | $\mathcal{O}(1)$ |
| [GLOS15] | 2-party | semi-honest | persistent | OT-hybrid | non-black-box OWF | $\mathcal{O}(1)$ |
| [GGMP16] | multi-party | semi-honest | persistent | OT-hybrid | black-box OWF | $\mathcal{O}(1)$ |
| [GLO15] | 2-party | semi-honest | persistent | OT-hybrid | black-box OWF | $\mathcal{O}(1)$ |
| [AHMR15] | 2-party | malicious | non-persistent | OT-hybrid | black-box OWF | $\mathcal{O}(t)$ |
| [HY16] | 2-party | malicious | weak persistent | OT-hybrid | non-black-box OWF | $\mathcal{O}(1)$ |
| [GGMP16] | multi-party | malicious | weak persistent | OT-hybrid | black-box OWF | $\mathcal{O}(1)$ |
| [this work] | 2-party | malicious | persistent | OT-hybrid, ROM | black-box OWF | $\mathcal{O}(1)$ |

Table 1: Comparison of this work with prior and concurrent work

## 2 Our Techniques

We build our protocol based on the previous semi-honest secure black-box garbled RAM construction [GLO15]. This serves as a good starting point in explaining the technical challenges that come up in realizing maliciously secure black-box garbled RAM.

**Abstract of [GLO15].** The construction of [GLO15] is complicated in that it involves details of the RAM computation model, structure of the tree-like garbled database consisting of garbled circuits as well as the way to properly concatenate these circuits, statistical ORAMs [DMN11, SCSL11, SvDS+13], etc. We first summarize this construction in an abstracted garbled RAM scheme, and then make it maliciously secure. More generally, if there are other constructions that can also be transformed into this abstracted form, it can be automatically made maliciously secure by our method.

[3] In this paper we focus on the standard persistent notion where later programs and inputs can be chosen adaptively.

At a high level the garbled RAM scheme can be described as follows. The garbled database and garbled programs consist of a collection of Yao's garbled circuits [Yao86], which are concatenated in a certain way. In particular, one garbled circuit may hard-code inside it certain public constants and (partial) input keys to other garbled circuits. The garbler generates all the garbled circuits concatenated properly and sends to the evaluator the garbled circuits along with partial input labels. The evaluator evaluates the garbled circuits one by one in a certain order, during which one garbled circuit may output (partial) input labels for other garbled circuits, enabling the evaluation of the next garbled circuit. This process proceeds until the evaluator obtains the output of the program.

**Technical challenges.** Starting from the construction of [GLO15], one possible method to make it maliciously secure is to compile it into a new protocol against malicious adversaries using standard techniques [GMW87]. The main idea is that the parties commit to their inputs, randomness and intermediate steps in the computation. Then they prove in zero-knowledge that the values inside the commitments were computed according to the protocol. The compiler technique is of great interest from a theoretical point of view, but unfortunately it is not really practical, because of the generic zero-knowledge proofs involved. Moreover, it will compromise the black-box use of cryptographic primitives.

Recall that in [GLO15], the garbler (Alice) generates the garbled database $\tilde{D}$ and garbled programs $\tilde{P}$ consisting of a collection of correlated garbled circuits, and sends to the evaluator (Bob). If Alice is malicious, she may generate incorrect circuits or circuits concatenated in a wrong way. To avoid Alice cheating, one possibility is to treat the garbled database and garbled program $(\tilde{D}, \tilde{P})$ as one large garbled circuit, and apply cut-and-choose techniques to it in a similar way to doing cut-and-choose on a single garbled circuit [LP07]. But cut-and-choose can only be done once for the database, meaning that we can only apply cut-and-choose for the first $(\tilde{D}, \tilde{P})$ pair, and cannot achieve execution of multiple programs on a persistent database. Therefore we turn to apply cut-and-choose on every single circuit, or even on every gate.

The key question that we are trying to answer is: How to enforce Alice to generate all the garbled circuits concatenated in a correct way? In particular, how to ensure that all the hard-coded parameters are correct? In order to check the correctness, we first pull out the hard-coded parameters in the circuit to be part of the input, and then define a public function specifying the correlation among all these circuits, in particular which input wires should take as input an input key to another garbled circuit, and which should take a public value as input, etc. The barrier becomes how to enforce the garbled circuits as well as the correlation between them to be correct.

**Consistency check by commitments.** A natural idea is to let Alice and Bob generate each garbled circuit correctly by cut-and-choose techniques [LP07, NO09, FJN+13, FJNT15]. Then we require Alice to help Bob "concatenate" these circuits satisfying the correlation requirements. Recall that in a Yao's garbled circuit, there are two labels per input wires, which together form the *input-garbling-keys* (which we also refer to as *input keys* in this paper) to a garbled circuit. When Bob evaluates the garbled circuit, he obtains one label per input wire. To ensure correct concatenation, the major task is to guarantee the consistency between input labels of one garbled circuit and input keys to another garbled circuit. For instance, the $i$-th input wire of circuit $X$ should be taking as input the $j$-th bit of circuit $Y$'s input keys. Let $(\mathsf{label}^0, \mathsf{label}^1)$ be the two labels of the $i$-th input wire of $X$, and $b$ be the $j$-th bit of $Y$'s input keys. In order to concatenate the two circuits, Alice must provide Bob with $\mathsf{label}^b$ *without* revealing $b$.

Our first attempt is requiring Alice to give *bit-by-bit* commitments of all the input keys. If Alice

commits honestly, then Bob holds a commitment of $b$ (denoted by $[b]$). In addition, we require Alice to commit to each label together with its corresponding bit, namely $\left[\mathsf{label}^0\right] || [0]$ and $\left[\mathsf{label}^1\right] || [1]$, in a randomly permuted order. When revealing label $\mathsf{label}^b$ to Bob, Alice picks $\left[\mathsf{label}^{b'}\right] || [b']$, opens $\left[\mathsf{label}^{b'}\right]$, and proves to Bob that $b' = b$. An additive/XOR-homomorphic commitment scheme suffices: Bob can compute $[b'] \oplus [b] \rightarrow [b \oplus b']$, and Alice opens the commitment to show that $b \oplus b' = 0$. If every bit-by-bit commitment is correct, then the above procedure ensures that Bob obtains $\mathsf{label}^b$. Now the question boils down to: How to enforce Alice to commit to every bit of the input keys honestly?

**Cut-and-choose on circuits:** Our next attempt is applying cut-and-choose on each single circuit [LP07] to ensure that every garbled circuit is generated correctly. The high level idea of [LP07] is to generate a correct garbled circuit as follows. Alice first generates a number of garbled circuits, and then Bob does cut-and-choose over these circuits, namely Bob randomly picks half of the circuits, asks Alice to reveal all the randomness of generating the picked ones, and checks if they are correctly generated. If all these circuits pass Bob's checking, then with high probability most of the remaining unchecked circuits are also correct. Bob then evaluates all the remaining circuits and takes a majority of the outputs.

As we discussed earlier, we require Alice to give bit-by-bit commitments of all the input keys, but currently we have no guarantee that Alice will do so honestly. In fact, even if she only committed to a single bit incorrectly, it will easily violate our consistency requirement. For the above example, Alice may commit to $\left[\overline{b}\right]$ and later open $\left[\mathsf{label}^{\overline{b}}\right]$ ($\overline{b}$ denotes the negation of bit $b$). To resolve this problem we might need another level of cut-and-choose.

Yet a more severe issue is that since Bob evaluates a number of replicate circuits and takes the majority of the outputs, the input length of the resulting garbled circuit is increased by a factor of the security parameter. This is fine for a single circuit, but if we apply [LP07] for each circuit in $\tilde{D}$ and $\tilde{P}$, where one circuit may take as input (partial) input keys to another circuit, the input length may grow exponentially in the number of circuits.

**Cut-and-choose on gates:** Now we consider cut-and-choose at a gate level, which has been known as LEGO [NO09]. The main idea of LEGO is as follows. Alice first produces a number of components and sends to Bob. Bob randomly picks a subset of the components to be checked, Alice sends the randomness used to generate them, and Bob checks the components. If Bob passes all the checking, then with high probability most of the remaining components are also correct. Bob then permutes the remaining components and use them to build a garbled circuit where each gate will be computed multiple times and Bob takes a majority vote on the output. To connect the remaining components we require Alice to open some commitments to Bob.

LEGO is a better fit for our setting than [LP07] in that the input length of a LEGO garbled circuit is the same as the original circuit, hence it would not grow drastically throughout the circuits. Moreover, since each wire is replicated multiple times, we are able to do more consistency checking among the commitments of replicated wires, and if Bob accepts all the checking, then we can achieve the guarantee that every bit-by-bit commitment is correct, as we elaborate in more detail below.

**XOR-homomorphic commitment scheme.** Next we describe in more detail how to guarantee the correctness of the bit-by-bit commitments. We start from the LEGO protocol of [NO09] and tailer the construction to our needs. There are two major modifications. First, an addi-

tive homomorphic commitment scheme is applied in LEGO, and we replace it with an XOR-homomorphic commitment scheme. The use of an XOR-homomorphic commitment scheme was proposed in [FJN+13, FJNT15] for improving efficiency, and we will see the advantage of this modification in our construction. Second, if Bob accepts all the checking in the LEGO protocol, by cut-and-choose it only guarantees that *most* of the remaining components are correct with high probability. We add more consistency checking among the commitments of input wires to ensure that *all* the commitments of input wires are correct with high probability.

We now discuss the intuition of the additional consistency checking via an example. Assume that an input wire $w$ has $\ell_{\mathsf{n}}+1$ replicate wires, with input keys $(\mathsf{label}^0_{w,0}, \mathsf{label}^1_{w,0})$, $(\mathsf{label}^0_{w,1}, \mathsf{label}^1_{w,1})$, $\cdots$, $(\mathsf{label}^0_{w,\ell_{\mathsf{n}}}, \mathsf{label}^1_{w,\ell_{\mathsf{n}}})$. Alice has provided a bit-by-bit commitment of all the input keys. Denote the first bit of $\mathsf{label}^0_{w,i}$ by $b_i$. As a demonstration, we explain how to ensure that Alice has committed to $[b_i]$ correctly. In the LEGO protocol, when concatenating components to build a garbled circuit, Bob obtains the difference between certain labels, in particular $\delta_{w,i} = \mathsf{label}^0_{w,i} \oplus \mathsf{label}^0_{w,0}$. The additional consistency checking is as follows. Note that $b_i \oplus b_0$ should be the first bit of $\delta_{w,i}$. Bob can compute $[b_i \oplus b_0] \leftarrow [b_i] \oplus [b_0]$. We require Alice to open the resulting commitment and Bob to check if it is equal to the first bit of $\delta_{w,i}$. Since cut-and-choose is done at a gate level, with high probability at least one of the replicate wires has correct bit-by-bit commitments. If Bob passes all the additional consistency checking, the correctness of a single wire will spread out to all the replicate wires. The above is merely one example of additional consistency checking, and in the protocol we need more consistency checking to ensure correctness of all the commitments.

**Practical efficiency.** We illustrate our idea of additional consistency checking on the original LEGO framework [NO09] and not on the more efficient LEGO protocols [FJN+13, FJNT15], because [NO09] is simpler and helps highlight our ideas better. We note that the focus of this work is the theoretical feasibility of achieving maliciously secure RAM computation protocols making only black-box use of cryptographic primitives, rather than practical efficiency. Nevertheless, it is worth pointing out that the LEGO protocols of [FJN+13, FJNT15] are compatible with all known optimizations for Yao's garbled circuits, e.g., point-and-permute [BMR90, MNP+04], free-XORs [KS08], garbled row reduction [NPS99, PSSW09, KMR14, ZRE15], etc.). We have left the goal of obtaining concrete efficiency improvements over our construction for future work. However, we believe that these improvements should be obtainable.

**Adaptive security.** Since we execute our protocol on a sequence of programs which run on a persistent database, the inputs of the adversary could be adaptively decided after the two parties have generated the garbled database and garbled programs which consist of garbled circuits. Thus we need to argue *adaptive security* of the garbled RAM. More precisely, security should hold even when the input on which the stored garbled RAM is computed can depend on the garbled RAM itself. Towards this goal, we first modify the LEGO garbled circuit to be adaptively secure in the random oracle model, the idea of which is similar as [BHR12a] on making Yao's garbled circuit adaptively secure in the random oracle model. Then we instantiate the garbled RAM with the adaptive secure LEGO garbled circuits to achieve an adaptive secure garbled RAM.

**Storage costs.** Finally, we mention that the client (Alice) can store the whole database on the cloud (Bob), and it is not necessary for her to store all the input keys, randomness, etc. on her disk. She may store all these things on the cloud signed and in an encrypted form (by private key encryption), and request for them when needed. Therefore, the client storage remains small after the garbled database has been created.

### 2.1 Roadmap

We now lay out a roadmap for the remainder of the paper. In Section 3 we give definitions for semi-honest secure and maliciously secure garbled RAM. In Section 4 we abstract the static secure black-box garbled RAM construction. We review the LEGO garbling scheme and describe our modified construction which is adaptive secure in Section 5, and present our new LEGO protocol in Section 6. Finally we give our secure RAM computation protocol in Section 7.

## 3 Preliminaries

In this section, we formally define the security of garbled RAM against semi-honest and malicious adversaries. For a brief description of the RAM model, circuit garbling schemes, and some building blocks needed in the construction, refer to Appendix A. In the following, let $\kappa$ be the computational security parameter for the commitment schemes, oblivious transfers, encryption schemes and hash functions used, and $s$ be the statistical security parameter.

### 3.1 Garbled RAM Scheme

In this section, we consider an extension of garbled circuits to the setting of RAM programs, as defined in [LO13b, GHL+14, GLOS15, GLO15]. In this setting the database data $D$ is garbled once and then many different garbled programs can be executed sequentially with the database changes persisting from one execution to the next. Note that all the previous work only defines static security for garbled RAM, and we also define adaptive security here.

**Syntax.** A garbled RAM scheme consists of four procedures $(\mathsf{GData}, \mathsf{GProg}, \mathsf{GInput}, \mathsf{GEval})$ with the following syntax:

- *Database Garbling:* $(\tilde{D}, s) \leftarrow \mathsf{GData}(1^\kappa, D)$ takes as input the security parameter $\kappa$ and database content $D \in \{0,1\}^M$, and outputs a garbled database $\tilde{D}$ and a key $s$.
- *Program Garbling:* $(\tilde{P}, s^{in}) \leftarrow \mathsf{GProg}(s, P, M, t, T)$ takes as input a key $s$ and a RAM program $P$ with database-size $M$ and run-time consisting $t$ CPU steps. We also provide $T$ indicating the cumulative number of CPU steps executed by all of the previous programs. It then outputs the garbled program $\tilde{P}$ and input-garbling-key $s^{in}$.
- *Input Garbling:* $\tilde{x} \leftarrow \mathsf{GInput}(x, s^{in})$ takes as input $x \in \{0,1\}^u$ and input keys $s^{in}$, and outputs the garbled input $\tilde{x}$.
- *Garbled Evaluation:* $y \leftarrow \mathsf{GEval}^{\tilde{D}}(\tilde{P}, \tilde{x})$ takes as input a garbled program $\tilde{P}$, garbled input $\tilde{x}$ and garbled database data $\tilde{D}$, and outputs a value $y$. We model $\mathsf{GEval}$ itself as a RAM program that can read and write to arbitrary locations of its database initially containing $\tilde{D}$.

**Efficiency.** We require the run-time of $\mathsf{GProg}$ and $\mathsf{GEval}$ to be $t \cdot \mathsf{poly}(\log M, \log t, \kappa)$, which also serves as the bound on the size of the garbled program $\tilde{P}$. Moreover, we require that the run-time of $\mathsf{GData}$ should be $M \cdot \mathsf{poly}(\log M, \kappa)$, which also serves as an upper bound on the size of $\tilde{D}$. Finally the running time of $\mathsf{GInput}$ is required to be $u \cdot \mathsf{poly}(\kappa)$.

**Correctness.** For correctness, we require that for any sequence of programs $P_1, \cdots, P_\ell$ with run-time $t_1, \cdots, t_\ell$, let $D \in \{0,1\}^M$ be any initial database data, let $x_1, \cdots, x_\ell$ be the inputs of the

programs and $(y_1, \cdots, y_\ell) = (P_1(x_1), \cdots, P_\ell(x_\ell))^D$ be the outputs given by the sequential execution of the programs. We have that:

$$\Pr\left[\left(\mathsf{GEval}(\tilde{P}_1, \tilde{x}_1), \cdots, \mathsf{GEval}(\tilde{P}_\ell, \tilde{x}_\ell)\right)^{\tilde{D}} = (y_1, \cdots, y_\ell)\right] = 1$$

where $(\tilde{D}, s) \leftarrow \mathsf{GData}(1^\kappa, D), (\tilde{P}_i, s_i^{in}) \leftarrow \mathsf{GProg}(s, P_i, M, t_i, T_i), \tilde{x}_i \leftarrow \mathsf{GInput}(x_i, s_i^{in})$, where $T_i = \sum_{j=1}^{i-1} t_j$ denotes the run-time of all programs prior to $P_i$.

