

# Adaptive Garbled Circuits and Garbled RAM from Non-Programmable Random Oracles\*

Cruz Barnum<sup>1</sup>, David Heath<sup>2</sup>, Vladimir Kolesnikov<sup>3</sup>, and Rafail Ostrovsky<sup>4</sup>

<sup>1</sup> cruzjb2@illinois.edu, UIUC

<sup>2</sup> daheath@illinois.edu, UIUC

<sup>3</sup> kolesnikov@gatech.edu, Georgia Tech

<sup>4</sup> rafail@cs.ucla.edu, UCLA

**Abstract.** Garbled circuit techniques that are secure in the adaptive setting – where inputs are chosen after the garbled program is sent – are motivated by practice, but they are notoriously difficult to achieve. Prior adaptive garbling is either impractically expensive or encrypts the entire garbled program with the output of a programmable random oracle (PRO), a strong assumption.

We present a simple framework for proving adaptive security of garbling schemes in the non-programmable random oracle (NPRO) model. NPRO is a milder assumption than PRO, and it is close to the assumption required by the widely used Free XOR extension. Our framework is applicable to a number of existing GC techniques, which are proved adaptively secure without modification.

As our main application of the framework, we construct and prove adaptively secure a garbling scheme for *tri-state circuits*, a recently proposed circuit model that captures both Boolean circuits and RAM programs (Heath et al., Crypto 2023). For a given TSC  $C$ , our garbling of  $C$  is at most  $|C| \cdot \lambda$  bits long, where  $\lambda$  is the security parameter. This implies both an adaptively secure garbled Boolean circuit scheme, as well an adaptively secure garbled RAM scheme where the garbling of  $T$  steps of a RAM program has size  $O(T \cdot \log^3 T \cdot \log \log T \cdot \lambda)$  bits.

Our scheme is concretely efficient: its Boolean circuit handling matches the performance of half-gates, and it is adaptively secure from NPRO.

**Keywords:** Adaptive Garbling · Garbled RAM · Multi-Party Computation · Non-Programmable Random Oracles

## 1 Introduction

Yao’s Garbled Circuit (GC) [Yao86] is a powerful cryptographic technique that allows two parties – a garbler  $G$  and an evaluator  $E$  – to securely evaluate

---

\* A previous version of this work required a modification to garbling schemes to prove they are adaptively secure: all RO calls were prefixed by a random seed. This version removes this requirement: we show adaptive security of several popular schemes *without modification*.

an arbitrary program  $\mathcal{P}$  on their joint private inputs. GC is foundational to secure two-party computation (2PC) and multiparty computation (MPC). The technique is noteworthy because it allows 2PC and MPC protocols that use only a small constant number of rounds, and because it relies almost entirely only on fast symmetric-key cryptographic primitives. GC is the most efficient secure computation approach in many settings, particularly those that involve two parties; studying its power, performance, and underlying assumptions is well-motivated by both theory and practice.

*Garbled RAM.* Typically, the evaluated program  $\mathcal{P}$  is a Boolean circuit. While Boolean circuits are powerful enough to represent any bounded function, the representation is often inefficient, in the sense that many natural programs blow up to large circuits. This is problematic because the cost of garbling typically scales linearly in the size of the circuit. The common sources of this blow-up are uses of complex looping/branching control flow and of complex data structures.

Garbled RAM (or GRAM, [LO13]) is a powerful GC extension that enables garbling of random access machine (RAM) programs. The GRAM primitive solves the above sources of blow-up, allowing for constant round 2PC/MPC protocols that handle complex programs.

Recent work [HKO23] showed that there exists a relatively simple circuit model – called tri-state circuits (TSCs) – that can efficiently emulate both Boolean circuits and RAM programs. We construct a scheme that garbles tri-state circuits, implying results for both garbled Boolean circuits and for garbled RAM. We emphasize that our scheme’s handling of Boolean circuits matches the cost of state-of-the-art half-gates garbling [ZRE15].<sup>5</sup>

*Garbling schemes and selective security.* For simplicity and modularity, GC techniques are often formalized as *garbling schemes* [BHR12b]. A garbling scheme factors evaluation of program  $\mathcal{P}$  on joint secret input  $x$  into four steps:

1. The parties encode their joint input  $x$  into a garbled form  $\tilde{x}$ .  $\tilde{x}$  is given to the GC evaluator  $E$ .
2. The GC garbler  $G$  encodes the program  $\mathcal{P}$  as a *garbled* program  $\tilde{\mathcal{P}}$ , and  $\tilde{\mathcal{P}}$  is also sent to  $E$ .
3.  $E$  evaluates  $\tilde{\mathcal{P}}$  on the garbled input, yielding garbled output  $\tilde{y} \leftarrow \text{Eval}(\tilde{\mathcal{P}}, \tilde{x})$ .
4. The parties *decode* the garbled output into its cleartext form  $y$ .

Of course,  $y$  should be equal to the result of simply running  $\mathcal{P}(x)$  in cleartext.

Security of garbling schemes is typically considered in the so-called *selective* setting. Security against (semi-honest) corrupted  $G$  is easy, as it essentially reduces to the security of Oblivious Transfer (OT). Security against corrupted  $E$  is more detailed. Consider the following interaction between  $G$  and  $E$ :

1.  $G$  garbles  $\mathcal{P}$  to obtain  $\tilde{\mathcal{P}}$ .

---

<sup>5</sup> [RR21] uses less communication than [ZRE15], but it uses significantly more computation. We consider both techniques state-of-the-art.

2.  $E$  sends a cleartext input  $x$  to  $G$ .
3.  $G$  sends to  $E$  the garbled input  $\tilde{x}$ , the garbled program  $\tilde{\mathcal{P}}$ , and information  $d$  needed to decode the output.

Security against a corrupted  $E$  is proved by considering this interaction and constructing a simulator that – from the program output  $y$  alone – can forge a garbled program, garbled input, and decoding information such that  $E$  cannot tell whether they are interacting with  $G$  or with the simulator. Existence of such a simulator proves that  $E$  learns *nothing beyond*  $y$  from GC evaluation.

The crucial detail of the above interaction is that  $E$  must select its input  $x$  *before* it sees the garbled program  $\tilde{\mathcal{P}}$ .

*Adaptive security.* Transmission of the garbled program  $\tilde{\mathcal{P}}$  is the main bottleneck of GC. One common practical mitigation is to move GC generation and transmission to the *offline* (or *preprocessing*) phase. In this way, we can do most of the work “overnight”, before inputs are ready. Once the parties obtain their inputs, they enter the *online* phase and quickly compute the desired output.

At first glance, garbling schemes seem ideal for the offline/online setting. Indeed,  $G$  can simply garble the program and send  $\tilde{\mathcal{P}}$  in advance; then, once inputs become available,  $G$  quickly conveys garbled inputs to  $E$ , who evaluates  $\tilde{\mathcal{P}}$  on  $\tilde{x}$  and learns the program output.

The security of such usage is *not implied* by our selective security game, so we need an updated game, where  $E$  chooses the input  $x$  after  $\tilde{\mathcal{P}}$  is sent over. This is the *adaptive security* game:

1.  $G$  garbles  $\mathcal{P}$  and sends  $\tilde{\mathcal{P}}$  to  $E$ .
2.  $E$  sends a cleartext input  $x$  to  $G$ .
3.  $G$  sends the garbled input  $\tilde{x}$  to  $E$ , as well as information  $d$  needed to decode the output.

Pushing transmission of  $\tilde{\mathcal{P}}$  to the offline phase requires that the GC scheme is secure in the context of this game.

Perhaps surprisingly, constructing garbling schemes that provably achieve this second notion is *notoriously difficult*. At a high level, this difficulty comes from the fact that  $E$  can base its input  $x$  on the garbled program itself. Proving this secure is a challenge, due to the nuanced nature of GC simulation.

*Cost accounting of adaptive GC schemes.* When constructing adaptively secure garbling schemes, we consider the cost of the offline and the online phases separately. The most important metrics are the offline and online communication costs. Ideally, offline phase communication should be as close as possible to the cost in the selective security setting. Thus, we ideally want a scheme whose offline communication is equal to the size of the underlying circuit representation, multiplied by the security parameter  $\lambda$ . In the online phase, we wish to pay online in terms of the number of input/output bits of the program.

Computation overhead is, of course, also important, and the computation used by both  $G$  and  $E$  should ideally be almost identical to their computation in the selective security setting.

*Summary of state-of-the-art adaptive GC.* The insecurity of standard GC (or, more precisely, the invalidity of existing GC selective-security proofs) when  $x$  may depend on  $\tilde{\mathcal{P}}$  has been observed relatively recently [BHR12a]. A number of solutions were proposed, which we review in detail in Section 1.3. Here, leading up to Section 1.1 we highlight two main approaches:

One, based only on one-way functions, requires high computational overhead (multiplicative factor  $O(w)$ , where  $w$  is the width of the evaluated circuit) both in online and offline phases [HJO<sup>+</sup>16]. This effectively negates the benefit of offline transmission in many settings.

The second is far more efficient, simply requiring to XOR the transmitted circuit with an output of a Random Oracle (RO), as described by [BHR12a]. This both requires modification to the selectively secure scheme, and it roughly doubles the computational cost. It would be far preferable to obtain adaptive security without increasing computation and, indeed, without modifying implementation. In addition the [BHR12a] proof requires that the simulator *program* the RO. This is a strong assumption, which cannot be met by any fixed function, and which is widely seen as much stronger than non-programmable RO.

No prior tri-state circuit constructions with adaptive security were previously proposed, although the above two approaches can undoubtedly be extended to TSCs – with their corresponding shortcomings.

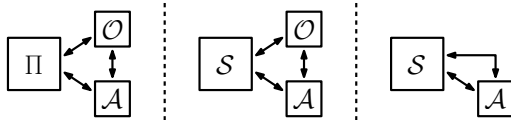
## 1.1 Our Contribution

Garbled circuit adaptive security is a well motivated and intensely studied problem. As we discuss in Section 1.2, the current state of the art offers to practitioners an unsatisfying menu of options when confronted with the need to use GC in the adaptive setting.

Many practical GC techniques are easily implemented and are selectively secure in the *non-programmable* random oracle (NPRO) model. As our main contribution, we provide a framework that allows to prove that such schemes are also *adaptively* secure, if they meet two conditions. These are satisfied by existing standard schemes, including Free XOR, half-gates, and even arithmetic techniques [BMR16]; see Appendix B. In short, our conditions require that (1) any RO queries issued while garbling the circuit are hidden by the resulting garbled circuit and (2) one can *resample* keys associated with a particular GC, such that the GC still evaluates correctly with these fresh keys – the scheme should be *rekeyable*. We highlight that these conditions are reasonably easy to prove, which would facilitate the adoption of our framework and indeed of the final protocols.

We apply our framework to the tri-state circuit model by (1) constructing a natural NPRO-based garbling scheme and (2) using our framework to prove this scheme achieves adaptive security. Thus, we obtain adaptively-secure garbling of both Boolean circuits and RAM programs in the NPRO model.

Our adaptively-secure TSC scheme matches the cost of state-of-the-art selectively secure schemes in terms of both communication and computation. Let  $C$  denote a TSC with input  $x$  and output  $y$ . Let  $\lambda$  be the security parameter,



**Fig. 1.** (Left) A real-world protocol using RO. The real-world adversary interacts directly with the RO. (Center) A simulation in the non-programmable RO model. Here, the simulator  $\mathcal{S}$  interacts with the RO independently of  $\mathcal{A}$ , so  $\mathcal{S}$  cannot respond to nor even learn  $\mathcal{A}$ 's RO queries. (Right) A simulation in the programmable RO model. Here,  $\mathcal{S}$  directly responds to  $\mathcal{A}$ 's oracle queries. See also [FLR<sup>+</sup>10].

which can be understood as the length of encryption keys (e.g. 128 bits). Our offline communication cost is  $\leq |C| \cdot \lambda$ , where the actual cost depends on the types of gates used. Our online communication cost is  $O((|x| + |y|) \cdot \lambda)$ , and is independent of the size of the program. In terms of computation, both  $G$  and  $E$  expend at most one call to the random oracle per tri-state gate.

When we compile Boolean gates to tri-state gates, our scheme's costs match the cost of the popular selectively-secure half-gates garbling scheme [ZRE15] in terms of both computation and communication. [ZRE15] is a state of the art Boolean scheme<sup>6</sup>, so our TSC-based Boolean handling matches what one would expect from more standard Boolean circuit garbling. In addition,  $T$  steps of a RAM program can be compiled to a tri-state circuit with  $O(T \cdot \log^3 T \cdot \log \log T)$  gates [HKO23].

## 1.2 Non-Programmable RO and Programmable RO

We discuss the relative strength of the PRO and NPRO assumptions and our motivation for seeking to weaken the assumption. We feel that prior to our work, the options for practical GC deployment were unsatisfying, and the system was brittle. Practitioners had three options:

*Option 1: Obey selective security, and perform all work in the online phase.* Even in settings where this is acceptable from the perspective of engineering/performance, delaying all work to the online phase is an undesirable constraint that is prone to be inadvertently violated. Indeed, envisioning larger systems incorporating MPC/GC, even implemented and maintained by a MPC-knowledgeable team, it is easy to foresee the temptation to modularize, optimize and multi-thread execution, separating GC generation/transmission from OT execution, eventually leading to order violation. We stress that such security errors are insidious and hard to find.

Relatedly, garbled circuit is a simple, powerful, and convenient object for standardization. There is already a preliminary effort by NIST to standardize threshold schemes, including more complex objects such as GC [MPT20,MPT23].

<sup>6</sup> The more recent [RR21] scheme uses less communication than [ZRE15], but it uses more computation.

Following discussion at the MPTS workshops, it seems impractical to standardize many possible combinations of GC and OT. Rather, GC, OT, and other related primitives are likely to be standardized separately. Clearly, a more robust, versatile, and resilient GC primitive would be much preferable for standardization than the more brittle one, subject to execution order constraints.

