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 milder assumption than PRO, and it is close to the assumption required by the widely used Free XOR extension. Our framework is applicable to a number of existing GC techniques, which are proved adaptively secure without modification.

As our main application, we construct and prove adaptively secure a garbling scheme for *tri-state circuits*, a model that captures both Boolean circuits and RAM programs (Heath et al., Crypto 2023). For TSC C, our garbling of C is at most $|C| \cdot \lambda$ bits long, for security parameter λ. This implies both an adaptively secure garbled Boolean circuit scheme, and an adaptively secure garbled RAM scheme where the garbling of a T-step 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

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

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 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.

[HKO23] showed that there exists a relatively simple circuit model – tri-state circuits (TSCs) – that can efficiently emulate both Boolean circuits and RAM programs. We construct a scheme that garbles TSCs, implying results for both garbled Boolean circuits and for garbled RAM. 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.

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

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

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

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

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

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

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

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

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

Cost accounting of adaptive GC schemes. When constructing adaptively secure garbling schemes, we consider the cost of the offline and the online phases separately. The most important metrics are the offline and online communication costs. Ideally, offline phase communication should be as close as possible to the cost in the selective security setting. Thus, we ideally want a scheme whose offline communication is equal to the size of the underlying circuit representation, multiplied by the security parameter 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 only on one-way functions, requires high computational overhead (multiplicative factor $O(w)$, where w is the width of the evaluated circuit) both in online and offline phases [HJO$^+$16]. This effectively negates the benefit of offline transmission in many settings.

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

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

1.1 Our Contribution

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

Many practical GC techniques are selectively secure in the *non-programmable* random oracle (NPRO) model. As our main contribution, we provide a framework that allows to prove that such schemes are also *adaptively* secure, if they meet two conditions. These are satisfied by existing standard schemes, including Free XOR, half-gates, three-halves gates [RR21], and even arithmetic techniques [BMR16,Hea24]; see Appendix C. In short, our conditions require that (1) any RO queries issued while garbling the circuit are hidden by the resulting garbled circuit and (2) one can *resample* keys associated with a particular GC, such that the GC still evaluates correctly with these fresh keys – the scheme should be *rekeyable*. We highlight that these conditions are reasonably easy to prove, which would facilitate the adoption of our framework and indeed of the final protocols.

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

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,

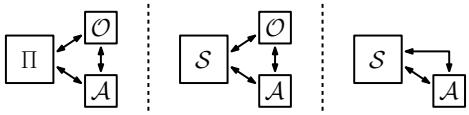


Fig. 1. (Left) A real-world protocol using RO. The real-world adversary interacts directly with the RO. (Center) A simulation in the non-programmable RO model. Here, the simulator $\mathcal{S}$ interacts with the RO independently of $\mathcal{A}$, so $\mathcal{S}$ cannot respond to nor even learn $\mathcal{A}$'s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

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

When we compile Boolean gates to tri-state gates, our scheme's costs match the cost of the popular selectively-secure half-gates garbling scheme [ZRE15] in terms of both computation and communication. [ZRE15] is a state-of-the-art Boolean scheme[6], so our TSC-based Boolean handling matches what one would expect from standard Boolean circuit garbling. Additionally, T steps of a RAM program can be compiled to a TSC 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 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

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

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

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

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

In contrast, NPRO is a milder assumption, which is widely used in practical cryptography. For instance, the standardized RSA-OAEP encryption scheme uses NPRO [BR95,FLR$^+$10,FOPS01,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 (PROM).

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

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

1.3 Related Work

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

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

One subtle, but central, aspect of this simulation is that we must replace gates in a specific order. This is so that we can base the indistinguishability of each hybrid on the PRF assumption. Indeed, 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 NPROM, we cannot circumvent the [AIKW13] lower bound.[7]

In concurrent work, [GYW+23] consider adaptive garbling of specific implementations of popular half-gates and three-halves schemes. They show that these schemes, instantiated with (previously designed) specific encryptions built on the

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

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

Adaptive garbling from one-way functions is a much harder task, so known solutions are substantially less efficient. [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 further, showing that GC techniques for reducing offline gate cost also work in the adaptive setting, achieving total online cost closer to that of a state-of-the-art garbling scheme. These techniques only work for circuits in NC^1, which is limiting. [KKPW21] showed that in the adaptive setting, Yao's scheme *must* suffer exponential security loss w.r.t. circuit depth. Thus, it seems log-depth circuits is the best possible for adaptive Yao's, unless the design is significantly changed.

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

The recent work of [BBK+23] achieves adaptive security from OWFs for a weaker notion of simulation security called *distributional simulation security*. The protocol in [BBK+23] is secure while only sending something which scales

with the size of input in the online phase, circumventing the [AIKW13] lower-bound. Our work uses the more standard notion of adaptivity.

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

Early GRAM schemes, e.g. [GLO15,GLOS15,GHL$^+$14,LO13], demonstrated important feasibility results, but they were not concerned with polylog performance factors, so their constructions are expensive. More recent constructions [HKO22,HKO23,PLS23] 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 computationally indistinguishable.
- $x \overset{s}{\approx} y$ denotes that distributions x and y are statistically close, i.e. indistinguishable to an unbounded adversary.
- $x \equiv y$ 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.
- Whenever $\mathcal{A}$ appears more than once in a game, assume $\mathcal{A}$ saves state.

2.2 Definition of Non-Programmable Random Oracle Model

We consider security in the non-programmable RO model (NPROM), as defined by [FLR$^+$10]. The NPROM specifies a constraint on formal reductions from one interactive Turing machine to another. In our case, it constrains the relationship between our ideal-world simulator $\mathcal{S}$ and the real-world adversary $\mathcal{A}$.

Formally, the adversary $\mathcal{A}$ interacts with the RO by writing queries to/reading responses from its oracle tape. In the *programmable* RO model's ideal world, the simulator controls this tape, such that it may read $\mathcal{A}$'s queries and arbitrarily choose RO responses – namely, it may program the RO. In the *non-programmable* RO model, even in the ideal world, the adversary's tape is instead connected to an externally defined oracle $\mathcal{O}$, and $\mathcal{O}$ responds directly; see Figure 1.

The informal rationale underlying the NPROM is that it more closely resembles the real-life setting where we heuristically instantiate RO with an externally-defined hash function. From a philosophical point of view, the simulator $\mathcal{S}$ can be viewed as an explanation of the interaction observed by the real-world adversary. In the NPROM, we, as the designers of $\mathcal{S}$, must explain this interaction in the context of a hash function that we cannot control. This seems to more closely resemble real life than does the PROM because we, of course, cannot control, e.g., the definition of SHA-3.