**Static security.** For static security, we require that there exists a PPT simulator $\mathsf{StatRamSim}$ such that for any PPT adversary $\mathcal{A}$, any initial database content $D \in \{0,1\}^M$, and any polynomially bounded $\ell$, the output of the following two experiments are computational indistinguishable:

Real experiment

- $(\tilde{D}, s) \leftarrow \mathsf{GData}(1^\kappa, D)$
- For $i = 1, 2, \cdots, \ell$
  - $(P_i, x_i) \leftarrow \mathcal{A}(1^\kappa, i)$
  - $(\tilde{P}_i, s_i^{in}) \leftarrow \mathsf{GProg}(s, P_i, M, t_i, T_i)$
  - $\tilde{x}_i \leftarrow \mathsf{GInput}(x_i, s_i^{in})$
- Output $\left(\tilde{D}, \left\{\left(\tilde{P}_i, \tilde{x}_i\right)\right\}_{i=1}^{\ell}\right)$

Simulated experiment

- $\left(\tilde{D}^{\mathsf{sim}}, \mathsf{state}_0\right) \leftarrow \mathsf{StatRamSim}\left(1^\kappa, 1^M\right)$
- For $i = 1, 2, \cdots, \ell$
  - $(P_i, x_i) \leftarrow \mathcal{A}(1^\kappa, i)$
  - $(\tilde{P}_i^{\mathsf{sim}}, \tilde{x}_i^{\mathsf{sim}}, \mathsf{state}_i) \leftarrow \mathsf{StatRamSim}(\mathsf{state}_{i-1}, P_i, 1^{t_i}, y_i)$

  (where $(y_1, \cdots, y_\ell) = (P_1(x_1), \cdots, P_\ell(x_\ell))^D$)
- Output $\left(\tilde{D}^{\mathsf{sim}}, \left\{\left(\tilde{P}_i^{\mathsf{sim}}, \tilde{x}_i^{\mathsf{sim}}\right)\right\}_{i=1}^{\ell}\right)$

**Adaptive security.** For adaptive security, we require that there exists a PPT simulator $\mathsf{AdaptRamSim}$ such that for any PPT adversary $\mathcal{A}$, any initial database content $D \in \{0,1\}^M$, and any polynomially bounded $\ell$, the output of the following two experiments are computational indistinguishable:

Real experiment

- $(\tilde{D}, s) \leftarrow \mathsf{GData}(1^\kappa, D)$
- For $i = 1, 2, \cdots, \ell$
  - $P_i \leftarrow \mathcal{A}\left(1^\kappa, \tilde{D}, \left\{(\tilde{P}_j, \tilde{x}_j)\right\}_{j=1}^{i-1}\right)$
  - $(\tilde{P}_i, s_i^{in}) \leftarrow \mathsf{GProg}(s, P_i, M, t_i, T_i)$
  - $x_i \leftarrow \mathcal{A}\left(1^\kappa, \tilde{D}, \left\{(\tilde{P}_j, \tilde{x}_j)\right\}_{j=1}^{i-1}, \tilde{P}_i\right)$
  - $\tilde{x}_i \leftarrow \mathsf{GInput}(x_i, s_i^{in})$
- Output $\left(\tilde{D}, \left\{\left(\tilde{P}_i, \tilde{x}_i\right)\right\}_{i=1}^{\ell}\right)$

Simulated experiment

- $\left(\tilde{D}^{\mathsf{sim}}, \mathsf{state}_0\right) \leftarrow \mathsf{AdaptRamSim}\left(1^\kappa, 1^M\right)$
- For $i = 1, 2, \cdots, \ell$
  - $P_i \leftarrow \mathcal{A}\left(1^\kappa, \tilde{D}, \left\{(\tilde{P}_j, \tilde{x}_j)\right\}_{j=1}^{i-1}\right)$
  - $(\tilde{P}_i^{\mathsf{sim}}, \mathsf{state}_i') \leftarrow \mathsf{AdaptRamSim}(\mathsf{state}_{i-1}, P_i, 1^{t_i}, y_i)$
  - $x_i \leftarrow \mathcal{A}\left(1^\kappa, \tilde{D}, \left\{(\tilde{P}_j, \tilde{x}_j)\right\}_{j=1}^{i-1}, \tilde{P}_i\right)$
  - $(\tilde{x}_i^{\mathsf{sim}}, \mathsf{state}_i) \leftarrow \mathsf{AdaptRamSim}(\mathsf{state}_i', y_i)$

  where $(y_1, \cdots, y_\ell) = (P_1(x_1), \cdots, P_\ell(x_\ell))^D$
- Output $\left(\tilde{D}^{\mathsf{sim}}, \left\{\left(\tilde{P}_i^{\mathsf{sim}}, \tilde{x}_i^{\mathsf{sim}}\right)\right\}_{i=1}^{\ell}\right)$

### 3.2 Garbled RAM Against Malicious Adversaries

We define security of a secure RAM computation protocol against malicious adversaries in the ideal/real world paradigm. The definition compares the output of a real execution to the output of an ideal computation involving a trusted third party, which we call ideal functionality. The ideal functionality receives the parties' inputs, computes the functionality on these inputs and returns their respective outputs. Loosely speaking, the protocol is secure if any real-world adversary can

be converted into an ideal-world adversary such that the output distributions are computationally indistinguishable.

**Database:** Alice inputs $(\mathsf{InitialData}, D)$. The ideal functionality parses it as $D \in \{0,1\}^M$.

**Program and input:** Alice inputs $(\mathsf{NewProgram}, \mathsf{A}, P^{\mathsf{A}}, x^{\mathsf{A}})$, Bob inputs $(\mathsf{NewProgram}, \mathsf{B}, P^{\mathsf{B}}, x^{\mathsf{B}})$. If $P^{\mathsf{A}} \neq P^{\mathsf{B}}$, then the ideal functionality outputs $\mathsf{disagreement!}$ to both parties and terminates. Otherwise, let $P = P^{\mathsf{A}}$ and parse $P$ as a program with input $x = (x^{\mathsf{A}}, x^{\mathsf{B}})$.

**Evaluation and output:** The ideal functionality computes $y = P^D(x)$, outputs $y$ to Bob, and updates $D$ for the execution of the next program.

**Corrupted party:** The corrupted party may deviate from its input, may abort the procedure by sending $\mathsf{abort!}$ to the ideal functionality, and can decide the time of message delivery. In addition, if Alice is corrupted, she can specify to the ideal functionality a set $\{i, \beta_i\}_{i \in I}$ where $I \subseteq \{1, \cdots, |x^{\mathsf{B}}|\}$ and $\beta_i \in \{0,1\}$, where $|x^{\mathsf{B}}|$ is the length of Bob's input. If $\beta_i = x_i^{\mathsf{B}}$ for every $i \in I$, then the ideal functionality outputs $\mathsf{correct!}$ to Alice. Otherwise, it outputs $\mathsf{wrong!}$ to Alice and outputs $\mathsf{Alice\ cheat!}$ to Bob.

Figure 1: The ideal functionality

**Execution in the ideal world.** We describe the ideal functionality $\mathcal{F}$ in Figure 1. Note that it is "insecure" similarly as [NO09] in the sense that it allows Alice to guess Bob's input bits, but if her guess is wrong then Bob is told that Alice is cheating. This models a standard problem in Yao's garbled circuits known as "selective failure attack", which can be solved by modifying the circuit being evaluated to first compute a function of a randomized encoding of Bob's input, where any $s$ bits are uniformly random and independent. This allows us to only argue that Alice can guess $s$ or more bits with probability at most $2^{-s}$, where $s$ is the statistical security parameter, since guessing fewer bits does not leak information. One method for this is given in [LP07]. The extra number of gates used is $\mathcal{O}\left(|x^{\mathsf{B}}| + s\right)$. From now on we focus on implementing the slightly insecure ideal functionality above.

**Execution in the real world.** We next consider the real world where the protocol $\Pi$ is executed. $\Pi$ consists of four protocols $(\Pi_{\mathsf{GData}}, \Pi_{\mathsf{GProg}}, \Pi_{\mathsf{GInput}}, \Pi_{\mathsf{GEval}})$ with the following functionality:

- *Database Garbling* $\Pi_{\mathsf{GData}}$. With Alice's input $D$, the protocol outputs a key $s$ to Alice and the garbled database $\tilde{D}$ to Bob.
- *Program Garbling* $\Pi_{\mathsf{GProg}}$. The program $P$ is known to both parties. With Alice's input $s$, the protocol outputs an input-garbling-key $s^{in}$ to Alice and the garbled program $\tilde{P}$ to Bob.
- *Input Garbling* $\Pi_{\mathsf{GInput}}$. With Alice's input $(s^{in}, x^{\mathsf{A}})$ and Bob's input $x^{\mathsf{B}}$, the protocol outputs the garbled input $\tilde{x}$ to Bob, where the input to the program is $x = (x^{\mathsf{A}}, x^{\mathsf{B}})$.
- *Garbled Evaluation* $\Pi_{\mathsf{GEval}}$. It is a procedure executed by Bob himself to compute $y = P^D(x)$ from $\tilde{D}, \tilde{P}, \tilde{x}$. The garbled database is updated during the evaluation for the execution of the next program.

**Efficiency.** Considering the communication and computation complexity, we require that the complexity of $\Pi_{\mathsf{GData}}$ be bounded by $M \cdot \mathsf{poly}(\log M, \kappa, s)$, and that of $\Pi_{\mathsf{GProg}}, \Pi_{\mathsf{GEval}}$ be bounded by $t \cdot \mathsf{poly}(\log M, \log t, \kappa, s)$. Besides, the complexity of $\Pi_{\mathsf{GInput}}$ is required to be $u \cdot \mathsf{poly}(\kappa, s)$. Moreover, the round complexity of $\Pi_{\mathsf{GData}}, \Pi_{\mathsf{GProg}}, \Pi_{\mathsf{GInput}}$ are required to be constant.

**Security against malicious adversaries.** We say that the protocol $\Pi$ is secure against malicious adversaries if for every pair of PPT adversary $\mathcal{A}$ in the real world, there exists a pair of PPT adversary $\mathcal{S}$ in the ideal world, such that with probability greater than $1 - 2^{-s}$, we have

$$\left\{\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}\left(D, \{P_i, x_i\}_{i=1}^{\ell}\right)\right\}_{D,\ell,\{P_i,x_i\}_{i=1}^{\ell}} \overset{c}{\approx} \left\{\text{REAL}_{\mathcal{A}}^{\Pi}\left(D, \{P_i, x_i\}_{i=1}^{\ell}\right)\right\}_{D,\ell,\{P_i,x_i\}_{i=1}^{\ell}}$$

where $\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}\left(D, \{P_i, x_i\}_{i=1}^{\ell}\right)$ denotes the output of $\mathcal{S}$ in the ideal world, and $\text{REAL}_{\mathcal{A}}^{\Pi}\left(D, \{P_i, x_i\}_{i=1}^{\ell}\right)$ denotes the output of $\mathcal{A}$ in the real world. Here $\ell$ is polynomially bounded, $D$ is the initial database content, and the programs are $P_i$ with input $x_i = (x_i^{\mathsf{A}}, x_i^{\mathsf{B}})$.

# 4 Black-Box Garbled RAM

In this section we abstract the construction of the black-box garbled RAM (GRAM) scheme [GLO15]. This abstraction captures the key aspects of the GRAM construction relevant to us. Additionally it avoids the details irrelevant for understanding our work.

At a high level the GRAM scheme can be described as follows. The garbled database and garbled programs consist of a collection of garbled circuits (GCs) concatenated in a certain way, which we will elaborate in more detail later. The garbler generates all the GCs concatenated properly and sends to the evaluator the GCs with partial labels. The evaluator evaluates the GCs one by one in a certain order, during which one GC may output (partial) labels for other GCs, enabling the evaluation of the next GC. This process proceeds until the evaluator obtains the output of the program.

To formalize and generalize the above scheme, we first define a uniform circuit needed in the construction, and then describe the scheme built on the uniform circuits. To avoid ambiguity, in the following the term *input keys* refers to both labels per input wire of a GC while *labels* refers to one label per wire.

## 4.1 Uniform Circuits

In the scheme, garbled database and garbled programs consist of a collection of GCs. We consider these circuits as *uniform circuits*, which all have the same topology. Now we give a brief description of these uniform circuits. The input of a uniform circuit consists of several parts, $\mathsf{const}, \mathsf{keys}, \mathsf{mem}, \mathsf{inp}, \mathsf{dyn}$, as shown in Figure 2. Note that it is not necessary to delve into the functionality of each circuit to understand the abstracted construction, so we omit the functionality and only give the interface.[4]

## 4.2 Garbled RAM Scheme

In this section we describe the black-box GRAM scheme built on uniform circuits. At a high level, garbled database and garbled programs consist of a collection of garbled uniform circuits which are *concatenated properly*. We will specify in the following how the generated GCs are concatenated, in particular what is value of every input wire of every circuit.

[4]For the readers who are familiar with [GLO15]: Each GC originally had two parameters $\mathsf{const}$ and $\mathsf{keys}$ hard-coded inside it, and had $\mathsf{mem}$, $\mathsf{inp}$ or $\mathsf{dyn}$ as input. Now $\mathsf{const}$ and $\mathsf{keys}$ are pulled out to be part of the input, and every circuit takes $\mathsf{mem}, \mathsf{inp}, \mathsf{dyn}$ as part of the input, so that all the circuits have the same topology.

$\mathsf{C}^{\mathsf{unif}}(\mathsf{const}, \mathsf{keys}, \mathsf{mem}, \mathsf{inp}, \mathsf{dyn})$:

- const: A public constant, specifying the functionality of the circuit.
- keys: A collection of (partial) input keys of certain other GCs. The information of which (partial) input keys it corresponds to is public.
- mem: Partial content of the database.
- inp: Partial content of the program input.
- dyn: Dynamic input. Its value is unknown at the time of garbling, and its label will be outputted by other circuits at runtime.

Figure 2: The interface of a uniform circuit

| category of $X[i]$ | $f(X,i)$ | value of $X[i]$ |
|---|---|---|
| const | $b = 0/1$ | $b$ |
| keys | $(Y, j)$ | $j$-th bit of input keys to $Y$ |
| mem | $j \in [M]$ | $D[j]$ |
| inp | $j \in [u]$ | $x[j]$ |
| dyn | null | unknown |

Table 2: The concatenation of the circuits

Recall that the database $D$ has size $M$, program $P$ has running time $t$. The input $x$ of program $P$ has length $u$. Let $\tilde{M}$ be the number of GCs needed for the garbled database and $\tilde{t}$ be the number of GCs needed for the garbled program.[5] First we number all the GCs, including those for the garbled database as well as the garbled programs. Denote the $i$-th input wire of circuit $X$ by $X[i]$. Then we define a public function $f$ specifying the value of every input wire of every circuit, as in Table 2. Given the output of $f$, the scheme is described in Figure 3.

The above garbled RAM constructed in [GLO15] is a static secure garbled RAM scheme that only makes a black-box use of one-way functions. In particular, it proves the following theorem.

**Theorem 1** (Static security of the garbled RAM scheme [GLO15])**.** *The garbled RAM scheme* $(\mathsf{GData}, \mathsf{GProg}, \mathsf{GInput}, \mathsf{GEval})$ *achieves efficiency, correctness, and static security as defined in Section 3.1. Moreover, the construction only makes a black-box use of one-way functions.*

# 5 Adaptive Secure Garbling Schemes

The circuit garbling protocol of [NO09] implies a special circuit garbling scheme, which we refer to in the following as LEGO garbling scheme, denoted by $(\mathsf{LegoGCircuit}, \mathsf{LegoGInput}, \mathsf{LegoEval}, \mathsf{LegoEvalCorrupt})$. It has the same syntax as the a secure circuit garbling scheme, but with a special structure of replicate gates. The LEGO garbling scheme satisfies correctness and static security defined in Appendix A.2. Moreover, the additional procedure $\mathsf{LegoEvalCorrupt}$ satisfies the robust correctness property, which we will define below.

In this section, we start from building units of the LEGO garbling scheme in Section 5.1, and review the garbling scheme of LEGO in Section 5.2. Then we present our modifications on the

[5] $\tilde{M}$ is proportional to $M$ with poly-logarithmic factors, and $\tilde{t}$ is proportional to $t$ with poly-logarithmic factors. For our purpose we do not need to specify the concrete numbers, but one may refer to [GLO15] for details.

**Database Garbling:** $(\tilde{D}, s) \leftarrow \mathsf{GData}(1^\kappa, D)$.

1. *Generating input keys:* Pick a PRF seed $s$ uniformly at random and generate all the input keys needed for the garbled circuits.
2. *Generating garbled circuits:* Generate all garbled uniform circuits $\{\tilde{\mathsf{C}}^i\}_{i=1}^{\tilde{M}}$ by a circuit garbling scheme.
3. *Generating partial labels:* Let $\mathcal{L}$ be the set of input labels consistent with $f$. In particular, for each wire $X[i]$ with category $\mathsf{const/keys/mem}$, pick the correct label for $X[i]$ according to $f(X, i)$ and $D$.
4. *Output:* $\tilde{D} = \left(\{\tilde{\mathsf{C}}^i\}_{i=1}^{\tilde{M}}, \mathcal{L}\right), s$.

**Program Garbling/Replenishing:** $(\tilde{P}, s^{in}) \leftarrow \mathsf{GProg}(s, P, M, t, T)$.

1. *Generating input keys:* Use $s$ to generates all the input keys needed for the new garbled circuits.
2. *Generating garbled circuits:* Let $N$ be the total number of previously generated GCs. Generate all new garbled uniform circuits $\{\tilde{\mathsf{C}}^i\}_{i=N+1}^{N+\tilde{t}}$ by a circuit garbling scheme.
3. *Generating partial labels:* Let $\mathcal{L}$ be the set of labels consistent with $f$ for all wires $X[i]$ with category $\mathsf{const/keys}$.
4. *Generating input-garbling-key:* Let $s^{in}$ be the set of input keys to all wires $X[i]$ with category $\mathsf{inp}$.
5. *Output:* $\tilde{P} = \left(\{\tilde{\mathsf{C}}^i\}_{i=N+1}^{N+\tilde{t}}, \mathcal{L}\right), s^{in}$.

**Input Garbling:** $\tilde{x} \leftarrow \mathsf{GInput}(x, s^{in})$.

1. *Parsing input-garbling-key:* Parse $s^{in}$ as partial input keys to currently generated GCs.
2. *Generating garbled input:* Let $\tilde{x}$ be the set of labels consistent with $f$ and $x$. In particular, for each wire $X[i]$ with category $\mathsf{inp}$ and $f(X, i) = j$, pick a label from $s^{in}$ according to $x[j]$.
3. *Output:* $\tilde{x}$.

**Garbled Evaluation:** $y \leftarrow \mathsf{GEval}^{\tilde{D}}(\tilde{P}, \tilde{x})$.

1. With all input labels of $\tilde{\mathsf{C}}^{N+1}$, start the evaluation from $\tilde{\mathsf{C}}^{N+1}$.
2. Evaluate the GCs one by one until no more GC can be evaluated.
   - One GC will output (partial) labels for other GCs.
   - Once obtaining all labels for a GC, evaluate that GC, and repeat.
3. The output of the last evaluated GC is $y$.

Figure 3: Garbled RAM scheme abstraction

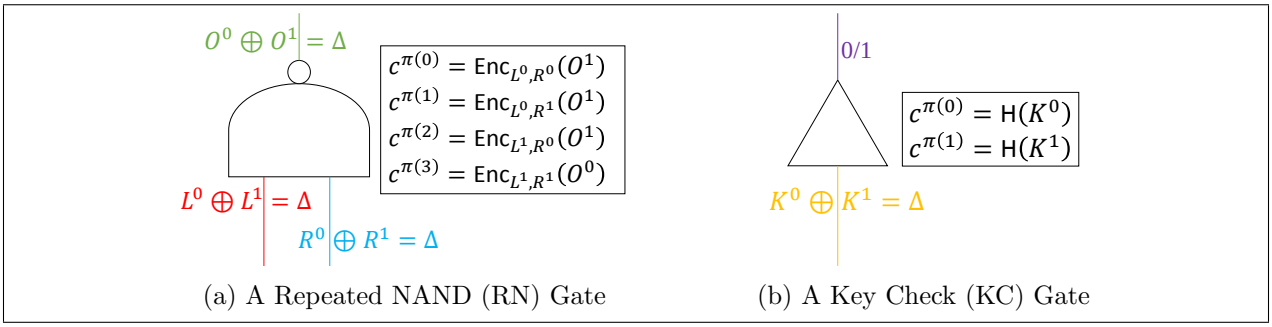
LEGO garbling scheme to make it adaptively secure in the random oracle model in Section 5.3. Finally, we instantiate the garbled RAM scheme in Section 4 with the adaptive LEGO garbling scheme to obtain an adaptive secure garbled RAM scheme in Section 5.4.