*Option 2: Assume a programmable random oracle (PRO); mask GCs with PRO.* Namely, use the PRO-based technique of [BHR12a]. This option roughly doubles the total computation cost, both for  $G$  and  $E$ , compared to selectively-secure GC, and to our solution.

Additionally, this assumes PRO. PRO is an unusually strong assumption, in that it clearly cannot be satisfied by any fixed function. PRO violates several impossibility results, e.g., enabling non-interactive non-committing encryption [Nie02] and adaptive GC with online phase independent of the program output size [BHR12a]. Both are impossible in the standard model [Nie02,AIKW13].

In contrast, NPRO is a milder assumption, which is widely used in practical cryptography. For instance, the standardized RSA-OAEP encryption scheme uses NPRO [BR95,FOPS01,FLR<sup>+</sup>10,MKJR16]. Notably, in the GC setting, the widely used Free-XOR key homomorphism relies on circular correlation robustness [CKKZ12], a slight weakening of the NPRO assumption.

NPRO and PRO are fundamentally different, and substantially stronger objections are raised against the use of PRO. For example, as pointed out in [CGH98], the NPRO assumption leaves open the possibility of seeking “reasonable notions of implementation” of RO, relative to which one can show the soundness of this methodology (at least, in some interesting cases). In particular, one could consider a more general notion of implementation, as a compiler that takes a scheme that works in the random oracle model and produces a scheme that works in the standard model (i.e., without a random oracle). Such line of work is ruled out in the PRO model.

*Option 3: Implement adaptive GC in the standard model.* While standard model adaptive GC remains of theoretical interest, the best known technique incurs multiplicative factor  $w$  overhead in terms of computation, where  $w$  is the width of the evaluated circuit. In many practical settings (e.g., laptops on the 1Gbps LAN), the speed of GC generation/evaluation is only about  $3\times$  faster than transmission. In this scenario, the online phase of the adaptively secure GC will be factor  $\approx w/3$  times *slower* than the entire selective GC evaluation.

We remark that improving the current state of the art in adaptive GC from just a PRF remains a challenge, and any results that improve the factor  $w$  overhead would be highly surprising.

### 1.3 Related Work

*Selective GC Security.* Even proofs of selective GC security are subtle. The classic proof of selective security from a PRF [LP09] proceeds by a hybrid argument where in each step we replace one real gate by a *simulated* gate. This simulated

gate is programmed such that it outputs a value consistent with real-world evaluation. Once each gate is replaced, the final distribution is statistically close to simulation, which does not depend on the real-world input  $x$ .

One subtle, but central, aspect of this simulation is that we must replace gates in a specific order. This is so that we can base the indistinguishability of each hybrid (and hence the proof) on the PRF assumption. Indeed, wire values (i.e., PRF keys) are used throughout the circuit, but PRF security only holds if the key had not been used elsewhere.

Jumping ahead, we remark that in the adaptive setting [LP09]’s intermediate simulated gates should output values that depend on the input  $x$ , but syntactically,  $x$  simply is not well defined at the time the simulated gate should be programmed. Prior works resolve this problem, but at significant cost.

*Adaptive Garbled Circuits.* [BHR12a] were the first to thoroughly investigate the problem of adaptive GC. They pointed out that existing GC schemes do not seem to admit a proof of adaptive security.

[BHR12a] also gave two constructions that *can* be proven adaptively secure. Their first construction should mostly be viewed as a proof of concept. It requires that  $G$  one-time pad the GC  $\tilde{C}$  before sending it to  $E$ . Then, in the online phase,  $G$  sends the one-time-pad mask to  $E$ , allowing  $E$  to decrypt the circuit and evaluate normally. In terms of online cost, this construction is poor, as it requires that  $G$  send a message proportional to the size of the garbled circuit.

This said, [BHR12a]’s one-time-pad-based construction does give important insight into *how* adaptivity can be achieved. In short, the one-time-pad mask allows  $G$  to *equivocate* the GC. Namely,  $G$  can unmask to  $E$  a (different) GC that *depends on  $E$ ’s choice of input  $x$* . This capability is, of course, not used in the real-world execution, but it *is* used by the simulator. Namely, in intermediate steps of the proof,  $G$  uses its ability to equivocate to open to  $E$  intermediate hybrid garbled circuits from the *selective* security proof [LP09]. In this way, the one-time-pad-based scheme admits a natural proof of adaptive security.

[BHR12a] also constructed a scheme that they proved secure by using programmable RO. In short, the simulator programs the RO to equivocate the GC, similar to the above. As already noted, this scheme circumvents a known lower bound on online communication cost of adaptively secure GC [AIKW13]. In particular, [AIKW13] showed that any standard model adaptive garbling scheme must have an online phase that scales at least with the size of the program’s output, but [BHR12a]’s RO construction only sends information proportional to the program’s *input*. In contrast, our simulator *does not* program the RO; in the NPRO model, we cannot circumvent the [AIKW13] lower bound.<sup>7</sup>

In concurrent work, [GYW<sup>+</sup>23] consider adaptive garbling of specific implementations of popular half-gates and three-halves schemes. They show that these

---

<sup>7</sup> If the decoding table is given in the online phase in the adaptive scheme from [BHR12a], then it is possible to show that their scheme is adaptively secure in the NPRO model. We emphasize that [BHR12a] still prescribes an RO mask on every ciphertext, which our analysis avoids.

schemes, instantiated with (previously designed) specific encryptions built on the random permutation model [GKW<sup>+</sup>20], achieve adaptive security. Their proof is focused on specific details of encryption based on fixed-key AES, and accordingly their proof details are intricate. In contrast, we aim to develop a generic approach or a framework that may be useful in large class of GC constructions. Our simulation methods are less customized and more general. In addition, our general framework applies to garbling of TSCs, and thus our results immediately imply an adaptively secure garbled RAM scheme in the NPRO model.

*Adaptive garbling from one-way functions* is a much harder task, so known solutions are substantially less efficient. [HJO<sup>+</sup>16] constructed an adaptive GC scheme with online cost sublinear in the circuit size and that assumes only the existence of one-way functions (OWFs). In particular, their online cost is  $O(w \cdot \text{poly}(\lambda))$ , where  $w$  denotes the *width* of the target circuit.

Much like [BHR12a]’s above one-time-pad-based scheme, [HJO<sup>+</sup>16]’s key idea is to allow  $G$  to equivocate the GC. To improve the equivocation, [HJO<sup>+</sup>16] defined and implemented a primitive called *somewhere equivocal encryption*. Somewhere equivocal encryption allows a sender to encrypt a long message, then later send a short key that decrypts the message, except that the sender can change the value of one secret position of the message. By encrypting a garbled circuit  $2w$  times with different somewhere equivocal keys,  $G$  can equivocate on up to  $2w$  gates.  $2w$  gates is sufficient, because [HJO<sup>+</sup>16] can equivocate two full *layers* of the circuit. Once the second layer is equivocated, the proof can remove equivocation from the first layer by changing the simulated gates to output 0. They then equivocate the next layer, and so on.

[HJO<sup>+</sup>16] also show a different order of equivocation that scales with the circuit depth  $d$ , but this strategy has exponential security loss in  $d$ .

While [HJO<sup>+</sup>16] is the state-of-the-art adaptive GC from OWFs, its online communication cost remains high and – far worse – the computational overhead imposed by the scheme is *multiplicative*. To evaluate each GC gate,  $E$  must in the online phase decrypt that gate  $O(w)$  times!

*Other Works in the Adaptive Setting.* [JW16] showed that Yao’s basic garbling scheme is adaptively secure for log-depth circuits. [JO20] pushed this result further, showing that classic GC techniques for reducing offline cost of each gate also work in the adaptive setting, making the total online cost closer to that of a state-of-art garbling scheme. These techniques only work for circuits in  $\text{NC}^1$ , which is very limiting. [KKPW21] showed that in the adaptive setting, Yao’s scheme *must* suffer exponential security loss wrt the depth of the circuit. Thus, it seems log-depth circuits is the best possible for adaptive Yao’s, unless the design is significantly changed.

[HJO<sup>+</sup>16,GOS18] demonstrate asymptotic improvement to adaptive GC in the standard model, but they are concretely expensive as they use both public key assumptions and non-black-box cryptography.

The recent work of [BBK<sup>+</sup>23] achieves adaptive security from OWFs for a weaker notion of simulation security called *distributional simulation security*.



The protocol in [BBK<sup>+</sup>23] is secure while only sending something which scales with the size of input in the online phase, circumventing the [AIKW13] lower-bound. Our work uses the more standard notion of adaptivity.

*Garbled RAM.* Garbled RAM [LO13] upgrades GC with the ability to handle RAM programs rather than circuits. In short, a Garbled RAM scheme allows the GC to perform oblivious random access to a large main memory where each access incurs amortized sublinear cost. Ideally, the per-access overhead should be at most polylogarithmic in the memory size.

Early GRAM schemes, e.g. [LO13,GHL<sup>+</sup>14,GLO15,GLOS15], demonstrated important feasibility results, but they were not concerned with polylog performance factors, so their constructions are expensive. More recent constructions [HKO22,PLS23,HKO23] target performance improvement, where the most recent result [HKO23] garbles only  $O(\log^3 n \cdot \log \log n)$  fan-in-two gates per access.

More interesting than [HKO23]’s cost is its formalism. [HKO23] shows that RAM computation can be emulated by a relatively simple circuit model called *tri-state circuits* (TSCs). To garble RAM, it suffices to garble tri-state gates.

Our result leverages the TSC model to achieve adaptively secure GRAM. We review the TSC model in Section 2.

## 2 Preliminaries

### 2.1 Notation

- $\lambda$  is a security parameter and can be understood as the length of GC labels.
- $x \approx y$  denotes that distributions  $x$  and  $y$  are indistinguishable to a PPT adversary.
- $x \approx_{\mu(\lambda)} y$  denotes that distributions  $x$  and  $y$  are indistinguishable, where the advantage of the adversary is bounded by  $\mu(\lambda)$ .
- $x \equiv y$  denotes that distributions  $x$  and  $y$  are identical.
- $\mathcal{O}$  denotes a (non-programmable) random oracle.
- We refer to wire id  $w$ . When clear from context, we will *overload*  $w$  to also mean the plaintext value on that wire.

### 2.2 Garbling Schemes

[BHR12b] defined the notion of a *garbling scheme*, which formalizes garbled circuit techniques as cryptographic primitives. We formalize our construction and our proof in the [BHR12b] framework.

**Definition 1 (Garbling Scheme [BHR12b]).** A *garbling scheme* for a class of circuits  $\mathcal{C}$  is a tuple of four procedures

$$(\text{Garble}, \text{Eval}, \text{Eval}, \text{Decode})$$

with the following interface:

- $\text{Garble}(1^\lambda, C) \rightarrow (\tilde{C}, e, d)$ : Garble a circuit  $C \in \mathcal{C}$ , producing garbled circuit  $\tilde{C}$ , input encoding string  $e$ , and output decoding string  $d$ .
- $\text{Encode}(e, x) \rightarrow \tilde{x}$ : Use the input encoding string  $e$  to encode input  $x$ .
- $\text{Eval}(\tilde{C}, \tilde{x}) \rightarrow \tilde{y}$ : Evaluate  $\tilde{C}$  on encoded input  $\tilde{x}$ , yielding encoded output  $\tilde{y}$ .
- $\text{Decode}(d, \tilde{y}) \rightarrow y$ : Use the output decoding string  $d$  to decode output  $\tilde{y}$ . If  $\tilde{y}$  is not a valid encoding, then Decode outputs  $\perp$ .

A correct garbling scheme implements the semantics of the circuit class  $\mathcal{C}$ :

**Definition 2 (Garbling Scheme Correctness [BHR12b]).** A garbling scheme is **correct** if for all circuits  $C \in \mathcal{C}$ , all inputs  $x$ , and for security parameter  $\lambda$ :

$$\text{Decode}(d, \text{Eval}(\tilde{C}, \text{Encode}(e, x))) = C(x) \quad \text{where } (\tilde{C}, e, d) \leftarrow \text{Garble}(1^\lambda, C)$$

Typically, garbling schemes are shown to satisfy selective notions called *obliviousness* and *privacy*. We consider *adaptive* variants of these; see next.

### 2.3 Definition of Adaptive Security

Our notion of adaptivity is based on definitions from [BHR12a] and [BHR12b]:

**Definition 3 (Adaptive Privacy).** A garbling scheme is **adaptively private** if for all circuits  $C$  computing a function  $f$  and for all stateful PPT adversaries  $\mathcal{A}$ , there exists a simulator  $\mathcal{S}$  such that the following quantity is negligible in  $\lambda$ :

$$\left| \Pr \left[ \text{Real}_{\text{priv}}^{\mathcal{A}, C}(1^\lambda) = 1 \right] - \Pr \left[ \text{Ideal}_{\text{priv}}^{\mathcal{A}, C}(1^\lambda) = 1 \right] \right|$$

where Real, Ideal are as follows:

$\text{Real}_{\text{priv}}^{\mathcal{A}, C}(1^\lambda)$	$\text{Ideal}_{\text{priv}}^{\mathcal{A}, C}(1^\lambda)$
1: $(\tilde{C}, e, d) \leftarrow \text{Garble}^\mathcal{O}(1^\lambda, C)$	1: $\tilde{C} \leftarrow \mathcal{S}^\mathcal{O}(1^\lambda, C)$
2: $x \leftarrow \mathcal{A}^\mathcal{O}(1^\lambda, \tilde{C})$	2: $x \leftarrow \mathcal{A}^\mathcal{O}(1^\lambda, \tilde{C})$
3: $\tilde{x} \leftarrow \text{Encode}(e, x)$	3: $(\tilde{x}, d) \leftarrow \mathcal{S}^\mathcal{O}(f(x))$
4: <b>return</b> $\mathcal{A}^\mathcal{O}(\tilde{x}, d)$	4: <b>return</b> $\mathcal{A}^\mathcal{O}(\tilde{x}, d)$

Adaptive privacy roughly states that  $\mathcal{A}$  cannot distinguish the real garbled circuit from a simulated one, even when it is allowed to adaptively choose its input, and even when it is given the string  $d$  that decodes the output.

