Adaptive Garbled Circuits and Garbled RAM from Non-Programmable Random Oracles

Cruz Barnum[1], David Heath[2], Vladimir Kolesnikov[3], and Rafail Ostrovsky[4]

[1] cruzjb2@illinois.edu, UIUC
[2] daheath@illinois.edu, UIUC
[3] kolesnikov@gatech.edu, Georgia Tech
[4] 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 much 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 GC techniques.

As our main goal and as an 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 λ 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 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].[5]

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 $\tilde{\mathcal{P}}$.
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

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

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 λ. 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 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+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 roughly doubles the total computational cost vs standard selectively-secure GC; however, the main issue is that the proof of adaptive security requires the simulator to dynamically define, or *program*, the RO. This is a strong assumption, which clearly 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 a simple condition. This condition is satisfied by many standard schemes, including Free XOR, half-gates, and even arithmetic techniques [BMR16]; see Appendix B. In short, our condition requires that 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 this condition is 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 λ be the security parameter, 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.

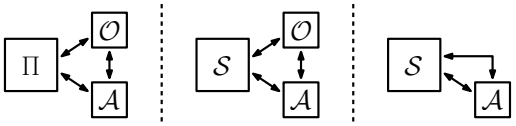


Fig. 1. (Left) A real-world protocol using RO. The real-world adversary interacts directly with the RO. (Center) A simulation in the non-programmable RO model. Here, the simulator $\mathcal{S}$ interacts with the RO independently of $\mathcal{A}$, so $\mathcal{S}$ cannot respond to nor even learn $\mathcal{A}$'s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

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. Note, 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]. 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.

Most importantly, 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 much milder assumption, which is widely used in practical cryptography. For instance, the standardized RSA-OAEP encryption scheme uses NPRO [BR95,FOPS01,FLR$^+$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. The computational overhead of this approach is multiplicative factor w, 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.

Adaptive garbling from one-way functions. [HJO$^+$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 $O(w \cdot \mathrm{poly}(\lambda))$, where w denotes the *width* of the target circuit.

Much like [BHR12a]'s above one-time-pad-based scheme, [HJO$^+$16]'s key idea is to allow G to equivocate the GC. To improve the equivocation, [HJO$^+$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$^+$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+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+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 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+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.

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+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

- λ 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*

$$(Garble, Eval, Eval, Decode)$$

with the following interface:

– $Garble(1^\lambda, C) \to (\tilde{C}, e, d)$*: Garble a circuit $C \in \mathcal{C}$, producing garbled circuit $\tilde{C}$,* input encoding string *e, and* output decoding string *d.*
– $Encode(e, x) \to \tilde{x}$*: Use the input encoding string e to encode input x.*
– $Eval(\tilde{C}, \tilde{x}) \to \tilde{y}$*: Evaluate $\tilde{C}$ on encoded input $\tilde{x}$, yielding encoded output $\tilde{y}$.*
– $Decode(d, \tilde{y}) \to y$*: 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 λ:*

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

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

2.3 Definitions of Adaptive Security

We consider two notions of adaptivity as defined by [BHR12a]. The first – *adaptive obliviousness* – is the simpler of the two, and is accordingly easier to prove. The second – *adaptive privacy* – is more directly applicable in the 2PC setting. We prove both. Our definitions are adapted from [BHR12a] and [BHR12b].

Roughly speaking, adaptive obliviousness states the adversary $\mathcal{A}$ cannot distinguish a real garbled circuit from a simulated one, even when it is allowed to choose its input adaptively.

Definition 3 (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 λ:*

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

where Real, Ideal *are as follows:*

$\text{Real}^{\mathcal{A},C}_{\text{obv}}(1^\lambda)$	$\text{Ideal}^{\mathcal{A},C}_{\text{obv}}(1^\lambda)$
$(\tilde{C}, e, d) \leftarrow \mathit{Garble}^{\mathcal{O}}(1^\lambda, C)$	$(\tilde{C}, \tilde{x}) \leftarrow \mathcal{S}^{\mathcal{O}}(1^\lambda, C)$
$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
$\tilde{x} \leftarrow \mathit{Encode}(e, x)$	**return** $\mathcal{A}^{\mathcal{O}}(\tilde{x})$
return $\mathcal{A}^{\mathcal{O}}(\tilde{x})$	

Under this definition, $\mathcal{A}$ is not given output decoding information d. Intuitively, this means that the garbled circuit reveals nothing to $\mathcal{A}$, not even information about the circuit output.

In constrast, 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.*

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

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

where Real, Ideal *are as follows:*

$\text{Real}^{\mathcal{A},C}_{\text{prv}}(1^\lambda)$	$\text{Ideal}^{\mathcal{A},C}_{\text{prv}}(1^\lambda)$
$(\tilde{C}, e, d) \leftarrow \mathit{Garble}^{\mathcal{O}}(1^\lambda, C)$	$\tilde{C} \leftarrow \mathcal{S}^{\mathcal{O}}(1^\lambda, C)$
$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
$\tilde{x} \leftarrow \mathit{Encode}(e, x)$	$y \leftarrow f(x)$
return $\mathcal{A}^{\mathcal{O}}(\tilde{x}, d)$	$(\tilde{x}, d) \leftarrow \mathcal{S}^{\mathcal{O}}(y)$
	return $\mathcal{A}^{\mathcal{O}}(\tilde{x}, d)$

Formally, adaptive obliviousness and adaptive privacy are *incomparable*; one does not imply the other. However, informally speaking, for typical schemes (including ours), privacy follows relatively easily once obliviousness is shown. Looking forward, our proof of adaptive privacy is a relatively straightforward extension of our proof of adaptive obliviousness.