## 5.1 LEGO Bricks

We first give a high-level overview of LEGO garbling scheme. Similar to Yao's garbling scheme [Yao86], every wire of the circuit is assigned with two labels, one representing the wire being 0, the other representing the wire being 1. Distinct from Yao's garbling scheme, each gate is repeatedly evaluated $\ell_{\mathsf{n}} + 1$ times, and then the output label is checked $\ell_{\mathsf{k}} + 1$ times for validity, where $\ell_{\mathsf{n}}, \ell_{\mathsf{k}}$ are the replication parameters. The replication is crucial for the cut-and-choose techniques. Moreover,

all the gates were first garbled separately, and then "soldered" together. Finally, in order to solder properly there is a *global difference* $\Delta$ between the two labels per wire. In particular, $\mathsf{label}_w^0$ and $\mathsf{label}_w^1$ for each wire $w$ has the property that $\mathsf{label}_w^0 \oplus \mathsf{label}_w^1 = \Delta$. Without loss of generality we assume that all gates are NAND gates in the following discussion.

**Components.** We start from two types of components needed in the garbling scheme.




(a) A Repeated NAND (RN) Gate (b) A Key Check (KC) Gate

Figure 4: Garbling components: RN and KC gates

*RN (Repeated NAND) gate:* A RN gate has two input wires and one output wire. Each wire is assigned with two labels with the global difference between them. Let $\pi$ be a random permutation of $\{0,1,2,3\}$. The garbled gate table consists of four encryptions as shown in Figure 8a, where $\mathsf{Enc}$ is a symmetric-key encryption procedure. Given the garbled gate table and two input labels $L^a, R^b$ $(a,b \in \{0,1\})$, one can obtain the corresponding output label $O^{a \otimes b}$, where the NAND operation is denoted by $\otimes$.

*KC (Key Check) gate:* A KC gate is assigned with two labels $K^0, K^1$ with the global difference between them. With a random permutation $\pi$ over $\{0,1\}$,[6] the garbled gate table is defined as in Figure 8b, where $\mathsf{H}$ is a hash function. Given the garbled gate table and one label $K$, the KC gate checks if $K$ is valid (i.e., $K \in \{K^0, K^1\}$).

**Garbling a single gate.** We next describe how to garble a single NAND gate $g$. The garbled gate $\tilde{g}$ consists of $\ell_{\mathsf{n}}+1$ RN gates and $\ell_{\mathsf{k}}+1$ KC gates. Let $\mathsf{RN}_{g,0}, \cdots, \mathsf{RN}_{g,\ell_{\mathsf{n}}}$ be the RN gates and $\mathsf{KC}_{g,0}, \cdots, \mathsf{KC}_{g,\ell_{\mathsf{k}}}$ be the KC gates, assigned with labels shown in Figure 5. $\delta$'s are define for the purpose of "soldering" the garbled gates together. The garbled gate table of $\tilde{g}$ is shown in Figure 6.

**Evaluating a (correct) garbled gate.** Given two input labels for a correctly constructed garbled gate, we can use the garbled gate table to evaluate it as follows.

- *Input:* $L_g^a, R_g^b$ $(a,b \in \{0,1\}$ are hidden values), (correct) garbled gate table of $\tilde{g}$.
- *Output:* $O_g^{a \otimes b}$.
- *Evaluation:*
  - Use $L_g^a, R_g^b$ and the garbled gate table to compute $L_{g,i}^a, R_{g,i}^b$ and the corresponding output label $O_{g,i}^{a \otimes b}$ for $\forall 0 \le i \le \ell_{\mathsf{n}}$. ($\{O_{g,i}^{a \otimes b}\}_{i=0}^{\ell_{\mathsf{n}}}$ are consistent with $\{\delta_{g,i}^O\}_{i=1}^{\ell_{\mathsf{n}}}$.)

[6] In principle $\pi$ can be viewed as a random bit. We define it as a permutation only for consistency with the definition of *RN* gates.

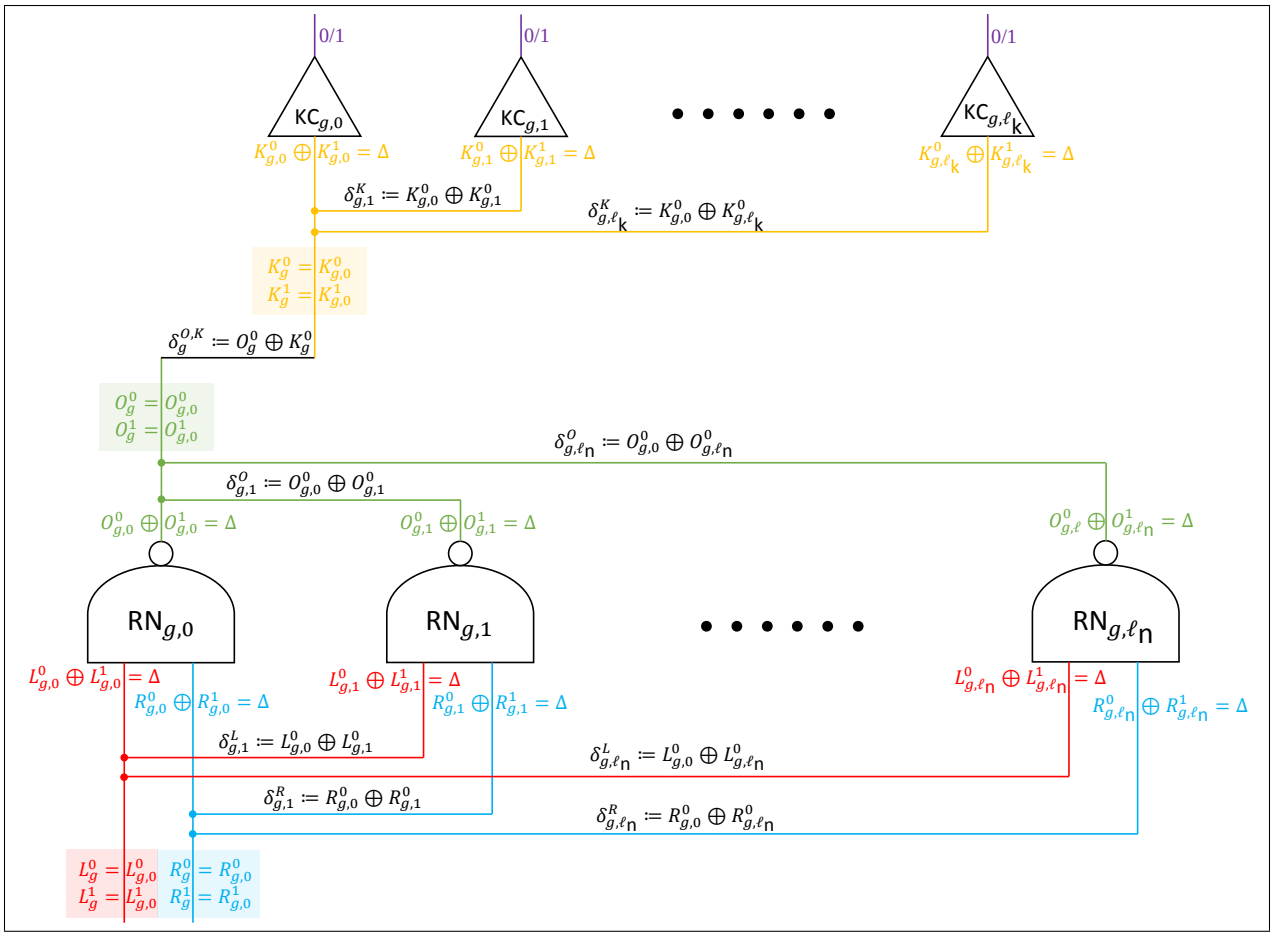



Figure 5: Garbling a single gate

Garbled gate table of $\mathsf{RN}_{g,i}, \forall i \in \{0, 1, \cdots, \ell_\mathsf{n}\}$;
Garbled gate table of $\mathsf{KC}_{g,i}, \forall i \in \{0, 1, \cdots, \ell_\mathsf{k}\}$;
Soldering of RN gates: $\delta^L_{g,i}, \delta^R_{g,i}, \delta^O_{g,i}, \forall i \in \{1, \cdots, \ell_\mathsf{n}\}$;
Soldering of KC gates: $\delta^K_{g,i}, \forall i \in \{1, \cdots, \ell_\mathsf{k}\}$;
Soldering of KC gates with RN gates: $\delta^{O,K}_g$.

Figure 6: Garbled gate table of $\tilde{g}$

- Let $O^{a\otimes b}_g := O^{a\otimes b}_{g,0}$ and $K^{a\otimes b}_g := O^{a\otimes b}_g \oplus \delta^{O,K}_g$. (Let $K^{a\otimes b}_{g,0} := K^{a\otimes b}_g$ and $K^{a\otimes b}_{g,i} := K^{a\otimes b}_{g,0} \oplus \delta^K_{g,i}$, then $K^{a\otimes b}_{g,i}$ can pass the checking of $\mathsf{KC}_{g,i}$ for $\forall 0 \le i \le \ell_\mathsf{k}$.)
- Output $O^{a\otimes b}_g$.

**Evaluating a corrupted garbled gate.** The garbled gate table can be corrupted in arbitrary ways, but we only consider (and later our protocol guarantees) the case where the labels

$\{L_g^b, R_g^b, O_g^b\}_{b=0,1}$ and the solders $\left\{\delta_{g,i}^L, \delta_{g,i}^R, \delta_{g,i}^O\right\}_{i=1}^{\ell_\mathsf{n}}, \left\{\delta_{g,i}^K\right\}_{i=1}^{\ell_\mathsf{k}}, \delta_g^{O,K}$ are all correct, and only the garbled tables of RN and KC gates could be incorrectly formed.

- *Input:* $L_g^a, R_g^b$ ($a, b \in \{0,1\}$ are hidden values), corrupted garbled gate table of $\tilde{g}$.
- *Output:* $O_g^{a \otimes b}$.
- *Evaluation:*
  - Use $L_g^a, R_g^b$ and the garbled gate table to compute $L_{g,i}^a, R_{g,i}^b$ and the corresponding output label $O_{g,i}^{a\otimes b}$ for $\forall 0 \le i \le \ell_\mathsf{n}$. Note that $L_{g,i}^a, R_{g,i}^b$ are guaranteed to be correct, but $O_{g,i}^{a\otimes b}$ could be wrong.
  - Let $\tilde{O}_{g,i}^{a\otimes b} := O_{g,i}^{a\otimes b} \oplus \delta_{g,i}^O$.
  - For every $i \in \{0, 1, \cdots, \ell_\mathsf{n}\}$, and for every $j \in \{0, 1, \cdots, \ell_\mathsf{k}\}$, let $\tilde{K}_{g,i,j} := \tilde{O}_{g,i}^{a\otimes b} \oplus \delta_g^{O,K} \oplus \delta_{g,j}^K$ (take $\delta_{g,0}^K = 0$) and use $\mathsf{KC}_{g,j}$ to test $\tilde{K}_{g,i,j}$.
  - Let $\mathcal{O} := \left\{\tilde{O}_{g,i}^{a\otimes b} | \tilde{K}_{g,i,j}\text{'s pass at least } \lceil \frac{\ell_\mathsf{k}}{2} \rceil \text{ KC gates}\right\}$
  - If $\mathcal{O} = \emptyset$ then output $\perp$; otherwise output an arbitrary element in $\mathcal{O}$.

### 5.2 Static Secure LEGO Garbling Scheme

Now we are ready to describe the LEGO garbling scheme ($\mathsf{LegoGCircuit}, \mathsf{LegoGInput}, \mathsf{LegoEval}$, $\mathsf{LegoEvalCorrupt}$) satisfying correctness and static security defined in Section A.2 and robust correctness defined below.

**Circuit garbling.** $\left(\tilde{\mathsf{C}}, s^{in}\right) \leftarrow \mathsf{LegoGCircuit}\,(1^\kappa, \mathsf{C})$.

Let $\mathsf{C}$ be a boolean circuit that consists of $n$ NAND gates, takes as input $x \in \{0,1\}^u$, and outputs $y \in \{0,1\}^v$. The circuit garbling procedure is as follows.

1. *Generating a global difference:* $\Delta \xleftarrow{\$} \{0,1\}^z$, where $z$ is the length of a label.
2. *Generating all the labels:* For each gate $g \in \mathsf{C}$, generate the following labels:

$$\begin{array}{ll} L_{g,i}^0, R_{g,i}^0, O_{g,i}^0 \xleftarrow{\$} \{0,1\}^z & \forall 0 \le i \le \ell_\mathsf{n} \\ L_{g,i}^1 := L_{g,i}^0 \oplus \Delta, R_{g,i}^1 := R_{g,i}^0 \oplus \Delta, O_{g,i}^1 := O_{g,i}^0 \oplus \Delta & \forall 0 \le i \le \ell_\mathsf{n} \\ K_{g,i}^0 \xleftarrow{\$} \{0,1\}^z & \forall 0 \le i \le \ell_\mathsf{k} \\ K_{g,i}^1 := K_{g,i}^0 \oplus \Delta & \forall 0 \le i \le \ell_\mathsf{k} \end{array}$$

   Let $\mathcal{W}_\mathsf{out}$ be the set of output wires of $\mathsf{C}$. For each $w \in \mathcal{W}_\mathsf{out}$, suppose it is the output wire of gate $g$. Let

$$O_g^0 := 0^z; O_g^1 := 1^z$$

   be the labels for $w$.

3. *Generating garbled gate tables:* Use the above labels to generate the garbled gate table of every $\tilde{g}$ as described in Section 5.1 and Figures 5, 6.

4. *Concatenating the garbled gates:* According to $\mathsf{C}$'s topology, solder output wires with input wires as shown in Figure 7. In particular, generate the following solders:

   If $g_1$'s output wire is $g$'s left input (denoted $g_1 \nearrow g$): $\delta^L_{g_1,g} := O^0_{g_1} \oplus L^0_g$;
   If $g_2$'s output wire is $g$'s right input (denoted $g \nwarrow g_2$): $\delta^R_{g_2,g} := O^0_{g_2} \oplus R^0_g$.

5. *Input keys:* Let $\mathcal{W}_{\mathsf{in}}$ be the set of input wires of $\mathsf{C}$. For each $w \in \mathcal{W}_{\mathsf{in}}$, suppose it is an input wire of gate $g$. If $w$ is $g$'s left input wire, then let $(\mathsf{label}^0_w, \mathsf{label}^1_w) := (L^0_g, L^1_g)$; otherwise let $(\mathsf{label}^0_w, \mathsf{label}^1_w) := (R^0_g, R^1_g)$. Let

$$s^{in} := \{(\mathsf{label}^0_w, \mathsf{label}^1_w)\}_{w \in \mathcal{W}_{\mathsf{in}}}$$

   be the input keys.

6. *Garbled circuit:* $\tilde{\mathsf{C}}$ consists of the following:

| | |
|---|---|
| All garbled gate tables: | $\{\text{garbled gate table of } \tilde{g}\}_{g \in \mathsf{C}}$ |
| Gate solders: | $\left\{\delta^L_{g_1,g_2}\right\}_{g_1,g_2 \in \mathsf{C}, g_1 \nearrow g_2}, \left\{\delta^R_{g_1,g_2}\right\}_{g_1,g_2 \in \mathsf{C}, g_2 \nwarrow g_1}$ |

7. *Output* $\tilde{\mathsf{C}}, s^{in}$.

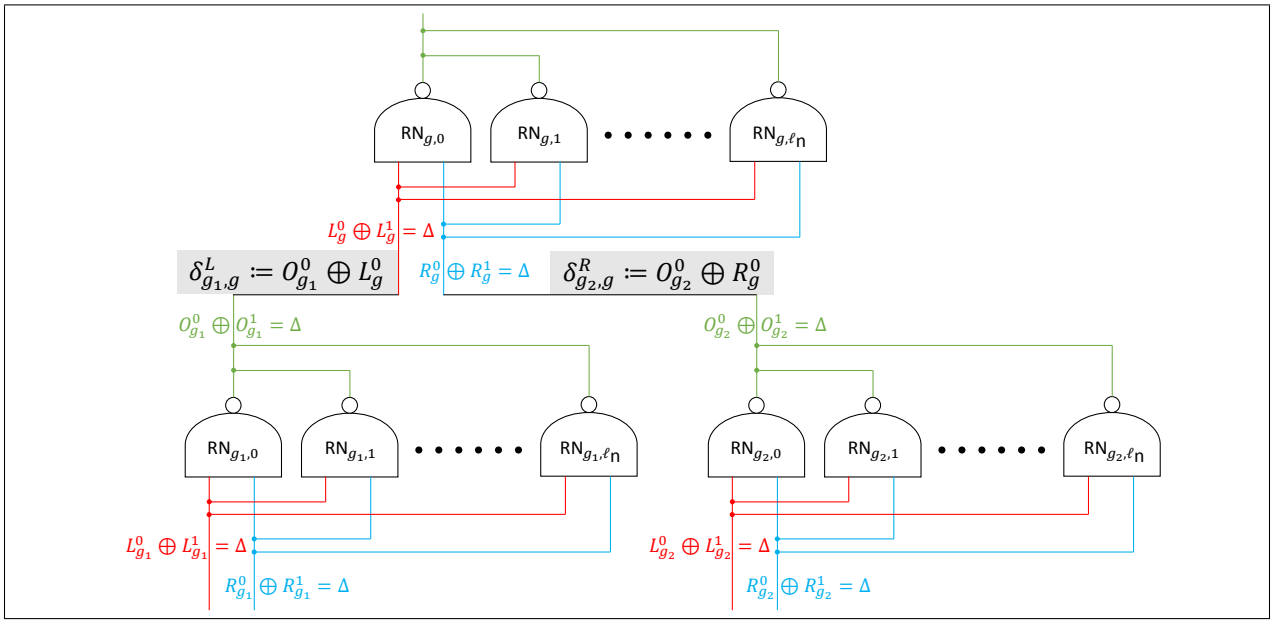


Figure 7: Soldering garbled gates

**Input garbling.** $\tilde{x} \leftarrow \mathsf{LegoGInput}(s^{in}, x)$.

1. Parse $s^{in}$ as $\left\{(\mathsf{label}^0_w, \mathsf{label}^1_w)\right\}_{w \in \mathcal{W}_{\mathsf{in}}}$.
2. For each $w \in \mathcal{W}_{\mathsf{in}}$, let $x[w]$ denote the input value of wire $w$.
3. Output $\tilde{x} = \left\{\mathsf{label}^{x[w]}_w\right\}_{w \in \mathcal{W}_{\mathsf{in}}}$.

**Evaluating a (correct) garbled circuit.** $y \leftarrow \mathsf{LegoEval}\left(\tilde{\mathsf{C}}, \tilde{x}\right)$.

Generally speaking, the evaluation is done gate by gate following the topology of the circuit. During the evaluation one label per wire is obtained, starting from input wires and ending up with output wires.

1. Initially every input wire has a label.
2. Evaluate the garbled circuit gate by gate until one label per output wire is obtained.
   - Once obtaining two labels $L_g^a, R_g^b$ ($a, b \in \{0,1\}$ are hidden values) of a garbled gate $\tilde{g}$, use the garbled gate table to evaluate $\tilde{g}$ as described in Section 5.1 (evaluating a *correct* garbled gate) and obtain $O_g^{a \otimes b}$.
   - If the output wire of $g$ is an input wire of another gate $g'$, use the gate solders to transform $O_g^{a \otimes b}$ into the label of that input wire of $g'$, and repeat.
3. Turn the output labels ($0^z$ or $1^z$) into an output $y$.