The literature also considers an alternative notion to privacy which is called *obliviousness*. Obliviousness differs from privacy in that the adversary is not allowed access to the output. That is, obliviousness roughly states that the garbled circuit alone should leak no information:

**Definition 4 (Adaptive Obliviousness).** A garbling scheme satisfies **adaptive obliviousness** if for all circuits  $C \in \mathcal{C}$  and all stateful PPT adversaries  $\mathcal{A}$ , there exists a simulator  $\mathcal{S}$  such that the following quantity is negligible in  $\lambda$ :

$$\left| \Pr \left[ \text{Real}_{\text{obv}}^{\mathcal{A}, C}(1^\lambda) = 1 \right] - \Pr \left[ \text{Ideal}_{\text{obv}}^{\mathcal{A}, C}(1^\lambda) = 1 \right] \right|$$

where Real, Ideal are as follows:

$\text{Real}_{\text{obv}}^{\mathcal{A}, C}(1^\lambda)$	$\text{Ideal}_{\text{obv}}^{\mathcal{A}, C}(1^\lambda)$
1: $(\tilde{C}, e, d) \leftarrow \text{Garble}^\mathcal{O}(1^\lambda, C)$	1: $(\tilde{C}, \tilde{x}) \leftarrow \mathcal{S}^\mathcal{O}(1^\lambda, C)$
2: $x \leftarrow \mathcal{A}^\mathcal{O}(1^\lambda, \tilde{C})$	2: $x \leftarrow \mathcal{A}^\mathcal{O}(1^\lambda, \tilde{C})$
3: $\tilde{x} \leftarrow \text{Encode}(e, x)$	3:
4: <b>return</b> $\mathcal{A}^\mathcal{O}(\tilde{x})$	4: <b>return</b> $\mathcal{A}^\mathcal{O}(\tilde{x})$

Formally, adaptive obliviousness and adaptive privacy are incomparable; one does not imply the other. However, informally, in typical garbling schemes (including all considered in this work), privacy is proved as a consequence of obliviousness. Since privacy is the more relevant property for applications in secure multiparty computation, we leave our proofs of adaptive obliviousness – which are almost identical to those of adaptive privacy – to Appendix A.

## 2.4 Garbled RAM and Tri-State Circuits

We provide a framework for achieving adaptive security from NPRO. As part of this contribution, we construct a scheme that captures much of the recent advances in practical GC, and we prove this scheme’s security in our framework.

[HKO23] formalized a model of computation called *tri-state circuits* (TSCs). TSCs are interesting for garbling because there exists an efficient (polylog overhead) reduction from RAM programs to the TSC model. The TSC model is straightforward to garble, and hence TSCs lead to natural constructions of Garbled RAM [LO13]. Our presented garbling scheme handles TSCs. Thus, we review relevant definitions. All definitions in this section are adapted from [HKO23].

**Definition 5 (Tri-state Circuit).** A **tri-state circuit** (TSC) is a circuit allowing cycles (i.e., its graph need not be acyclic) with three gate types: XORs ( $\oplus$ ), buffers ( $/$ ), and joins ( $\bowtie$ ). Each wire carries a value in the set  $\{0, 1, \mathcal{Z}, \mathbf{X}\}$ . The semantics of each gate type are as follows:

$\oplus$	$\mathcal{Z}$ 0 1 $\mathbf{X}$	$/$	$\mathcal{Z}$ 0 1 $\mathbf{X}$	$\bowtie$	$\mathcal{Z}$ 0 1 $\mathbf{X}$
$\mathcal{Z}$	$\mathcal{Z}$ $\mathcal{Z}$ $\mathcal{Z}$ $\mathbf{X}$	$\mathcal{Z}$	$\mathcal{Z}$ $\mathcal{Z}$ $\mathcal{Z}$ $\mathbf{X}$	$\mathcal{Z}$	$\mathcal{Z}$ 0 1 $\mathbf{X}$
0	$\mathcal{Z}$ 0 1 $\mathbf{X}$	0	$\mathcal{Z}$ $\mathcal{Z}$ 0 $\mathbf{X}$	0	0 0 $\mathbf{X}$ $\mathbf{X}$
1	$\mathcal{Z}$ 1 0 $\mathbf{X}$	1	$\mathcal{Z}$ $\mathcal{Z}$ 1 $\mathbf{X}$	1	1 $\mathbf{X}$ 1 $\mathbf{X}$
$\mathbf{X}$	$\mathbf{X}$ $\mathbf{X}$ $\mathbf{X}$ $\mathbf{X}$	$\mathbf{X}$	$\mathbf{X}$ $\mathbf{X}$ $\mathbf{X}$ $\mathbf{X}$	$\mathbf{X}$	$\mathbf{X}$ $\mathbf{X}$ $\mathbf{X}$ $\mathbf{X}$

We assume each gate has some distinct gate ID  $gid$ . A TSC has  $n$  input wires and  $m$  output wires. TSCs may use a distinguished wire, named 1, which carries constant 1. Circuit execution on input  $x \in \{0,1\}^n$  proceeds as follows: (1) Store  $\mathcal{Z}$  on each non-input wire, (2) store  $x$  on the input wires, (3) repeatedly and arbitrarily choose some gate  $g$  and update  $g$ 's output wire according to  $g$ 's function and input wires. Once there remains no gate whose execution would change a wire, halt and output the state of the circuit output wires.

Buffer gates act as “switches”. Each buffer  $x/y$  has two inputs: a *control wire*  $y$  and a *data wire*  $x$ . If the control wire  $y$  holds one, then the buffer “closes”, connecting its data wire  $x$  to its output; if the control wire holds zero, the buffer remains open, and the output wire is unassigned. Join gates allow us to connect wires together. For instance, we can connect the output of two buffers such that the joined output wire takes the value of whichever buffer is closed.

The tri-state value  $\mathcal{Z}$  roughly denotes the idea “this wire does not have a value” and the value  $\mathbf{X}$  denotes “an error has occurred”. In this work we consider a natural restriction of TSCs which requires that (1) no errors occur and (2) every wire ultimately acquires a Boolean value:

**Definition 6 (Total Tri-State Circuit).** *A tri-state circuit  $C$  is **total** if on every input  $x$ , the following holds. Change the semantics of joins such that they are multidirectional. To execute gate  $z \leftarrow x \bowtie y$ , update the value of each wire  $x, y, z$  with joined value  $(x \bowtie y \bowtie z)$ .  $C$  is **total** if after completing circuit execution with these semantics, every circuit wire is assigned a Boolean value.*

The interesting capability of TSCs is that gates execute in data-dependent orders. This capability is precisely what enables efficient RAM emulation. To take advantage of this in the GC setting, we must inform the GC evaluator  $E$  of the order they should execute gates. [HKO23] show that to make this work, it suffices to reveal to  $E$  every buffer control wire.

Revealing control wires to  $E$  complicates simulation of  $E$ 's view. We must somehow argue that even though  $E$  learns all control wires, we can still simulate. [HKO23] solve this by extending TSC input to additionally include specially constructed randomness. The random part of the input can be used as masks on control bits. With the addition of masks and careful circuit design, particular tri-state circuits can then be shown to be *oblivious*, i.e. that the control wire values can be simulated. We garble oblivious TSCs, so we give the relevant definitions:

**Definition 7 (Randomized Tri-State Circuit).** *A **randomized tri-state circuit** is a pair consisting of a tri-state circuit  $C$  and a distribution of bit-strings  $D$ . The execution of a randomized tri-state circuit on input  $x$  is defined by randomly sampling a string  $r$  from  $D$ , then running  $C$  on  $x$  and  $r$ :*

$$(C, D)(x) = C(x; r) \quad \text{where } r \leftarrow_{\$} D$$

**Definition 8 (Controls).** *Let  $C$  be a tri-state circuit with input  $x \in \{0,1\}^n$ . The **controls of  $C$  on  $x$** , denoted  $\text{controls}(C, x) \in \{0,1\}^*$ , is the set of all buffer control wire values (each labeled by its gate ID) after executing  $C(x)$ .*

**Definition 9 (Oblivious tri-state circuit).** Let  $\{(C_i, D_i) : i \in \mathbb{N}\}$  denote a family of randomized tri-state circuits. The family is considered **oblivious** if for any two inputs  $x, y \in \{0, 1\}^n$  the following holds:

$$\{ \text{controls}(C_\sigma, (x; r)) \mid r \leftarrow \$ D_\sigma \} \stackrel{s}{\approx} \{ \text{controls}(C_\sigma, (y; r)) \mid r \leftarrow \$ D_\sigma \}$$

### 3 Technical Overview

This section explains our approach at a high level, providing sufficient detail for informal understanding. Sections 4 and 5 formalize the ideas explained here.

#### 3.1 Tri-State Circuit Construction

Our main contribution is a framework for proving adaptive security of garbling schemes in the NPRO model. To make this contribution concrete, we formalize a particularly useful garbling scheme, and we prove this scheme fits into our framework. Our scheme garbles circuits from the recently proposed tri-state circuit (TSC) model; see Section 2.4.

In short, a TSC includes three types of gates, and the gates execute in a data-dependent order. This data-dependent execution is powerful enough to support efficient emulation of RAM programs.

*Wire keys.* Our TSC handling starts with Free-XOR-based wire keys [KS08]. Namely, to garble a TSC  $C$ , the garbler  $G$  samples for each circuit wire  $w$  a length- $\lambda$  key  $\mathbf{k}_w^0$ . This key encodes a logical zero on wire  $w$ .  $G$  then samples a single length- $\lambda$  global correlation  $\Delta$ , and for each wire  $w$ ,  $G$  defines the encoding of logical one as follows:

$$\mathbf{k}_w^1 = \mathbf{k}_w^0 \oplus \Delta$$

Note that this means that if we overload the name of a wire with its runtime value, at runtime  $E$  holds the following key:

$$\mathbf{k}_w = \mathbf{k}_w^0 \oplus w \cdot \Delta$$

Recall that TSCs also allow wires to hold a distinguished value  $\mathcal{Z}$ . We encode  $\mathcal{Z}$  by  $E$  holding *no key at all*.

*Gate handling.* The function of each TSC gate-type was explained in Section 2.4. Roughly, our garbling of gates is as follows:

- **XOR gates** are handled simply by XORing the input labels. This is the Free XOR optimization [KS08].
- **Buffers** of the form  $z \leftarrow x/y$  have two inputs: a *control* wire  $y$  and a *data* wire  $x$ .  $G$  defines the key for the buffer’s output wire as follows:

$$\mathbf{k}_z^0 = \mathbf{k}_x^0 \oplus \mathcal{O}(\mathbf{k}_y^1, \text{gid})$$

Here *gid* is a nonce. This use of RO ensures that  $E$  can compute a key for the output wire iff the control wire  $y$  holds logical one.

- **Joins** of the form  $z \leftarrow x \bowtie y$  connect together the inputs  $x$  and  $y$  such that if either wire is non- $\mathcal{Z}$ , the output wire acquires the non- $\mathcal{Z}$  input value.  $G$  handles joins by simply setting the output key as  $k_z^0 = k_x^0$ . This trivially enables  $E$  to translate an  $x$  key to a  $z$  key. To allow  $E$  to translate a  $y$  key to a  $z$  key,  $G$  includes in the GC the particular string  $k_x \oplus k_y$ .  $E$  XORs this difference with its  $y$  key to obtain an appropriate  $z$  key.

We refer the reader to Section 5 for further details on our TSC handling.

### 3.2 Proof of Security

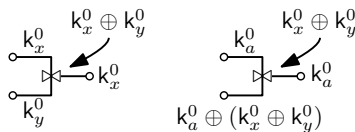
Our main contribution shows that typical GC schemes built from a random oracle are also *adaptively secure* in the NPRO model. For example, our above basic TSC scheme is adaptively secure with no change in implementation. Our framework for proving RO-based schemes adaptively secure requires two properties of the GC scheme: (1) the scheme should be *rekeyable* and (2) the scheme should be *query hiding*. In the following, we explain these properties in the context of our TSC scheme.

*On our (non-) use of programmability.* Our security proof observes that programmability is not needed for our construction. A main observation of our proof is that it *is okay* to program the RO in intermediate hybrids of our proof, so long as the real-world and ideal-world games do not program the oracle. Our real-world and ideal-world handling of GC uses a RO as a uniformly sampled function, and no programming is required. Hence, our construction is secure in the NPRO model. In other words, our definition of security is formalized with respect to an RO that cannot be programmed. *This alone* defines the security properties of our scheme; the method by which we prove real-ideal indistinguishability is irrelevant.

*Rekeying a Garbled Circuit.* In a typical GC proof in the *selective* setting, we would use a hybrid argument to rewrite parts of the GC to “hard-code” its behavior, forcing GC gates to output keys consistent with evaluation under the circuit input  $x$ ; see e.g. [LP09]. In the *adaptive* setting, such a hybrid argument is impossible: at the time the adversary receives the GC, the input  $x$  is not defined.

Our proof of adaptive security observes that while we cannot use a hybrid argument to change the GC, we *can* change the *keys* associated with the GC. Consider garbled circuit  $\tilde{C}$ , and let  $K$  be the collection of wire keys chosen by  $G$  while garbling  $\tilde{C}$ . In our TSC construction – and in many RO-based GC schemes – it is possible to choose a fresh collection of keys  $K'$  that are independent of  $K$  and that preserve circuit semantics when executed with garbling  $\tilde{C}$ . We call this process of replacing GC keys a *rekeying* of the GC.