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 $\{0, 1, \mathcal{Z}, \boldsymbol{\times}\}$. The semantics of each gate type are as follows:*

$\oplus$	$\mathcal{Z}$	0	1	$\boldsymbol{\times}$
$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\boldsymbol{\times}$
0	$\mathcal{Z}$	0	1	$\boldsymbol{\times}$
1	$\mathcal{Z}$	1	0	$\boldsymbol{\times}$
$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$

$/$	$\mathcal{Z}$	0	1	$\boldsymbol{\times}$
$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\boldsymbol{\times}$
0	$\mathcal{Z}$	$\mathcal{Z}$	0	$\boldsymbol{\times}$
1	$\mathcal{Z}$	$\mathcal{Z}$	1	$\boldsymbol{\times}$
$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$

$\bowtie$	$\mathcal{Z}$	0	1	$\boldsymbol{\times}$
$\mathcal{Z}$	$\mathcal{Z}$	0	1	$\boldsymbol{\times}$
0	0	0	$\boldsymbol{\times}$	$\boldsymbol{\times}$
1	1	$\boldsymbol{\times}$	1	$\boldsymbol{\times}$
$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$	$\boldsymbol{\times}$

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 $\boldsymbol{\times}$ 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:*

$$(C, D)(x) = C(x; r) \qquad \text{where } r \leftarrow\!\!\$\ D$$

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:*

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

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-λ key k_w^0. This key encodes a logical zero on wire w. G then samples a single length-λ global correlation Δ, and for each wire w, G defines the encoding of logical one as follows:

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

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

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

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:

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

  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 $\mathsf{k}_z^0 = \mathsf{k}_x^0$. 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 $\mathsf{k}_x \oplus \mathsf{k}_y$. E XORs this difference with its y key to obtain an appropriate z key.

We refer the reader to Section 4 for further details on our TSC handling.

3.2 Proof of Security

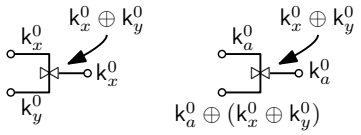
Our main contribution shows that typical GC schemes built from a random oracle are also *adaptively secure* in the NPRO model. For example, our above basic TSC scheme is adaptively secure with no change in implementation.

On our (non-) use of programmability. Our proof of security 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 an 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 $\mathsf{k}^0_x, \mathsf{k}^0_y$; the output wire is similarly labelled by k^0_x. To enable evaluation in all cases, G includes in $\tilde{C}$ the specific string $\mathsf{k}^0_x \oplus \mathsf{k}^0_y$. If E only holds a runtime key k_y for the bottom input wire, they can use this string to *translate* that input key to an appropriate output key:

$$(\mathsf{k}^0_y \oplus y \cdot \Delta) \oplus (\mathsf{k}^0_x \oplus \mathsf{k}^0_y) = \mathsf{k}^0_x \oplus y \cdot \Delta$$

We start rekeying this gate by replacing its top input key k^0_x with some freshly sampled key k^0_a. Similarly, we replace the output wire key by k^0_a. 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^0_y by a key that is specifically chosen to preserve semantics: $\mathsf{k}^0_a \oplus (\mathsf{k}^0_x \oplus \mathsf{k}^0_y)$. 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:

$$((\mathsf{k}^0_a \oplus (\mathsf{k}^0_x \oplus \mathsf{k}^0_y)) \oplus y \cdot \Delta) \oplus (\mathsf{k}^0_x \oplus \mathsf{k}^0_y) = \mathsf{k}^0_a \oplus y \cdot \Delta$$

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 we can sample fresh GC keys K' *after* the adversary has chosen its input x, and this circumvents the main difficulty of the adaptive setting. From here, in proving adaptive obliviousness, we use a hybrid argument to show that the following four worlds are indistinguishable:

1. **Real world:** $\mathcal{A}$ sees the GC $\tilde{C}$ and chooses its input x. We encode x and give the encoding to $\mathcal{A}$.
2. $\mathcal{A}$ sees the GC $\tilde{C}$ and chooses its input x. We then rekey the GC and use the new keys to encode x, programming the RO such that gates execute consistently with the encoding of x. We give the rekeyed encoding of x to $\mathcal{A}$.
3. $\mathcal{A}$ sees the GC $\tilde{C}$ and chooses its input x. We then rekey the GC, *ignore x, and encode the all zeros input*, programming the RO such that gates execute consistently with the encoding of 0. We give the rekeyed encoding of 0 to $\mathcal{A}$.

4. **Ideal world:** $\mathcal{A}$ sees the GC $\tilde{C}$ and chooses its input x. We ignore x, encode the all zeros input, and give the encoding to $\mathcal{A}$.

The above hybrid argument is relatively generic. In our formalism, we define a procedure Rekey along with security properties on Rekey. *Any* garbling scheme that satisfies these properties is adaptively secure. The above proof sketch summarizes our proof of adaptive *obliviousness* (Definition 3); the proof of adaptive *privacy* (Definition 4) is slightly more complex, but the main idea is unchanged.

4 Our 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 5 proves this scheme adaptively secure.

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

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

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* Δ:

$$\Delta \leftarrow\!\$\ \{0,1\}^{\lambda-1}1$$

Δ is chosen by the garbler G and is hidden from the evaluator E.

For each circuit wire w, the G maintains a key $\mathsf{k}_w^0 \in \{0,1\}^\lambda$. This key encodes logical zero. The encoding of logical one is *defined* as $\mathsf{k}_w^1 = \mathsf{k}_w^0 \oplus \Delta$. 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:

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

Recall that in a TSC, wires can carry Boolean value, or they can carry value $\mathcal{Z}$ or ✗. Following [HKO23], $\mathcal{Z}$ is encoded by E's *lack of a key*. ✗ 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 ✗.

$\text{Garble}^{\mathcal{O}}(1^\lambda, (C, D))$

$\tilde{C} \leftarrow \textit{emptymap}$
$e, d \leftarrow \textit{emptyvec}$
$\Delta \leftarrow\!\!\$\ \{0,1\}^{\lambda-1} \mid\mid 1$
$r \leftarrow\!\!\$\ D$
for each input wire w **do**
  $\mathsf{k}_w^0 \leftarrow\!\!\$\ \{0,1\}^\lambda$
  append $(\mathsf{k}_w^0, \mathsf{k}_w^0 \oplus \Delta)$ to e
for each i-th random input w **do**
  $\mathsf{k}_w^0 \leftarrow r_i \cdot \Delta$
for $(g, gid) \in C$ **do**
  if $g = (z := x/y)$ **do**
    $\tilde{C}[gid] \leftarrow \text{lsb}(\mathsf{k}_y^0)$
    $\mathsf{k}_z^0 \leftarrow \mathcal{O}(\mathsf{k}_y^0 \oplus \Delta, gid) \oplus \mathsf{k}_x^0$
  elseif $g = (z := x \bowtie y)$ **do**
    $\tilde{C}[gid] \leftarrow \mathsf{k}_x^0 \oplus \mathsf{k}_y^0$
    $\mathsf{k}_z^0 \leftarrow \mathsf{k}_x^0$
  elseif $g = (z := x \oplus y)$ **do**
    $\mathsf{k}_z^0 \leftarrow \mathsf{k}_x^0 \oplus \mathsf{k}_y^0$
for each i-th output wire w **do**
  append to d
    $(\text{lsb}(\mathsf{k}_w^0), \mathcal{O}(\mathsf{k}_w^0, i), \mathcal{O}(\mathsf{k}_w^1, i))$
return $(\tilde{C}, e, d)$