**Evaluating a corrupted garbled circuit.** $y = \mathsf{LegoEvalCorrupt}\left(\tilde{\mathsf{C}}^{\mathsf{cor}}, \tilde{x}\right)$.

The garbled circuit can be corrupted in arbitrary ways, but again, we only consider (and later our protocol guarantees) the case where only the garbled gate tables of RN and KC gates could go wrong.

The evaluation goes the same way as $\mathsf{LegoEval}$ except replacing the evaluation of a *correct* garbled gate by evaluation of a *corrupted* garbled gate as describe in Section 5.1. If in any step the evaluation of a garbled gate outputs $\perp$, then output $\perp$. Otherwise continue the process until obtaining one label per output wire, and turn output labels into an output $y$.

**Correctness and security.**

We observe that the following theorem is implicitly proved in [NO09]. Intuitively, the robust correctness property states that even if there are some, though not too many, incorrect garbled gates in the garbled circuit, correctness still holds.

**Theorem 2** (Robust correctness of LEGO garbling scheme [NO09])**.** *For any circuit* $\mathsf{C}$ *and input* $x$*, let* $\tilde{\mathsf{C}}^{\mathsf{cor}}$ *be a corrupted LEGO garbled circuit and let* $\tilde{x}$ *be the garbled input of* $x$*. If each garbled gate* $\tilde{g}^{\mathsf{cor}} \in \tilde{\mathsf{C}}^{\mathsf{cor}}$ *consists of at least* 1 *correct garbled RN gate and at least* $\lceil \frac{\ell_{\mathsf{k}}}{2} \rceil$ *correct garbled KC gates, then*

$$\mathsf{C}(x) = \mathsf{LegoEvalCorrupt}\left(\tilde{\mathsf{C}}^{\mathsf{cor}}, \tilde{x}\right).$$

For static security of the LEGO garbling scheme, we provide a stronger security guarantee where the simulator can generate a garbled circuit with specified solders. We state the theorem below and defer the security proof to Appendix C.1.

**Theorem 3** (Static security of LEGO garbling scheme)**.** *There exits a PPT simulator* $\mathsf{StatLegoSim}$ *such that for any* $\mathsf{C}, x$*, we have that:*

$$\left(\tilde{\mathsf{C}}, \tilde{x}\right) \overset{c}{\approx} \mathsf{StatLegoSim}\left(1^\kappa, \mathsf{C}, \mathsf{C}(x)\right)\big|_{\mathcal{S}}$$

*where* $\left(\tilde{\mathsf{C}}, s^{in}\right) \leftarrow \mathsf{LegoGCircuit}\left(1^\kappa, \mathsf{C}\right), \tilde{x} \leftarrow \mathsf{LegoGInput}(s^{in}, x)$*, and* $\mathsf{StatLegoSim}\left(1^\kappa, \mathsf{C}, \mathsf{C}(x)\right)\big|_{\mathcal{S}}$ *denotes the output of* $\mathsf{StatLegoSim}$ *when the solders of the garbled circuit is restricted to*

$$\mathcal{S} := \left\{ \left\{ \left\{\delta_{g,i}^L, \delta_{g,i}^R, \delta_{g,i}^O\right\}_{i=1}^{\ell_{\mathsf{n}}}, \left\{\delta_{g,i}^K\right\}_{i=1}^{\ell_{\mathsf{k}}}, \delta_g^{O,K} \right\}_{g \in \mathsf{C}}, \left\{\delta_{g_1,g_2}^L\right\}_{g_1,g_2 \in \mathsf{C}, g_1 \nearrow g_2}, \left\{\delta_{g_1,g_2}^R\right\}_{g_1,g_2 \in \mathsf{C}, g_2 \nwarrow g_1} \right\}$$

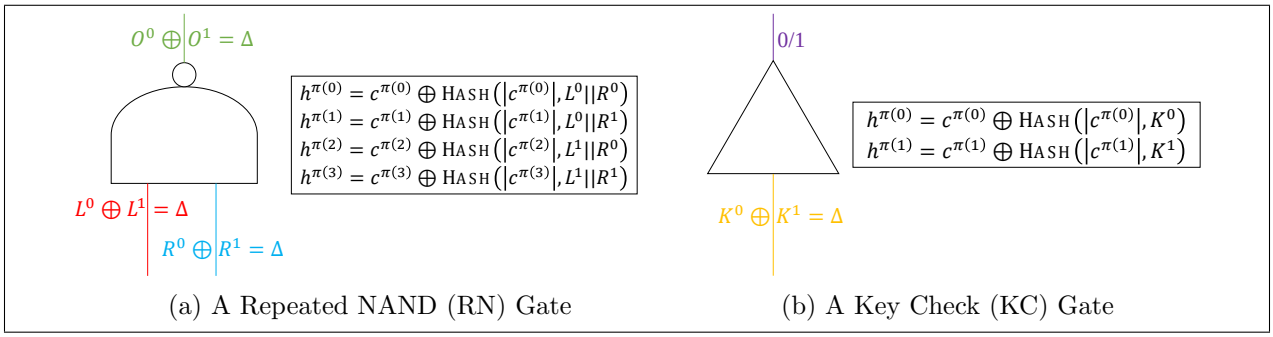
*which is sampled uniformly at random.*

### 5.3 Adaptive Secure LEGO Garbling Scheme

In this section we modify the static secure LEGO garbling scheme to be adaptive secure in the random oracle model. First of all, to extend our definitions of garbling scheme privacy to adaptive security in the random oracle model [BR93], we follow the treatment of [BHR12b, BHR12a]. An ROM garbling scheme is a garbling scheme whose algorithms have access to an oracle HASH called the random oracle. The model is obtained by adding the following procedure HASH to the real/simulated experiments defined in Appendix A.2.

**procedure** HASH$(\gamma, w)$
  **if** $\mathsf{hash}[\gamma, w] = \perp$ **then**
    **if** in the real experiment **then**
      $\mathsf{hash}[\gamma, w] \xleftarrow{\$} \{0,1\}^\gamma$
    **else**
      $\mathsf{hash}[\gamma, w] \leftarrow \mathsf{AdaptCircSim}(\mathsf{RO}, \gamma, w)$
  **return** $\mathsf{hash}[\gamma, w]$

**New components.**




(a) A Repeated NAND (RN) Gate (b) A Key Check (KC) Gate

Figure 8: Garbling components for adaptive LEGO garbling scheme

First we modify the garbled RN and KC gates by adding random oracles, as shown in Figure 8. As before, given the garbled gate table of an RN gate and two input labels $L^a, R^b$ $(a, b \in \{0,1\})$, one can obtain the corresponding output label $O^{a \otimes b}$. Give the garbled table of a KC gate and one label $K$, one can check if $K$ is valid (i.e., $K \in \{K^0, K^1\}$).

**New scheme.**

Now we modify the static secure LEGO garbling scheme to be an adaptive secure one ($\mathsf{AdaptLegoGCircuit}$, $\mathsf{AdaptLegoGInput}$, $\mathsf{AdaptLegoEval}$, $\mathsf{AdaptLegoEvalCorrupt}$) with the following syntax:

$$\left(\tilde{\mathsf{C}}_\mathsf{a}, s_\mathsf{a}^{in}\right) \leftarrow \mathsf{AdaptLegoGCircuit}\,(1^\kappa, \mathsf{C})$$
$$\tilde{x}_\mathsf{a} \leftarrow \mathsf{AdaptLegoGInput}(s_\mathsf{a}^{in}, x)$$
$$y \leftarrow \mathsf{AdaptLegoEval}(\tilde{\mathsf{C}}_\mathsf{a}, \tilde{x}_\mathsf{a})$$

There are two modifications compared to ($\mathsf{LegoGCircuit}$, $\mathsf{LegoGInput}$, $\mathsf{LegoEval}$, $\mathsf{LegoEvalCorrupt}$):

1. When generating garbled gate tables in $\mathsf{AdaptLegoGCircuit}$, now generate the new garbled RN and KC gate tables as in Figure 8.
2. When evaluating garbled gates in $\mathsf{AdaptLegoEval}$ and $\mathsf{AdaptLegoEvalCorrupt}$, now evaluate the new garbled RN and KC gates.

**Correctness and security.**

Correct garbled gates remain correct for evaluation in the adaptive LEGO garbling scheme, hence the robust correctness still holds.

**Theorem 4** (Robust correctness of adaptive LEGO garbling scheme)**.** *For any circuit* $\mathsf{C}$ *and input* $x$*, let* $\tilde{\mathsf{C}}_{\mathsf{a}}^{\mathsf{cor}}$ *be a corrupted LEGO garbled circuit in the adaptive LEGO garbling scheme and let* $\tilde{x}_{\mathsf{a}}$ *be the garbled input of* $x$*. If each garbled gate* $\tilde{g}_{\mathsf{a}}^{\mathsf{cor}} \in \tilde{\mathsf{C}}_{\mathsf{a}}^{\mathsf{cor}}$ *consists of at least* 1 *correct garbled RN gate and at least* $\lceil \frac{\ell_{\mathsf{k}}}{2} \rceil$ *correct garbled KC gates, then*

$$\mathsf{C}(x) = \mathsf{AdaptLegoEvalCorrupt}\left(\tilde{\mathsf{C}}_{\mathsf{a}}^{\mathsf{cor}}, \tilde{x}_{\mathsf{a}}\right).$$

We prove the adaptive security for the new scheme. At a high level, the simulator $\mathsf{AdaptLegoSim}$ will first generate the garbled circuit honestly when seeing the circuit. After seeing the output of the circuit, $\mathsf{AdaptLegoSim}$ generates a new simulated garbled circuit by $\mathsf{StatLegoSim}$, and uses the random oracles to transform the previously generated garbled circuit into the newly simulated one. See Appendix C.2 for the detailed proof.

**Theorem 5** (Adaptive security of adaptive LEGO garbling scheme)**.** *There exists a PPT simulator* $\mathsf{AdaptLegoSim}$ *such that for any PPT adversary* $\mathcal{A}$ *and any circuit* $\mathsf{C}$*, the output of the following two experiments are computational indistinguishable:*

*Real experiment*

- $\left(\tilde{\mathsf{C}}_{\mathsf{a}}, s_{\mathsf{a}}^{in}\right) \leftarrow \mathsf{AdaptLegoGCircuit}\,(1^{\kappa}, \mathsf{C})$
- $x \leftarrow \mathcal{A}(1^{\kappa}, \mathsf{C}, \tilde{\mathsf{C}}_{\mathsf{a}})$
- $\tilde{x}_{\mathsf{a}} \leftarrow \mathsf{AdaptLegoGInput}(s_{\mathsf{a}}^{in}, x)$
- *Output* $(\tilde{\mathsf{C}}_{\mathsf{a}}, \tilde{x}_{\mathsf{a}})$

*Simulated experiment*

- $\left(\tilde{\mathsf{C}}_{\mathsf{a}}^{\mathsf{sim}}, \mathsf{state}\right) \leftarrow \mathsf{AdaptLegoSim}\,(1^{\kappa}, \mathsf{C})$
- $x \leftarrow \mathcal{A}(1^{\kappa}, \mathsf{C}, \tilde{\mathsf{C}}_{\mathsf{a}})$
- $\tilde{x}_{\mathsf{a}}^{\mathsf{sim}} \leftarrow \mathsf{AdaptLegoSim}(\mathsf{state}, \mathsf{C}(x))$
- *Output* $(\tilde{\mathsf{C}}_{\mathsf{a}}^{\mathsf{sim}}, \tilde{x}_{\mathsf{a}}^{\mathsf{sim}})$

### 5.4 Adaptive Secure Garbled RAM

The garbled RAM scheme was instantiated with Yao's garbling scheme in [GLO15]. In this section we will instantiate it with the aforementioned static/adaptive secure LEGO garbling schemes. Out key observation is that [GLO15] makes a black-box use of a secure circuit garbling scheme $(\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{Eval})$, which can be instantiated using the LEGO schemes.

First we instantiate the garbled RAM with the static LEGO garbling scheme $(\mathsf{LegoGCircuit}, \mathsf{LegoGInput}, \mathsf{LegoEval})$ given in Section 5.2. Note that the static LEGO garbling scheme has the same syntax as the circuit garbling scheme. As mentioned in Theorem 3, it is a static secure circuit garbling scheme. Moreover, the scheme only makes black-box use of one-way functions. Therefore, instantiating the garbled RAM with the static LEGO garbling scheme would give us a static secure garbled RAM scheme.

Our next step is instantiating the garbled RAM with the adaptive LEGO garbling scheme $(\mathsf{AdaptLegoGCircuit}, \mathsf{AdaptLegoGInput}, \mathsf{AdaptLegoEval})$ in Section 5.3. It is an adaptive secure circuit

garbling scheme as proved in Theorem 5, and it also makes black-box use of one-way functions. We observe that in the security proof of [GLO15] if we replace the static secure circuit garbling scheme with an adaptive secure one, the resulting garbled RAM is also adaptive secure. The following theorem summarizes the security of the resulting garbled RAM.

**Theorem 6** (Adaptive secure garbled RAM)**.** *Instantiating the garbled RAM of [GLO15] with the adaptive LEGO garbling scheme gives a garbled RAM scheme* $(\mathsf{GData}_{\mathsf{Lego}}, \mathsf{GProg}_{\mathsf{Lego}}, \mathsf{GInput}_{\mathsf{Lego}}, \mathsf{GEval}_{\mathsf{Lego}})$ *with a simulator* $\mathsf{AdaptRamSim}_{\mathsf{Lego}}$ *that achieves efficiency, correctness, and adaptive security as defined in Section 3.1.*

# 6 Generating an Adaptive LEGO Garbled Circuit

In this section, we provide a protocol between two parties to generate an adaptive LEGO garbled circuit. The protocol is based on the original LEGO protocol in [NO09], and it differs from the LEGO protocol in that we add more commitments and consistency checking in the protocol. After the additional consistency checking, as we will see in Theorem 8, all the commitments of input wires are guaranteed to be correct. Looking ahead, this property is crucial for our construction of secure RAM computation protocol to ensure that the garbler generates the garbled memory and garbled programs concatenated properly. In the following we first present the protocol in Section 6.1, and then give some useful analysis in Section 6.2.

## 6.1 The New LEGO Protocol

In this section, we present the protocol between Alice and Bob where Alice plays the role of a garbler, and Bob obtains an adaptive LEGO garbled circuit at the end of the protocol. For notations, we use $[\cdot]$ to denote a commitment, and $\overline{b}$ to denote a negation of bit $b$.

**Global difference.** Alice samples a global difference $\Delta$ and a randomizer $r_\Delta$, generates the commitment $[\Delta; r_\Delta]$, sends $[\Delta]$ to Bob, and gives a zero-knowledge UC-secure proof of knowledge of $\Delta$ (see Appendix B for details).

**Component production.** Let $\mathsf{C}$ be a circuit with $n$ NAND gates. Let $N_{\mathsf{n}} = (\ell_{\mathsf{n}} + 1)n, N_{\mathsf{k}} = (\ell_{\mathsf{k}} + 1)n$.

*Generating garbled RN gates.* Alice generates $\phi_{\mathsf{n}} N_{\mathsf{n}}$ garbled RN gates (as described in Section 5.3). For each garbled RN gate, Alice sample $\pi_{\mathsf{L}}, \pi_{\mathsf{R}}, \pi_{\mathsf{O}} \xleftarrow{\$} \{0,1\}$ and sends the following to Bob. Note that the commitments in steps 3 and 4 are additional compared to the original LEGO protocol.

1. Commitment of the zero labels and permutation:

$$\left[L^0\right], \left[R^0\right], \left[O^0\right], [\pi].$$

   (Note that Bob can compute the one labels by himself: $\left[L^1\right] \leftarrow \left[L^0\right] \oplus [\Delta]$, $\left[R^1\right] \leftarrow \left[R^0\right] \oplus [\Delta]$, $\left[O^1\right] \leftarrow \left[O^0\right] \oplus [\Delta]$.)

2. The garbled gate table.

$$h^{\pi(0)} = \mathsf{Enc}_{L^0,R^0}(O^1) \oplus \textsc{Hash}\left(\left|\mathsf{Enc}_{L^0,R^0}(O^1)\right|, L^0||R^0\right);$$
$$h^{\pi(1)} = \mathsf{Enc}_{L^0,R^1}(O^1) \oplus \textsc{Hash}\left(\left|\mathsf{Enc}_{L^0,R^1}(O^1)\right|, L^0||R^1\right);$$

$$h^{\pi(2)} = \mathsf{Enc}_{L^1,R^0}(O^1) \oplus \textsc{Hash}\left(\left|\mathsf{Enc}_{L^1,R^0}(O^1)\right|, L^1||R^0\right);$$
$$h^{\pi(3)} = \mathsf{Enc}_{L^1,R^1}(O^0) \oplus \textsc{Hash}\left(\left|\mathsf{Enc}_{L^1,R^1}(O^0)\right|, L^1||R^1\right).$$

3. Commitment of $(L^0, L^1)$, $(R^0, R^1)$, and $(O^0, O^1)$ in permuted orders:

$$[L^{\pi_\mathsf{L}}] \,||\, [\pi_\mathsf{L}], \left[L^{\overline{\pi_\mathsf{L}}}\right] \,||\, [\overline{\pi_\mathsf{L}}];$$
$$[R^{\pi_\mathsf{R}}] \,||\, [\pi_\mathsf{R}], \left[R^{\overline{\pi_\mathsf{R}}}\right] \,||\, [\overline{\pi_\mathsf{R}}];$$
$$[O^{\pi_\mathsf{O}}] \,||\, [\pi_\mathsf{O}], \left[O^{\overline{\pi_\mathsf{O}}}\right] \,||\, [\overline{\pi_\mathsf{O}}].$$

4. Write the labels bit by bit as follows (where $z$ is the length of a label):

$$L^0||L^1 = \ell^{0,1}\ell^{0,2}\cdots\ell^{0,z}||\ell^{1,1}\ell^{1,2}\cdots\ell^{1,z};$$
$$R^0||R^1 = r^{0,1}r^{0,2}\cdots r^{0,z}||r^{1,1}r^{1,2}\cdots r^{1,z};$$
$$O^0||O^1 = o^{0,1}o^{0,2}\cdots o^{0,z}||o^{1,1}o^{1,2}\cdots o^{1,z}.$$

   Bit-by-bit commitment of all labels:

$$\left\{\left[\ell^{b,u}\right], \left[r^{b,u}\right], \left[o^{b,u}\right]\right\}_{b\in\{0,1\},1\le u\le z}.$$

*Generating garbled KC gates.* Alice generates $\phi_\mathsf{k} N_\mathsf{k}$ KC gates (as described in Section 5.3). For each garbled KC gate, Alice sends the following to Bob:

1. Commitment of the zero key and permutation

$$\left[K^0\right], [\pi].$$

   (Note that Bob can compute the one key by himself: $\left[K^1\right] \leftarrow \left[K^0\right] \oplus [\Delta]$.)