For an example of rekeying, consider the following TSC join gate:



On the left, we depict a join as garbled by  $G$ , where we label the input wires with  $G$ 's keys  $k_x^0, k_y^0$ ; the output wire is similarly labelled by  $k_x^0$ . To enable evaluation in all cases,  $G$  includes in  $\tilde{C}$  the specific string  $k_x^0 \oplus k_y^0$ . If  $E$  only holds a runtime key  $k_y$  for the bottom input wire, it can use this string to *translate* that input key to an appropriate output key:

$$(k_y^0 \oplus y \cdot \Delta) \oplus (k_x^0 \oplus k_y^0) = k_x^0 \oplus y \cdot \Delta$$

We start rekeying this gate by replacing its top input key  $k_x^0$  with some freshly sampled key  $k_a^0$ . Similarly, we replace the output wire key by  $k_a^0$ . To complete the rekeying, we must ensure the semantics of the gate are preserved. Namely, if  $E$  obtains some key on the bottom wire, it should be able to *translate* that input key to an appropriate output key. To ensure this works, we rekey the bottom wire as well, replacing  $k_y^0$  by a key that is specifically chosen to preserve semantics:  $k_a^0 \oplus (k_x^0 \oplus k_y^0)$ . Thus, if  $E$  obtains a key on the bottom input wire  $y$ , it can use the GC string to appropriately translate to an output key:

$$((k_a^0 \oplus (k_x^0 \oplus k_y^0)) \oplus y \cdot \Delta) \oplus (k_x^0 \oplus k_y^0) = k_a^0 \oplus y \cdot \Delta$$

We can prove that this rekeying of the gate is indistinguishable from the original keying of the gate, and by extending this strategy we can replace *all* GC keys.

The benefit of rekeying is that it allows us to define security properties for *specific*, i.e. fixed, garbled circuits. More precisely, the standard definitions of selective GC security [BHR12b] consider distributions of garbled circuits. For instance, the standard definition of GC obliviousness roughly states that a randomly garbled circuit should be indistinguishable from the output of a simulator. Our notion of rekeying allows us to instead *quantify* over garbled circuits. For instance, we can state that for a particular garbled circuit, the rekeying of *that* garbled circuit should be indistinguishable from some other distribution. This shift from probabilistically-defined GCs to universally quantified GCs is critical, because it allows us to make meaningful claims about the adversary's ability to distinguish in the adaptive game's *online* phase, when the garbled circuit is, indeed, fixed.

*Query hiding.* Recall that we use a random oracle  $\mathcal{O}$  to construct a garbled circuit  $\tilde{C}$ . Then, in the adaptive security game's *offline* phase, the adversary obtains  $\tilde{C}$  and is allowed access to the same random oracle  $\mathcal{O}$ . One concern is that the adversary might *guess* a RO query that intersects with the queries issued when garbling. If this happens, then the adversary might, for instance, be able to partially decrypt  $\tilde{C}$ , and use this side information to choose an input  $x$  that in the later *online* phase helps it distinguish real from ideal.

To show security, we therefore must show that such a guess is unlikely, in the sense that the garbled circuit  $\tilde{C}$  does not “help” the adversary to guess problematic queries. More precisely, we formalize a property – query hiding – whereby the adversary should not be able to detect when we *remove* from the random oracle all queries issued by Garble; see Section 4. At the very highest

level, we show that our TSC garbling scheme is query-hiding due to the fact that all keys are uniformly chosen, and not included in the GC itself. By plugging together the ability to rekey the garbled circuit with query hiding, we are able to obtain a proof of adaptive security.

*Intuition underlying the proof.* In short, our proof combines query hiding and rekeying to show security. By applying query hiding, we show that the adversary’s chosen input  $x$  cannot depend on the content of the random oracle. However, it still might depend on the garbled circuit itself. From here, rekeyability explicitly decouples the garbled circuit from its wire keys as well as the content of the RO, allowing us to argue that we can replace the adversary’s chosen input  $x$  by, say, the all zeros string, without the adversary noticing. This ultimately leads to a complete proof of security.

## 4 Adaptive Security of Rekeyable Garbling Schemes

This section formalizes the notions of **query-hiding garbling schemes** and **rekeyable garbling schemes**. Then, we show schemes satisfying these notions are adaptively secure. In Section 5, we will see that our garbling of tri-state circuits is query-hiding and rekeyable, and in Appendix B, we discuss other garbling schemes that satisfy our notions. Namely, we discuss Free XOR [KS08], half-gates [ZRE15], and garbled arithmetic gadgets [BMR16].

### 4.1 Additional Notation

Recall that in adaptive GC we split evaluation into an offline and an online phase.  $\mathcal{A}$  has access to the RO even in the offline phase, when it has seen the circuit garbling, but before we send the encoded input. As part of our security hybrids, we show that the adversary cannot detect if we remove oracle queries used to garble the circuit. This is useful, because it allows us to reason that the adversary cannot use work it performs in the offline phase to help it distinguish in the online phase. Formalizing this will require us to change the programming of an RO, so we define appropriate notation.

**Notation 1** (Explicitly Programmed Random Oracle). *Let  $\mathcal{O}$  be a random oracle outputting strings in  $\{0, 1\}^\lambda$ , and let  $\delta$  be a partial map from oracle queries to oracle responses. The **explicitly programmed oracle**  $\mathcal{O}^\delta$  is defined as follows:*

$$\mathcal{O}^\delta(x) = \begin{cases} r & \text{if } (x, r) \in \delta \\ \mathcal{O}(x) & \text{otherwise} \end{cases}$$

We also introduce a shorthand for simply sanitizing certain oracle locations.

**Notation 2** (Sanitized Oracle). *Let  $\mathcal{O}$  be a random oracle outputting strings in  $\{0, 1\}^\lambda$ , and let  $\delta$  be a partial map from oracle queries to oracle responses. The*



**sanitized oracle**  $\mathcal{O}^{-\delta}$  is defined as follows:

$$\mathcal{O}^{-\delta} = \begin{cases} \mathcal{O}'(x) & \text{if } \exists r \text{ s.t. } (x, r) \in \delta \\ \mathcal{O}(x) & \text{otherwise} \end{cases}$$

for second sampled random oracle  $\mathcal{O}'$ . Informally,  $\mathcal{O}^{-\delta}$  replaces the outputs from queries in  $\delta$  with fresh uniform values.

It will also be convenient to extract from the formal procedure `Garble` its oracle queries:

**Notation 3** (Garbler Queries). For an application  $\text{Garble}^{\mathcal{O}}(1^\lambda, C)$ , let  $\delta$  denote the partial map from oracle queries issued by `Garble` to corresponding oracle responses. We sometimes extend the output of `Garble` with this partial map:

$$\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$$

Our security properties of rekeyable garbling schemes rely on the ability to rekey any fixed garbled circuit  $\tilde{C}$  (we universally quantify over all GCs). To properly formalize such notions, we will need the ability to talk about the set of all possible garbled circuits from a particular garbling scheme:

**Notation 4** (Garble Support). Let  $C \in \mathcal{C}$  be a circuit. Let  $\text{Supp}(\text{Garble}(1^\lambda, C))$  denote the **support** of `Garble` on input  $C$ . Formally,  $\text{Supp}(\text{Garble}(1^\lambda, C))$  is the support of the distribution defined by the following procedure:

$\mathcal{G}(\lambda, C)$
<hr style="border: 0; border-top: 1px solid black; margin: 0;"/>
1: Sample random oracle $\mathcal{O}$
2: $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
3: <b>return</b> $\tilde{C}$

## 4.2 Properties on Garbling Schemes

This section formalizes important definitions and properties we require on a garbling scheme to show adaptive privacy.

We start with *query hiding*, which roughly states that a garbled circuit  $\tilde{C}$  does not leak the oracle queries that were used to construct it:

**Definition 10 (Query Hiding).** Let  $\Pi$  be a garbling scheme. We say that  $\Pi$  is **query hiding** if for all adversaries  $\mathcal{A}$  and all circuits  $C$ , the following ensembles are statistically close (in  $\lambda$ ):

$\text{QueryHiding}^{\mathcal{A}, C}(\lambda)$	$\text{QueryHiding}^{\mathcal{A}, C}(\lambda)$
<hr style="border: 0; border-top: 1px solid black; margin: 0;"/>	<hr style="border: 0; border-top: 1px solid black; margin: 0;"/>
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}(1^\lambda, C)$	2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}(1^\lambda, C)$
3: <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3: <b>return</b> $\mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$

That is, by simply observing  $\tilde{C}$ , the adversary cannot tell if it is interacting with an oracle containing query/response pairs as used by Garble, or if it is interacting with an oracle that is completely independent of  $\tilde{C}$ .

Next, we define *rekeyability* of a garbling scheme. Roughly, a garbling scheme is rekeyable if there is a way to sample fresh keys that are consistent with  $\tilde{C}$ :

**Definition 11 (Rekeyability).** *Let  $\Pi$  be a garbling scheme.  $\Pi$  is **rekeyable** if the following holds. There must exist a poly-time<sup>8</sup> procedure *Rekey* with the following form:*

$$(\delta, e, d) \leftarrow \text{Rekey}(\lambda, C, \tilde{C}),$$

That is, on input a circuit  $C$  and a garbled circuit  $\tilde{C}$ , *Rekey* outputs a query map  $\delta$ , an input encoding string  $e$ , and an output decoding string  $d$ . *Rekey* must be such that for all circuits  $C$ , the ensembles described by the following experiments are identical:

<i>Rekeyable</i> <sup>R</sup> ( $1^\lambda, C$ )	<i>Rekeyable</i> <sup>I</sup> ( $1^\lambda, C$ )
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$	2: $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
3:	3: $(\delta', e', d') \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$
4: <b>return</b> $(\delta, \tilde{C}, e, d)$	4: <b>return</b> $(\delta', \tilde{C}, e', d')$

Rekeyability alone is not sufficient to prove adaptive security, and even selective security combined with rekeyability seems insufficient to achieve adaptive security. Intuitively, it is not clear how to obtain rekey privacy (see next), which is stated wrt a fixed GC, from GC privacy, which is probabilistic over sampled GCs. We accordingly define the notion of a garbling scheme that *privately rekeys*:

**Definition 12 (Privately Rekeying).** *Let  $\Pi$  be a rekeyable garbling scheme (Definition 11). We say that  $\Pi$  **privately rekeys** if for all circuits  $C$  computing some function  $f$ , there exists a simulator  $\mathcal{S}$  s.t. for all  $x$ , for all garbled circuits  $\tilde{C} \in \text{Supp}(\text{Garble}(1^\lambda, C))$ , and for all PPT adversaries  $\mathcal{A}$ , the following ensembles are statistically close in  $\lambda$ :*

<i>Real</i> <sup>A,C,<math>\tilde{C},x</math></sup> ( $\lambda$ )	<i>Ideal</i> <sup>A,S,C,<math>\tilde{C},x</math></sup> ( $\lambda$ )
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $(\delta, e, d) \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$	2: $(\delta, e, d) \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$
3: $\tilde{x} \leftarrow \text{Encode}(e, x)$	3: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
4:	4: $d' \leftarrow \mathcal{S}(1^\lambda, f(x), d)$
5: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d)$	5: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d')$

<sup>8</sup> For our work we implement the rekey procedure since our proofs rely on properties of a candidate construction; however, it is not technically necessary that *Rekey* is efficient. In fact, an inefficient *Rekey* procedure should always exist, since reverse sampling any distribution is always well-formed.

Roughly speaking, the above privacy simulator  $\mathcal{S}$  forges an output decoding table  $d'$  such that the garbled circuit correctly evaluates to the expected result  $f(x)$ , even though the input is an encoding of 0.

### 4.3 Adaptive Security

We now prove our main theorem, which connects our notions of query hiding and rekeyability with adaptive security:

**Theorem 1 (Adaptive Privacy from Private Rekeying and Query Hiding).** *Let  $\Pi$  be a privately rekeyable (Definition 12) garbling scheme that is query hiding (Definition 10).  $\Pi$  is adaptively private.*

*Proof.* By construction of a simulator:

$\mathcal{S}^{\mathcal{O}}(1^\lambda, C)$
1: $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
2: <b>return</b> $\tilde{C}$
3: // Second call; receive $f(x)$ from caller
4: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5: $d' \leftarrow \mathcal{S}_p(1^\lambda, y, d)$
6: <b>return</b> $(\tilde{x}, d')$

Here,  $\mathcal{S}_p$  is the privacy simulator  $\mathcal{S}$  provided by Definition 12. In the offline phase,  $\mathcal{S}$  simply garbles a circuit normally. In the online phase,  $\mathcal{S}$  (1) encodes the all-zeros input and (2) uses  $\Pi$ 's privacy simulator  $\mathcal{S}_p$  to forge an output decoding string  $d$  that convincingly decodes to the correct output  $f(x)$ .

Now, we show that the real-world and ideal-world experiments are indistinguishable. We restate the experiments, in-lining the handling of our simulator:

$\text{Real}_{\text{priv}}^{\mathcal{A}, C}(\lambda)$	$\text{Ideal}_{\text{priv}}^{\mathcal{A}, C}(\lambda)$
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$	2: $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
4: $\tilde{x} \leftarrow \text{Encode}(e, x)$	4: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5:	5: $d' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d)$
6: <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d)$	6: <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d')$

To demonstrate indistinguishability, we proceed by a hybrid argument. We give six hybrids:

$F_{\text{priv},R}^{\mathcal{A},C}(\lambda)$	$F_{\text{priv},S}^{\mathcal{A},C}(\lambda)$
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$	2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
3: $x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$	3: $x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$
4: $\tilde{x} \leftarrow \text{Encode}(e, x)$	4: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5:	5: $d' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d)$
6: <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d)$	6: <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d')$

$G_{\text{priv},R}^{\mathcal{A},C}(\lambda)$	$G_{\text{priv},S}^{\mathcal{A},C}(\lambda)$
1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$	1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$
2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$	2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$
3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
4: $\tilde{x} \leftarrow \text{Encode}(e, x)$	4: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5:	5: $d' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d)$
6: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d)$	6: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d')$

$H_{\text{priv},R}^{\mathcal{A},C}(\lambda)$	$H_{\text{priv},S}^{\mathcal{A},C}(\lambda)$
1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$	1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$
2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$	2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$
3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
4: $(\delta', e', d') \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$	4: $(\delta', e', d') \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$
5: $\tilde{x} \leftarrow \text{Encode}(e', x)$	5: $\tilde{x} \leftarrow \text{Encode}(e', \mathbf{0})$
6:	6: $d'' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d')$
7: <b>return</b> $\mathcal{A}^{\mathcal{O}^{\delta'}}(1^\lambda, \tilde{x}, d')$	7: <b>return</b> $\mathcal{A}^{\mathcal{O}^{\delta'}}(1^\lambda, \tilde{x}, d'')$

We find it most direct to argue indistinguishability by “meeting in the middle”. Namely, our proof proceeds starting from the real/ideal ensemble, and at each step of our proof, we show the current real/ideal ensemble is indistinguishable from some respective intermediate ensemble. To conclude, we show that these two final ensembles are indistinguishable from one another.