$\text{Encode}(e, x)$

$\tilde{x} \leftarrow \textit{emptyvec}$
for each i-th input $x[i]$
  $(\mathsf{k}^0, \mathsf{k}^1) \leftarrow e[i]$
  append $\mathsf{k}^{x[i]}$ to $\tilde{x}$
return $\tilde{x}$

$\text{Eval}^{\mathcal{O}}((C, D), \tilde{C}, \tilde{x})$

for each nonrandom input wire w **do**
  $\mathsf{k}_w \leftarrow \tilde{x}[w]$
for each random input wire w **do**
  $\mathsf{k}_w \leftarrow 0^\lambda$
for $(g, gid) \in C$ where g is ready **do**
  if $g = (z := x/y)$ **do**
    $ctrl \leftarrow \text{lsb}(\mathsf{k}_y) \oplus \tilde{C}[gid]$
    if $ctrl = 1$ **do**
      $\mathsf{k}_z \leftarrow \mathcal{O}(\mathsf{k}_y, gid) \oplus \mathsf{k}_x$
  elseif $g = (z := x \bowtie y)$ **do**
    if x is set **do**$\mathsf{k}_z \leftarrow \mathsf{k}_x$
    else $\mathsf{k}_z \leftarrow \mathsf{k}_y \oplus \tilde{C}[gid]$
  elseif $g = (z := x \oplus y)$ **do**
    $\mathsf{k}_z \leftarrow \mathsf{k}_x \oplus \mathsf{k}_y$
$\tilde{y} \leftarrow \textit{emptyvec}$
for each output wire w **do**
  append k_w to $\tilde{y}$
return $\tilde{y}$

$\text{Decode}(d, \tilde{y})$

$y \leftarrow \textit{emptyvec}$
for each i-th output label $\tilde{y}[i]$
  $(p, \mathsf{k}^0, \mathsf{k}^1) \leftarrow d[i]$
  if $\mathcal{O}(\tilde{y}[i], i) = \mathsf{k}^0 \wedge \text{lsb}(\tilde{y}[i]) = p$ **do**
    append 0 to y
  elseif $\mathcal{O}(\tilde{y}[i], i) = \mathsf{k}^1$ **do**
    append 1 to y
  else
    return $\perp$
return y

Fig. 2. Our NPRO-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 10). Our scheme is adaptively secure.

Point and Permute. Our global offset Δ 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 $\mathsf{k}_w^0 \oplus \Delta$ 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 $\mathrm{lsb}(\mathsf{k}_w^0)$ to the GC. At runtime, E computes the lsb of its key $\mathrm{lsb}(\mathsf{k}_w^0 \oplus w \cdot \Delta)$, XORs the result with $\mathrm{lsb}(\mathsf{k}_w^0)$, 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 as keys which are attached to the garbling $\tilde{C}$.

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:

$$\mathsf{k}_z^0 = \mathsf{k}_x^0 \oplus \mathsf{k}_y^0 \qquad \text{XOR Garble}$$

At runtime and by our invariant, E holds keys $\mathsf{k}_x \oplus x \cdot \Delta$ and $\mathsf{k}_y \oplus y \cdot \Delta$. E simply XORs its keys together, yielding a correct encoding for wire z:

$$(\mathsf{k}_x^0 \oplus x \cdot \Delta) \oplus (\mathsf{k}_y^0 \oplus y \cdot \Delta) = \mathsf{k}_z^0 \oplus (x \oplus y) \cdot \Delta \qquad \text{XOR Evaluate}$$

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:

$$\mathsf{k}_z^0 = \mathcal{O}(\mathsf{k}_y^0 \oplus \Delta, \mathit{gid}) \oplus \mathsf{k}_x^0 \qquad \text{Buffer Garble}$$

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 $\mathsf{k}_x^0 \oplus x \cdot \Delta$ and $\mathsf{k}_y^0 \oplus y \cdot \Delta$. If the control wire y holds one, E holds $\mathsf{k}_y^0 \oplus \Delta$, so E can compute the correct encoding for the output wire z:

$$\mathcal{O}(\mathsf{k}_y^0 \oplus \Delta, \mathit{gid}) \oplus (\mathsf{k}_x^0 \oplus x \cdot \Delta) = \mathsf{k}_z^0 \oplus x \cdot \Delta \qquad \text{Buffer Evaluate}$$

If the control wire y holds zero – and because E does not know Δ – 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:

$$\mathrm{lsb}(\mathsf{k}_y^0) \qquad \text{Buffer GC String}$$

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:

$$\mathsf{k}_z^0 = \mathsf{k}_x^0 \qquad \text{Join Garble}$$

(This choice is arbitrary; G could also set $\mathsf{k}_z^0 = \mathsf{k}_y^0$.) G also includes in the GC a length-λ ciphertext that allows E to *translate* y keys to z keys:

$$\mathsf{k}_y^0 \oplus \mathsf{k}_x^0 \qquad \text{Join GC String}$$

At runtime, suppose E holds key $\mathsf{k}_x^0 \oplus x \cdot \Delta$, or E holds $\mathsf{k}_y^0 \oplus y \cdot \Delta$, 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:

$$\mathsf{k}_z^0 \oplus z \cdot \Delta = \begin{cases} \mathsf{k}_x^0 \oplus x \cdot \Delta & \text{if } x \text{ set} \\ (\mathsf{k}_x^0 \oplus \mathsf{k}_y^0) \oplus (\mathsf{k}_y^0 \oplus y \cdot \Delta) & \text{if } y \text{ set} \end{cases} \qquad \text{Join Evaluate}$$

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 10 (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	≤ 1
Join ($\bowtie$)	λ	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 if $O((n+m) \cdot \lambda)$.