2. The garbled gate table.

$$h^{\pi(0)} = \mathsf{H}(K^0) \oplus \textsc{Hash}\left(\left|\mathsf{H}(K^0)\right|, K^0\right);$$
$$h^{\pi(1)} = \mathsf{H}(K^1) \oplus \textsc{Hash}\left(\left|\mathsf{H}(K^1)\right|, K^1\right).$$

**Component checking.** Bob randomly picks $(\phi_\mathsf{n} - 1)N_\mathsf{n}$ RN gates and $(\phi_\mathsf{k} - 1)N_\mathsf{k}$ KC gates to be checked, and sends to Alice. Note that Alice cannot simply send to Bob all the randomness used to generate the garbled gates being checked, because revealing both the zero label and one label of a wire will leak $\Delta$ to Bob, which compromises the security of LEGO garbled circuits completely. Hence Bob randomly picks one label per wire to check.

*Checking RN gates.* For each RN gate to be checked, Bob randomly picks $b_\mathsf{L}, b_\mathsf{R} \xleftarrow{\$} \{0,1\}$, computes $b_\mathsf{O} = b_\mathsf{L} \otimes b_\mathsf{R}$, and sends $b_\mathsf{L}, b_\mathsf{R}, b_\mathsf{O}$ to Alice.

1. *Checking the commitment of labels and permutation:* Alice opens the commitment

$$\left[L^{b_\mathsf{L}}\right], \left[R^{b_\mathsf{R}}\right], \left[O^{b_\mathsf{O}}\right], [\pi].$$

2. *Checking the garbled gate table:* Bob computes $b = 2 \cdot b_{\mathsf{L}} + b_{\mathsf{R}}$, and checks

$$h^{\pi(b)} = \mathsf{Enc}_{L^{b_{\mathsf{L}}}, R^{b_{\mathsf{R}}}}(O^{b_{\mathsf{O}}}) \oplus \textsc{Hash}\left(\left|\mathsf{Enc}_{L^{b_{\mathsf{L}}}, R^{b_{\mathsf{R}}}}(O^{b_{\mathsf{O}}})\right|, L^{b_{\mathsf{L}}} || R^{b_{\mathsf{R}}}\right).$$

3. *Checking the commitment of* $(L^0, L^1)$, $(R^0, R^1)$, *and* $(O^0, O^1)$ *in permuted orders:* Alice opens the commitment

$$\begin{array}{lll} [\pi_{\mathsf{L}}], & \left[L^{b_{\mathsf{L}}}\right] || [b_{\mathsf{L}}]; & (\text{If } \pi_{\mathsf{L}} = b_{\mathsf{L}}, \text{ then open } [L^{\pi_{\mathsf{L}}}] || [\pi_{\mathsf{L}}]; \text{ otherwise } \left[L^{\overline{\pi_{\mathsf{L}}}}\right] || [\overline{\pi_{\mathsf{L}}}]) \\ [\pi_{\mathsf{R}}], & \left[R^{b_{\mathsf{R}}}\right] || [b_{\mathsf{R}}]; & (\text{If } \pi_{\mathsf{R}} = b_{\mathsf{R}}, \text{ then open } [L^{\pi_{\mathsf{R}}}] || [\pi_{\mathsf{R}}]; \text{ otherwise } \left[L^{\overline{\pi_{\mathsf{R}}}}\right] || [\overline{\pi_{\mathsf{R}}}]) \\ [\pi_{\mathsf{O}}], & \left[O^{b_{\mathsf{O}}}\right] || [b_{\mathsf{O}}]. & (\text{If } \pi_{\mathsf{O}} = b_{\mathsf{O}}, \text{ then open } [L^{\pi_{\mathsf{O}}}] || [\pi_{\mathsf{O}}]; \text{ otherwise } \left[L^{\overline{\pi_{\mathsf{O}}}}\right] || [\overline{\pi_{\mathsf{O}}}]) \end{array}$$

Bob checks consistency of

$$\begin{array}{l} \left[L^{b_{\mathsf{L}}}\right] || [b_{\mathsf{L}}] \text{ with previously revealed } L^{b_{\mathsf{L}}}; \\ \left[R^{b_{\mathsf{R}}}\right] || [b_{\mathsf{R}}] \text{ with previously revealed } R^{b_{\mathsf{R}}}; \\ \left[O^{b_{\mathsf{O}}}\right] || [b_{\mathsf{O}}] \text{ with previously revealed } O^{b_{\mathsf{O}}}. \end{array}$$

4. *Checking the commitment of every bit of labels:* Alice opens the following commitment

$$\left\{\left[\ell^{b_{\mathsf{L}},u}\right], \left[r^{b_{\mathsf{R}},u}\right], \left[o^{b_{\mathsf{O}},u}\right]\right\}_{u=1}^{z}.$$

Bob checks the consistency with previously revealed $L^{b_{\mathsf{L}}}, R^{b_{\mathsf{R}}}, O^{b_{\mathsf{O}}}$.

*Checking KC gates.* For each KC gate to be checked, Bob randomly picks $b_{\mathsf{K}} \xleftarrow{\$} \{0,1\}$, sends to Alice.

1. *Checking the commitment of labels and permutation:* Alice opens the commitment

$$\left[K^{b_{\mathsf{K}}}\right], [\pi].$$

2. *Checking the garbled gate table:* Bob checks

$$h^{\pi(b_{\mathsf{K}})} = \mathsf{H}(K^{b_{\mathsf{K}}}) \oplus \textsc{Hash}\left(|\mathsf{H}(K^{b_{\mathsf{K}}})|, K^{b_{\mathsf{K}}}\right).$$

**Soldering.** Bob randomly permutes the remaining $N_{\mathsf{n}}$ RN gates and $N_{\mathsf{k}}$ KC gates, constructs the garbled circuit such that each garbled gate consists of $(\ell_{\mathsf{n}} + 1)$ RN gates and $(\ell_{\mathsf{k}} + 1)$ KC gates. Alice helps Bob solder the gates.

1. *Soldering RN gates and KC gates of one garbled gate:* For each gate $g \in \mathsf{C}$, recall that the solders consist of the following:

$$\begin{array}{l} \text{Soldering of RN gates: } \delta^{L}_{g,i}, \delta^{R}_{g,i}, \delta^{O}_{g,i}, \forall i \in \{1, \cdots, \ell_{\mathsf{n}}\}; \\ \text{Soldering of KC gates: } \delta^{K}_{g,i}, \forall i \in \{1, \cdots, \ell_{\mathsf{k}}\}; \\ \text{Soldering of KC gates with RN gates: } \delta^{O,K}_{g}. \end{array}$$

Bob can compute by himself the commitment of all the above solders:

$$\begin{aligned}
&\left[\delta^L_{g,i}\right] \leftarrow \left[L^0_{g,i}\right] \oplus \left[R^0_{g,0}\right];\\
&\left[\delta^R_{g,i}\right] \leftarrow \left[R^0_{g,i}\right] \oplus \left[R^0_{g,0}\right];\\
&\left[\delta^O_{g,i}\right] \leftarrow \left[O^0_{g,i}\right] \oplus \left[O^0_{g,0}\right];\\
&\left[\delta^K_{g,i}\right] \leftarrow \left[K^0_{g,i}\right] \oplus \left[K^0_{g,0}\right];\\
&\left[\delta^{O,K}_{g}\right] \leftarrow \left[O^0_{g,0}\right] \oplus \left[K^0_{g,0}\right].
\end{aligned}$$

Alice opens all the commitment.

2. *Soldering garbled gates together:* Recall that the gate solders consist of the following:

$$\left\{\delta^L_{g_1,g_2}\right\}_{g_1,g_2\in\mathsf{C},g_1\nearrow g_2}, \left\{\delta^R_{g_1,g_2}\right\}_{g_1,g_2\in\mathsf{C},g_2\nwarrow g_1}.$$

Bob can compute by himself the commitment of these solders:

$$\begin{aligned}
&\forall g_1 \nearrow g_2 : \left[\delta^L_{g_1,g_2}\right] \leftarrow \left[O^0_{g_1,0}\right] \oplus \left[L^0_{g_2,0}\right];\\
&\forall g_2 \nwarrow g_1 : \left[\delta^R_{g_1,g_2}\right] \leftarrow \left[O^0_{g_1,0}\right] \oplus \left[R^0_{g_2,0}\right].
\end{aligned}$$

Alice opens the commitment.

**Output table.** For each output wire $w \in \mathcal{W}_{\mathsf{out}}$, suppose it is the output wire of gate $g$. Bob has the following commitment:

$$\begin{aligned}
&\left[L^{\pi_{\mathsf{L},0}}_{g,0}\right] || \left[\pi_{\mathsf{L},0}\right], \left[L^{\overline{\pi_{\mathsf{L},0}}}_{g,0}\right] || \left[\overline{\pi_{\mathsf{L},0}}\right];\\
&\left[L^{\pi_{\mathsf{L},1}}_{g,1}\right] || \left[\pi_{\mathsf{L},1}\right], \left[L^{\overline{\pi_{\mathsf{L},1}}}_{g,1}\right] || \left[\overline{\pi_{\mathsf{L},1}}\right];\\
&\vdots\\
&\left[L^{\pi_{\mathsf{L},\ell_\mathsf{n}}}_{g,\ell_\mathsf{n}}\right] || \left[\pi_{\mathsf{L},\ell_\mathsf{n}}\right], \left[L^{\overline{\pi_{\mathsf{L},\ell_\mathsf{n}}}}_{g,\ell_\mathsf{n}}\right] || \left[\overline{\pi_{\mathsf{L},\ell_\mathsf{n}}}\right].
\end{aligned}$$

**Input consistency checking.** For each input wire $w \in \mathcal{W}_{\mathsf{in}}$, without loss of generality assume it is the left input wire of gate $g$. Bob has the following commitments, as shown in Figure 9.

$$\begin{aligned}
&\left[L^{\pi_{\mathsf{L},0}}_{g,0}\right] || \left[\pi_{\mathsf{L},0}\right]; \left[L^{\overline{\pi_{\mathsf{L},0}}}_{g,0}\right] || \left[\overline{\pi_{\mathsf{L},0}}\right];\\
&\left[L^{\pi_{\mathsf{L},1}}_{g,1}\right] || \left[\pi_{\mathsf{L},1}\right]; \left[L^{\overline{\pi_{\mathsf{L},1}}}_{g,1}\right] || \left[\overline{\pi_{\mathsf{L},1}}\right];\\
&\vdots\\
&\left[L^{\pi_{\mathsf{L},\ell_\mathsf{n}}}_{g,\ell_\mathsf{n}}\right] || \left[\pi_{\mathsf{L},\ell_\mathsf{n}}\right]; \left[L^{\overline{\pi_{\mathsf{L},\ell_\mathsf{n}}}}_{g,\ell_\mathsf{n}}\right] || \left[\overline{\pi_{\mathsf{L},\ell_\mathsf{n}}}\right].\\
&\left[\ell^{0,1}_{g,0}\right], \left[\ell^{0,2}_{g,0}\right], \cdots, \left[\ell^{0,z}_{g,0}\right]; \left[\ell^{1,1}_{g,0}\right], \left[\ell^{1,2}_{g,0}\right], \cdots, \left[\ell^{1,z}_{g,0}\right];\\
&\left[\ell^{0,1}_{g,1}\right], \left[\ell^{0,2}_{g,1}\right], \cdots, \left[\ell^{0,z}_{g,1}\right]; \left[\ell^{1,1}_{g,1}\right], \left[\ell^{1,2}_{g,1}\right], \cdots, \left[\ell^{1,z}_{g,1}\right];\\
&\vdots
\end{aligned}$$

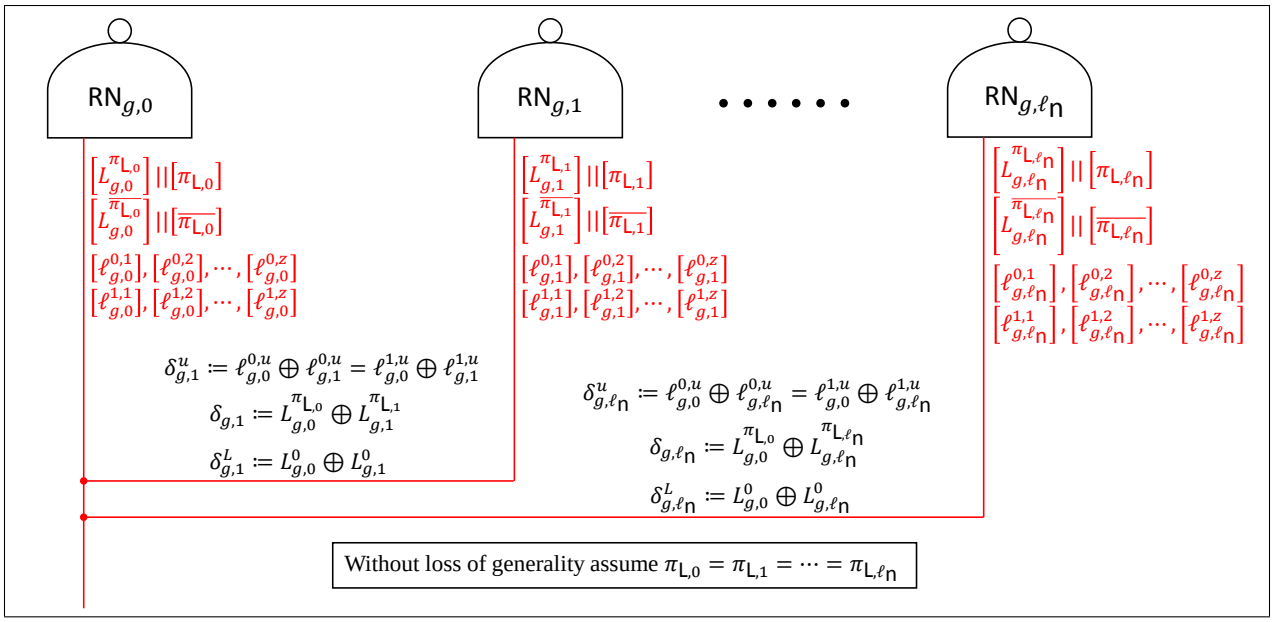



Figure 9: Consistency checking of an input wire

$$\left[\ell_{g,\ell_{\mathsf{n}}}^{0,1}\right], \left[\ell_{g,\ell_{\mathsf{n}}}^{0,2}\right], \cdots, \left[\ell_{g,\ell_{\mathsf{n}}}^{0,z}\right]; \left[\ell_{g,\ell_{\mathsf{n}}}^{1,1}\right], \left[\ell_{g,\ell_{\mathsf{n}}}^{1,2}\right], \cdots, \left[\ell_{g,\ell_{\mathsf{n}}}^{1,z}\right].$$

Alice and Bob execute the following consistency checking. This step is a crucial change compared to the original LEGO protocol. Looking ahead, in Theorem 8 we will see that if Bob passes all the input consistency checking, then all the commitments of input wires are correct with high probability.

1. *Revealing the relation between* $\{\pi_{\mathsf{L},i}\}_{i=0}^{\ell_{\mathsf{n}}}$*:* For every $1 \le i \le \ell_{\mathsf{n}}$, Bob can compute the following

   $$[\pi_{\mathsf{L},0} \oplus \pi_{\mathsf{L},i}] \leftarrow [\pi_{\mathsf{L},0}] \oplus [\pi_{\mathsf{L},i}]$$

   Alice opens the commitment. Bob checks if $\pi_{\mathsf{L},0} \oplus \pi_{\mathsf{L},i} = 0$ or 1, and obtains the relation between $\{\pi_{\mathsf{L},i}\}_{i=0}^{\ell_{\mathsf{n}}}$, namely $\pi_{\mathsf{L},i} = \pi_{\mathsf{L},0}$ or $\overline{\pi_{\mathsf{L},0}}$. Without loss of generality assume that

   $$\pi_{\mathsf{L},0} = \pi_{\mathsf{L},1} = \cdots = \pi_{\mathsf{L},\ell_{\mathsf{n}}}.$$

2. *Checking the difference between labels is consistent with previously revealed* $\left\{\delta_{g,i}^{L}\right\}_{i=1}^{\ell_{\mathsf{n}}}$*:* For every $1 \le i \le \ell_{\mathsf{n}}$, Bob can compute

   $$[\delta_{g,i}] \leftarrow \left[L_{g,0}^{\pi_{\mathsf{L},0}}\right] \oplus \left[L_{g,i}^{\pi_{\mathsf{L},i}}\right].$$

   Alice opens the commitment, and Bob checks if $\delta_{g,i} = \delta_{g,i}^{L}$.

3. *Checking each pair of labels has a difference* $\Delta$ *in between:* For every $0 \le i \le \ell_{\mathsf{n}}$, Bob can compute

   $$[0] \,||\, [0] \leftarrow \left(\left[L_{g,i}^{\pi_{\mathsf{L},i}}\right] || [\pi_{\mathsf{L},i}]\right) \oplus \left(\left[L_{g,i}^{\overline{\pi_{\mathsf{L},i}}}\right] || [\overline{\pi_{\mathsf{L},i}}]\right) \oplus [\Delta] \,||\, [1].$$

   Alice opens the commitment, and Bob checks.

4. *Checking the difference between bit-by-bit commitment is consistent with previously revealed* $\left\{\delta_{g,i}^{L}\right\}_{i=1}^{\ell_{\mathsf{n}}}$*:* For every $1 \leq i \leq \ell_{\mathsf{n}}, b \in \{0,1\}, 1 \leq u \leq z$, Bob can compute

$$\left[\delta_{g,i}^{u}\right] \leftarrow \left[\ell_{g,0}^{b,u}\right] \oplus \left[\ell_{g,i}^{b,u}\right].$$

Alice opens the commitment, and Bob checks if $\delta_{g,i}^{u}$ is equal to the $u$-th bit of previously revealed $\delta_{g,i}^{L}$.

**Parameters.** We pick the parameters $\ell_{\mathsf{k}}, \ell_{\mathsf{n}}, \phi_{\mathsf{n}}, \phi_{\mathsf{k}}, z$ such that the proofs of Theorems 7, 8 go through in Appendices C.3,C.4. One may refer to [NO09] for a detailed discussion of parameters choice.

### 6.2 Analysis

The main idea in the above cut-and-choose protocol is that if Bob accepts all the checking, then with high probability there are not too many incorrect gates in total. With these small amount of incorrect gates, there will be at least one RN gates and $\lceil \frac{\ell_{\mathsf{k}}}{2} \rceil$ KC gates per gate with probability exponentially close to 1.

**Theorem 7.** *Assume that Alice is corrupted and Bob is honest. In generating an adaptive LEGO garbled circuit* $\tilde{\mathsf{C}}$*, if Bob accepts all the checking with probability greater than* $2^{-s}$*, then with probability greater than* $1 - 2^{-s}$ *there are at least* 1 *correct garbled RN gate and* $\lceil \frac{\ell_{\mathsf{k}}}{2} \rceil$ *correct garbled KC gates per garbled gate in* $\tilde{\mathsf{C}}$*.*

The above theorem states that if Bob accepts all the component checking and consistency checking, then most of the garbled gates in the garbled circuit are correct with high probability. In the following theorem, we further show that all the commitments of input wires are correct with high probability.