More formally, the NPROM is motivated by a separation between it and the PROM, as there exist constructions that are (1) possible in the PROM and (2) impossible in both the standard model and NPROM, suggesting that the NPROM is closer to reality; see also discussion in Section 1.2. Our results also hold for an even weaker model called Auxiliary Input ROM from [Unr07]. This model and proof that our scheme is compatible can be found in Appendix D.

2.3 Garbling Schemes

[BHR12b] defined the notion of a *garbling scheme*, and we formalize our construction and 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 $(Garble, Encode, Eval, Decode)$ 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 garbling scheme is (perfectly) **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.

2.4 Definition of Adaptive Security

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

Definition 2 (Adaptive Privacy). *A garbling scheme is **adaptively private** if for all circuits C computing a function f, there exists a simulator $\mathcal{S}$ such that for all stateful PPT adversaries $\mathcal{A}$ the following quantity is negligible in λ:*

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

where Real, Ideal *are as follows:*

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

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

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

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

2.5 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 3 (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 $\{0, 1, \mathcal{Z}, \text{✗}\}$. The semantics of each gate type are as follows:*

$\oplus$	$\mathcal{Z}$	0	1	✗
$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	✗
0	$\mathcal{Z}$	0	1	✗
1	$\mathcal{Z}$	1	0	✗
✗	✗	✗	✗	✗

$/$	$\mathcal{Z}$	0	1	✗
$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	$\mathcal{Z}$	✗
0	$\mathcal{Z}$	$\mathcal{Z}$	0	✗
1	$\mathcal{Z}$	$\mathcal{Z}$	1	✗
✗	✗	✗	✗	✗

$\bowtie$	$\mathcal{Z}$	0	1	✗
$\mathcal{Z}$	$\mathcal{Z}$	0	1	✗
0	0	0	✗	✗
1	1	✗	1	✗
✗	✗	✗	✗	✗

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 ✗ 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 4 (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 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 5 (Oblivious tri-state circuit). *A **randomized tri-state circuit** is a pair consisting of a tri-state circuit C and a distribution of bitstrings 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:*

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

*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 $C(x)$. 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 prove it fits into our framework. Our scheme garbles the tri-state circuit (TSC) model; see Section 2.5.

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 $\mathsf{k}_w^1 = \mathsf{k}_w^0 \oplus \Delta$ 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 2.5. 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 5 for further details on our TSC handling.

3.2 Proof of Security

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

On our (non-) use of programming. Recall from Section 2.2 that the NPRO model constrains the relationship between interactive Turing machines $\mathcal{S}$ and $\mathcal{A}$. Namely, the simulator $\mathcal{S}$ may not program responses to $\mathcal{A}$'s RO queries.

In our security proof, we perform a standard real/ideal comparison, where we define hybrid distributions bridging the two worlds. In these hybrids, we reason about the content of the random oracle's truth table by "programming" it. For instance, we reason that a certain interaction with a true random oracle is statistically close to one with the same oracle, but where some rows of the truth table have been replaced by fresh randomness.

One natural philosophical question is to consider whether or not this use of "programming" should be allowed in the NPROM.

We first emphasize that this use of "programming" is *formally* in the NPROM. The model simply constrains that the *simulator* cannot program the oracle. As a proof technique, we are free to reason in our thought experiments about the syntactic content of the random oracle's truth table. 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, and the method by which we prove real-ideal indistinguishability is irrelevant.

Still, at a philosophical level, one might wonder if it is appropriate that the NPROM does not rule out the use of such proof techniques. We argue that it should not rule them out, and that our use of "programming" in thought experiments is consistent with the informal rationale underlying NPRO.

When we heuristically instantiate a hash function we, of course, cannot control that hash function's truth table. However, we *of course* can *reason* about that truth table, which is heuristically assumed to be random. When reasoning about the truth table, we can, e.g., compare to another table that is different from – but statistically close to – the original table. This is no different than standard cryptographic reasoning, for example thinking about a pseudorandom function (PRF) as if it were a truly random function, and using this reasoning in a thought experiment to "program" the PRF.

Extending this example closer to our protocol, let's reframe our NPRO-based protocol (and our proof) into a RO-less protocol involving an additional (incorruptible) party $\mathcal{R}$ who responds to players' RO queries. $\mathcal{R}$ initializes by sampling a secret PRF key k; on each query x, $\mathcal{R}$ responds with PRF output $F(k, x)$. $\mathcal{A}$ interacts directly with $\mathcal{R}$, even in the ideal world. We can reason about the truth table $F(k, \cdot)$ as if it is a random table, by the definition of PRF security. If we envision instantiating each call to our hash function with $\mathcal{R}$, our proof of security

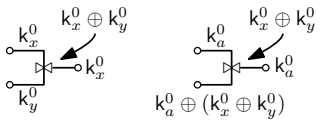
– which uses "programming" in the hybrid steps – will go through, simply by applying PRF security. Appendix E explains this reasoning in greater detail.

This example supports our thesis that it is appropriate to use programming in *reasoning* about real/ideal indistinguishability. In particular, an illuminating interpretation of the philosophy underlying the NPROM is that an RO $\mathcal{O}$ black-box instantiates this party $\mathcal{R}$, and hence we can use the same techniques to reason about $\mathcal{O}$ as we could to reason about $\mathcal{R}$.

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 $\mathcal{A}$ 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 input wires are labelled by G's keys $\mathsf{k}^0_x, \mathsf{k}^0_y$; the output is labelled by k^0_x. To enable evaluation in all cases, G includes in $\tilde{C}$ the string $\mathsf{k}^0_x \oplus \mathsf{k}^0_y$. If E only holds a runtime key k_y for the bottom input, it can use this string to *translate* that input key to an appropriate output:

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

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

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

To show security, we therefore must show that such a guess is unlikely, in the sense that the garbled circuit $\tilde{C}$ does not "help" $\mathcal{A}$ to guess problematic queries. More precisely, we formalize a property – query hiding – whereby $\mathcal{A}$ should not be able to detect when we *remove* from the random oracle all queries issued by Garble; see Section 4. At the very highest level, we show that our TSC garbling scheme is query-hiding due to the fact that all keys are uniformly chosen, and not included in the GC itself. By plugging together the ability to rekey the garbled circuit with query hiding, we are able to obtain a proof of adaptive security.

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

4 Adaptive Security of Rekeyable Garbling Schemes

This section formalizes **query-hiding garbling schemes** and **rekeyable garbling schemes**, and we show schemes satisfying these notions are adaptively secure. In Section 5, we will see that our garbling of TSCs is query-hiding and rekeyable; Appendix C discusses other garbling schemes that satisfy our notions.