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 $\mathsf{k}_x^0 \oplus x \cdot \Delta$ 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 Adaptive Security of Rekeyable Garbling Schemes

This section formalizes the notion of **rekeyable garbling schemes**, as well as security properties on these schemes. Then, we show that (1) rekeyable garbling schemes are adaptively secure and (2) our TSC-based construction (Section 4) is rekeyable. In Appendix B, we discuss other garbling schemes that are rekeyable (when compiled; see next). Namely, we Free XOR [KS08], half-gates [ZRE15], and garbled arithmetic gadgets [BMR16] are rekeyable.

5.1 Additional Notation

We start with additional useful 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. To prove security, it is convenient to *seed* the RO with some uniformly chosen seed s, and to only deliver s to $\mathcal{A}$ in the online phase. This makes it easy to prove that $\mathcal{A}$ cannot issue useful

queries in the offline phase, unless $\mathcal{A}$ is exceedingly lucky. We provide relevant notation that seeds an RO with s, effectively producing a fresh RO:

Notation 1 (Seeded Random Oracle)**.** *Let $\mathcal{O}$ be a random oracle. Then for some λ-length string s, we say $\mathcal{O}[s]$ is the oracle seeded at s:*

$$\mathcal{O}[s](x) = \mathcal{O}(s, x)$$

To accompany Notation 1, we present a (trivial) compiler that transforms a garbling scheme by using a seeded oracle in place of an unseeded oracle. We emphasize that this compilation is straightforward.

Construction 2 (Seeded Oracle Rekeyable Garbling)**.** *Let Π be a garbling scheme with Rekey. Then we give the seeded scheme Π_s:*

$\mathit{Garble}'^{\mathcal{O}}(1^\lambda, C)$

1 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$

2 : $(\tilde{C}, e, d) \leftarrow \mathit{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$

3 : **return** $(\tilde{C}, (s, e), d)$

$$\mathit{Encode}'((s,e),x) := (s, \mathit{Encode}(e,x))$$
$$\mathit{Eval}'^{\mathcal{O}}(\tilde{C}, (s, \tilde{x})) := \mathit{Eval}^{\mathcal{O}[s]}(\tilde{C}, \tilde{x})$$
$$\mathit{Decode}'(d, \tilde{y}) := \mathit{Decode}(d, \tilde{y})$$

For simplicity of notation, we allow the challenger to manage seeds s.

Remark 1. We note that seeding the oracle, in the most natural instantiation, increases the input length of $\mathcal{O}$. Depending on the specific scheme, this may or may not impact concrete performance costs. We leave as future work finding an efficient instantiation based on fixed-key AES, likely along the line of work [GKWY20].

Next, recall that our approach to adaptive security requires that we can *rekey* the GC. To ensure the GC continues to decrypt correctly even with freshly chosen keys, our intermediate proof hybrids program entries of the RO. We define appropriate notation for an RO that is programmed at particular rows:

Notation 2 (Explicitly Programmed Random Oracle)**.** *Let $\mathcal{O}$ be a random oracle outputting strings in $\{0,1\}^\lambda$, and let δ be a partial map from oracle queries to oracle responses. The **explicitly programmed oracle** $\mathcal{O}^\delta$ is defined as follows:*

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

Notation 3. *By $\mathcal{O}^{\delta[s]}$, we mean programming the oracle $\mathcal{O}$ such that when the oracle is* later *seeded with s, the resulting* seeded *oracle is programmed with δ. Namely, the following correctness condition holds:*

$$(\mathcal{O}^{\delta[s]})[s] = (\mathcal{O}[s])^\delta$$

In our security games, we provide to $\mathcal{A}$ access to the unseeded oracle, which $\mathcal{A}$ then seeds. Notation 3 allows us to discuss providing to the hybrid-world adversary access to the unseeded programmed oracle, which is later seeded (by $\mathcal{A}$) and behaves according to the programming.

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:

Definition 11 (Garble Support). *Let $C \in \mathcal{C}$ be a circuit. Let $Supp(Garble(C))$ denote the **support** of Garble on input C. Formally, $Supp(Garble(C))$ is the support of the distribution defined by the following procedure:*

$\mathcal{G}(\lambda, C)$

1 : *Sample random oracle* $\mathcal{O}$
2 : $(\tilde{C}, e, d) \leftarrow Garble^{\mathcal{O}}(1^\lambda, C)$
3 : **return** $\tilde{C}$

5.2 Rekeyable Garbling Schemes

This section formalizes the notion of rekeyable garbling schemes, as well as their security properties. We formalize the Rekey procedure for a rekeyable scheme:

Definition 12 (Rekey). *Let Π be a garbling scheme. A **rekey procedure** Rekey for Π is a procedure that outputs (1) a partial map δ mapping oracle queries to oracle responses, (2) an input encoding string e, and (3) an output decoding string d. The correctness of Rekey is defined by the following experiment: For all circuits $C \in \mathcal{C}$, all inputs x, the following experiment outputs 1 with probability $1 - \mu(\lambda)$, for some negligible function μ:*

$RekeyCorrect(\lambda, C, x)$

1 : *Sample random oracles* $\mathcal{O}_1, \mathcal{O}_2$
2 : $(\tilde{C}, e, d) \leftarrow Garble^{\mathcal{O}_1}(1^\lambda, C)$
3 : $(\delta, e', d') \leftarrow Rekey(1^\lambda, C, \tilde{C})$
4 : $\tilde{x} \leftarrow Encode(e', x)$
5 : $\tilde{y} \leftarrow Eval^{\mathcal{O}_2^\delta}(\tilde{C}, \tilde{x})$
6 : **return** $C(x) = Decode(d', \tilde{y})$

In words, this definition states that a rekey procedure modifies GC keys and oracle queries – but not the GC itself – such that the GC still correctly evaluates. Note that our definition uses two different oracles $\mathcal{O}_1$ and $\mathcal{O}_2$. This ensures that the programming of the RO needed to enable correct evaluation is

fully described by δ. In other words, Eval should not depend meaningfully on queries to the non-programmed oracle inputs.

Our first security property on Rekey is *Rekey Indistinguishability*. In short, this property states that the adversary should not be able to distinguish the keys associated with a garbled circuit from a rekeying of the same garbled circuit.