**Theorem 8.** *Assume that Alice is corrupted and Bob is honest. In generating an adaptive LEGO garbled circuit* $\tilde{\mathsf{C}}$*, if Bob accepts all the checking with probability greater than* $2^{-s}$*, then with probability greater than* $1 - 2^{-s}$ *all the commitments of all the input wires of* $\tilde{\mathsf{C}}$ *are correct.*

## 7 Our Construction

In this section we give our construction of the secure RAM computation protocol. The high level intuition of the protocol is as follows. It is built on the garbled RAM scheme we abstracted in Section 4 and instantiated in Section 5.4. Recall that in the garbled RAM scheme, we defined uniform circuits in Section 4.1 as building units. In the procedure of database garbling and program garbling, the garbler generates a collection of garbled uniform circuits with partial labels consistent with a public function $f$, which indicates the concatenation of the uniform circuits. The evaluator evaluates the garbled circuits (GCs) one by one in a certain order, during which one GC may output (partial) labels for other GCs, enabling the evaluation of the next GC. This process proceeds until the evaluator obtains the output of the program.

In the protocol Alice will play the role of a garbler and Bob will play the role of an evaluator. They apply the protocol in Section 6.1 to generate the garbled uniform circuits in parallel. After

Bob obtains all the GCs, Alice provides him with partial labels consistent with the public function $f$. We will elaborate in more detail how Alice gives the partial labels in Section 7.1. Given that Alice is providing the correct labels, Bob, as an evaluator, can evaluate the garbled RAM as in the garbled RAM scheme.

## 7.1 Generating Partial Labels

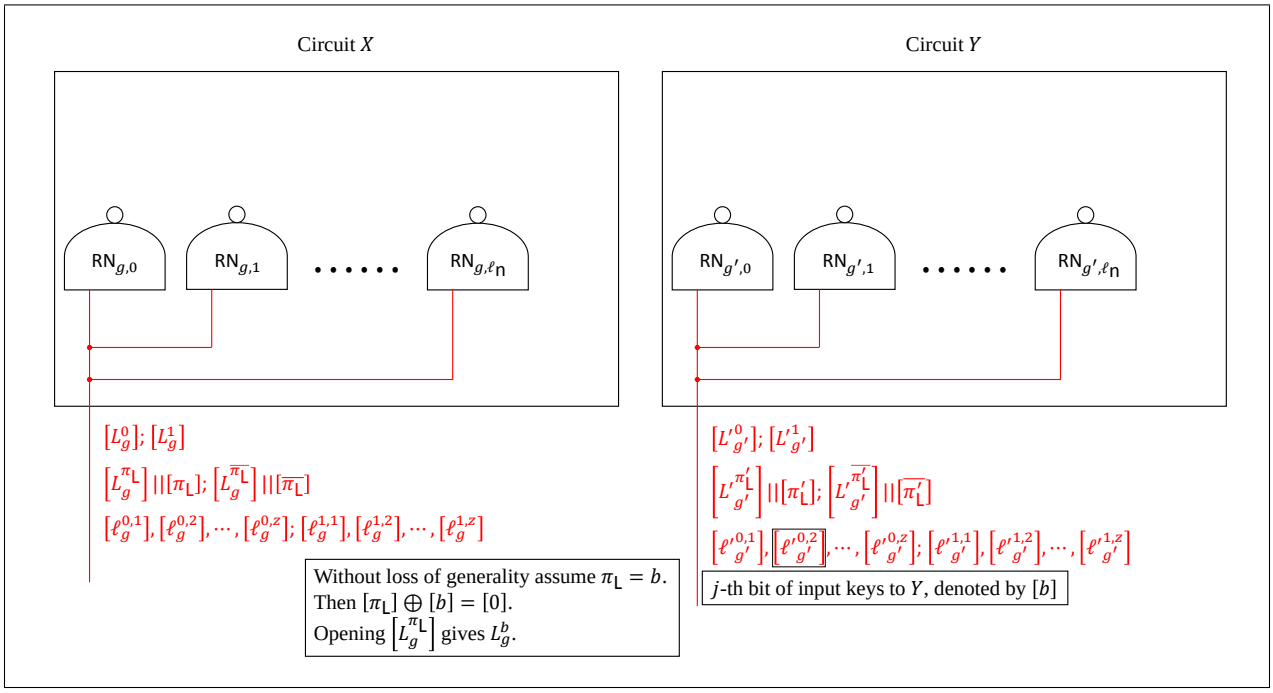



Figure 10: Concatenation of two garbled circuits

Suppose that Alice and Bob runs the protocol in Section 6.1 to generate a collection of garbled uniform circuits in parallel. In this section we describe how Alice provides Bob with partial labels to these garbled circuits consistent with the public function $f$. For each input wire $X[i]$ (the $i$-th input wire of circuit $X$), without loss of generality assume it is the left input wire of gate of $g$. Recall that Bob has the following commitment of its input keys, as shown in Figure 10.

$$\begin{gathered}\left[L_g^0\right]; \left[L_g^1\right]; \\ \left[L_g^{\pi_{\mathsf{L}}}\right] || [\pi_{\mathsf{L}}]; \left[L_g^{\overline{\pi_{\mathsf{L}}}}\right] || [\overline{\pi_{\mathsf{L}}}]; \\ \left[\ell_g^{0,1}\right], \left[\ell_g^{0,2}\right], \cdots, \left[\ell_g^{0,z}\right]; \left[\ell_g^{1,1}\right], \left[\ell_g^{1,2}\right], \cdots, \left[\ell_g^{1,z}\right].\end{gathered}$$

Recall that in Table 2 we defined five categories of inputs: $\mathsf{const}, \mathsf{keys}, \mathsf{mem}, \mathsf{inp}, \mathsf{dyn}$. We explain in the following how Bob obtains the labels for each category of inputs:

- $\mathsf{const}$: Let $b = f(X, i)$ be value of $X[i]$. Alice simply opens the following commitment:

$$\left[\ell_g^{b,1}\right], \left[\ell_g^{b,2}\right], \cdots, \left[\ell_g^{b,z}\right].$$

- keys: Let $(Y, j) := f(X, i)$, then the value of $X[i]$ is the $j$-th bit of input keys to circuit $Y$. As illustrated in Figure 10, Bob holds a commitment of that bit, denoted by $[b]$. Without loss of generality assume $\pi_{\mathsf{L}} = b$. Alice points out the fact that $\pi_{\mathsf{L}} = b$ to Bob, and Bob can compute

$$[\pi_{\mathsf{L}} \oplus b] \leftarrow [\pi_{\mathsf{L}}] \oplus [b] \,.$$

  Alice opens the above commitment and Bob checks if $\pi_{\mathsf{L}} \oplus b = 0$. If Bob accepts the checking, then Alice opens $\left[L_g^{\pi_{\mathsf{L}}}\right]$ and Bob obtains $L_g^b$.

- mem: Let $j := f(X, i)$ be the location of the database and let $b := D[j]$ be the value of $X[i]$. Alice first gives a zero-knowledge proof of knowledge of $L_g^0$ and $L_g^1$ (see Appendix B for more details). Then Alice sends the label $L_g^b$ to Bob.

- inp: Let $j := f(X, i)$ and $b := x[j]$ be the value of $X[i]$.

  - If $x[j]$ is an input bit from Alice, then Alice first gives a zero-knowledge proof of knowledge of $L_g^0$ and $L_g^1$, and then sends the label $L_g^b$ to Bob.
  - Otherwise the two parties run an OT where Alice inputs the openings of $\left[L_g^0\right]$ and $\left[L_g^1\right]$ and Bob inputs $b$, allowing Bob to obtain $L_g^b$.

- dyn: The label will be produced by another garbled circuit during runtime. There is no need for Bob to obtain a label for it before evaluation.

### 7.2 Our Protocol

Our secure RAM computation protocol $\Pi = (\Pi_{\mathsf{GData}}, \Pi_{\mathsf{GProg}}, \Pi_{\mathsf{GInput}}, \Pi_{\mathsf{GEval}})$ is described in Figure 11.

### 7.3 Security

In this section we prove the following theorem.

**Theorem 9.** *The protocol* $\Pi = (\Pi_{\mathsf{GData}}, \Pi_{\mathsf{GProg}}, \Pi_{\mathsf{GInput}}, \Pi_{\mathsf{GEval}})$ *is a secure RAM computation protocol against malicious adversaries satisfying the security and the efficiency requirements of Section 3.2.*

**Security Against Malicious Alice.**

We first consider the case where Bob is honest and Alice is cheating. We construct a simulator $\mathcal{S}$ (ideal-world adversary) which has access to Alice and the ideal functionality $\mathcal{F}$, and can emulate the adversary in the real world. Intuitively, the simulator $\mathcal{S}$ plays the role of Bob and runs the protocol with Alice, emulating the communication between Alice and Bob in the real world.

We now describe the simulator $\mathcal{S}$ in detail. In the simulation, if $\mathcal{S}$ rejects the component checking or consistency checking, then it inputs abort! to $\mathcal{F}$. Otherwise $\mathcal{S}$ accepts all the checking. In that case, if Bob accepts all the checking with probability no more than $2^{-s}$ in the real world, since it is negligible in the security parameter, $\mathcal{S}$ can do anything and does not affect security.

If Bob accepts all the checking with probability greater than $2^{-s}$ in the real world, then by Theorem 7, in each garbled circuit there are at least 1 RN gate and $\lceil \frac{\ell_{\mathsf{k}}}{2} \rceil$ KC gates per garbled gate with probability greater than $1 - 2^{-s}$. By union bound, there are at least 1 RN gate and $\lceil \frac{\ell_{\mathsf{k}}}{2} \rceil$ KC gates per garbled gate for every garbled circuit with overwhelming probability. By Theorem 4, $\mathcal{S}$

**Database Garbling $\Pi_{\mathsf{GData}}$.**

1. *Generating input keys:* Alice picks a PRF seed $s$ uniformly at random and generates all input keys needed for the garbled circuits.
2. *Generating garbled circuits:* The two parties run the protocol in Section 6.1 to generate all the garbled uniform circuits $\{\tilde{\mathsf{C}}^i\}_{i=1}^{\tilde{M}}$ in parallel.
3. *Generating partial labels:* For each wire $X[i]$ with category const/keys/mem, Alice provides Bob with labels consistent with $f(X, i)$ and her database content $D$ as in Section 7.1.

**Program Garbling $\Pi_{\mathsf{GProg}}$.** The new program $P$ with running time $t$ is known to both parties.

1. *Generating input keys:* Alice uses $s$ to generates all input keys needed for the new garbled circuits.
2. *Generating garbled circuits:* Let $N$ be the total number of previously generated GCs. The two parties run the protocol in Section 6.1 to generate all new garbled uniform circuits $\{\tilde{\mathsf{C}}^i\}_{i=N+1}^{N+\tilde{t}}$ in parallel.
3. *Generating partial labels:* For each wire $X[i]$ with category const/keys, Alice provides Bob with labels consistent with $f(X, i)$ and as in Section 7.1.
4. *Generating input-garbling-key:* Let $s^{in}$ be the set of input keys to all wires $X[i]$ with category inp, and Alice keeps it.

**Input Garbling $\Pi_{\mathsf{GInput}}$.** The input to the program is $x = (x^{\mathsf{A}}, x^{\mathsf{B}})$.

1. *Parsing input-garbling-key:* Alice parses $s^{in}$ as partial input keys to currently generated GCs.
2. *Generating garbled input:* For each wire $X[i]$ with category inp, Alice provides Bob with labels consistent with $f(X, i)$ and $(x^{\mathsf{A}}, x^{\mathsf{B}})$ as in Section 7.1.

**Garbled Evaluation $\Pi_{\mathsf{GEval}}$.**

- Bob uses $\tilde{D}, \tilde{P}, \tilde{x}$ to compute $y \leftarrow \mathsf{GEval}_{\mathsf{Lego}}{}^{\tilde{D}}(\tilde{P}, \tilde{x})$.

Figure 11: Secure RAM computation protocol $\Pi$

can evaluate all the garbled circuits, and hence the garble RAM, correctly, given that the garbled circuits are concatenated properly.

**Extracting Alice's input:** For each input wire with category mem, without loss of generality assume it is the left input wire of gate $g$, and its value is $D[j]$. Recall that Alice gives a zero-knowledge proof of knowledge of $L_g^0$ and $L_g^1$, and provides a label $L$ corresponding to $D[j]$. $\mathcal{S}$ can extract both labels to match with $L$ and figure out the bit $b$ of the database content. More precisely, if $L_g^b = L$, then set $D[j] := b$. If neither label matches $L$, then send abort! to $\mathcal{F}$. In such a way $\mathcal{S}$ can extract every bit of $D$ and $x^{\mathsf{A}}$ and sends to $\mathcal{F}$.

**Guessing Bob's input:** When $\mathcal{S}$ runs OT with Alice to get labels of Bob's input, $\mathcal{S}$ can extract both values that Alice feeds into OT, and determine whether they are correct openings of the commitment. If both values are incorrect, then in the real world Bob always sees an bad opening and terminates. In the ideal world $\mathcal{S}$ sends abort! to $\mathcal{F}$. Otherwise, let $I$ be the indices of OTs for which Alice inputs exactly one bad value, and let $\beta_i$ be the value of $x_i^{\mathsf{B}}$ which would lead Bob to not see the bad value. If Bob terminates in the real world, then Alice knows that $b_i \neq \beta_i$ for some $i \in I$. If Bob does not terminate, then Alice knows that $x_i^{\mathsf{B}} = \beta$ for all $i \in I$. $\mathcal{S}$ could accomplish the same in the ideal world by inputting $I$ and $\{\beta_i\}_{i \in I}$ to $\mathcal{F}$.

**Evaluation:** As we have already argued, $\mathcal{S}$ can evaluate the garble RAM correctly given the fact

that the garbled circuits are concatenated properly, namely all the partial labels that $\mathcal{S}$ obtains are correct. We only need to consider the input wires with categories const and keys. By Theorem 8, the commitments of every input wire are correct with overwhelming probability. For every input wire $w$, without loss of generality assume it is the left input wire of gate $g$. If $w$'s category is const, assume its value is $b$. Recall that Alice opens the bit-by-bit commitment of $L_g^b$, hence the label is correct with overwhelming probability. If $w$'s category is keys, assume its value is the $j$-th bit of input keys to circuit $Y$. $\mathcal{S}$ holds a correct commitment of that bit, denoted by $[b]$. Without loss of generality assume $\pi_{\mathsf{L}} = b$. Alice points out the fact that $\pi_{\mathsf{L}} = b$, and $\mathcal{S}$ can compute $[\pi_{\mathsf{L}} \oplus b] \leftarrow [\pi_{\mathsf{L}}] \oplus [b]$. Alice opens that commitment and $\left[L_g^{\pi_{\mathsf{L}}}\right]$. Since all the commitments are correct, if $\mathcal{S}$ accepts the checking of $\pi_{\mathsf{L}} \oplus b = 0$, then $\pi_{\mathsf{L}} = b$ and $\mathcal{S}$ obtains $L_g^b$ with overwhelming probability.

**Output:** $\mathcal{S}$ can evaluate the garbled RAM correctly, and obtain the output $y$. $\mathcal{S}$ then outputs $y$ along with the output of Alice.

**Security Against Malicious Bob.**

Now we consider when Alice is honest and Bob is cheating. Since we are running our protocol on a sequence of programs which run on a persistent database, the input of Bob could be adaptively decided after the two parties have generated the garbled database and garbled program. Thus we need to argue adaptive security against malicious Bob.

Recall Theorem 6, there exists a simulator $\mathsf{AdaptRamSim}_{\mathsf{Lego}}$ for the garbled RAM scheme $(\mathsf{GData}_{\mathsf{Lego}}, \mathsf{GProg}_{\mathsf{Lego}}, \mathsf{GInput}_{\mathsf{Lego}}, \mathsf{GEval}_{\mathsf{Lego}})$ that achieves adaptive security as defined in Section 3.1. In particular, $\mathsf{AdaptRamSim}_{\mathsf{Lego}}$ has the following procedures

$$\begin{aligned}
&\left(\tilde{D}^{\mathsf{sim}}, \mathsf{state}_0\right) \leftarrow \mathsf{AdaptRamSim}_{\mathsf{Lego}}\left(1^\kappa, 1^M\right);\\
&(\tilde{P}_i^{\mathsf{sim}}, \mathsf{state}_i') \leftarrow \mathsf{AdaptRamSim}_{\mathsf{Lego}}(\mathsf{state}_{i-1}, P_i, 1^{t_i}, y_i);\\
&(\tilde{x}_i^{\mathsf{sim}}, \mathsf{state}_i) \leftarrow \mathsf{AdaptRamSim}_{\mathsf{Lego}}(\mathsf{state}_i', y_i).
\end{aligned}$$

And the garbled RAM generated by $\mathsf{AdaptRamSim}_{\mathsf{Lego}}$ is indistinguishable from an honestly generated garbled RAM.

The simulator $\mathcal{S}$ plays the role of Alice and runs the protocol $\Pi_{\mathsf{GData}}$ and $\Pi_{\mathsf{GProg}}$ with Bob to generate $\tilde{D}^{\mathsf{sim}}$ and $\tilde{P}_i^{\mathsf{sim}}$, without knowing Alice's input. After $\Pi_{\mathsf{GData}}$ and $\Pi_{\mathsf{GProg}}$, Bob will obtain the labels corresponding to his input $x_i^{\mathsf{B}}$ via OT. $\mathcal{S}$ can extract $x_i^{\mathsf{B}}$ out of it, send to the $\mathcal{F}$, and obtain the output $y_i$. Then $\mathcal{S}$ provides Bob with the simulated garbled input $\tilde{x}^{\mathsf{sim}}$. By Theorem 6, the garbled RAM generated as above is computationally indistinguishable from Bob's view in the real world.

# References


[AHMR15] Arash Afshar, Zhangxiang Hu, Payman Mohassel, and Mike Rosulek. How to efficiently evaluate RAM programs with malicious security. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 702–729, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th Annual ACM Symposium on Theory of Computing*, pages 479–488, Philadephia, PA, USA, May 22–24, 1996. ACM Press.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, pages 103–112, 1988.

[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.

[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.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[CR73] Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *J. Comput. Syst. Sci.*, 7(4):354–375, 1973.

[DMN11] Ivan Damgård, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious RAM without random oracles. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 144–163, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

[FJN+13] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 537–556, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[FJNT15] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. TinyLEGO: An interactive garbling scheme for maliciously secure two-party computation. Cryptology ePrint Archive, Report 2015/309, 2015. `http://eprint.iacr.org/2015/309`.

[FJNT16] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic UC commitments. 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 542–565, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal of Computing*, 29(1):1–28, 1999.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GGMP16] Sanjam Garg, Divya Gupta, Peihan Miao, and Omkant Pandey. Secure multiparty RAM computation in constant rounds. In *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, Lecture Notes in Computer Science, Beijing, China, November 1–3, 2016. 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.

[GKK+12] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *CCS*, 2012.

[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.

[GLOV12] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd Annual Symposium on Foundations of Computer Science*, pages 51–60, New Brunswick, NJ, USA, October 20–23, 2012. IEEE Computer Society Press.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 182–194, New York City, NY, USA, May 25–27, 1987. ACM Press.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Proceedings of Eurocrypt 2006, volume 4004 of LNCS*, pages 339–358. Springer, 2006.