4.1 Additional Notation

Adaptive GC splits evaluation into an offline and an online phase. $\mathcal{A}$ accesses the RO even in the offline phase, when it has seen the circuit garbling, but before

we send the encoded input. In our security proof, we show that the adversary cannot detect if we remove oracle queries used to garble the circuit. This is useful, because it allows us to reason that $\mathcal{A}$ cannot use work it performs in the offline phase to help it distinguish in the online phase. Formalizing this requires us to change rows of an RO, so we define appropriate notation.

Notation 1. *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. We denote by $\mathcal{O}^\delta$ the following programming of the oracle's truth table:*

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

We will also sometimes rerandomize certain rows in $\mathcal{O}$'s truth table:

Notation 2. *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. We denote by $\mathcal{O}^{-\delta}$ the following rerandomization of the oracle's truth table:*

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

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

It will also be convenient to extract from the formal procedure Garble its oracle queries. From here on, we will write $\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$ to denote that δ captures RO queries/responses occurring in Garble.

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

Notation 3 (Garble Support)**.** *Let $C \in \mathcal{C}$ be a circuit. Let $\mathrm{Supp}(\mathrm{Garble}(1^\lambda, C))$ denote the **support** of Garble on input C. Formally, $\mathrm{Supp}(\mathrm{Garble}(1^\lambda, C))$ is the support of the distribution defined by the procedure that (1) samples a fresh RO $\mathcal{O}$, (2) uses $\mathcal{O}$ to construct a garbled circuit and associated encoding/decoding strings $(\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$, and (3) returns $\tilde{C}$:*

4.2 Properties on Garbling Schemes

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

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

Definition 6 (Query Hiding). *Let Π be a garbling scheme. We say that Π is **query hiding** if for all polynomially-query-bounded adversaries $\mathcal{A}$ and all circuits C, then for the following game:*

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

The probability that $\mathcal{A}$ makes a query in δ is $\mathsf{negl}(\lambda)$. That is, an adversary that only observes $\tilde{C}$ cannot make queries that the garbler made.

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

Definition 7 (Rekeyability). *Let Π be a garbling scheme. Π is **rekeyable** if the following holds. There must exist a poly-time*[8] *procedure Rekey:*

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

On input a circuit C and a garbled circuit $\tilde{C}$, Rekey outputs a query map δ, an input encoding string e, and an output decoding string d. The ensembles described by the following experiments must be identically distributed:

$\mathit{Rekeyable}^R(1^\lambda, C)$		$\mathit{Rekeyable}^I(1^\lambda, C)$	
1 :	*Sample random oracle* $\mathcal{O}$	1 :	*Sample random oracle* $\mathcal{O}$
2 :	$\delta, (\tilde{C}, e, d) \leftarrow \mathit{Garble}^{\mathcal{O}}(1^\lambda, C)$	2 :	$\delta, (\tilde{C}, e, d) \leftarrow \mathit{Garble}^{\mathcal{O}}(1^\lambda, C)$
3 :		3 :	$(\delta', e', d') \leftarrow \mathit{Rekey}(1^\lambda, C, \tilde{C})$
4 :	**return** $(\delta, \tilde{C}, e, d)$	4 :	**return** $(\delta', \tilde{C}, e', d')$

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

Definition 8 (Privately Rekeying). *Let Π be a rekeyable garbling scheme (Definition 7). Π **privately rekeys** if for all circuits C computing function f, there exists a simulator $\mathcal{S}$ s.t. for all x, all GCs $\tilde{C} \in \mathit{Supp}(\mathit{Garble}(1^\lambda, C))$, and all* PPT *adversaries $\mathcal{A}$, the following ensembles are statistically close in λ:*

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

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

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

4.3 Adaptive Security

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

Theorem 1 (Adaptive Privacy from Private Rekeying and Query Hiding). *Let Π be a privately rekeyable (Definition 8) garbling scheme that is query hiding (Definition 6). Π is adaptively private.*

Proof. By construction of a simulator:

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

Here, $\mathcal{S}_p$ is the privacy simulator $\mathcal{S}$ provided by Definition 8. In the offline phase, $\mathcal{S}$ simply garbles a circuit normally. In the online phase, $\mathcal{S}$ (1) encodes the all-zeros input and (2) uses Π's privacy simulator $\mathcal{S}_p$ to forge an output decoding string d that convincingly decodes to the correct output $f(x)$.

Now, we show that the real-world and ideal-world experiments are indistinguishable. In Figure 2, we restate adaptive privacy's real/ideal experiments, in-lining the definition of our simulator. Figure 2 also specifies six hybrid distributions used to show indistinguishability.

We find most direct to argue by "meeting in the middle". Namely, we proceed starting from the real/ideal ensemble, and at each step, we show the current real/ideal ensemble is indistinguishable from some respective intermediate ensemble. We conclude by showing these two final ensembles are indistinguishable.

$\mathrm{Real}^{\mathcal{A},C}_{\mathrm{prv}}(\lambda)$	$\mathrm{Ideal}^{\mathcal{A},C}_{\mathrm{prv}}(\lambda)$
Sample random oracle $\mathcal{O}$	Sample random oracle $\mathcal{O}$
$(\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$	$(\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$
$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
$\tilde{x} \leftarrow \mathrm{Encode}(e, x)$	$\tilde{x} \leftarrow \mathrm{Encode}(e, \mathbf{0})$
	$d' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d)$
return $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d)$	**return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d')$

$\mathrm{F}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)$	$\mathrm{F}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)$
Sample random oracle $\mathcal{O}$	Sample random oracle $\mathcal{O}$
$\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$	$\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$
$x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$	$x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$
$\tilde{x} \leftarrow \mathrm{Encode}(e, x)$	$\tilde{x} \leftarrow \mathrm{Encode}(e, \mathbf{0})$
	$d' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d)$
return $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d)$	**return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x}, d')$

$\mathrm{G}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)$	$\mathrm{G}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)$
Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$	Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$
$\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}'}(1^\lambda, C)$	$\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}'}(1^\lambda, C)$
$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
$\tilde{x} \leftarrow \mathrm{Encode}(e, x)$	$\tilde{x} \leftarrow \mathrm{Encode}(e, \mathbf{0})$
	$d' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d)$
return $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d)$	**return** $\mathcal{A}^{\mathcal{O}^\delta}(1^\lambda, \tilde{x}, d')$

$\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)$	$\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)$
Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$	Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$
$\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}'}(1^\lambda, C)$	$\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}'}(1^\lambda, C)$
$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	$x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
$(\delta', e', d') \leftarrow \mathrm{Rekey}(1^\lambda, C, \tilde{C})$	$(\delta', e', d') \leftarrow \mathrm{Rekey}(1^\lambda, C, \tilde{C})$
$\tilde{x} \leftarrow \mathrm{Encode}(e', x)$	$\tilde{x} \leftarrow \mathrm{Encode}(e', \mathbf{0})$
	$d'' \leftarrow \mathcal{S}_p(1^\lambda, f(x), d')$
return $\mathcal{A}^{\mathcal{O}^{\delta'}}(1^\lambda, \tilde{x}, d')$	**return** $\mathcal{A}^{\mathcal{O}^{\delta'}}(1^\lambda, \tilde{x}, d'')$

Fig. 2. Real, ideal, and hybrid distributions used in our proof of Theorem 1.