*Removing offline oracle queries.* In our crucial proof step, we argue the following:

$$\{\text{Real}_{\text{priv}}^{\mathcal{A},C}(\lambda)\} \approx \{F_{\text{priv},R}^{\mathcal{A},C}(\lambda)\} \quad \text{and} \quad \{\text{Ideal}_{\text{priv}}^{\mathcal{A},C}(\lambda)\} \approx \{F_{\text{priv},S}^{\mathcal{A},C}(\lambda)\}$$

In this step we remove from the adversary’s oracle all queries issued by the call to `Garble`, but only in the offline phase. Jumping ahead, our informal objective

is to show that the adversary’s chosen input  $x$  must be independent of Garble’s random oracle queries.

Of course, there is a difference between games Real and  $G_{\text{priv},R}$  (resp. Ideal and  $G_{\text{priv},I}$ ). Indeed, in the former game the adversary is given access to the same oracle twice, and in the latter the adversary is given access to two oracles that differ by  $\delta$ . Thus, if the adversary can guess a query in  $\delta$ , it can distinguish the two games, since in the first game it will see the oracle respond consistently in the offline and online phases, and in the second game it will see the oracle “disagree with itself”.

Query hiding (Definition 10) is precisely what we need to show that the adversary cannot guess a query in  $\delta$ . Indeed, an unbounded adversary with only polynomial oracle queries has at most  $\text{negl}(\lambda)$  chance to sample a point in  $\delta$ . As such, the adversary only has a  $\text{negl}(\lambda)$  probability of determining if it was given  $\mathcal{O}$  or  $\mathcal{O}^{-\delta}$  on line 3 of both the Real games and G.

*Rearranging the oracles.* In our second step, we perform a “refactoring” of the random oracle queries. In particular, we argue:

$$\{F_{\text{priv},R}^{A,C}(\lambda)\} \equiv \{G_{\text{priv},R}^{A,C}(\lambda)\} \quad \text{and} \quad \{F_{\text{priv},S}^{A,C}(\lambda)\} \equiv \{G_{\text{priv},S}^{A,C}(\lambda)\}$$

Indeed, games  $G$  and  $H$  are identically distributed, as they are merely a rearrangement of the random oracle queries. The output distribution of  $\text{Garble}^{\mathcal{O}}$  in line 2 of hybrid G is independent of the programmed oracle  $\mathcal{O}^{-\delta}$  on line 2. As such, we can rewrite lines 2 and 3 to use different oracles all together to get hybrid H. On line 6, we note that the only part of  $\mathcal{O}'$  that  $\text{Garble}^{\mathcal{O}'}$  depends on are exactly those queries in  $\delta$ . As such, the garbler may as well have used  $\mathcal{O}^{\delta}$  to garble. After this step, it is clear that the adversary’s chosen input  $x$  must be independent of the random oracle  $\mathcal{O}'$  used to garble, since the adversary is not allowed access to  $\mathcal{O}'$ .

*Rekeying the garbled circuit.* Next, we use the rekeyability (Definition 11) of the scheme to sample fresh keys associated with the particular garbled circuit  $\tilde{C}$ . In particular, we argue:

$$\{G_{\text{priv},R}^{A,C}(\lambda)\} \equiv \{H_{\text{priv},R}^{A,C}(\lambda)\} \quad \text{and} \quad \{G_{\text{priv},S}^{A,C}(\lambda)\} \equiv \{H_{\text{priv},S}^{A,C}(\lambda)\}$$

This is immediate by the rekeyability property.

*Privacy.* Finally, we bridge from the “real world”, where we encode  $x$ , to the “ideal world”, where we encode 0:

$$\{H_{\text{priv},R}^{A,C}(\lambda)\} \equiv \{H_{\text{priv},S}^{A,C}(\lambda)\}$$

First, notice that the outputs of garbling –  $\tilde{C}$ ,  $\delta$ ,  $e$ , and  $d$  – are independent of  $\mathcal{O}$ . Thus, the choice of  $x$  is also independent of  $\delta$ ,  $e$ , and  $d$ , except insofar as they are constrained by the garbled circuit  $\tilde{C}$ . We can capture this sentiment by treating  $\tilde{C}$  and  $x$  as if they are universally quantified. But this setting is exactly

what is considered in private rekeying (Definition 12). Applying private rekeying thus discharges the proof.

Any query hiding, privately rekeyable garbling scheme is adaptively private.  $\square$

## 5 Our Adaptively-Secure Garbling of Tri-State Circuits

This section formalizes our handling of tri-state circuits. Recall, we formalize our approach as a *garbling scheme* (Definition 1). Section 4 proves this scheme adaptively secure.

**Construction 1** (Garbled TSCs from NPRO). *Our garbling scheme is the collection of algorithms*

(*Garble, Encode, Eval, Decode*)

*formalized in Figure 2 and described in the remainder of this section.*

In short, our approach is arguably the natural garbling of TSCs [HKO23]. Our main contribution is proving that this natural scheme is adaptively secure.

*Wire keys.* Our scheme uses Free-XOR-style GC keys [KS08]. In non-Free-XOR-based garbling, each wire is assigned *two* keys. Free XOR instead assigns only *one* key, and then chooses a single global *offset*  $\Delta$ :

$$\Delta \leftarrow_{\$} \{0, 1\}^{\lambda-1}$$

$\Delta$  is chosen by the garbler  $G$  and is hidden from the evaluator  $E$ .

For each circuit wire  $w$ , the  $G$  maintains a key  $k_w^0 \in \{0, 1\}^\lambda$ . This key encodes logical zero. The encoding of logical one is *defined* as  $k_w^1 = k_w^0 \oplus \Delta$ . The advantage of this encoding is that XOR gates can be garbled without any corresponding garbled gate: XORs are “free”. Keys on input wires are sampled uniformly; all other keys are derived from the input keys (and calls to the RO).

The crucial invariant of GC evaluation is that for each wire  $w$ , the evaluator will hold only *one* key. In general, the evaluator cannot distinguish the zero-key  $k_w^0$  from the one-key  $k_w^1$ , forming the basis of GC security. We write  $k_w$  to mean a key held by  $E$  corresponding to the logical value on wire  $w$ ;  $k_w$  could be either  $k_w^0$  or  $k_w^1$ . If we overload  $w$  as both the name of the wire and the Boolean value on that wire, the following holds:

$$k_w = k_w^0 \oplus w \cdot \Delta$$

Recall that in a TSC, wires can carry Boolean value, or they can carry value  $\mathcal{Z}$  or  $\mathcal{X}$ . Following [HKO23],  $\mathcal{Z}$  is encoded by  $E$ 's *lack of a key*.  $\mathcal{X}$  denotes that some error occurred in the circuit, and our construction only supports circuits that are free of errors (Definition 6). Hence, we do not need to encode  $\mathcal{X}$ .

<div style="border: 1px solid black; padding: 5px;"> <p>Garble<sup>O</sup>(1<sup>λ</sup>, (C, D))</p> <hr/> <pre> 1 : <math>\tilde{C} \leftarrow \text{emptymap}</math> 2 : <math>e, d \leftarrow \text{emptyvec}</math> 3 : <math>\Delta \leftarrow_{\\$} \{0, 1\}^{\lambda-1} \parallel 1</math> 4 : <math>r \leftarrow_{\\$} D</math> 5 : <b>for</b> each input wire <math>w</math> <b>do</b> 6 :   <math>k_w^0 \leftarrow_{\\$} \{0, 1\}^\lambda</math> 7 :   append <math>(k_w^0, k_w^0 \oplus \Delta)</math> to <math>e</math> 8 : <b>for</b> each <math>i</math>-th random input <math>w</math> <b>do</b> 9 :   <math>k_w^0 \leftarrow r_i \cdot \Delta</math> 10 : <b>for</b> <math>(g, gid) \in C</math> <b>do</b> 11 :   <b>if</b> <math>g = (z := x/y)</math> <b>do</b> 12 :     <math>\tilde{C}[gid] \leftarrow \text{lsb}(k_y^0)</math> 13 :     <math>k_z^0 \leftarrow \mathcal{O}(k_y^0 \oplus \Delta, gid) \oplus k_x^0</math> 14 :   <b>elseif</b> <math>g = (z := x \boxtimes y)</math> <b>do</b> 15 :     <math>\tilde{C}[gid] \leftarrow k_x^0 \oplus k_y^0</math> 16 :     <math>k_z^0 \leftarrow k_x^0</math> 17 :   <b>elseif</b> <math>g = (z := x \oplus y)</math> <b>do</b> 18 :     <math>k_z^0 \leftarrow k_x^0 \oplus k_y^0</math> 19 : <b>for</b> each <math>i</math>-th output wire <math>w</math> <b>do</b> 20 :   append to <math>d</math> 21 :   <math>(\text{lsb}(k_w^0), \mathcal{O}(k_w^0, i), \mathcal{O}(k_w^1, i))</math> 22 : <b>return</b> <math>(\tilde{C}, e, d)</math> </pre> </div>	<div style="border: 1px solid black; padding: 5px;"> <p>Eval<sup>O</sup>((C, D), <math>\tilde{C}, \tilde{x}</math>)</p> <hr/> <pre> 1 : <b>for</b> each nonrandom input wire <math>w</math> <b>do</b> 2 :   <math>k_w \leftarrow \tilde{x}[w]</math> 3 : <b>for</b> each random input wire <math>w</math> <b>do</b> 4 :   <math>k_w \leftarrow 0^\lambda</math> 5 : <b>for</b> <math>(g, gid) \in C</math> where <math>g</math> is ready <b>do</b> 6 :   <b>if</b> <math>g = (z := x/y)</math> <b>do</b> 7 :     <math>ctrl \leftarrow \text{lsb}(k_y) \oplus \tilde{C}[gid]</math> 8 :     <b>if</b> <math>ctrl = 1</math> <b>do</b> 9 :       <math>k_z \leftarrow \mathcal{O}(k_y, gid) \oplus k_x</math> 10 :    <b>elseif</b> <math>g = (z := x \boxtimes y)</math> <b>do</b> 11 :      <b>if</b> <math>x</math> is set <b>do</b> <math>k_z \leftarrow k_x</math> 12 :      <b>else</b> <math>k_z \leftarrow k_y \oplus \tilde{C}[gid]</math> 13 :    <b>elseif</b> <math>g = (z := x \oplus y)</math> <b>do</b> 14 :      <math>k_z \leftarrow k_x \oplus k_y</math> 15 : <math>\tilde{y} \leftarrow \text{emptyvec}</math> 16 : <b>for</b> each output wire <math>w</math> <b>do</b> 17 :   append <math>k_w</math> to <math>\tilde{y}</math> 18 : <b>return</b> <math>\tilde{y}</math> </pre> </div>
<div style="border: 1px solid black; padding: 5px;"> <p>Encode(<math>e, x</math>)</p> <hr/> <pre> 1 : <math>\tilde{x} \leftarrow \text{emptyvec}</math> 2 : <b>for</b> each <math>i</math>-th input <math>x[i]</math> 3 :   <math>(k^0, k^1) \leftarrow e[i]</math> 4 :   append <math>k^{x[i]}</math> to <math>\tilde{x}</math> 5 : <b>return</b> <math>\tilde{x}</math> </pre> </div>	<div style="border: 1px solid black; padding: 5px;"> <p>Decode(<math>d, \tilde{y}</math>)</p> <hr/> <pre> 1 : <math>y \leftarrow \text{emptyvec}</math> 2 : <b>for</b> each <math>i</math>-th output label <math>\tilde{y}[i]</math> 3 :   <math>(p, k^0, k^1) \leftarrow d[i]</math> 4 :   <b>if</b> <math>\mathcal{O}(\tilde{y}[i], i) = k^0 \wedge \text{lsb}(\tilde{y}[i]) = p</math> <b>do</b> 5 :     append 0 to <math>y</math> 6 :   <b>elseif</b> <math>\mathcal{O}(\tilde{y}[i], i) = k^1</math> <b>do</b> 7 :     append 1 to <math>y</math> 8 :   <b>else</b> 9 :     <b>return</b> <math>\perp</math> 10 : <b>return</b> <math>y</math> </pre> </div>

**Fig. 2.** Our NPRG-based garbling scheme for oblivious tri-state circuits  $(C, D)$ . Our scheme transmits one ciphertext per join gate and one bit per buffer; XOR gates are “free” [KS08].  $E$  evaluates gates in a data dependent order, choosing gates that are *ready* (Definition 13). Our scheme is adaptively secure.

*Point and Permute.* Our global offset  $\Delta$  has least significant bit (lsb) 1. This ensures that for each wire  $w$ , the lsb of the zero-key  $k_w^0$  and of the one-key  $k_w^0 \oplus \Delta$  are *different*. This is useful because it allows us to view the lsb of the zero-key  $k_w^0$  and the lsb of the active key  $k_w$  as an *XOR secret share* of  $w$ 's semantic value. This arrangement is a classic trick called point and permute [BMR90].

Point and permute makes it simple for  $G$  to reveal particular wire values to  $E$ . To reveal the value of some wire  $w$ ,  $G$  simply attaches the bit  $\text{lsb}(k_w^0)$  to the GC. At runtime,  $E$  computes the lsb of its key  $\text{lsb}(k_w^0 \oplus w \cdot \Delta)$ , XORs the result with  $\text{lsb}(k_w^0)$ , and by construction obtains  $w$ .

Revealing particular wire values is central to the handling of TSCs. Specifically, we reveal to  $E$  the value of each buffer's control wire. Note when the considered TSC is *oblivious* (Definition 9), it is safe to reveal these values: the information  $E$  learns can be simulated.