[GOSV14] Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 515–524, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[HY16] Carmit Hazay and Avishay Yanai. Constant-round maliciously secure two-party computation in the RAM model. In *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*, Lecture Notes in Computer Science, Beijing, China, November 1–3, 2016. Springer, Heidelberg, Germany.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions for secure computation. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing*, pages 99–108, Seattle, WA, USA, May 21–23, 2006. ACM Press.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 8–26, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

[LO13a] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 377–396, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany.

[LO13b] 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.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78, Barcelona, Spain, May 20–24, 2007. 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.

[MNP+04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. Fairplay — secure two-party computation system. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386. Springer, Heidelberg, Germany, March 15–17, 2009.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *29th Annual ACM Symposium on Theory of Computing*, pages 294–303, El Paso, TX, USA, May 4–6, 1997. ACM Press.

[Ost90] Rafail Ostrovsky. Efficient computation on oblivious RAMs. In *22nd Annual ACM Symposium on Theory of Computing*, pages 514–523, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 403–418. Springer, Heidelberg, Germany, March 15–17, 2009.

[SCSL11] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $o((\log n)^3)$ worst-case cost. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 197–214, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[SvDS+13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 299–310, Berlin, Germany, November 4–8, 2013. ACM Press.

[Wee10] Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st Annual Symposium on Foundations of Computer Science*, pages 531–540, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press.

[WHC+14] Xiao Shaun Wang, Yan Huang, T.-H. Hubert Chan, Abhi Shelat, and Elaine Shi. SCORAM: Oblivious RAM for secure computation. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14: 21st Conference on Computer and Communications Security*, pages 191–202, Scottsdale, AZ, USA, November 3–7, 2014. 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.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

# A Building Blocks

In this section we fix notations for RAM computation, provide definitions for static and adaptive secure circuit garbling schemes, and discuss the XOR-homomorphic commitment scheme employed in our constructions.

## A.1 RAM Model

For a program $P$ with database of size $M$ we denote the initial contents of the database data by $D \in \{0,1\}^M$. Additionally, the program gets a "short" input $x \in \{0,1\}^u$, which we alternatively think of as the initial state of the program. We use the notation $P^D(x)$ to denote the execution of program $P$ with initial database contents $D$ and input $x$. The program $P$ can read from and write to various locations in database $D$ throughout its execution.[7]

We will also consider the case where several different programs are executed sequentially and the database persists between executions. We denote this process as $(y_1, y_2, \cdots) = (P_1(x_1), P_2(x_2), \cdots)^D$ to indicate that first $P_1^D(x_1)$ is executed, resulting in some database contents $D_1$ and output $y_1$, then $P_2^{D_1}(x_2)$ is executed resulting in some database contents $D_2$ and output $y_2$ etc. As an example, imagine that $D$ is a huge database and the programs $P_i$ are database queries that can read and possibly write to the database and are parameterized by some values $x_i$.

## A.2 Circuit Garbling Scheme

Garbled circuits was first constructed by Yao [Yao86] (see e.g. Lindell and Pinkas [LP09] and Bellare et al. [BHR12b] for a detailed proof and further discussion). A circuit garbling scheme is a tuple of PPT algorithms $(\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{Eval})$. Very roughly $\mathsf{GCircuit}$ is the circuit garbling procedure, $\mathsf{GInput}$ is the input garbling procedure, and $\mathsf{Eval}$ is the evaluation procedure.

**Definition 1** (Circuit garbling scheme)**.** *A circuit garbling scheme consists of three procedures* $(\mathsf{GCircuit}, \mathsf{GInput}, \mathsf{Eval})$ *with the following syntax:*

- $\left(\tilde{\mathsf{C}}, s^{in}\right) \leftarrow \mathsf{GCircuit}\left(1^\kappa, \mathsf{C}\right)$*: On input a security parameter* $\kappa$*, a circuit* $\mathsf{C}$*, it outputs a* garbled circuit $\tilde{\mathsf{C}}$ *and an* input-garbling-key $s^{in}$*.*
- $\tilde{x} \leftarrow \mathsf{GInput}(s^{in}, x)$*: On input the input-garbling-key* $s^{in}$ *and* $x$*, it outputs the* garbled input $\tilde{x}$*.*
- $y = \mathsf{Eval}\left(\tilde{\mathsf{C}}, \tilde{x}\right)$*: On input a garbled circuit* $\tilde{\mathsf{C}}$ *and a garbled input* $\tilde{x}$*, it outputs* $y$*.*

**Correctness.** *For correctness, we require that for any circuit* $\mathsf{C}$ *and input* $x$*, we have that:*

$$\Pr\left[\mathsf{C}(x) = \mathsf{Eval}\left(\tilde{\mathsf{C}}, \tilde{x}\right)\right] = 1.$$

*where* $\left(\tilde{\mathsf{C}}, s^{in}\right) \leftarrow \mathsf{GCircuit}\left(1^\kappa, \mathsf{C}\right), \tilde{x} \leftarrow \mathsf{GInput}(s^{in}, x)$.

[7] In general, the distinction between what to include in the program $P$, the database data $D$ and the short input $x$ can be somewhat arbitrary. However as motivated by our applications we will typically be interested in a setting where that data $D$ is large while the size of the program $|P|$ and input length $u$ is small.

**Static security.** *For static security, we require that there exists a PPT simulator* $\mathsf{StatCircSim}$ *such that for any* $\mathsf{C}, x$*, we have that:*

$$\left(\tilde{\mathsf{C}}, \tilde{x}\right) \overset{c}{\approx} \mathsf{StatCircSim}\left(1^{\kappa}, \mathsf{C}, \mathsf{C}(x)\right)$$

*where* $\left(\tilde{\mathsf{C}}, s^{in}\right) \leftarrow \mathsf{GCircuit}\left(1^{\kappa}, \mathsf{C}\right), \tilde{x} \leftarrow \mathsf{GInput}(s^{in}, x)$.

**Adaptive security.** *For adaptive security, we require that there exists a PPT simulator* $\mathsf{AdaptCircSim}$ *such that for any PPT adversary* $\mathcal{A}$ *and any circuit* $\mathsf{C}$*, the output of the following two experiments are computational indistinguishable:*

*Real experiment*

- $\left(\tilde{\mathsf{C}}, s^{in}\right) \leftarrow \mathsf{GCircuit}\left(1^{\kappa}, \mathsf{C}\right)$
- $x \leftarrow \mathcal{A}(1^{\kappa}, \mathsf{C}, \tilde{\mathsf{C}})$
- $\tilde{x} \leftarrow \mathsf{GInput}(s^{in}, x)$
- *Output* $(\tilde{\mathsf{C}}, \tilde{x})$

*Simulated experiment*

- $\left(\tilde{\mathsf{C}}^{\mathsf{sim}}, \mathsf{state}\right) \leftarrow \mathsf{AdaptCircSim}\left(1^{\kappa}, \mathsf{C}\right)$
- $x \leftarrow \mathcal{A}(1^{\kappa}, \mathsf{C}, \tilde{\mathsf{C}})$
- $\tilde{x}^{\mathsf{sim}} \leftarrow \mathsf{AdaptCircSim}(\mathsf{state}, \mathsf{C}(x))$
- *Output* $(\tilde{\mathsf{C}}^{\mathsf{sim}}, \tilde{x}^{\mathsf{sim}})$

### A.3 Commitment Scheme

In our construction we use an XOR-homomorphic commitment scheme, which is UC secure in the OT-hybrid model, as a building block. The ideal functionality $\mathcal{F}_{\mathsf{HCOM}}$ is defined in Figure 12.

**Setup:** Alice and Bob both input $(\mathsf{init}, z)$.

**Commit:** When receiving a message $(\mathsf{commit}, i, m_i)$ from Alice where $m_i \in \{0,1\}^z$, $\mathcal{F}_{\mathsf{HCOM}}$ stores the pair $(i, m_i)$.

**Open:** When receiving a message $(\mathsf{open}, I)$ from Alice where $I$ is a set of integers, if $\forall i \in I$, a pair $(i, m_i)$ was stored previously, then $\mathcal{F}_{\mathsf{HCOM}}$ sends $(\mathsf{Open}, I, \bigoplus_{i \in I} m_i)$ to Bob.

Figure 12: Ideal functionality $\mathcal{F}_{\mathsf{HCOM}}$

We work in the $\mathcal{F}_{\mathsf{HCOM}}$-hybrid model, and essentially in the OT-hybrid model. We instantiate the additive homomorphic commitment scheme from [FJNT16] in the field $\mathbb{F}_2$, and commit messages bit by bit to achieve XOR homomorphism. For simplicity of notation we write $[m; r]$ (or $[m]$) for a commitment of $m$ with randomizer $r$.

## B Zero-Knowledge Proof of Knowledge via OT

During the protocol Alice needs to prove in zero-knowledge that she knows an opening of a commitment $[\Delta; r_{\Delta}]$. In this section we present a way to reduce UC zero-knowledge proof of knowledge to UC OT. The protocol allows the simulator to extract $\Delta$.

The protocol is the following: Alice picks at random $K_0, r_0$. Define $K_1 = K_0 \oplus \Delta, r_1 = r_0 \oplus \Delta$. Then Alice sends $[K_0]$ to Bob, and offers $((K_0, r_0), (K_1, r_1))$ to the OT. Bob chooses a random bit $b \in \{0,1\}$ and accepts if $K_b, r_b$ is an correct opening of $[K_b]$ (Note that Bob can compute $[K_1] \leftarrow [K_0] \oplus [\Delta]$ by himself). They repeat the protocol $s$ times in parallel. Therefore, Alice

cannot guess Bob's choice with probability better than $2^{-s}$. If Alice is corrupted the simulator gets to see both $K_0$ and $K_1$, and can therefore compute $\Delta = K_1 \oplus K_0$.

# C Postponed Proofs

## C.1 Proof of Theorem 3

*Proof.* We first construct the simulator $\mathsf{StatLegoSim}$. Given $\mathsf{C}, \mathsf{C}(x)$ and all the solders

$$\mathcal{S} := \left\{ \left\{ \left\{\delta^L_{g,i}, \delta^R_{g,i}, \delta^O_{g,i}\right\}_{i=1}^{\ell_\mathsf{n}}, \left\{\delta^K_{g,i}\right\}_{i=1}^{\ell_\mathsf{k}}, \delta^{O,K}_g \right\}_{g\in\mathsf{C}}, \left\{\delta^L_{g_1,g_2}\right\}_{g_1,g_2\in\mathsf{C}, g_1 \nearrow g_2}, \left\{\delta^R_{g_1,g_2}\right\}_{g_1,g_2\in\mathsf{C}, g_2 \nwarrow g_1} \right\},$$

$\mathsf{StatLegoSim}$ generates the garbled circuit and garble input as follows.

1. *Generating a global difference:* $\Delta \xleftarrow{\$} \{0,1\}^z$.
2. *Generating all the labels:* Generate the following labels gate by gate for each gate $g \in \mathsf{C}$, from input gates to output gates.

$$\begin{array}{ll}
\text{If } g_1 \nearrow g, \text{ then } L^0_{g,0} := O^0_{g_1,0} \oplus \delta^L_{g_1,g}; \text{ else } L^0_{g,0} \xleftarrow{\$} \{0,1\}^z & \\
\text{If } g \nwarrow g_1, \text{ then } R^0_{g,0} := O^0_{g_1,0} \oplus \delta^R_{g_1,g}; \text{ else } R^0_{g,0} \xleftarrow{\$} \{0,1\}^z & \\
O^0_{g,0} \xleftarrow{\$} \{0,1\}^z & \\
L^0_{g,i} := L^0_{g,0} \oplus \delta^L_{g,i}, R^0_{g,i} := R^0_{g,0} \oplus \delta^R_{g,i}, O^1_{g,i} := O^0_{g,0} \oplus \delta^O_{g,i} & \forall 1 \le i \le \ell_\mathsf{n} \\
L^1_{g,i} := L^0_{g,i} \oplus \Delta, R^1_{g,i} := R^0_{g,i} \oplus \Delta, O^1_{g,i} := O^0_{g,i} \oplus \Delta & \forall 0 \le i \le \ell_\mathsf{n} \\
K^0_{g,0} := O^0_{g,0} \oplus \delta^{O,K}_g & \\
K^0_{g,i} := K^0_{g,0} \oplus \delta^K_{g,i} & \forall 1 \le i \le \ell_\mathsf{k} \\
K^1_{g,i} := K^0_{g,i} \oplus \Delta & \forall 0 \le i \le \ell_\mathsf{k}
\end{array}$$

   Let $\mathcal{W}_\mathsf{out}$ be the set of output wires of $\mathsf{C}$. For each $w \in \mathcal{W}_\mathsf{out}$, let the value of $w$ be $b$. Suppose it is the output wire of gate $g$. Let

$$O^0_g := b^z; O^1_g := \overline{b}^z$$

   be the labels for $w$.

3. *Generating garbled gate tables:* Use the above labels to generate the garbled gate table of every $\tilde{g}$ as described in Section 5.1 and Figures 5, 6 except that the garbled table of a RN is the following:

$$\begin{aligned}
c^{\pi(0)} &= \mathsf{Enc}_{L^0,R^0}(O^0); \\
c^{\pi(1)} &= \mathsf{Enc}_{L^0,R^1}(O^0); \\
c^{\pi(2)} &= \mathsf{Enc}_{L^1,R^0}(O^0); \\
c^{\pi(3)} &= \mathsf{Enc}_{L^1,R^1}(O^0).
\end{aligned}$$

   And the the garbled table of a KC is the following:

$$\begin{aligned}
c^{\pi(0)} &= \mathsf{H}(K^0); \\
c^{\pi(1)} &= \mathsf{H}(K^0).
\end{aligned}$$

4. *Garbled circuit:* $\tilde{\mathsf{C}}^{\mathsf{sim}}$ consists all the garbled tables and all solders $\mathcal{S}$.

5. *Garbled input:* Let $\mathcal{W}_{\mathsf{in}}$ be the set of input wires of $\mathsf{C}$. For each $w \in \mathcal{W}_{\mathsf{in}}$, suppose it is an input wire of gate $g$. If $w$ is $g$'s left input wire, then let $(\mathsf{label}_w^0, \mathsf{label}_w^1) := (L_g^0, L_g^1)$; otherwise let $(\mathsf{label}_w^0, \mathsf{label}_w^1) := (R_g^0, R_g^1)$. Let

$$\tilde{x}^{\mathsf{sim}} = \left\{\mathsf{label}_w^0\right\}_{w \in \mathcal{W}_{\mathsf{in}}}$$

be the garbled input.

6. *Output* $\tilde{\mathsf{C}}^{\mathsf{sim}}, \tilde{x}^{\mathsf{sim}}$.

To argue that the simulated $(\tilde{\mathsf{C}}^{\mathsf{sim}}, \tilde{x}^{\mathsf{sim}})$ is indistinguishable from an honestly generated garbled circuit and garbled input, the idea is similar to arguing security in Yao's garbled circuit [Yao82]. First we replace gate by gate the unevaluated ciphertexts in the garbled gate table by random strings. Security of the encryption schemes would guarantee that the garbled circuit remains indistinguishable from an honestly generated one. Then we switch the zero and one labels to be consistent with $(\tilde{\mathsf{C}}^{\mathsf{sim}}, \tilde{x}^{\mathsf{sim}})$. Finally we conclude the proof by changing the unevaluated ciphertexts to be the same as $\tilde{\mathsf{C}}^{\mathsf{sim}}$. □

## C.2 Proof of Theorem 5

*Proof.* Given the static secure LEGO garbling scheme with simulator $\mathsf{StatLegoSim}$, we first construct the simulator $\mathsf{AdaptLegoSim}$ as follows. The main ideal is that after seeing the output $\mathsf{C}(x)$, $\mathsf{AdaptLegoSim}$ can manipulate the output of the random oracles and enforce the previously generated garbled circuit to be a newly simulated one.

- $\left(\tilde{\mathsf{C}}_{\mathsf{a}}^{\mathsf{sim}}, \mathsf{state}\right) \leftarrow \mathsf{AdaptLegoSim}\left(1^{\kappa}, \mathsf{C}\right).$
  - $\left(\tilde{\mathsf{C}}_{\mathsf{a}}, s_{\mathsf{a}}^{in}\right) \leftarrow \mathsf{AdaptLegoGCircuit}\left(1^{\kappa}, \mathsf{C}\right).$
  - $\mathsf{state} \leftarrow \tilde{\mathsf{C}}_{\mathsf{a}}$.
  - Output $\left(\tilde{\mathsf{C}}_{\mathsf{a}}, \mathsf{state}\right)$.
- $\tilde{x}^{\mathsf{sim}} \leftarrow \mathsf{AdaptLegoSim}(\mathsf{state}, \mathsf{C}(x))$.
  - Let

$$\mathcal{S} := \left\{ \left\{ \left\{\delta_{g,i}^L, \delta_{g,i}^R, \delta_{g,i}^O\right\}_{i=1}^{\ell_{\mathsf{n}}}, \left\{\delta_{g,i}^K\right\}_{i=1}^{\ell_{\mathsf{k}}}, \delta_g^{O,K} \right\}_{g \in \mathsf{C}}, \left\{\delta_{g_1,g_2}^L\right\}_{g_1,g_2 \in \mathsf{C}, g_1 \nearrow g_2}, \left\{\delta_{g_1,g_2}^R\right\}_{g_1,g_2 \in \mathsf{C}, g_2 \nwarrow g_1} \right\}$$

    be the set of all solders of $\tilde{\mathsf{C}}_{\mathsf{a}}$.
  - $\left(\tilde{\mathsf{C}}^{\mathsf{sim}}, \tilde{x}^{\mathsf{sim}}\right) \leftarrow \mathsf{StatLegoSim}\left(1^{\kappa}, \mathsf{C}, \mathsf{C}(x)\right)\big|_{\mathcal{S}}$
  - Use $\tilde{x}^{\mathsf{sim}}$ to simulate the evaluation of $\tilde{\mathsf{C}}_{\mathsf{a}}$ gate by gate as follows:
    * Let $L_g^a, R_g^b$ be the input labels of $\tilde{g}_{\mathsf{a}} \in \tilde{\mathsf{C}}_{\mathsf{a}}$.
    * Use $\delta_{g,i}^L, \delta_{g,i}^R$ to compute $L_{g,i}^a, R_{g,i}^b$ for $\forall 0 \le i \le \ell_{\mathsf{n}}$.
    * For $\forall 0 \le i \le \ell_{\mathsf{n}}$,

· Let $c^t$ ($t \in \{0,1,2,3\}$) be the ciphertext of $\mathsf{RN}_{g,i} \in \tilde{\mathsf{C}}^{\mathsf{sim}}$ that can be decrypted by $L^a_{g,i}, R^b_{g,i}$, and let $O^{a\otimes b}_{g,i}$ be the output label, namely $c^t = \mathsf{Enc}_{L^a_{g,i},R^b_{g,i}}(O^{a\otimes b}_{g,i})$.

· $\mathsf{hash}\left[|c^t|, L^a_{g,i}||R^b_{g,i}\right] \leftarrow h^t \oplus c^t$.

∗ Note that $\{O^{a\otimes b}_{g,i}\}^{\ell_\mathsf{n}}_{i=0}$ are consistent with $\{\delta^O_{g,i}\}^{\ell_\mathsf{n}}_{i=1}$.