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

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

In this step we remove from $\mathcal{A}$'s oracle all queries issued by the call to Garble, but only in the offline phase. Jumping ahead, our informal objective is to show that $\mathcal{A}$'s chosen input x must be independent of Garble's random oracle queries.

Of course, there is a difference between games Real and $\mathrm{F}_{\mathrm{prv},R}$ (resp. Ideal and $\mathrm{F}_{\mathrm{prv},I}$). In the former game $\mathcal{A}$ is given access to the same oracle twice, and in the latter $\mathcal{A}$ is given access to two oracles that differ by δ. Thus, if $\mathcal{A}$ can guess a query in δ, it can distinguish the two games, since in the first game it will see the oracle respond consistently in the offline and online phases, and in the second game it will see the oracle "disagree with itself".

Query hiding (Definition 6) is precisely what we need to show that $\mathcal{A}$ cannot guess a query in δ. Indeed, an unbounded adversary with only polynomial oracle queries has at most $\mathsf{negl}(\lambda)$ chance to sample a point in δ. As such, $\mathcal{A}$ only has a $\mathsf{negl}(\lambda)$ probability of determining if it was given $\mathcal{O}$ or $\mathcal{O}^{-\delta}$ on line 3.

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

$$\{\mathrm{F}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)\} \equiv \{\mathrm{G}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)\} \qquad \text{and} \qquad \{\mathrm{F}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)\} \equiv \{\mathrm{G}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)\}$$

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

Rekeying the GC. We use rekeyability (Definition 7) to sample fresh keys associated with the garbled circuit $\tilde{C}$. The following is immediate by rekeyability:

$$\{\mathrm{G}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)\} \equiv \{\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)\} \qquad \text{and} \qquad \{\mathrm{G}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)\} \equiv \{\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)\}$$

Privacy. Finally, we bridge from the "real world", where we encode x, to the "ideal world", where we encode 0: $\{\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},R}(\lambda)\} \overset{\mathrm{s}}{\approx} \{\mathrm{H}^{\mathcal{A},C}_{\mathrm{prv},S}(\lambda)\}$. Notice that the outputs of garbling – $\delta, \tilde{C}, e, d$ – are independent of $\mathcal{O}$. Thus, the choice of x is also independent of δ, e, d, except insofar as they are constrained by garbled circuit $\tilde{C}$. We can capture this sentiment by treating $\tilde{C}$ and x as if they are universally quantified. This is exactly the setting considered in private rekeying (Definition 8). Applying private rekeying thus discharges the proof. Any query hiding, privately rekeyable garbling scheme is adaptively private. □

5 Our Adaptively-Secure Garbling of Tri-State Circuits

This section formalizes our handling of tri-state circuits as a garbling scheme (Definition 1), and it proves the scheme is query hiding and rekeyable.

Construction 1 (Garbled TSCs from NPRO)**.** *Our garbling scheme is the collection of algorithms* $(\mathit{Garble}, \mathit{Encode}, \mathit{Eval}, \mathit{Decode})$ *described in the following.*

For completeness, Appendix B gives a more direct specification of our algorithms, but all relevant details are given in the following text. 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]. Free XOR starts by sampling a single global *offset* $\Delta \leftarrow\!\!\$\ \{0,1\}^{\lambda-1}1$. Δ is chosen by the garbler G and is hidden from the evaluator E. For each wire w, the G maintains a key $\mathsf{k}_w^0 \in \{0,1\}^\lambda$ encoding 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 a 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 holds *one* key. E cannot distinguish a 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 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 4). Hence, we do not need to encode ✗.

Point and Permute. Δ has least significant bit (lsb) 1. This ensures that for each wire w, the lsb of zero-key k_w^0 and of one-key $\mathsf{k}_w^0 \oplus \Delta$ are different. This allows us to view the lsb of zero-key k_w^0 and the lsb of active key k_w as an *XOR share* of w's value. This classic trick is 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 5), it is safe to reveal these values: the information E learns can be simulated.

Distribution sampling. Recall that an oblivious tri-state circuit (Definition 5) consists of two parts: a circuit C and a *distribution* on bits D. To evaluate C, we must sample D. This is straightforward: G locally samples $r \leftarrow\!\!\$\ D$, then encodes r to ensure that E's encoding of each random input is the all zero key.

Gate handling. Recall (from Section 2.5) 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: $\mathsf{k}_z^0 = \mathsf{k}_x^0 \oplus \mathsf{k}_y^0$. 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 5).

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 4).

Order of evaluation. Recall that in TSCs, gates can fire in data-dependent orders. Thus, we must specify how E chooses which gate to activate at each step. At each step of evaluation, our E will 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 9 (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 is $O((n+m) \cdot \lambda)$.

Garbling scheme procedures. Our garbling scheme procedures, given in Appendix B, 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.

Decode includes two extra details: (1) we store in the decoding string d not the output keys themselves, but rather applications of RO to those keys, and (2) we include in d the lsb of the zero key. These details are respectively needed to ensure the scheme is *authentic* [BHR12b] and *perfectly correct*.

5.1 Tri-State Circuit Adaptive Security

We show Construction 1 is query hiding and can be rekeyed privately. For simplicity, we refer to Construction 1 as Π_{TSC}. Combined with the results in this section, Theorem 1 implies that Π_{TSC} is an adaptive private garbling scheme.

Lemma 1 (Π_{TSC} is Query Hiding). *Π_{TSC} is query hiding (Definition 6).*

Proof. Consider the garbling of a circuit $\delta, (\tilde{C}, e, d) \leftarrow \text{Garble}^{\mathcal{O}}(1^\lambda, C)$. Recall that query hiding roughly states that $\mathcal{A}$ cannot distinguish two worlds. In the first world, we give $\mathcal{A}$ $\tilde{C}$ and oracle access to $\mathcal{O}$; in the second, we give the advesary $\tilde{C}$ and oracle access to $\mathcal{O}^{-\delta}$ – an oracle with Garble's queries removed.