The details here are relatively subtle, as to make a meaningful definition, we need to *fix* a particular garbled circuit $\tilde{C}$ and reason about the ability for the adversary to distinguish keys associated with *that* GC. This becomes particularly subtle when we consider that two calls to Garble might yield the *same* garbled circuit $\tilde{C}$, but they might associate with that GC different keys, due to sampling different randomness and using different random oracles. Our definition roughly states that, given $\tilde{C}$, the keys generated by Garble (such that Garble output $\tilde{C}$) are indistinguishable from resampled keys chosen by Rekey (on input $\tilde{C}$).

Definition 13 (Rekey Indistinguishability). *Let Π be a garbling scheme with rekey procedure Rekey. We say Rekey* ***indistinguishably rekeys*** *if for all garbled circuits $\tilde{C} \in \mathit{Supp}(\mathit{Garble}(C))$, for all* PPT *adversaries $\mathcal{A}$, the following quantity is bounded from above by a negligible function $\mu(\lambda)$:*

$$\left|\begin{array}{l}\Pr\left[\mathcal{A}^{\mathcal{O}}(1^\lambda, e, d) = 1 \;\middle|\; (\mathcal{O}, \tilde{C}', e, d) \leftarrow \mathit{Setup}^R(\lambda, C) \wedge \tilde{C}' = \tilde{C}\right] \\ \quad - \Pr\left[\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, e', d') = 1 \;\middle|\; (\mathcal{O}, \delta, e', d') \leftarrow \mathit{Setup}^I(\lambda, C, \tilde{C})\right]\end{array}\right|$$

where $\mathit{Setup}^R, \mathit{Setup}^I$ are defined as follows:

$\mathit{Setup}^R(\lambda, C)$	$\mathit{Setup}^I(\lambda, C, \tilde{C})$
1 : *Sample random oracle* $\mathcal{O}$	1 : *Sample random oracle* $\mathcal{O}$
2 : $(\tilde{C}', e, d) \leftarrow \mathit{Garble}^{\mathcal{O}}(1^\lambda, C)$	2 : $(\delta, e', d') \leftarrow \mathit{Rekey}(1^\lambda, C, \tilde{C})$
3 : **return** $(\mathcal{O}, \tilde{C}', e, d)$	3 : **return** $(\mathcal{O}, \delta, e', d')$

When $\mathcal{A}$ is unbounded and the above holds, we say that Rekey ***statistically rekeys****. If μ is the zero function, we say that Rekey* ***perfectly rekeys****.*

We point out that natural RO-based schemes tend to *perfectly* rekey. Informally, this is because natural schemes choose keys uniformly or by calling the RO. We provide statistical and computational notions for completeness.

Rekey indistinguishability is not sufficient to prove adaptive security, and even selective security combined with rekey indistinguishability seems insufficient to achieve adaptive security. Intuitively, it is not clear how to obtain rekey obliviousness (see next), which is stated wrt a fixed GC, from GC obliviousness, which is probabilistic over sampled GCs.

Recall from Section 2.3 that we consider two forms of adaptive security: obliviousness and privacy. We define two corresponding security notions on rekeyable schemes. Jumping ahead, we note that a scheme that supports rekey indistinguishability and that satisfies the following obliviousness notion (resp. privacy notion) is adaptively oblivious (resp. private); see Theorems 1 and 2.

Definition 14 (Rekey Obliviousness). *Rekey **obliviously rekeys** if for all circuits $C \in \mathcal{C}$, all inputs x, all garbled circuits $\tilde{C} \in Supp(Garble(C))$, and all* PPT *adversaries $\mathcal{A}$:*

$$\{Real^{\mathcal{A},C,\tilde{C},x}(\lambda)\} \approx_{\mu(\lambda)} \{Ideal^{\mathcal{A},C,\tilde{C}}(\lambda)\},$$

where $\mu(\lambda)$ is a negligible function and where the games are defined as follows:

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

The above obliviousness definition corresponds to GC obliviousness, which concerns an evaluator who sees a GC and input encoding $\tilde{x}$. The GC *privacy* game provides security against the evaluator who sees a GC, input encoding $\tilde{x}$, *and* output decoding table d. Correspondingly, we define rekey privacy as a modification of rekey obliviousness. The requirement is that Rekey must be compatible with a way to simulate a decoding table d', which will allow to decode to arbitrary output values. In standard schemes, such a decoding table simulation is easily achieved simply by appropriately reassigning key-value correspondence.

Definition 15 (Rekey Privacy). *Rekey **privately rekeys** if for all circuits $C \in \mathcal{C}$ computing some function f, there exists a simulator $\mathcal{S}$ s.t. for all x, for all garbled circuits $\tilde{C} \in Supp(Garble(C))$, and for all* PPT *adversaries $\mathcal{A}$:*

$$\{Real^{\mathcal{A},C,\tilde{C},x}(\lambda)\} \approx_{\mu(\lambda)} \{Ideal^{\mathcal{A},C,\tilde{C}}(\lambda)\},$$

where $\mu(\lambda)$ is a negligible function and the games are defined as follows.

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

5.3 Adaptive Security

In this section, we show that any garbling scheme with an indistinguishable, oblivious (resp. private) rekey procedure satisfies adaptive obliviousness (resp. privacy), when the scheme is compiled with Construction 2.

Theorem 1 (Adaptive Obliviousness from Rekeying). *Let Π be a garbling scheme that rekeys indistinguishably and obliviously (Definitions 13 and 14). The compiled (via Construction 2) scheme Π_{seed} is adaptively oblivious.*

Proof. In short, the properties on Rekey are precisely what is needed to show obliviousness. In more detail, recall that the definition of adaptive obliviousness requires existence of a simulator (Definition 3). Our simulator is straightforward:

$\mathcal{S}^{\mathcal{O}}(1^\lambda, C)$

1 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
2 : $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
3 : **return** $(\tilde{C}, \tilde{x})$

Now, we must show that the real-world and ideal-world experiments are indistinguishable. We restate the experiments, inlining the handling of our simulator and of our seeded oracle compiler (Construction 2):

$\text{Real}^{\mathcal{A},C}_{\text{obv}}(\lambda)$	$\text{Ideal}^{\mathcal{A},C}_{\text{obv}}(\lambda)$
1 : Sample random oracle $\mathcal{O}$	1 : Sample random oracle $\mathcal{O}$
2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$	2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$
3 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$	3 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$
4 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	4 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
5 : $\tilde{x} \leftarrow \text{Encode}(e, x)$	5 : $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
6 : **return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, s, \tilde{x})$	6 : **return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, s, \tilde{x})$