*Distribution sampling.* Recall that an oblivious tri-state circuit (Definition 9) consists of two parts: a circuit description  $C$  and a *distribution* on bits  $D$ . To evaluate the circuit, we must sample  $D$ . This is straightforward:  $G$  locally samples  $r \leftarrow D$ , then encodes  $r$  to ensure that the evaluator's encoding of each random input is the all zero key.

*Gate handling.* Recall (from Section 2.4) that TSCs support three gate types: XORs, buffers, and joins. We show how to handle each.

**XORs.** Consider an XOR gate  $z := x \oplus y$ . Our handling of XOR gates is straightforward, due to our use of Free-XOR-style keys. To start,  $G$  defines the key for  $z$  as the XOR of the input keys:

$$k_z^0 = k_x^0 \oplus k_y^0 \quad \text{XOR Garble}$$

At runtime and by our invariant,  $E$  holds keys  $k_x \oplus x \cdot \Delta$  and  $k_y \oplus y \cdot \Delta$ .  $E$  simply XORs its keys together, yielding a correct encoding for wire  $z$ :

$$(k_x^0 \oplus x \cdot \Delta) \oplus (k_y^0 \oplus y \cdot \Delta) = k_z^0 \oplus (x \oplus y) \cdot \Delta \quad \text{XOR Evaluate}$$

XOR gates are “free” in that the GC does not grow with XOR gates.

**Buffers.** Consider a buffer  $z := x/y$ . Here  $x$  is the buffer's data wire, and  $y$  is the control. If  $y$  holds a 1, then  $E$  should obtain a key on the output wire that matches the data wire; if not, then the output should hold  $\mathcal{Z}$ , so  $E$  should obtain *no key*. Accordingly,  $G$  defines  $z$ 's output key as follows:

$$k_z^0 = \mathcal{O}(k_y^0 \oplus \Delta, gid) \oplus k_x^0 \quad \text{Buffer Garble}$$

Here,  $gid$  is the buffer's gate-specific nonce. In words,  $G$  encrypts the data key  $k_x^0$  with the control wire's one-key.

At runtime and by our invariant,  $E$  holds keys  $k_x^0 \oplus x \cdot \Delta$  and  $k_y^0 \oplus y \cdot \Delta$ . If the control wire  $y$  holds one,  $E$  holds  $k_y^0 \oplus \Delta$ , so  $E$  can compute the correct encoding for the output wire  $z$ :

$$\mathcal{O}(k_y^0 \oplus \Delta, gid) \oplus (k_x^0 \oplus x \cdot \Delta) = k_z^0 \oplus x \cdot \Delta \quad \text{Buffer Evaluate}$$



If the control wire  $y$  holds zero – and because  $E$  does not know  $\Delta$  –  $E$  cannot decrypt the output key, and thus holds no key at all.

Note crucially that  $E$ 's evaluation is thus *conditioned on the control wire*  $y$ . To correctly evaluate,  $E$  must know  $y$ , so  $G$  must allow  $E$  to decrypt  $y$ . Accordingly,  $G$  attaches to the GC a single extra bit:

$$\text{lsb}(k_y^0) \qquad \text{Buffer GC String}$$

Per our discussion of point and permute,  $E$  simply XORs this value with its own key to decrypt the cleartext value of the control wire. Recall,  $E$  can learn all buffer controls due to TSC obliviousness (Definition 9).

**Joins.** Consider a join  $z := x \bowtie y$ . By TSC semantics,  $E$  should learn an encoding of the output if it holds an encoding for *either* input wire. To handle this,  $G$  simply defines the key of the output wire as the key for input  $x$ :

$$k_z^0 = k_x^0 \qquad \text{Join Garble}$$

(This choice is arbitrary;  $G$  could also set  $k_z^0 = k_y^0$ .)  $G$  also includes in the GC a length- $\lambda$  ciphertext that allows  $E$  to *translate*  $y$  keys to  $z$  keys:

$$k_y^0 \oplus k_x^0 \qquad \text{Join GC String}$$

At runtime, suppose  $E$  holds key  $k_x^0 \oplus x \cdot \Delta$ , or  $E$  holds  $k_y^0 \oplus y \cdot \Delta$ , or both. If  $E$  holds a key for a wire, we say that wire is set.  $E$  acts conditionally, depending on which wire is set:

$$k_z^0 \oplus z \cdot \Delta = \begin{cases} k_x^0 \oplus x \cdot \Delta & \text{if } x \text{ set} \\ (k_x^0 \oplus k_y^0) \oplus (k_y^0 \oplus y \cdot \Delta) & \text{if } y \text{ set} \end{cases} \qquad \text{Join Evaluate}$$

Note that it is impossible for  $x$  and  $y$  to hold mismatched Boolean values, as this would imply the TSC is not total (Definition 6).

*Order of evaluation.* As described above, buffers sometimes place keys on their output wires, and sometimes they do not. This means that in a TSC, the order in which  $E$  evaluates the circuit can depend on the data in the wires. Indeed, this data dependence is crucial in the efficient handling of RAM programs.

At each step of evaluation,  $E$  can choose any gate that is *ready*, and evaluate that gate. Recall that a wire is *set* if  $E$  holds a key on that wire:

**Definition 13 (Ready).** A TSC gate  $g$  is **ready** if one of the following holds:

- $g$  is an XOR  $z := x \oplus y$  where  $x$  and  $y$  are set and  $z$  is not set.
- $g$  is a buffer  $z := x/y$  where  $x$  and  $y$  are set and  $z$  is not set.
- $g$  is a join  $z := x \bowtie y$  where  $x$  or  $y$  are set (or both) and  $z$  is not set.

*Offline/Online and Costs.* In our construction,  $G$  sends the GC to  $E$  in the offline phase. In the online phase,  $G$  simply sends (1) an encoding of the input and (2) an output decoding table that allows to decode the output.

We summarize the communication and computation costs of each TSC gate:

	Comm. (bits)	$G$ queries to $\mathcal{O}$	$E$ queries to $\mathcal{O}$
XOR ( $\oplus$ )	0	0	0
Buffer ( $/$ )	1	1	$\leq 1$
Join ( $\bowtie$ )	$\lambda$	0	0

Thus, our total offline communication cost is  $\leq (|C| \cdot \lambda)$  bits. Our online cost scales with the circuit’s number of inputs  $n$  and the number of outputs  $m$ . Specifically, the online communication cost is  $O((n + m) \cdot \lambda)$ .

*Garbling scheme procedures.* Our garbling scheme procedures (Figure 2) are merely a formalization of the above handling. *Garble* describes  $G$ ’s actions, and *Eval* describes  $E$ ’s actions. *Encode* formalizes how the circuit input should be encoded, which by our use of Free-XOR-style labels simply maps each input  $x$  to  $k_x^0 \oplus x \cdot \Delta$  for some uniform  $k_x^0$ .

The decoding of outputs (formally, *Decode*) includes one additional detail: we store in the output decoding string  $d$  not the output wire keys themselves, but rather applications of the random oracle to the output wire keys. This is needed to ensure the garbling scheme is *authentic* [BHR12b]. Namely, this choice of *Decode* ensures that even a malicious  $E$  cannot forge output keys that correctly decode, except by running *Eval* as intended. As a final detail, we include in the output decoding string  $d$  the lsb of the zero key; this ensures the scheme is *perfectly correct*; without this addition, there is some negligible probability that the two output keys will hash to the same string.

### 5.1 Tri-State Circuit Adaptive Security

We show that our TSC garbling scheme (Construction 1) is query hiding and can be rekeyed privately. For simplicity, we refer to Construction 1 as  $\Pi_{\text{TSC}}$ . Combined with the results in this section, Theorem 2 imply that  $\Pi_{\text{TSC}}$  is an adaptive private garbling scheme.

**Lemma 1 ( $\Pi_{\text{TSC}}$  is Query Hiding).**  $\Pi_{\text{TSC}}$  is query hiding (Definition 10).

*Proof.* Consider the garbling of a circuit:

$$\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$$

Recall that query hiding roughly states that the adversary should not be able to distinguish two worlds. In the first world, we give the adversary  $\tilde{C}$  and oracle access to  $\mathcal{O}$ ; in the second, we give the adversary  $\tilde{C}$  and oracle access to  $\mathcal{O}^{-\delta}$  – an oracle with *Garble*’s queries removed.

The challenge in proving this property is that the garbled circuit  $\tilde{C}$  might provide to the adversary some “help” in making problematic oracle queries. We

must show that this is *not* the case, and that the adversary gains from  $\tilde{C}$  no advantage in guessing queries in  $\delta$ .

To show  $\Pi_{\text{TSC}}$  is query hiding, we first note that the offline garbled circuit  $\tilde{C}$  consists only of (1) join-gate ciphertexts of the form  $k_x^0 \oplus k_y^0$  and (2) point and permute bits associated with control keys to buffer gates. We also note that all the queries to the random oracle  $\mathcal{O}$  contain a key in them, so to show query hiding, we will reduce the adversary's ability to guess a problematic query to the adversary's ability to guess either the global secret  $\Delta$ , or some specific garbled circuit key – say, the first input zero key  $k_0^0$ .

Indeed, consider the following thought experiment: Rather than giving  $\tilde{C}$  to the adversary, suppose that for each circuit wire  $x$  we give to the adversary (1) the point and permute bit of  $k_x^0$  and (2) a ciphertext  $k_0^0 \oplus k_x^0$ . Note that the adversary could construct  $\tilde{C}$  from this information themselves, since garbled material joining arbitrary wires  $x$  and  $y$  can be expressed as follows:

$$k_x^0 \oplus k_y^0 = (k_0^0 \oplus k_x^0) \oplus (k_0^0 \oplus k_y^0)$$

Thus, the information conveyed to the adversary in this thought experiment is strictly more than that in the garbled circuit, since we can construct the latter from the former.

In this thought experiment, we can reason that if the adversary finds *any* key  $k_x^b$ , then they can compute a key on the

$$k_0^b = (k_0^0 \oplus k_x^0) \oplus k_x^b.$$

As such, an adversary that guesses some query made by  $\text{Garble}^{\mathcal{O}}$  can also output  $k_0^b$  for some  $b$  with the exact same probability. But by construction  $k_0^0$  and  $k_0^1$  are uniform in the choices of  $k_0^0$  and  $\Delta = k_0^1 \oplus k_0^0$  (Figure 2). Thus,  $\mathcal{A}$  gains no more than  $\text{negl}(\lambda)$  advantage in guessing  $k_0^0$  or  $k_0^1$ , which implies that the adversary cannot guess a query given only  $\tilde{C}$  w.p. better than  $\text{negl}(\lambda)$ .

Thus, this worst-case thought experiment hides garbling queries. Since the worst-case world gives strictly more information to the adversary than  $\tilde{C}$ ,  $\Pi_{\text{TSC}}$  is query hiding.  $\square$

**Lemma 2.**  $\Pi_{\text{TSC}}$  is rekeyable (Definition 11).

*Proof.* By construction of a rekeying procedure  $\text{Rekey}$ ; see Figure 3.

Recall that rekeyability roughly states that  $\text{Rekey}$  should output fresh keys that are identically distributed to keys generated by  $\text{Garble}$ . In short, we can arrange this due to the fact that all constraints imposed by the garbled circuit are linear. Thus, we can use keys to choose the garbled circuit, or vice versa.

More precisely,  $\text{Garble}$  and  $\text{Rekey}$  sample keys from the same distribution. Indeed,  $\Delta$  is identically distributed by construction. Further, each zero-key is also identically distributed. Indeed, the only difference between the two procedures is that  $\text{Garble}$  chooses keys from start to finish by calling  $\mathcal{O}$  and constructing  $\tilde{C}$ , whereas  $\text{Rekey}$  works backwards from  $\tilde{C}$  and chooses keys. Because the RO

$\text{Rekey}_{\text{TSC}}(1^\lambda, (C, D), \tilde{C})$

---

```

1 :  $r \leftarrow \$ D$ 
2 :  $\Delta \leftarrow \$ \{0, 1\}^{\lambda-1} \parallel 1$ 
3 : while not all keys are assigned do
4 :   arbitrarily select some unassigned wire and uniformly assign one of its keys,
5 :   subject to the following constraints:
6 :   for each join  $z \leftarrow x \bowtie y$  with  $\tilde{C}$  string  $row : (k_z^0 = k_x^0) \wedge (k_x^0 \oplus k_y^0 = row)$ 
7 :   for each buffer  $z \leftarrow x/y$  with  $\tilde{C}$  bit  $p : \text{lsb}(k_y^0) = p$ 
8 :   for each XOR  $z \leftarrow x \oplus y : k_z = k_x \oplus k_y$ 
9 :   for each  $i$ -th randomized input wire  $w : k_w^0 = r[i] \cdot \Delta$ 
10 :  for each wire  $w : k_w^0 \oplus k_w^1 = \Delta$ 
11 :  for each  $i$ -th input wire  $w$  do append to  $e$   $(k_w^0, k_w^0 \oplus \Delta)$ 
12 :  for each  $i$ -th output wire  $w$  do append to  $d$   $(\mathcal{O}(k_w^0, i), \mathcal{O}(k_w^0 \oplus \Delta, i))$ 
13 :  for each buffer  $z \leftarrow x/y$  do append to  $\delta$   $((k_y^1, gid), k_x^0 \oplus k_z^0)$ 
14 :  return  $(\delta, e, d)$ 

```

**Fig. 3.** The Rekey procedure for Construction 1. In short, our rekeying uniformly samples keys, subject to linear constraints imposed by the garbled circuit. The procedure outputs (1) a programming string  $\delta$ , (2) an input encoding string  $e$ , and (3) an output decoding string  $d$ . It is straightforward that the rekeying can be computed in linear time; at each step we (1) pick an arbitrary wire with an unassigned key, (2) uniformly sample the unconstrained bits of that unassigned key, and (3) use the chosen key to propagate constraints (by setting appropriate key bits) through connected gates.

is not in scope for Rekey, keys in both worlds are uniformly distributed, other than constraints imposed by  $\tilde{C}$ . But all constraints imposed by  $\tilde{C}$  are *linear*, so it does not matter if  $\tilde{C}$  is chosen with respect to the keys, or if keys are chosen with respect to  $\tilde{C}$ . Thus, .