The challenge in proving this property is that the GC $\tilde{C}$ might provide to $\mathcal{A}$ some "help" in making problematic oracle queries. We must show that this is *not* the case, and that $\mathcal{A}$ gains from $\tilde{C}$ no advantage in guessing queries in δ.

We first note that the offline garbled circuit $\tilde{C}$ consists only of (1) join-gate ciphertexts of the form $\mathsf{k}_x^0 \oplus \mathsf{k}_y^0$ and (2) point and permute bits associated with control keys to buffer gates. We also note that all queries to $\mathcal{O}$ involve a key. To show query hiding, we reduce $\mathcal{A}$'s ability to guess a query to $\mathcal{A}$'s ability to guess either Δ, or some wire key – say, the first input key k_0^0.

Consider the following thought experiment: Rather than giving $\tilde{C}$ to $\mathcal{A}$, suppose that for each wire x we give to $\mathcal{A}$ (1) the point and permute bit of k_x^0 and (2) a ciphertext $\mathsf{k}_0^0 \oplus \mathsf{k}_x^0$. Note that $\mathcal{A}$ could construct $\tilde{C}$ from this information, since garbled material joining arbitrary wires x and y can be expressed as follows:

$$\mathsf{k}_x^0 \oplus \mathsf{k}_y^0 = (\mathsf{k}_0^0 \oplus \mathsf{k}_x^0) \oplus (\mathsf{k}_0^0 \oplus \mathsf{k}_y^0)$$

Thus, the information conveyed to $\mathcal{A}$ in this experiment is strictly more than that in the garbled circuit, since we can construct the latter from the former.

In this experiment, we can reason that if $\mathcal{A}$ finds *any* key k_x^b, then they can compute a key on the first wire: $\mathsf{k}_0^b = (\mathsf{k}_0^0 \oplus \mathsf{k}_x^0) \oplus \mathsf{k}_x^b$. Thus, an adversary that guesses some query made by $\text{Garble}^{\mathcal{O}}$ can also output k_0^b for some b with the same probability. But k_0^0 and k_0^1 are uniform in the choices of k_0^0 and $\Delta = \mathsf{k}_0^1 \oplus \mathsf{k}_0^0$ (Figure 4). Thus, $\mathcal{A}$ gains no more than $\mathsf{negl}(\lambda)$ advantage in guessing k_0^0 or k_0^1, implying that $\mathcal{A}$ cannot guess a query given only $\tilde{C}$ w.p. better than $\mathsf{negl}(\lambda)$.

Thus, this worst-case experiment hides garbling queries. Since the worst-case world gives strictly more information to $\mathcal{A}$ than $\tilde{C}$, Π_{TSC} is query hiding. □

Lemma 2. *Π_{TSC} is rekeyable (Definition 7).*

Proof. By construction of a rekeying procedure Rekey; see Figure 3.

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

More precisely, Garble and Rekey sample keys from the same distribution. Indeed, Δ is identically distributed, and each zero-key is also identically distributed. The only difference between the two procedures is that Garble chooses

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

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

Fig. 3. The Rekey procedure for Construction 1. Rekey uniformly samples keys, subject to linear constraints imposed by $\tilde{C}$. It outputs (1) a programming string δ, (2) an input encoding string e, and (3) an output decoding string d. 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.

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, 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 $\tilde{C}$. □

Lemma 3 (Π_{TSC} Privately Rekeys). *Π_{TSC} privately rekeys (Definition 8).*

Proof. Privately rekeying (Definition 8) requires that we show indistinguishability of the following for all $\tilde{C} \in \text{Supp}(\text{Garble}(1^\lambda, C))$ and inputs x:

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

We introduce similar games which convey the intuition that $\mathcal{A}$ cannot distinguish when it is not allowed to decrypt the output of the GC (it is not given d). These are similar to the notion of GC *obliviousness* [BHR12b]:

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

Privacy from obliviousness. In our scheme Π_{TSC}, if Real_{obv} and Ideal_{obv} are indistinguishable, then we almost immediately achieve privacy as well. In particular, our simulator $\mathcal{S}$ in Ideal_{prv} takes the decoding table d and permutes the entries to find d' so that the encoded output properly decodes:

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

Thus, it suffices to show that Real_{obv} and Ideal_{obv} are indistinguishable.

Uniform Δ. Consider the information revealed to $\mathcal{A}$ in the online phase, for now ignoring $\mathcal{A}$'s RO queries. $\mathcal{A}$ is given the following:

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

for every wire w_i with cleartext wire value b_i. The TSC is total (Definition 4), so $\mathcal{A}$ indeed sees a key on *each* wire. Note that the control bits $\text{Controls}(C, x)$ (resp. $\text{Controls}(C, \mathbf{0})$) are implied by the keys on control wires $\mathsf{k}_c^{b_c}$ together with revealed buffer point-and-permute bits included as part of $\tilde{C}$.

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

Game Indistinguishability. Without RO queries, we note that the distributions $\text{Controls}(C, x)$ and $\text{Controls}(C, \mathbf{0})$ are statistically close (Definition 5); the one way $\mathcal{A}$ can get a larger advantage is by querying the RO at $\mathcal{O}(\mathsf{k}_c^1 || gid)$ for some wire w_c that controls a open buffer, i.e. a buffer where $\mathcal{A}$ was given key k_c^0.

With this query $\mathcal{A}$ would be able to determine whether $\mathsf{k}_i^{b_i} = \mathcal{O}(\mathsf{k}_c^1 || gid) \oplus \mathsf{k}_j^{b_j}$ for data input (resp. output) wire w_i (resp. w_j) associated with the buffer. If any such query was leaked to $\mathcal{A}$, then $\mathcal{A}$ could distinguish the control bits; otherwise,

$\mathcal{A}$ would have no help distinguishing the two games Real_{obv} or Ideal_{obv}, since the keys together with $\tilde{C}$ are independent of the evaluation.

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

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

Corollary 1. *Π_{TSC} is an adaptively private garbling scheme.*

References

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Berlin, Heidelberg, August 2013.

[BBK+23] Estuardo Alpírez Bock, Chris Brzuska, Pihla Karanko, Sabine Oechsner, and Kirthivaasan Puniamurthy. Adaptive distributional security for garbling schemes with o (— x—) online complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 139–171. Springer, 2023.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Berlin, 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, Berlin, 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, Berlin, 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, Berlin, 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, Berlin, 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, Berlin, Heidelberg, May 2014.