To show indistinguishability, we proceed by a hybrid argument. We formalize the two hybrids needed to complete our proof:

$\text{H}^{\mathcal{A},C}_{\text{obv},R}(\lambda)$	$\text{H}^{\mathcal{A},C}_{\text{obv},S}(\lambda)$
1 : Sample random oracle $\mathcal{O}$	1 : Sample random oracle $\mathcal{O}$
2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$	2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$
3 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$	3 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$
4 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	4 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
5 : $s' \leftarrow\!\!\$\ \{0,1\}^\lambda$	5 : $s' \leftarrow\!\!\$\ \{0,1\}^\lambda$
6 : $(\delta, e', d') \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$	6 : $(\delta, e', d') \leftarrow \text{Rekey}(1^\lambda, C, \tilde{C})$
7 : $\tilde{x} \leftarrow \text{Encode}(e', x)$	7 : $\tilde{x} \leftarrow \text{Encode}(e', \mathbf{0})$
8 : **return** $\mathcal{A}^{\mathcal{O}^{\delta[s']}}(1^\lambda, s', \tilde{x})$	8 : **return** $\mathcal{A}^{\mathcal{O}^{\delta[s']}}(1^\lambda, s', \tilde{x})$

Our intermediate hybrids (1) sample fresh RO seeds s', (2) rekey the garbled circuit, and (3) give to the adversary the rekeyed input, as well as access to a

seeded, programmed RO $\mathcal{O}^{\delta[s']}$. Using a fresh seed s' makes it easy to argue that the adversary cannot issue useful queries in the offline phase, since nothing $\mathcal{A}$ sees in the offline phase is related to s'. When $\mathcal{A}$ does not know a seed s, s', it is as if $\mathcal{A}$ does not yet have access to the RO.

Now, we argue indistinguishability as follows: We show the real-world game is indistinguishable from $\mathrm{H}_{\mathrm{obv},R}$, and we show that the ideal-world game is indistinguishable from $\mathrm{H}_{\mathrm{obv},S}$. To complete the proof, we show $\mathrm{H}_{\mathrm{obv},R}$ is indistinguishable from $\mathrm{H}_{\mathrm{obv},S}$.

Game-Hybrid Steps. We first argue the following:

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

The main difference between $\mathrm{Real}_{\mathrm{obv}}$ and $\mathrm{Ideal}_{\mathrm{obv}}$ from the respective H_{obv} games is the addition of the line

$$(\delta, e', d') \leftarrow \mathrm{Rekey}(1^\lambda, C, \tilde{C}),$$

as well as the use of e' instead of e, d' instead of d, and $\mathcal{O}^{\delta[s']}[s']$ instead of $\mathcal{O}[s]$. That is, all we need to prove is that the hybrid distribution $(\mathcal{O}^{\delta[s']}[s'], e', d')$ is indistinguishable from game distribution $(\mathcal{O}[s], e, d)$, i.e. that a PPT distinguisher with corresponding oracle access cannot distinguish (e, d) from (e', d'). If we ignore information gleaned from the offline phase, this immediately follows from rekey indistinguishability (Definition 13).

Of course, we must account for the offline phase. $\mathcal{A}$ gains no advantage from it if $\mathcal{A}$ does not make queries related to the seeds s or s'. Since the seeds are uniformly sampled strings, $\mathcal{A}$ has negligible probability of guessing s or s' in the offline phase, so it is as if $\mathcal{A}$ is not given the oracle until the online phase. Thus, the above argument based on rekey indistinguishability goes through.

Hybrid-Hybrid Step. We now show:

$$\{\mathrm{H}^{\mathcal{A},C}_{\mathrm{obv},R}(\lambda)\} \approx \{\mathrm{H}^{\mathcal{A},C}_{\mathrm{obv},S}(\lambda)\}$$

Since the garbled circuit $\tilde{C}$ comes from the same distribution in both games, consider the two hybrids on some fixed $\tilde{C} \in \mathrm{Supp}(\mathrm{Garble}(C))$. Because $\tilde{C}$ is fixed, $\mathcal{A}$'s choice of input x must come from the same distribution in both worlds. Therefore, we can apply rekey obliviousness (Definition 14) to show that lines 6 and 7 of each hybrid are indistinguishable. $\mathcal{A}$'s distinguishing advantage is at most the $\mu(\lambda)$ advantage from rekey obliviousness.

Thus, the considered garbling scheme is adaptively oblivious. □

Our considered properties on rekeyable garbling schemes also imply adaptive privacy. The proofs and techniques are very similar to those in Theorem 1. Any garbling scheme that rekeys indistinguishably and privately is adaptively private. We prove the following in Appendix A.

Theorem 2 (Adaptive Privacy from Rekeying). *Let Π be a garbling scheme that rekeys indistinguishably and privately (Definitions 13 and 15). Then compiled scheme Π_{seed} is an adaptively private garbling scheme (Definition 4).*

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

$r \leftarrow\!\!\$\ D$

$\Delta \leftarrow\!\!\$\ \{0,1\}^{\lambda-1} \,||\, 1$

while not all keys are assigned **do**

  arbitrarily select some unassigned wire and uniformly assign one of its keys,

  subject to the following constraints:

  for each join $z \leftarrow x \bowtie y$ with $\tilde{C}$ string $row : (\mathsf{k}_z^0 = \mathsf{k}_x^0) \wedge (\mathsf{k}_x^0 \oplus \mathsf{k}_y^0 = row)$

  for each buffer $z \leftarrow x/y$ with $\tilde{C}$ bit $p :\ \text{lsb}(\mathsf{k}_y^0) = p$

  for each XOR $z \leftarrow x \oplus y :\ \mathsf{k}_z = \mathsf{k}_x \oplus \mathsf{k}_y$

  for each i-th randomized input wire $w : \mathsf{k}_w^0 = r[i] \cdot \Delta$

  for each wire $w :\ \mathsf{k}_w^0 \oplus \mathsf{k}_w^1 = \Delta$

for each i-th input wire w **do** append to e $(\mathsf{k}_w^0, \mathsf{k}_w^0 \oplus \Delta)$

for each i-th output wire w **do** append to d $(\mathcal{O}(\mathsf{k}_w^0, i), \mathcal{O}(\mathsf{k}_w^0 \oplus \Delta, i))$

for each buffer $z \leftarrow x/y$ **do** append to δ $((\mathsf{k}_y^1, gid), \mathsf{k}_x^0 \oplus \mathsf{k}_z^0)$

return (δ, 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 δ, (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.