∗ Use $\delta^{O,K}_g$ and $\delta^K_{g,i}$ to compute $K^{a\otimes b}_{g,i}$ for $\forall 0 \le i \le \ell_\mathsf{k}$.

∗ For $\forall 0 \le i \le \ell_\mathsf{k}$,

· Let $c^t$ ($t \in \{0,1\}$) be the ciphertext of $\mathsf{KC}_{g,i} \in \tilde{\mathsf{C}}^{\mathsf{sim}}$ that can be checked by $K^{a\otimes b}_{g,i}$.

· $\mathsf{hash}\left[|c^t|, K^{a\otimes b}_{g,i}\right] \leftarrow h^t \oplus c^t$.

∗ Let $O^{a\otimes b}_g := O^{a\otimes b}_{g,0}$, and continue the evaluation using gate solders $\left\{\delta^L_{g_1,g_2}\right\}_{g_1,g_2\in\mathsf{C},g_1\nearrow g_2}$, $\left\{\delta^R_{g_1,g_2}\right\}_{g_1,g_2\in\mathsf{C}}$

– Output $\tilde{x}^{\mathsf{sim}}$.

- $\mathsf{hash}[\gamma, w] \leftarrow \mathsf{AdaptLegoSim}(\mathsf{RO}, \gamma, w)$.

– If $\mathsf{hash}[\gamma, w] = \perp$ then $\mathsf{hash}[\gamma, w] \xleftarrow{\$} \{0,1\}^\gamma$.

– Output $\mathsf{hash}[\gamma, w]$.

Note that in the procedure of $\mathsf{AdaptLegoSim}(\mathsf{state}, \mathsf{C}(x))$, by manipulating the output of random oracles properly $\mathcal{S}$ can fool the evaluator gate by gate into evaluating on the later simulated $\tilde{\mathsf{C}}^{\mathsf{sim}}$ and $\tilde{x}^{\mathsf{sim}}$. Therefore, given an adversary that can distinguish the real and simulated experiments in the adaptive LEGO garbling scheme, we can build an adversary that can distinguish $\left(\tilde{\mathsf{C}}^{\mathsf{sim}}, \tilde{x}^{\mathsf{sim}}\right)$ from honestly generated $\left(\tilde{\mathsf{C}}, \tilde{x}\right)$, which breaks the static secure LEGO garbling scheme. □

### C.3 Proof of Theorem 7

In this section we prove the following two lemmas, and combining the two lemmas we can immediately conclude Theorem 7.

**Lemma 10.** *Assume that Alice is corrupted and Bob is honest. In generating an adaptive LEGO garbled circuit $\tilde{\mathsf{C}}$, if Bob accepts all the checking with probability greater than $2^{-s}$, then there exists a constant $c$ such that at most $c \cdot s$ RN gates and $c \cdot s$ KC gates have incorrect commitments or incorrect garbled gate tables.*

**Lemma 11.** *Assume that Alice is corrupted and Bob is honest. In generating an adaptive LEGO garbled circuit $\tilde{\mathsf{C}}$, if Alice generates at most $c \cdot s$ RN gates and $c \cdot s$ KC gates with incorrect commitments or incorrect garbled gate tables, then with probability higher than $1 - 2^{-s}$ there are at least 1 correct garbled RN gate and $\lceil \frac{\ell_\mathsf{k}}{2} \rceil$ correct garbled KC gates per garbled gate in $\tilde{\mathsf{C}}$.*

*Proof of Lemma 10.* Recall that $N_\mathsf{n} = (\ell_\mathsf{n} + 1)n$ is the number of RN gates needed in $\tilde{\mathsf{C}}$. Alice first generates $\phi_\mathsf{n} N_\mathsf{n}$ garbled RN gates, $(\phi_\mathsf{n} - 1)N_\mathsf{n}$ among which will be checked. Assume that there are $\alpha$ incorrectly formed gates in the $\phi_\mathsf{n} N_\mathsf{n}$ RN gates. Bob randomly picks $(\phi_\mathsf{n} - 1)N_\mathsf{n}$ gates to check. For each incorrect gate, if Bob picks it, then he will accept the component checking of that gate with probability $\le \frac{3}{4}$.

Pr[Bob accepts component checking of all the RN gates]

$$
\begin{aligned}
&= \sum_{i=0}^{\alpha} \Pr[\text{Bob picks } i \text{ incorrect RN gates}] \times \Pr[\text{Bob accepts component checking of } i \text{ incorrect gates}] \\
&\leq \sum_{i=0}^{\alpha} \frac{\binom{\alpha}{i}\binom{\phi_{\mathsf{n}} N_{\mathsf{n}} - \alpha}{(\phi_{\mathsf{n}}-1)N_{\mathsf{n}} - i}}{\binom{\phi_{\mathsf{n}} N_{\mathsf{n}}}{(\phi_{\mathsf{n}}-1)N_{\mathsf{n}}}} \cdot \left(\frac{3}{4}\right)^i \\
&= \sum_{i=0}^{\alpha} \binom{\alpha}{i} \cdot \left(\frac{3}{4}\right)^i \cdot \frac{\prod_{j=0}^{\alpha-i-1}(N_{\mathsf{n}} - j)\prod_{j=0}^{i-1}((\phi_{\mathsf{n}}-1)N_{\mathsf{n}} - j)}{\prod_{j=0}^{\alpha-1}(\phi_{\mathsf{n}} N_{\mathsf{n}} - j)} \\
&\leq \sum_{i=0}^{\alpha} \binom{\alpha}{i} \cdot \left(\frac{3}{4}\right)^i \cdot \left(\frac{N_{\mathsf{n}}}{\phi_{\mathsf{n}} N_{\mathsf{n}} - \alpha}\right)^{\alpha-i} \cdot \left(\frac{(\phi_{\mathsf{n}}-1)N_{\mathsf{n}}}{\phi_{\mathsf{n}} N_{\mathsf{n}} - \alpha}\right)^i.
\end{aligned}
$$

If $\alpha > c_1 \phi_{\mathsf{n}} N_{\mathsf{n}}$ for some constant $0 \leq c_1 < 1$, then $\mathbb{E}[\text{bad gates picked by Bob}] = \frac{\alpha(\phi_{\mathsf{n}}-1)}{\phi_{\mathsf{n}}} > c_1(\phi_{\mathsf{n}}-1)N_{\mathsf{n}}$, and each incorrect gate will fail the checking with probability $\geq \frac{1}{4}$. By Chernoff bound and appropriate value of $c$ (depending on $s$ and $\phi_{\mathsf{n}}$), Bob will accept with probability $< 2^{-s}$. So we can assume that $\alpha \leq c_1 \phi_{\mathsf{n}} N_{\mathsf{n}}$.

$$
\begin{aligned}
&\Pr[\text{Bob accepts component checking of all the RN gates}] \\
&\leq \sum_{i=0}^{\alpha} \binom{\alpha}{i} \cdot \left(\frac{3}{4}\right)^i \cdot \left(\frac{N_{\mathsf{n}}}{(1-c_1)\phi_{\mathsf{n}} N_{\mathsf{n}}}\right)^{\alpha-i} \cdot \left(\frac{(\phi_{\mathsf{n}}-1)N_{\mathsf{n}}}{(1-c_1)\phi_{\mathsf{n}} N_{\mathsf{n}}}\right)^i \\
&\leq \sum_{i=0}^{\alpha} \binom{\alpha}{i} \cdot \left(\frac{3}{4}\right)^i \cdot \left(\frac{\max\{\phi_{\mathsf{n}}-1, 1\}}{(1-c_1)\phi_{\mathsf{n}}}\right)^{\alpha} \\
&= \left(\frac{\max\{\phi_{\mathsf{n}}-1, 1\}}{(1-c_1)\phi_{\mathsf{n}}}\right)^{\alpha} \cdot \left(\frac{7}{4}\right)^{\alpha} \leq c_2^{\alpha}
\end{aligned}
$$

where $0 < c_2 < 1$ for appropriate values of $c_1$ and $\phi_{\mathsf{n}}$. Since Bob accepts all the component checking with probability $> 2^{-s}$, $\Pr[\text{Bob accepts component checking of all the RN gates}] > 2^{-s}$,

$$
\alpha \geq \frac{\ln 2}{-\ln c_2} s = c \cdot s.
$$

By the same argument for KC gates and by choosing the value of $\phi_{\mathsf{k}}$ properly, we know that there are at most $c \cdot s$ incorrect KC gates. □

*Proof of Lemma 11.* There are $N_{\mathsf{n}} = (\ell_{\mathsf{n}}+1)n$ RN gates among which at most $c \cdot s$ are incorrect, and $N_{\mathsf{k}} = (\ell_{\mathsf{k}}+1)n$ KC gates among which at most $c \cdot s$ are incorrect. For each gate $g \in \mathsf{C}$,

$$
\begin{aligned}
&\Pr[g \text{ has} \leq \ell_{\mathsf{n}} \text{ correct KC gates or has no correct RN gate}] \\
&\leq \Pr[g \text{ has} \geq \ell_{\mathsf{n}}+1 \text{ incorrect KC gates}] + \Pr[g \text{ has } \ell_{\mathsf{n}}+1 \text{ incorrect RN gates}] \\
&\leq \sum_{i=\ell_{\mathsf{n}}+1}^{\ell_{\mathsf{k}}+1} \binom{\ell_{\mathsf{k}}+1}{i} \left(\frac{c \cdot s}{N_{\mathsf{k}}}\right)^i + \left(\frac{c \cdot s}{N_{\mathsf{n}}}\right)^{\ell_{\mathsf{n}}+1} \\
&\leq \left(\frac{c \cdot s}{N_{\mathsf{k}}}\right)^{\ell_{\mathsf{n}}+1} \cdot \sum_{i=\ell_{\mathsf{n}}+1}^{\ell_{\mathsf{k}}+1} \binom{\ell_{\mathsf{k}}+1}{i} + \left(\frac{c \cdot s}{N_{\mathsf{n}}}\right)^{\ell_{\mathsf{n}}+1}
\end{aligned}
$$

$$\leq \left(\frac{c \cdot s}{(\ell_{\mathsf{k}}+1)n}\right)^{\ell_{\mathsf{n}}+1} \cdot 2^{\ell_{\mathsf{k}}} + \left(\frac{c \cdot s}{(\ell_{\mathsf{n}}+1)n}\right)^{\ell_{\mathsf{n}}+1}$$
$$\leq \left(\frac{4c \cdot s}{(\ell_{\mathsf{n}}+1)n}\right)^{\ell_{\mathsf{n}}+1}.$$

Hence

$$\begin{aligned}&\Pr[\exists g \in \mathsf{C} \text{ with } \leq \ell_{\mathsf{n}} \text{ correct KC gates or with no correct RN gate}]\\ \leq & n\left(\frac{4c \cdot s}{(\ell_{\mathsf{n}}+1)n}\right)^{\ell_{\mathsf{n}}+1} \leq 2^{-s} \text{ (for large enough } \ell_{\mathsf{n}}).\end{aligned}$$

□

## C.4 Proof of Theorem 8

*Proof.* Assume that there are at least 1 correct RN gate per garbled gate in $\tilde{\mathsf{C}}$. For each input wire $w \in \mathcal{W}_{\mathsf{in}}$, without loss of generality assume that it is the left input wire of gate $g$. Let $\mathsf{RN}_{g,0}, \cdots, \mathsf{RN}_{g,\ell_{\mathsf{n}}}$ be the RN gates for $\tilde{g}$. Recall that Bob has the following commitments:

$$\begin{gathered}\left[L_{g,0}^{\pi_{\mathsf{L},0}}\right] || \left[\pi_{\mathsf{L},0}\right]; \left[L_{g,0}^{\overline{\pi_{\mathsf{L},0}}}\right] || \left[\overline{\pi_{\mathsf{L},0}}\right];\\ \left[L_{g,1}^{\pi_{\mathsf{L},1}}\right] || \left[\pi_{\mathsf{L},1}\right]; \left[L_{g,1}^{\overline{\pi_{\mathsf{L},1}}}\right] || \left[\overline{\pi_{\mathsf{L},1}}\right];\\ \vdots\\ \left[L_{g,\ell_{\mathsf{n}}}^{\pi_{\mathsf{L},\ell_{\mathsf{n}}}}\right] || \left[\pi_{\mathsf{L},\ell_{\mathsf{n}}}\right]; \left[L_{g,\ell_{\mathsf{n}}}^{\overline{\pi_{\mathsf{L},\ell_{\mathsf{n}}}}}\right] || \left[\overline{\pi_{\mathsf{L},\ell_{\mathsf{n}}}}\right].\\ \left[\ell_{g,0}^{0,1}\right], \left[\ell_{g,0}^{0,2}\right], \cdots, \left[\ell_{g,0}^{0,z}\right]; \left[\ell_{g,0}^{1,1}\right], \left[\ell_{g,0}^{1,2}\right], \cdots, \left[\ell_{g,0}^{1,z}\right];\\ \left[\ell_{g,1}^{0,1}\right], \left[\ell_{g,1}^{0,2}\right], \cdots, \left[\ell_{g,1}^{0,z}\right]; \left[\ell_{g,1}^{1,1}\right], \left[\ell_{g,1}^{1,2}\right], \cdots, \left[\ell_{g,1}^{1,z}\right];\\ \vdots\\ \left[\ell_{g,\ell_{\mathsf{n}}}^{0,1}\right], \left[\ell_{g,\ell_{\mathsf{n}}}^{0,2}\right], \cdots, \left[\ell_{g,\ell_{\mathsf{n}}}^{0,z}\right]; \left[\ell_{g,\ell_{\mathsf{n}}}^{1,1}\right], \left[\ell_{g,\ell_{\mathsf{n}}}^{1,2}\right], \cdots, \left[\ell_{g,\ell_{\mathsf{n}}}^{1,z}\right].\end{gathered}$$

We claim that all the commitments are correct.

Since there is at least 1 correct RN gate for $\tilde{g}$, let $\mathsf{RN}_{g,t}$ be the correct gate. Then all the commitments of $\mathsf{RN}_{g,t}$ are correct. Now we compare the commitments of $\mathsf{RN}_{g,0}$ with $\mathsf{RN}_{g,t}$, and we will see that input consistency checking ensures the correctness of commitments of $\mathsf{RN}_{g,0}$.

First, when revealing the relation between $\{\pi_{\mathsf{L},i}\}_{i=0}^{\ell_{\mathsf{n}}}$, Bob computes the following

$$\left[\pi_{\mathsf{L},0} \oplus \pi_{\mathsf{L},t}\right] \leftarrow \left[\pi_{\mathsf{L},0}\right] \oplus \left[\pi_{\mathsf{L},t}\right].$$

Alice opens the commitment. Bob checks if $\pi_{\mathsf{L},0} \oplus \pi_{\mathsf{L},t} = 0$ or 1, and obtains the relation between $\pi_{\mathsf{L},0}$ and $\pi_{\mathsf{L},t}$, namely $\pi_{\mathsf{L},t} = \pi_{\mathsf{L},0}$ or $\overline{\pi_{\mathsf{L},0}}$. Since $\pi_{\mathsf{L},t} = 0/1$ and $[\pi_{\mathsf{L},t}]$ is correct, $[\pi_{\mathsf{L},0}]$ is also a correct commitment of 0/1. Without loss of generality assume that $\pi_{\mathsf{L},0} = \pi_{\mathsf{L},t}$.

Second, when checking the difference between labels is consistent with previously revealed $\left\{\delta_{g,i}^{L}\right\}_{i=1}^{\ell_{\mathsf{n}}}$, Bob computes

$$\left[\delta_{g,t}\right] \leftarrow \left[L_{g,0}^{\pi_{\mathsf{L},0}}\right] \oplus \left[L_{g,t}^{\pi_{\mathsf{L},t}}\right].$$

Alice opens the commitment, and Bob checks if $\delta_{g,t} = \delta^L_{g,t}$. Hence we conclude that $L^{\pi_{\mathsf{L},0}}_{g,0} = L^{\pi_{\mathsf{L},t}}_{g,t} \oplus \delta_{g,t}$ is the label of $\pi_{\mathsf{L},0}$, and the commitment $\left[L^{\pi_{\mathsf{L},0}}_{g,0}\right]$ is correct.

$L^0_{g,0} = L^0_{g,t} \oplus \delta^L_{g,t}$, and the commitment $\left[L^0_{g,0}\right]$ is correct.

Third, when checking that each pair of labels has a difference $\Delta$ in between, Bob computes

$$[0] \,||\, [0] \leftarrow \left(\left[L^{\pi_{\mathsf{L},i}}_{g,t}\right] \,||\, [\pi_{\mathsf{L},t}]\right) \oplus \left(\left[L^{\overline{\pi_{\mathsf{L},t}}}_{g,t}\right] \,||\, [\overline{\pi_{\mathsf{L},t}}]\right) \oplus [\Delta] \,||\, [1]\,.$$

Alice opens the commitment, and Bob checks. Since $[\pi_{\mathsf{L},0}]$ is also a correct commitment of $0/1$, $[\overline{\pi_{\mathsf{L},0}}]$ is also a correct commitment of $0/1$, and is the opposite of $[\pi_{\mathsf{L},0}]$. Since $L^{\pi_{\mathsf{L},0}}_{g,0} = L^{\pi_{\mathsf{L},t}}_{g,t} \oplus \delta_{g,t}$ is the label of $\pi_{\mathsf{L},0}$ and the commitment $\left[L^{\pi_{\mathsf{L},0}}_{g,0}\right]$ is correct, we conclude that $L^{\overline{\pi_{\mathsf{L},0}}}_{g,0} = L^{\pi_{\mathsf{L},0}}_{g,0} \oplus \Delta$ is the label of $\overline{\pi_{\mathsf{L},0}}$, and the commitment $\left[L^{\overline{\pi_{\mathsf{L},0}}}_{g,0}\right]$ is also correct.

Forth, when checking that the difference between bit-by-bit commitment is consistent with previously revealed $\left\{\delta^L_{g,i}\right\}_{i=1}^{\ell_{\mathsf{n}}}$, for every $b \in \{0,1\}, 1 \le u \le z$, Bob can compute

$$\left[\delta^u_{g,t}\right] \leftarrow \left[\ell^{b,u}_{g,0}\right] \oplus \left[\ell^{b,u}_{g,t}\right].$$

Alice opens the commitment, and Bob checks if $\delta^u_{g,t}$ is equal to the $u$-th bit of previously revealed $\delta^L_{g,t}$. Since $\left[\ell^{c,u}_{g,t}\right]$ is correct, we conclude that $\ell^{c,u}_{g,0} = \delta^u_{g,t} \oplus \ell^{c,u}_{g,t}$, and $\left[\ell^{c,u}_{g,0}\right]$ is also correct.

Since the input consistency checking is done between $\mathsf{RN}_{g,0}$ and $\mathsf{RN}_{g,i}$ for every $i$, using the same argument we can conclude that all the commitments should be correct if Bob passes the checking.

Finally, if Bob accepts all the checks with probability greater than $2^{-s}$, by Theorem 7 we know that with probability greater than $1 - 2^{-s}$ there exist at least 1 correct RN gate per garbled gate in $\tilde{\mathsf{C}}$. Hence concludes the proof. □