[GKW+20] Chun Guo, Jonathan Katz, Xiao Wang, Chenkai Weng, and Yu Yu. Better concrete security for half-gates garbling (in the multi-instance setting). In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 793–822. Springer, Cham, August 2020.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. In Venkatesan Guruswami, editor, *56th FOCS*, pages 210–229. IEEE Computer Society Press, October 2015.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 449–458. ACM Press, June 2015.

[GOS18] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 515–544. Springer, Cham, August 2018.

[GYW+23] Xiaojie Guo, Kang Yang, Xiao Wang, Yu Yu, and Zheli Liu. Unmodified half-gates is adaptively secure - so is unmodified three-halves. Cryptology ePrint Archive, Report 2023/1528, 2023.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 3–31. Springer, Cham, May 2024.

[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, Berlin, 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, Cham, May / June 2022.

[HKO23] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Tri-state circuits - A circuit model that captures RAM. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 128–160. Springer, Cham, August 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, Cham, December 2020.

[JSW17] Zahra Jafargholi, Alessandra Scafuro, and Daniel Wichs. Adaptively indistinguishable garbled circuits. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 40–71. Springer, Cham, November 2017.

[JW16] Zahra Jafargholi and Daniel Wichs. Adaptive security of Yao's garbled circuits. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 433–458. Springer, Berlin, 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, Cham.

[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, Berlin, 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, Berlin, 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, Berlin, Heidelberg, August 2002.

[PLS23] Andrew Park, Wei-Kai Lin, and Elaine Shi. NanoGRAM: Garbled RAM with $\widetilde{O}(\log N)$ overhead. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 456–486. Springer, Cham, April 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, Cham.

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 205–223. Springer, Berlin, Heidelberg, August 2007.

[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, Berlin, Heidelberg, April 2015.

Appendices

A Adaptive Obliviousness

Definition 10 (Adaptive Obliviousness). *A garbling scheme satisfies **adaptive obliviousness** if for all circuits $C \in \mathcal{C}$, there exists a simulator $\mathcal{S}$, such that for all stateful PPT adversaries $\mathcal{A}$ the following quantity is negligible in λ:*

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

where Real, Ideal *are as follows:*

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

Definition 11 (Obliviously Rekeying). *Let Π be a rekeyable garbling scheme (Definition 7). We say that Π **obliviously rekeys** if for all circuits C computing function f, for all x, for all garbled circuits $\tilde{C} \in \mathit{Supp}(\mathit{Garble}(1^\lambda,C))$, and for all* PPT *adversaries $\mathcal{A}$, the following ensembles are statistically close in λ:*

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

Lemma 4 ($\Pi_{\mathbf{TSC}}$ Obliviously Rekeys). *Π_{TSC} obliviously rekeys (Definition 11).*

Proof. Recall the hybrid of oblivious rekeying (Definition 11), which we would like to show indistinguishability for all $\tilde{C} \in \mathrm{Supp}(\mathrm{Garble}(1^\lambda,C))$ and inputs x.

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

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

for every wire w_i and hidden semantic value over the wire b_i, recalling that TSC is total (Definition 4). Note that the control bits $\text{Controls}(C, x)$ or $\text{Controls}(C, \mathbf{0})$ are implied by having access to the keys $\mathsf{k}_c^{b_c}$ associated with control wires w_c and the buffer leaked pnp bit from $\tilde{C}$.

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

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

With this query the adversary would be able to determine whether

$$\mathsf{k}_i^{b_i} = \mathcal{O}(\mathsf{k}_c^1 || gid) \oplus \mathsf{k}_j^{b_j},$$

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

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

Theorem 2 (Adaptive Obliviousness from Oblivious Rekeying and Query Hiding). *Let Π be an obliviously rekeyable (Definition 11) garbling scheme that is query hiding (Definition 6). Π is adaptively oblivious.*

Proof. By construction of a simulator:

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

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

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

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

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

$\mathrm{F}^{\mathcal{A},C}_{\mathrm{obv},R}(\lambda)$	$\mathrm{F}^{\mathcal{A},C}_{\mathrm{obv},S}(\lambda)$
1 : Sample random oracle $\mathcal{O}$	1 : Sample random oracle $\mathcal{O}$
2 : $\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$	2 : $\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$
3 : $x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$	3 : $x \leftarrow \mathcal{A}^{\mathcal{O}^{-\delta}}(1^\lambda, \tilde{C})$
4 : $\tilde{x} \leftarrow \mathrm{Encode}(e, x)$	4 : $\tilde{x} \leftarrow \mathrm{Encode}(e, \mathbf{0})$
5 : **return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x})$	5 : **return** $\mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{x})$

$\mathrm{G}^{\mathcal{A},C}_{\mathrm{obv},R}(\lambda)$	$\mathrm{G}^{\mathcal{A},C}_{\mathrm{obv},S}(\lambda)$
1 : Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$	1 : Sample random oracles $\mathcal{O}$ and $\mathcal{O}'$
2 : $\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}'}(1^\lambda, C)$	2 : $\delta, (\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}'}(1^\lambda, C)$
3 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	3 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
4 : $\tilde{x} \leftarrow \mathrm{Encode}(e, x)$	4 : $\tilde{x} \leftarrow \mathrm{Encode}(e, \mathbf{0})$
5 : **return** $\mathcal{A}^{\mathcal{O}^{\delta}}(1^\lambda, \tilde{x})$	5 : **return** $\mathcal{A}^{\mathcal{O}^{\delta}}(1^\lambda, \tilde{x})$

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

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

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

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

In this step we remove from the adversary's oracle all queries issued by the call to Garble, but only in the offline phase. Jumping ahead, our informal objective is to show that the adversary's chosen input x must be independent of Garble's random oracle queries.

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

Query hiding (Definition 6) is precisely what we need to show that the adversary cannot guess a query in δ. Indeed, an unbounded adversary with only polynomial oracle queries has at most $\mathsf{negl}(\lambda)$ chance to sample a point in δ. As such, the adversary only has a $\mathsf{negl}(\lambda)$ probability of determining if it was given $\mathcal{O}$ or $\mathcal{O}^{-\delta}$ on line 3 of both the Real games and G.

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

$$\{\mathrm{F}^{\mathcal{A},C}_{\mathrm{obv},R}(\lambda)\} \equiv \{\mathrm{G}^{\mathcal{A},C}_{\mathrm{obv},R}(\lambda)\} \qquad \text{and} \qquad \{\mathrm{F}^{\mathcal{A},C}_{\mathrm{obv},S}(\lambda)\} \equiv \{\mathrm{G}^{\mathcal{A},C}_{\mathrm{obv},S}(\lambda)\}$$

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

Rekeying the garbled circuit. Next, we use the rekeyability (Definition 7) of the scheme to sample fresh keys associated with the particular garbled circuit $\tilde{C}$. In particular, we argue:

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

This is immediate by the rekeyability property.