5.4 Tri-State Circuit Rekeying

We show that our TSC garbling scheme (Construction 1), can be rekeyed (perfectly) indistinguishably, obliviously, and privately. For simplicity, we refer to Construction 1 as Π_{TSC}. Combined with the results in this section, Theorems 1 and 2 imply that Π_{TSC} is an adaptively oblivious and private garbling scheme.

Rekeying. Recall that the TSC garbler G produces zero-keys for wires as follows:

- **Input wires:** G uniformly samples the key.
- **Buffer wires:** G chooses the output key based on an RO query.
- **XOR output wires:** G defines the output key as XOR of the input keys.
- **Join output wires:** G defines the output key as one of the input keys.

To choose the one-keys, recall that G samples Δ.

Recall that to rekey a GC $\tilde{C}$, we must sample fresh keys consistent with the GC and a (programmed) oracle. In the context of the TSC construction, this means that our new keys must respect linear constraints imposed by the GC itself. Our garbling of TSCs includes three types of strings that our rekey procedure must respect (see Figure 2):

– **Joins:** For each join gate with input keys $\mathsf{k}_x^0, \mathsf{k}_y^0$, $\tilde{C}$ includes the string $\mathsf{k}_x^0 \oplus \mathsf{k}_y^0$.
– **Buffers:** For each buffer with control key k_y^0, $\tilde{C}$ includes the bit $\mathrm{lsb}(\mathsf{k}_y^0)$.
– **Random inputs:** For each random input wire w corresponding to sampled input bit r, $\tilde{C}$ includes the key $\mathsf{k}_w^0 \oplus r \cdot \Delta$.

In short, our construction's Rekey procedure simply uniformly samples a fresh correlation Δ, then uniformly samples keys, subject to the above constraints. Recall, our construction works with oblivious TSCs, so it takes as input a TSC C and a distribution D. Figure 3 lists the full procedure.

Theorem 3 (Rekeying Garbled Tri-State Circuits). *Let (C_i, D_i) be an oblivious and total tri-state circuit. Rekey_{TSC} (Figure 3) rekeys with perfectly indistinguishability, obliviousness, and privacy.*

Proof. We prove each property.

Correctness is verified by inspection: the GC evaluates to the correct value.

Perfect Indistinguishability. Given some fixed garbled circuit $\tilde{C}$, Garble and Rekey sample keys from the same distribution.

Δ is identically distributed by construction. Further, each zero-key is also identically distributed. Indeed, the only difference between the two procedures is that Garble chooses keys from start to finish by calling $\mathcal{O}$ and constructing $\tilde{C}$, whereas Rekey works backwards from $\tilde{C}$ and chooses keys. Because the RO is not in scope for rekey indistinguishability, 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, Rekey satisfies perfect indistinguishability.

Obliviousness. Recall that obliviousness (Definition 3) states that the evaluator cannot distinguish a rekeying of a garbled circuit used to encode a chosen input x from a rekeying used to encode the all-zeros string.

During evaluation, one key on each wire will be revealed to the evaluator (due to tri-state circuit totality, Definition 6). By inspection, we find that we can compute the collection of *every* GC key K from C, $\tilde{C}$, Δ, and the control bits $\mathrm{Controls}(C, x)$ (resp. $\mathrm{Controls}(C, \mathbf{0})$). (In more detail, the controls are determined by x or $\mathbf{0}$ together with some randomly sampled $r \leftarrow\!\!\$\ D$; the rekey procedure chooses a fresh r.) In other words, the only information that can possibly be used to distinguish these two worlds is Δ and the control bits; all other information is fixed or can be derived. But Δ is uniform in both worlds, and our above indistinguishability argument shows that *any* Δ could have produced $\tilde{C}$. Thus, only the control bits can be used to distinguish. But the tri-state circuit is oblivious (Definition 9), so the controls are statistically close between these worlds. Thus, Rekey satisfies obliviousness.

Privacy. Privacy (Definition 4) is satisfied in the same manner as obliviousness, except that we also find that d can be derived from $y = C(x)$ and all keys K. Thus, Rekey satisfies privacy. □

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.

[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+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+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.

[GKWY20] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy*, pages 825–841. IEEE Computer Society Press, May 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.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *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 Ingólfsdóttir, 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 $\widetilde{O}(\log N)$ 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 Proof of Adaptive Privacy

We provide the deferred proof of Theorem 2:

Theorem 2 (Adaptive Privacy from Rekeying). *Let Π be a garbling scheme that rekeys indistinguishably and privately (Definitions 13 and 15). The compiled (via Construction 2) scheme Π_{seed} is adaptively private.*

Proof. In short, the properties of Rekey are exactly what is needed to show privacy. The following proof is very similar to that of Theorem 1.

Recall that the difference between obliviousness and privacy is that in the latter, $\mathcal{A}$ is given access to an output decoding string d, which allows $\mathcal{A}$ to observe the garbled circuit outputting some specific value. To achieve indistinguishability, the GC output must be consistent with $\mathcal{A}$'s adaptively chosen input x; $\mathcal{A}$ must see $C(x)$. This is not hard to arrange, as we send the decoding string d only *after* $\mathcal{A}$ chooses its input, so in the ideal world we simply use a simulator to program d to output the appropriate value. Other details of the proof are essentially identical to that of Theorem 1.