□

**Lemma 3** ( $\Pi_{\text{TSC}}$  Privately Rekeys).  $\Pi_{\text{TSC}}$  privately rekeys (Definition 12).

*Proof.* Privately rekeying (Definition 12) requires that we show indistinguishability of the following for all  $\tilde{C} \in \text{Supp}(\text{Garble}(1^\lambda, C))$  and inputs  $x$ :

$\text{Real}_{\text{prv}}^{\mathcal{A}, C, \tilde{C}, x}(\lambda)$	$\text{Ideal}_{\text{prv}}^{\mathcal{A}, \mathcal{S}, C, \tilde{C}, x}(\lambda)$
1 : Sample random oracle $\mathcal{O}$	1 : Sample random oracle $\mathcal{O}$
2 : $(\delta, e, d) \leftarrow \text{Rekey}(\lambda, C, \tilde{C})$	2 : $(\delta, e, d) \leftarrow \text{Rekey}(\lambda, C, \tilde{C})$
3 : $\tilde{x} \leftarrow \text{Encode}(e, x)$	3 : $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
4 :	4 : $d' \leftarrow \mathcal{S}(1^\lambda, f(x), d)$
5 : <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d)$	5 : <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d')$

We introduce similar games which convey the intuition that the adversary should also not be able to distinguish when it is not allowed to decrypt the output of the garbled circuit (it is not given  $d$ ). These are similar to the notion of garbled circuit *obliviousness* [BHR12b]:

$\text{Real}_{obv}^{\mathcal{A}, C, \tilde{C}, x}(\lambda)$	$\text{Ideal}_{obv}^{\mathcal{A}, S, C, \tilde{C}}(\lambda)$
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $(\delta, e, d) \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$	2: $(\delta, e, d) \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$
3: $\tilde{x} \leftarrow \text{Encode}(e, x)$	3: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
4: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x})$	4: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x})$

*Privacy from obliviousness.* In our scheme  $\Pi_{\text{TSC}}$ , if  $\text{Real}_{obv}$  and  $\text{Ideal}_{obv}$  are indistinguishable, then we can construct also show privacy. In particular, our simulator  $\mathcal{S}$  in  $\text{Ideal}_{prv}$  takes the decoding table  $d$  and permutes the entries to find  $d'$  so that the encoded output properly decodes:

$$\text{Decode}(d', \text{Eval}(\tilde{C}, \text{Encode}(e, \mathbf{0}))) = f(x)$$

Thus, it suffices to show that  $\text{Real}_{obv}$  and  $\text{Ideal}_{obv}$  are indistinguishable.

*Uniform  $\Delta$ .* Consider the information revealed to the adversary in the online phase, for now ignoring the adversary's RO queries. The adversary is given the following:

$$\text{Online}_{\tilde{C}, x} = \{\tilde{C}, x, (\mathbf{k}_i^{b_i})_{\forall w_i}\},$$

for every wire  $w_i$  with cleartext wire value  $b_i$ . The TSC is total (Definition 6), so the adversary indeed sees a key on *each* wire. Note that the control bits  $\text{Controls}(C, x)$  (resp.  $\text{Controls}(C, \mathbf{0})$ ) are implied by the keys on control wires  $\mathbf{k}_c^{b_c}$  together with revealed buffer point-and-permute bits included as part of  $\tilde{C}$ .

Recall from Figure 3 that to rekey  $\tilde{C}$ ,  $\text{Rekey}_{\text{TSC}}$  samples new randomized circuit inputs, new uniform keys for some wires, and a new delta  $\Delta$ . After the adversary learns corresponding online information, but before it issues any RO queries, we find that while key randomness is leaked to the adversary, keys associated with open buffer gates (i.e., those buffers whose control wire is 0) are not. Notice that the global secret  $\Delta$  is *unconstrained*, despite the information sent to the adversary, and because  $\text{Rekey}_{\text{TSC}}$  samples  $\Delta$  independently of  $(C, D)$ , we find that the distribution of  $\Delta$  conditioned on seeing  $\text{Online}_{\tilde{C}, x}$  remains uniform.

*Game Indistinguishability.* Without any query access, we note that the distributions  $\text{Controls}(C, x)$  and  $\text{Controls}(C, \mathbf{0})$  are statistically close (Definition 9); the one way the adversary can get a larger advantage is by querying the RO at  $\mathcal{O}(\mathbf{k}_c^1 || \text{gid})$  for some wire  $w_c$  that controls a open buffer, i.e. a buffer where the adversary was given key  $\mathbf{k}_c^0$ .

With this query the adversary would be able to determine whether

$$k_i^{b_i} = \mathcal{O}(k_c^1 || gid) \oplus k_j^{b_j},$$

for data input (resp. output) wire  $w_i$  (resp.  $w_j$ ) associated with the buffer. If any such query was leaked to the adversary, then the adversary could distinguish the control bits; otherwise, the adversary would have no help distinguishing the two games  $\text{Real}_{obv}$  or  $\text{Ideal}_{obv}$ , since the keys together with  $\tilde{C}$  are independent of the evaluation.

Thus, to gain non-negligible advantage, the adversary must successfully query at least one open buffer gate. If the adversary were able to make such a query, then the adversary could also compute  $\Delta = k_c^0 \oplus k_c^1$ , and vice versa. Since  $\Delta$  is uniform and independent of the RO, we find that the probability that the adversary finds  $\Delta$  given only polynomially many oracle queries is  $\text{negl}(\lambda)$ .  $\Pi_{TSC}$  is adaptively private. □

By combining Theorem 2 with Lemmas 1 to 3 we obtain the following result:

**Corollary 1.**  *$\Pi_{TSC}$  is an adaptively private garbling scheme.*

## References

- [AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Heidelberg, August 2013.
- [BBK<sup>+</sup>23] Estuardo Alpírez Bock, Chris Brzuska, Pihla Karanko, Sabine Oechsner, and Kirthivaasan Puniyamurthy. Adaptive distributional security for garbling schemes with  $o(-x-)$  online complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 139–171. Springer, 2023.
- [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, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Heidelberg, December 2012.
- [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 2012*, pages 784–796. ACM Press, October 2012.
- [BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.
- [BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.

- [BR95] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *EUROCRYPT'94*, volume 950 of *LNCS*, pages 92–111. Springer, Heidelberg, May 1995.
- [CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.
- [CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the “free-XOR” technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012.
- [FLR<sup>+</sup>10] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 303–320. Springer, Heidelberg, December 2010.
- [FOPS01] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. RSA-OAEP is secure under the RSA assumption. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 260–274. Springer, Heidelberg, August 2001.
- [GHL<sup>+</sup>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, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 405–422. Springer, Heidelberg, May 2014.
- [GKW<sup>+</sup>20] Chun Guo, Jonathan Katz, Xiao Wang, Chenkai Weng, and Yu Yu. Better concrete security for half-gates garbling (in the multi-instance setting). In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 793–822. Springer, Heidelberg, August 2020.
- [GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. In Venkatesan Guruswami, editor, *56th FOCS*, pages 210–229. IEEE Computer Society Press, October 2015.
- [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 ACM STOC*, pages 449–458. ACM Press, June 2015.
- [GOS18] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 515–544. Springer, Heidelberg, August 2018.
- [GYW<sup>+</sup>23] Xiaojie Guo, Kang Yang, Xiao Wang, Yu Yu, and Zheli Liu. Unmodified half-gates is adaptively secure; so is unmodified three-halves. *IACR EPrint Archive*, 2023.
- [HJO<sup>+</sup>16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016.
- [HKO22] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. EpiGRAM: Practical garbled RAM. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 3–33. Springer, Heidelberg, May / June 2022.
- [HKO23] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Tri-state circuits: A circuit model that captures RAM. In *Crypto*, 2023.

- [JO20] Zahra Jafargholi and Sabine Oechsner. Adaptive security of practical garbling schemes. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 741–762. Springer, Heidelberg, December 2020.
- [JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao’s garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 433–458. Springer, Heidelberg, October / November 2016.
- [KKPW21] Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Daniel Wichs. Limits on the adaptive security of yao’s garbling. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 486–515, Virtual Event, August 2021. Springer, Heidelberg.
- [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 Ingólfssdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.
- [LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 719–734. Springer, Heidelberg, May 2013.
- [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.
- [MKJR16] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016.
- [MPT20] NIST workshop on multi-party threshold schemes 2020. National Institute of Standards and Technology, 2020.
- [MPT23] NIST workshop on multi-party threshold schemes 2023. National Institute of Standards and Technology, 2023.
- [Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126. Springer, Heidelberg, August 2002.
- [PLS23] Andrew Park, Wei-Kai Lin, and Elaine Shi. NanoGRAM: Garbled RAM with  $\tilde{O}(\log N)$  overhead. In *EUROCRYPT 2023, Part I*, LNCS, pages 456–486. Springer, Heidelberg, June 2023.
- [RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Heidelberg.
- [Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.
- [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, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.



# Appendices

## A Adaptive Obliviousness

**Definition 14 (Obliviously Rekeying).** Let  $\Pi$  be a rekeyable garbling scheme (Definition 11). We say that  $\Pi$  **obliviously rekeys** if for all circuits  $C$  computing some function  $f$ , for all  $x$ , for all garbled circuits  $\tilde{C} \in \text{Supp}(\text{Garble}(1^\lambda, C))$ , and for all PPT adversaries  $\mathcal{A}$ , the following ensembles are statistically close in  $\lambda$ :

$\text{Real}^{\mathcal{A}, C, \tilde{C}, x}(\lambda)$	$\text{Ideal}^{\mathcal{A}, C, \tilde{C}}(\lambda)$
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $(\delta, e, d) \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$	2: $(\delta, e, d) \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$
3: $\tilde{x} \leftarrow \text{Encode}(e, x)$	3: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
4: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x})$	4: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x})$

**Lemma 4 ( $\Pi_{\text{TSC}}$  Obliviously Rekeys).**  $\Pi_{\text{TSC}}$  obliviously rekeys (Definition 14).

*Proof.* Recall the hybrid of oblivious rekeying (Definition 14), which we would like to show indistinguishability for all  $\tilde{C} \in \text{Supp}(\text{Garble}(1^\lambda, C))$  and inputs  $x$ .

*Uniform  $\Delta$ .* We first take stock of what is revealed to the adversary in the online phase without adversary-directed queries. The adversary at least has access to the following distribution, note that  $\tilde{C}$  is fixed:

$$\text{Online}_{\tilde{C}, x} = \{\tilde{C}, x, (\mathbf{k}_i^{b_i})_{\forall w_i}\},$$

for every wire  $w_i$  and hidden semantic value over the wire  $b_i$ , recalling that TSC is total (Definition 6). Note that the control bits  $\text{Controls}(C, x)$  or  $\text{Controls}(C, \mathbf{0})$  are implied by having access to the keys  $\mathbf{k}_c^{b_c}$  associated with control wires  $w_c$  and the buffer leaked pnp bit from  $\tilde{C}$ .

Recall from Figure 3 that to rekey  $\tilde{C}$ ,  $\text{Rekey}_{\text{TSC}}$  samples new bits from  $(C, D)$ , new uniform keys for some wires, and a new delta  $\Delta$ . After revealing the keys as described above, but before the adversary is allowed any other queries, we find that the key randomness is leaked to the adversary, but not the bits from  $(C, D)$  associated with buffer gates with wires set to 0, and independently  $\Delta$ . We point out that because  $\Delta$  is still unconstrained despite the online phase information, and because  $\Delta$  is drawn independently from  $(C, D)$ , we find that the distribution of  $\Delta$  conditioned on seeing  $\text{Online}_{\tilde{C}, x}$  is still uniform.

*Game Indistinguishability.* Without any query access, we note that the distributions  $\text{Controls}(C, x)$  and  $\text{Controls}(C, \mathbf{0})$  are statistically close; however, the one way that the adversary can use queries to get a larger advantage is by querying the oracle at  $\mathcal{O}(k_c^1 || gid)$  for some wire  $w_c$  that leads into a buffer and that the adversary was given key  $k_c^0$  during evaluation.

With this query the adversary would be able to determine whether

$$k_i^{b_i} = \mathcal{O}(k_c^1 || gid) \oplus k_j^{b_j},$$

for data input (resp. output) wire  $w_i$  (resp.  $w_j$ ) associated with the buffer. If all such queries were leaked to the adversary, then the adversary would be able to distinguish the control bits; otherwise, the adversary would have no help distinguishing the two games Real or Ideal, since all the other information in  $\tilde{C}$  and the keys is independent of the evaluation.