Privacy. Finally, we bridge from the "real world", where we encode x, to the "ideal world", where we encode 0:

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

First, notice that the outputs of garbling – $\tilde{C}$, δ, e, and d – are independent of $\mathcal{O}$. Thus, the choice of x is also independent of δ, e, and d, except insofar as they are constrained by the garbled circuit $\tilde{C}$. We can capture this sentiment by treating $\tilde{C}$ and x as if they are universally quantified. But this setting is exactly what is considered in oblivious rekeying (Definition 11). Applying oblivious rekeying thus discharges the proof.

Any query hiding, obliviously rekeyable garbling scheme is adaptively oblivious.

□

B TSC Garbling Procedures

Our NPRO-based TSC garbling scheme is described in full detail in Section 2.3. For completeness, Figure 4 additionally includes precise algorithms for each garbling scheme procedure.

C Existing Schemes are Adaptive

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

C.1 Free XOR

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

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

For each XOR gate $z \leftarrow x \oplus y$, the garbler sets the output wire key $\mathsf{k}^0_z = \mathsf{k}^0_x \oplus \mathsf{k}^0_y$. For each AND gate $z \leftarrow x \cdot y$, the garbler generates four garbled rows[9],

[9] 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.

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

$\tilde{C} \leftarrow \textit{emptymap}$
$e, d \leftarrow \textit{emptyvec}$
$\Delta \leftarrow\!\!\$\ \{0,1\}^{\lambda-1} \mid\mid 1$
$r \leftarrow\!\!\$\ D$
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. 4. 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 9). Our scheme is adaptively secure.

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 5. *Let Π denote the garbling scheme of [KS08] described above. Π is query hiding (Definition 6).*

Proof Sketch. The proof that keys garbled in [KS08] hide queries is very similar, if not simpler, than for TSC (Lemma 1). Like with TSC, keys are generated either from uniform, or as a linear combination of other keys. Note that all the ciphertexts in $\tilde{C}$ given to the adversary are of the form $\mathcal{O}(q) \oplus \mathsf{k}_x^b$ for some query q. When the adversary is allowed no queries, then $\tilde{C}$ perfectly encrypts what the keys could be, since the outputs of the RO is uniform. Informally, this continues to be the case until the adversary hits a query q used to encrypt some key. Since the keys are uniform or linear in nature, we find that the adversary cannot guess any of the keys and by extension any oracle query with better than w.p. $\mathsf{negl}(\lambda)$. □

Lemma 6. *Let Π denote the garbling scheme of [KS08] described above. Π supports a rekey procedure (Definition 7) that privately and obliviously rekeys (Definitions 8 and 11).*

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

The procedure described above is a rekeyer (Definition 7): the keys are clearly drawn from the same distribution in the garbling and the rekeying. The proof that this rekey procedure satisfies rekey obliviousness and privacy is almost identical to the proof given in Lemma 2, except that the proof is arguably simpler, since we need not concern ourselves with tri-state circuit obliviousness. □

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

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

C.2 Half-Gates is Rekeyable

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

Lemma 7. *Let Π denote the garbling scheme of [ZRE15]. Π is query hiding (Definition 6) and supports a rekey procedure (Definition 7) that privately and obliviously rekeys (Definitions 8 and 11).*

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

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

C.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 8. *Let Π denote the garbling scheme of [BMR16]. Π is query hiding (Definition 6).*

Proof Sketch. Because the garbled circuit $\tilde{C}$ is a collection of RO outputs XORed with the linear keys in $\mathbb{Z}_p^\ell$, we find that the oracle queries that take in these linear and uniform keys should be hidden for exactly the same reason as in Lemma 5. □

Lemma 9. *Let Π denote the garbling scheme of [BMR16]. Π supports a rekey procedure (Definition 7) that privately and obliviously rekeys (Definitions 8 and 11).*

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

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

C.4 Switch Systems are Rekeyable

The switch system garbling scheme of [Hea24] was inspired by TSCs, with the extra insight that most of the gates can be bi-directional and arithmetic. TSCs are actually a special case of a switch system. Switch systems are also rekeyable for exactly the same reasons that TSCs are, since switch systems are still garbled in a TSC-like order with XOR replaced with addition. We chose to work with TSCs in this paper because they are closer to a traditional circuit-garbling scheme and still support GRAM.

Lemma 10. *Let Π denote the garbling scheme of [Hea24]. Π is query hiding (Definition 6) and supports a rekey procedure (Definition 7) that privately and obliviously rekeys (Definitions 8 and 11).*

Proof Sketch. The proof that the switch system technique of [Hea24] is rekeyable is very similar to our TSC proof, with the linearity in $\mathbb{Z}_2$ replaced with linearity in $\mathbb{Z}_{2^k}$.

Rather than meticulously proving that switch systems are rekeyable and query hiding, we simply point out that these properties are implied by Lemmas 1 and 2. Thus, the [Hea24] scheme is adaptively secure if its hash function is modeled by a NPRO. □

C.5 Three-Halves is Rekeyable

The three-have construction of [RR21] can be viewed in two parts: a method for correlating half-sized labels with a hash function that results in statistically-hiding the inputs and gate (given a controlled view) *and separately* the implementation of this hash function with some appropriate primitive (e.g. correlation-robust hash functions). To put more formally, the slice and dice method is statistically secure if we only give the adversary hash queries that correspond to an honest evaluation. As such, the elements in the slice and dice method are statistically rekeyable since there should exist with high probability another gate, input, and randomness pair that explains the information that the adversary

sees. As such, when we use an NPRO in place of the hash function, we note that we can program the output corresponding with some other circuit with statistical success for honest queries and with further negligible loss in security when the adversary can query on polynomially many points[10].

Lemma 11. *Let Π denote the garbling scheme of [RR21]. Π is query hiding (Definition 6).*

Proof Sketch. Because the garbled circuit $\tilde{C}$ is a collection of RO outputs and a solution to a private matrix, all the labels are linearly dependent by some uniform and unknown value to the adversary. As such, the keys are hidden for exactly the same reason as in Lemma 5. □

Lemma 12. *Let Π denote the garbling scheme of [RR21]. Π supports a rekey procedure (Definition 7) that privately and obliviously rekeys (Definitions 8 and 11).*

Proof Sketch. As discussed above, the protocol is rekeyable since for any view, there statistically exists other secrets that explain the view but are consistent with every other combination of input key and gate semantics.