In more detail, recall that the definition of adaptive privacy requires existence of a (stateful) simulator. Our simulator is straightforward:

$\mathcal{S}^{\mathcal{O}}(1^\lambda, C)$

1 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$
2 : **return** $\tilde{C}$
3 : // In second call $\mathcal{S}^{\mathcal{O}}(y)$ for $C(x) = y$.
4 : $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
5 : $d' \leftarrow \mathcal{S}_p(1^\lambda, y, d)$
6 : **return** $(\tilde{x}, d')$

Now, we must show that the real-world and ideal-world experiments are indistinguishable. We restate the experiments, inlining the handling of our simulator and of our seeded oracle compiler (Construction 2):

$\text{Real}^{\mathcal{A},C}_{\text{prv}}(\lambda)$	$\text{Ideal}^{\mathcal{A},\mathcal{S}_p,C}_{\text{prv}}(\lambda)$
1 : Sample random oracle $\mathcal{O}$	1 : Sample random oracle $\mathcal{O}$
2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$	2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$
3 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$	3 : $(\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$
4 : $x \leftarrow \mathcal{A}^{\mathcal{O}_1}(1^\lambda, \tilde{C})$	4 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
5 : $\tilde{x} \leftarrow \text{Encode}(e, x)$	5 : $\tilde{x} \leftarrow \text{Encode}(e, \mathbf{0})$
6 : **return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, s, \tilde{x}, d)$	6 : $d' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d)$
	7 : **return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, s, \tilde{x}, d')$

To show indistinguishability, we proceed by a hybrid argument. We formalize the two hybrids needed to complete our proof:

$\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)$	$\mathrm{H}^{\mathcal{A},\mathcal{S}_p,C}_{\mathrm{prv},S}(\lambda)$
1 : Sample random oracle $\mathcal{O}$	1 : Sample random oracle $\mathcal{O}$
2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$	2 : $s \leftarrow\!\!\$\ \{0,1\}^\lambda$
3 : $(\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$	3 : $(\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}[s]}(1^\lambda, C)$
4 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	4 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
5 : $s' \leftarrow\!\!\$\ \{0,1\}^\lambda$	5 : $s' \leftarrow\!\!\$\ \{0,1\}^\lambda$
6 : $(\delta, e', d') \leftarrow \mathrm{Rekey}(1^\lambda, C, \tilde{C})$	6 : $(\delta, e', d') \leftarrow \mathrm{Rekey}(1^\lambda, C, \tilde{C})$
7 : $\tilde{x} \leftarrow \mathrm{Encode}(e', x)$	7 : $\tilde{x} \leftarrow \mathrm{Encode}(e', \mathbf{0})$
8 : **return** $\mathcal{A}^{\mathcal{O}^{\delta[s']}}(1^\lambda, s', \tilde{x}, d')$	8 : $d'' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d')$
	9 : **return** $\mathcal{A}^{\mathcal{O}^{\delta[s']}}(1^\lambda, s', \tilde{x}, d'')$

As in Theorem 1, our hybrids (1) sample fresh seeds s', (2) rekey the garbled circuit, and (3) give to the adversary the rekeyed input, as well as access to a seeded, programmed RO.

Game-Hybrid Step. We first argue the following:

$$\{\mathrm{Real}^{\mathcal{A},C}_{\mathrm{prv}}(\lambda)\} \approx \{\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)\} \qquad \{\mathrm{Ideal}^{\mathcal{A},\mathcal{S}_p,C}_{\mathrm{prv}}(\lambda)\} \approx \{\mathrm{H}^{\mathcal{A},\mathcal{S}_p,C}_{\mathrm{prv},S}(\lambda)\}$$

The argument as to why this indistinguishability holds is identical to the Game-Hybrid reasoning argument in the obliviousness proof (Theorem 1).

Hybrid-Hybrid Step. Finally, we argue:

$$\{\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)\} \approx \{\mathrm{H}^{\mathcal{A},\mathcal{S}_p,C}_{\mathrm{prv},S}(\lambda)\}$$

Since the garbled circuit $\tilde{C}$ comes from the same distribution in both games, consider the two hybrids on some fixed $\tilde{C} \in \mathrm{Supp}(\mathrm{Garble}(C))$. Because $\tilde{C}$ is fixed, $\mathcal{A}$'s choice of input x must come from the same distribution in both worlds. Therefore, we can apply rekey privacy (Definition 15) to show that lines 6 and 7 in $\mathrm{H}_{\mathrm{prv},R}$ and lines 6 through 8 in $\mathrm{H}_{\mathrm{prv},S}$ are indistinguishable. The adversary's distinguishing advantage is at most the $\mu(\lambda)$ advantage of rekey privacy.

Thus, the considered garbling scheme is adaptively private. □

B Popular Existing Schemes are Rekeyable

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 is Rekeyable

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 4. In particular, for each input wire and each AND gate output wire w, the scheme samples a uniformly random label $\mathsf{k}_w^0 \leftarrow\!\$\ \{0,1\}^\lambda$. Additionally, the garbler samples a global offset $\Delta \leftarrow\!\$\ \{0,1\}^{\lambda-1} \mid\mid 1$, and for each wire w, the garbler sets $\mathsf{k}_w^1 = \mathsf{k}_w^0 \oplus \Delta$.

For each XOR gate $z \leftarrow x \oplus y$, the garbler sets the output wire key $\mathsf{k}_z^0 = \mathsf{k}_x^0 \oplus \mathsf{k}_y^0$. For each AND gate $z \leftarrow x \cdot y$, the garbler generates four garbled rows[6], which are added to the garbled circuit:

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

Lemma 1. *Let Π denote the garbling scheme of [KS08] described above, compiled by applying Construction 2. Π supports a rekey procedure that with perfect indistinguishability, obliviousness, and privacy.*

Proof Sketch. The roadmap to proving that (the compiled via Construction 2 version of) Free-XOR is rekeyable is similar to that of rekeyability of TSC (Theorem 3), but with even simpler steps. The rekey procedure simply chooses wire keys in the same manner as garble: sample a fresh Δ, 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.

Perfect rekey indistinguishability thus comes for free: 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 Theorem 3, except that the proof is arguably simpler, since we need not concern ourselves with tri-state circuit obliviousness. Thus, the [KS08] scheme (when compiled with Construction 2) is adaptively secure if their hash function is modeled by a NPRO. □

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:

[6] 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.

Lemma 2. *Let Π denote the garbling scheme of [ZRE15], compiled by applying Construction 2. Π supports a rekey procedure that with perfect indistinguishability, obliviousness, and privacy.*

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, we simply point out that its rekeyability is implied by Theorem 3. Thus, the [ZRE15] scheme (when compiled with Construction 2) is adaptively secure if its hash function is modeled by a NPRO. □

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 Δ has $\approx \lambda$ bits of entropy. For simplicity, we henceforth simply say $\mathbb{Z}_p^\ell$ where ℓ denotes the number of $\mathbb{Z}_p$ elements needed to acquire λ 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 Δ.

[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 3. *Let Π denote the garbling scheme of [BMR16], compiled by applying Construction 2. Π supports a rekey procedure that with perfect indistinguishability, obliviousness, and privacy.*

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 (when compiled with Construction 2) is adaptively secure if their hash function is modeled by a NPRO. □