As such, the only hope for the adversary to get more than statistical advantage in distinguishing Real or Ideal is to make at least one query to such a buffer gate; however, we point out that if the adversary were able to make such a query, then the adversary could also compute  $\Delta = k_c^0 \oplus k_c^1$ , and vice versa. Since  $\Delta$  is drawn from uniform and independent of the random oracle output randomness, we find that the probability that the adversary finds  $\Delta$  given only polynomially many oracle queries is  $\text{negl}(\lambda)$ . As such, we find that the two games are indistinguishable.  $\square$

**Theorem 2 (Adaptive Obliviousness from Oblivious Rekeying and Query Hiding).** *Let  $\Pi$  be an obliviously rekeyable (Definition 14) garbling scheme that is query hiding (Definition 10).  $\Pi$  is adaptively oblivious.*

*Proof.* By construction of a simulator:

$\mathcal{S}^{\mathcal{O}}(1^\lambda, C)$
1 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$ 2 : <b>return</b> $(\tilde{C}, \tilde{x})$ 3 : // Second call; receive $f(x)$ from caller 4 : $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$ 5 : <b>return</b> $(\tilde{x})$

Now, we show that the real-world and ideal-world experiments are indistinguishable. We restate the experiments, in-lining the handling of our simulator:

$\text{Real}_{\text{obv}}^{\mathcal{A}, C}(\lambda)$	$\text{Ideal}_{\text{obv}}^{\mathcal{A}, C}(\lambda)$
1 : Sample random oracle $\mathcal{O}$	1 : Sample random oracle $\mathcal{O}$
2 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$	2 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
3 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
4 : $\tilde{x} \leftarrow \text{Encode}(e, x)$	4 : $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5 : <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x})$	5 : <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x})$

To demonstrate indistinguishability, we proceed by a hybrid argument. We give six hybrids:

$F_{\text{obv},R}^{\mathcal{A},C}(\lambda)$	$F_{\text{obv},S}^{\mathcal{A},C}(\lambda)$
1: Sample random oracle $\mathcal{O}$	1: Sample random oracle $\mathcal{O}$
2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$	2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
3: $x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$	3: $x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$
4: $\tilde{x} \leftarrow \text{Encode}(e, x)$	4: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5: <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x})$	5: <b>return</b> $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x})$

$G_{\text{obv},R}^{\mathcal{A},C}(\lambda)$	$G_{\text{obv},S}^{\mathcal{A},C}(\lambda)$
1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$	1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$
2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$	2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$
3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
4: $\tilde{x} \leftarrow \text{Encode}(e, x)$	4: $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x})$	5: <b>return</b> $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x})$

$H_{\text{obv},R}^{\mathcal{A},C}(\lambda)$	$H_{\text{obv},S}^{\mathcal{A},C}(\lambda)$
1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$	1: Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$
2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$	2: $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}'}(1^\lambda, C)$
3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3: $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
4: $(\delta', e', d') \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$	4: $(\delta', e', d') \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$
5: $\tilde{x} \leftarrow \text{Encode}(e', x)$	5: $\tilde{x} \leftarrow \text{Encode}(e', \mathbf{0})$
6: <b>return</b> $\mathcal{A}^{\mathcal{O}^{\delta'}}(1^\lambda, \tilde{x})$	6: <b>return</b> $\mathcal{A}^{\mathcal{O}^{\delta'}}(1^\lambda, \tilde{x})$

We find it most direct to argue indistinguishability by “meeting in the middle”. Namely, our proof proceeds starting from the real/ideal ensemble, and at each step of our proof, we show the current real/ideal ensemble is indistinguishable from some respective intermediate ensemble. To conclude, we show that these two final ensembles are indistinguishable from one another.

*Removing offline oracle queries.* In our crucial proof step, we argue the following:

$$\{\text{Real}_{\text{obv}}^{\mathcal{A},C}(\lambda)\} \approx \{F_{\text{obv},R}^{\mathcal{A},C}(\lambda)\} \quad \text{and} \quad \{\text{Ideal}_{\text{obv}}^{\mathcal{A},C}(\lambda)\} \approx \{F_{\text{obv},S}^{\mathcal{A},C}(\lambda)\}$$

In this step we remove from the adversary’s oracle all queries issued by the call to `Garble`, but only in the offline phase. Jumping ahead, our informal objective

is to show that the adversary’s chosen input  $x$  must be independent of Garble’s random oracle queries.

Of course, there is a difference between games Real and  $G_{\text{obv},R}$  (resp. Ideal and  $G_{\text{obv},I}$ ). Indeed, in the former game the adversary is given access to the same oracle twice, and in the latter the adversary is given access to two oracles that differ by  $\delta$ . Thus, if the adversary can guess a query in  $\delta$ , it can distinguish the two games, since in the first game it will see the oracle respond consistently in the offline and online phases, and in the second game it will see the oracle “disagree with itself”.

Query hiding (Definition 10) is precisely what we need to show that the adversary cannot guess a query in  $\delta$ . Indeed, an unbounded adversary with only polynomial oracle queries has at most  $\text{negl}(\lambda)$  chance to sample a point in  $\delta$ . As such, the adversary only has a  $\text{negl}(\lambda)$  probability of determining if it was given  $\mathcal{O}$  or  $\mathcal{O}^{-\delta}$  on line 3 of both the Real games and  $G$ .

*Rearranging the oracles.* In our second step, we perform a “refactoring” of the random oracle queries. In particular, we argue:

$$\{F_{\text{obv},R}^{A,C}(\lambda)\} \equiv \{G_{\text{obv},R}^{A,C}(\lambda)\} \quad \text{and} \quad \{F_{\text{obv},S}^{A,C}(\lambda)\} \equiv \{G_{\text{obv},S}^{A,C}(\lambda)\}$$

Indeed, games  $G$  and  $H$  are identically distributed, as they are merely a rearrangement of the random oracle queries. The output distribution of  $\text{Garble}^{\mathcal{O}}$  in line 2 of hybrid  $G$  is independent of the programmed oracle  $\mathcal{O}^{-\delta}$  on line 2. As such, we can rewrite lines 2 and 3 to use different oracles all together to get hybrid  $H$ . On line 5, we note that the only part of  $\mathcal{O}'$  that  $\text{Garble}^{\mathcal{O}'}$  depends on are exactly those queries in  $\delta$ . As such, the garbler may as well have used  $\mathcal{O}^{\delta}$  to garble. After this step, it is clear that the adversary’s chosen input  $x$  must be independent of the random oracle  $\mathcal{O}'$  used to garble, since the adversary is not allowed access to  $\mathcal{O}'$ .

*Rekeying the garbled circuit.* Next, we use the rekeyability (Definition 11) of the scheme to sample fresh keys associated with the particular garbled circuit  $\tilde{C}$ . In particular, we argue:

$$\{G_{\text{obv},R}^{A,C}(\lambda)\} \equiv \{H_{\text{obv},R}^{A,C}(\lambda)\} \quad \text{and} \quad \{G_{\text{obv},S}^{A,C}(\lambda)\} \equiv \{H_{\text{obv},S}^{A,C}(\lambda)\}$$

This is immediate by the rekeyability property.

*Privacy.* Finally, we bridge from the “real world”, where we encode  $x$ , to the “ideal world”, where we encode 0:

$$\{H_{\text{obv},R}^{A,C}(\lambda)\} \equiv \{H_{\text{obv},S}^{A,C}(\lambda)\}$$

First, notice that the outputs of garbling –  $\tilde{C}$ ,  $\delta$ ,  $e$ , and  $d$  – are independent of  $\mathcal{O}$ . Thus, the choice of  $x$  is also independent of  $\delta$ ,  $e$ , and  $d$ , except insofar as they are constrained by the garbled circuit  $\tilde{C}$ . We can capture this sentiment

by treating  $\tilde{C}$  and  $x$  as if they are universally quantified. But this setting is exactly what is considered in oblivious rekeying (Definition 14). Applying oblivious rekeying thus discharges the proof.

Any query hiding, obviously rekeyable garbling scheme is adaptively oblivious. □

## B Existing Schemes are Adaptive

As another application of our framework, we argue that a range of existing GC schemes are adaptively secure, if their underlying hash function is implemented with an NPRO.

### B.1 Free XOR

We show that the Free XOR construction [KS08], as written, is rekeyable. The Free XOR scheme is relatively straightforward.

First, the handling of wire keys is the same as described in Section 5. In particular, for each input wire and each AND gate output wire  $w$ , the scheme samples a uniformly random label  $k_w^0 \leftarrow \{0, 1\}^\lambda$ . Additionally, the garbler samples a global offset  $\Delta \leftarrow \{0, 1\}^{\lambda-1} \parallel 1$ , and for each wire  $w$ , the garbler sets  $k_w^1 = k_w^0 \oplus \Delta$ .

For each XOR gate  $z \leftarrow x \oplus y$ , the garbler sets the output wire key  $k_z^0 = k_x^0 \oplus k_y^0$ . For each AND gate  $z \leftarrow x \cdot y$ , the garbler generates four garbled rows<sup>9</sup>, which are added to the garbled circuit:

$$\begin{aligned} & \mathcal{O}(k_x^0, k_y^0, gid) \oplus k_z^0 \\ & \mathcal{O}(k_x^0, k_y^1, gid) \oplus k_z^0 \\ & \mathcal{O}(k_x^1, k_y^0, gid) \oplus k_z^0 \\ & \mathcal{O}(k_x^1, k_y^1, gid) \oplus k_z^1 \end{aligned}$$

**Lemma 5.** *Let  $\Pi$  denote the garbling scheme of [KS08] described above.  $\Pi$  is query hiding (Definition 10).*

*Proof Sketch.* The proof that keys garbled in [KS08] hide queries is very similar, if not simpler, than for TSC (Lemma 1). Like with TSC, keys are generated either from uniform, or as a linear combination of other keys. Note that all the ciphertexts in  $\tilde{C}$  given to the adversary are of the form  $\mathcal{O}(q) \oplus k_x^b$  for some query  $q$ . When the adversary is allowed no queries, then  $\tilde{C}$  perfectly encrypts what the keys could be, since the outputs of the RO is uniform. Informally, this continues to be the case until the adversary hits a query  $q$  used to encrypt some key.

<sup>9</sup> For simplicity, we list the rows in an unpermuted order; the full scheme would permute the rows according to point-and-permute bits. The permutation of rows is not relevant to our current discussion.

Since the keys are uniform or linear in nature, we find that the adversary cannot guess any of the keys and by extension any oracle query with better than w.p.  $\text{negl}(\lambda)$ .  $\square$

**Lemma 6.** *Let  $\Pi$  denote the garbling scheme of [KS08] described above.  $\Pi$  supports a rekey procedure (Definition 11) that privately and obliviously rekeys (Definitions 12 and 14).*

*Proof Sketch.* The roadmap to proving that Free-XOR is rekeyable is similar to that of rekeyability of TSC (Lemma 2), but with even simpler steps. The rekey procedure simply chooses wire keys in the same manner as garble: sample a fresh  $\Delta$ , uniformly sample a zero-key on each input wire and each AND gate output, and compute the zero-key on each XOR gate output by XORing the input keys. From here, the rekey procedure programs the RO in the obvious way, ensuring that each combination of AND gate input keys decrypts to the appropriate output key.

The procedure described above is a rekeyer (Definition 11): the keys are clearly drawn from the same distribution in the garbling and the rekeying. The proof that this rekey procedure satisfies rekey obliviousness and privacy is almost identical to the proof given in Lemma 2, except that the proof is arguably simpler, since we need not concern ourselves with tri-state circuit obliviousness.  $\square$

Thus, the [KS08] scheme is adaptively secure if their hash function is modeled by a NPRO.

**Corollary 2.**  *$\Pi$  denoting the garbling scheme of [KS08] is an adaptively private and adaptively oblivious.*

## B.2 Half-Gates is Rekeyable

The popular half-gates scheme [ZRE15] first demonstrated how to garble AND gates for only two ciphertexts. The scheme is rekeyable, given the hash function is modeled as an NPRO:

**Lemma 7.** *Let  $\Pi$  denote the garbling scheme of [ZRE15].  $\Pi$  is query hiding (Definition 10) and supports a rekey procedure (Definition 11) that privately and obliviously rekeys (Definitions 12 and 14).*

*Proof Sketch.* The proof that the half-gates technique of [ZRE15] is rekeyable is very similar to the above Free XOR proof, and to our TSC proof. Indeed, TSCs in some sense formally capture the exact procedures of the half-gates scheme. This was shown in [HKO23], where they give an oblivious AND gate construction that uses exactly two join gates, matching the cost of half-gates (indeed, the underlying handling is the same as half-gates).

Rather than meticulously proving that half-gates is rekeyable and query hiding, we simply point out that these properties are implied by Lemmas 1 and 2. Thus, the [ZRE15] scheme is adaptively secure if its hash function is modeled by a NPRO.  $\square$

### B.3 Arithmetic Gadgets is Rekeyable

[BMR16] demonstrated interesting garbling techniques for a limited class of arithmetic circuits. In particular, they generalize the Free XOR technique to small prime fields. Their construction allows wires over various moduli  $\mathbb{Z}_p$  for prime  $p$ , and to do this, they sample a Free XOR correlation for each modulus  $p$ , each consisting of repeated  $\mathbb{Z}_p$  elements, such that each such  $\Delta$  has  $\approx \lambda$  bits of entropy. For simplicity, we henceforth simply say  $\mathbb{Z}_p^\ell$  where  $\ell$  denotes the number of  $\mathbb{Z}_p$  elements needed to acquire  $\lambda$  bits of entropy.

To select keys for wires containing a  $\mathbb{Z}_p$  element, [BMR16] choose the zero-key uniformly from  $\mathbb{Z}_p^\ell$ ; the one-key, two-key, three-key, ...,  $p - 1$ -key are chosen by adding multiples of the appropriate correlation  $\Delta$ .

[BMR16] provide three operations on wires: addition (modulo  $p$ ), scaling by a constant (modulo  $p$ ), and *projection*. The projection operation allows mapping each of the  $p$  possible keys to some specified output key, potentially in a different modulus. The projection operation is implemented simply by hashing the input key, and using the hash to encrypt the respective output key, then including that ciphertext in the garbled circuit (these ciphertexts are shuffled according to point and permute).

**Lemma 8.** *Let  $\Pi$  denote the garbling scheme of [BMR16].  $\Pi$  is query hiding (Definition 10).*

*Proof Sketch.* Because the garbled circuit  $\tilde{C}$  is a collection of RO outputs XORed with the linear keys in  $\mathbb{Z}_p^\ell$ , we find that the oracle queries that take in these linear and uniform keys should be hidden for exactly the same reason as in Lemma 8.  $\square$

**Lemma 9.** *Let  $\Pi$  denote the garbling scheme of [BMR16].  $\Pi$  supports a rekey procedure (Definition 11) that privately and obliviously rekeys (Definitions 12 and 14).*

*Proof Sketch.* Similar to our discussion of Free XOR, it is clear that this scheme is rekeyable. Indeed, all keys are either uniformly sampled, combined in a linear manner, or encrypted under RO. Thus, we can construct a Rekey procedure that resamples arithmetic keys uniformly, while respecting the constraints imposed by addition gates and scalar gates. The fact that this Rekey procedure is perfectly indistinguishable, oblivious, and private is proved in almost the exact same manner as Free XOR.

Thus, the [BMR16] scheme is adaptively secure if their hash function is modeled by a NPRO.  $\square$