Thus, the [RR21] scheme is adaptively secure if their hash function is modeled by a NPRO. □

D Auxiliary Input Random Oracle Model

Our framework can also be easily adapted to work even in the (non-programmable) auxiliary input ROM of [Unr07], in which the polynomial-time adversary is allowed to depend arbitrarily on the sampled RO.[11] This model is more restrictive than the NPROM in that it captures adversaries that know properties of the hash function in advance. The model rules out protocols that, for example, are secure in the NPROM, but rely on concrete queries resulting in uniform outputs. For instance, if a protocol relies on $\mathcal{O}(0)$ to output some uniform value, this could be a problem in the real world, where a fixed hash function always return the same string, which may lead to prepossessing attacks.

D.1 Security in the Auxiliary Input NPROM

Rather than re-proving each of our theorems, we argue the main reasons that our results clearly work in the auxiliary input model. [Unr07] shows an equivalence between their model and one in which some polynomial number of entries in

[10] In the proof of security in [RR21], the adversary would be able to break the protocol if they were able to get the labels corresponding to some other input, hence we can say that they are not findable when we use an RO.

[11] For unbounded adversaries with polynomial numbers of queries, the adversary instead receives a polynomially-sized string that depends arbitrarily on the sampled RO.

the RO are fixed in advance (and known to the adversary), and where all other entries are drawn uniformly.

Recall Definition 6, which forces that the queries a garbler makes not be guessable by an unbounded adversary with polynomially many queries to the RO. Say for reference that the adversary always queries the pre-set points in the RO (which could be arbitrary, but poly-bounded in size). This implies that garbling schemes that satisfy Definition 6 avoid hitting the bad pre-set points in the RO. Because oracle queries that are not pre-set are uniformly drawn, and because our framework runs either the honest or resamples some other honest execution of the protocol conditionally, the protocol is negligibly likely to hit one of these bad pre-set queries throughout the hybrids, and of course in the real or ideal worlds.

E Additional Discussion of NPROM

NPRO as a Hash Function. We expand our discussion of the NPROM. The intuition behind the ROM and indeed the [FLR$^+$10] formalism is that we would like to model a hash function that the adversary does not look inside. We can formalize this notion by imagining the ROM as a third party that runs a PRF with some key that is hidden from every party (even the game itself). In the programmable ROM's ideal world, we corrupt this third party, which corresponds to corrupting the real-world hash function, an unusual assumption. NPROM circumvents this by forcing the third party to be honest and to answer queries even in the ideal world (where the simulator does not get the PRF key). In other words, the NPROM can be understood as viewing a hash function as some obfuscation of a keyed PRF, where the key was chosen when the function was created. For example, we can imagine that the secret key that SHA-3 hides was chosen when SHA-3 was designed, and it is now hidden from everyone.

It is this model that [FLR$^+$10] captures. For example, let $G_L^{(\cdot)}(\lambda)$ and $G_L^{(\cdot)}(\lambda)$ be some games that each use an oracle (as the third party), and suppose we wish to show indistinguishability between them. Then for a properly sampled key k for the PRF, the following is a standard definition that we can reason about:

$$G_L^{F_{\mathsf{k}}(\cdot)} \stackrel{?}{\approx} G_R^{F_{\mathsf{k}}(\cdot)}$$

Recall that in the definition of a PRF, a keyed PRF (whose key is uniform and hidden from the adversary) should be indistinguishable from a truly uniform function. As such, we have that a valid proof of this definition might proceed as follows:

$$G_L^{F_{\mathsf{k}}(\cdot)} \approx G_L^{f(\cdot)} \equiv G_L^{\mathcal{O}(\cdot)} \stackrel{?}{\approx} G_R^{\mathcal{O}(\cdot)} \equiv G_R^{f(\cdot)} \approx G_R^{F_{\mathsf{k}}(\cdot)}$$

We then note that definitions of the form

$$G_L^{\mathcal{O}(\cdot)} \stackrel{?}{\approx} G_R^{\mathcal{O}(\cdot)}$$

for games G_L and G_R using a fixed NPROM is exactly what the [FLR+10] framework is designed to support. Put another way, [FLR+10] is sufficient also as a model for fixed uniform functions even for standard model proofs and even for any arbitrary method for showing indistinguishability between $G_L^{\mathcal{O}(\cdot)}$ and $G_R^{\mathcal{O}(\cdot)}$.

Implications for Simulation- and Indistinguishability-based Security. One other unintuitive and seemingly problematic notion that arises from [FLR+10] in garbling is that PROM simulation-based security immediately implies NPROM indistinguishability-based security for adaptive garbled circuits. To give more context, in an indistinguishability-based definition as in [JSW17], the adversary provides two inputs x_0 and x_1 such that $f(x_0) = f(x_1)$ for the chosen functionality, and it must distinguish two real-worlds in which the challenger either provides labels for x_0 or for x_1. The lower bound from [AIKW13] does not apply to protocols in the indistinguishability-based definitions; however, the potentially unintuitive behavior comes from the following analysis. Say that we have a simulation-based garbling scheme such that

$$\text{Real}_{\mathcal{A}}(\lambda) \approx \text{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$$

for some efficient simulator $\mathcal{S}$, programming or non-programming. Then we can show indistinguishability-based security for the same protocol using the following sketch. Let $\text{Real}_{\mathcal{A},b}$ be the real world in which the adversary submits two inputs as mentioned and the b-th input is always chosen to encode by the challenger, then

$$\text{Real}_{\mathcal{A},0}(\lambda) \approx \text{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda) \approx \text{Real}_{\mathcal{A},1}(\lambda).$$

If we carry out the above transformation with a protocol which uses the NPROM and is simulation secure, the implied protocol for indistinguishability-based security will be left with the same online-phase cost; however, if we start with a PROM simulation-secure protocol, we could circumvent this issue. The concern is that because we used programming in the hybrids (i.e. pivoting on PROM simulation security) that this allows us to show something (succinct indistinguishability-based security) when this would otherwise not normally be a transformation that can be done from NPROM simulation-based security. As it happens, the same work [JSW17] also shows that there also exists a transformation from certain simulation-based garbling schemes **in the standard model** to get an indistinguishability-based garbling scheme from the same assumptions without the decoding table (the formal part that adds the extra cost, see Definition 1). This transformation can be applied to protocols that rely on so-called pebbling-based security proofs, which all the protocols featured in this work use. This is to say that this interaction was predicted in the standard model and is **reinforced** rightly in the NPROM model from [FLR+10].