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

Cruz Barnum
University of Illinois Urbana-Champaign, IL, USA

David Heath
University of Illinois Urbana-Champaign, IL, USA

Vladimir Kolesnikov
Georgia Institute of Technology, Atlanta, GA, USA

Rafail Ostrovsky
University of California Los Angeles, CA, USA

Abstract

Garbled circuit techniques secure in the adaptive setting – where inputs are chosen after a garbled program is sent – are motivated by practice, but they are difficult to achieve. Prior adaptive garbling is either impractically expensive or encrypts the garbled program with the output of a programmable random oracle (PRO). This latter approach introduces a strong model *and* incurs computational overhead.

We present a simple framework for proving adaptive security of garbling schemes in the non-programmable random oracle (NPRO) model. NPRO is a milder model 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* (and hence incurring no overhead).

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.

2012 ACM Subject Classification Security and privacy → Cryptography; Security and privacy → Symmetric cryptography and hash functions

Keywords and phrases Adaptive Garbling, Garbled RAM, Random Oracle Model

Digital Object Identifier 10.4230/LIPIcs.ITC.2026.11

Related Version *Full Version*: `https://eprint.iacr.org/2023/1527` [4]

1 Introduction

Yao's Garbled Circuit (GC) is a cryptographic technique allowing two parties – a garbler G and an evaluator E – to securely evaluate arbitrary programs $\mathcal{P}$ on their joint private inputs [37]. GC is a foundational cryptographic primitive with various applications, especially in the fields of 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 on fast symmetric-key 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 (GRAM) is a GC extension that enables garbling of random access machine (RAM) programs [29]. GRAM solves the above sources of blow-up, allowing constant round protocols that handle complex programs.

[21] showed that there exists a relatively simple circuit model – tri-state circuits (TSCs) – that can efficiently emulate both Boolean circuits and RAM programs. Roughly, efficient emulation of RAM is possible because TSC gates evaluate in a data-dependent order. We construct a TSC garbling scheme that in particular handles data-dependent execution order, 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 [38].[1]

Garbling schemes and selective security

For simplicity and modularity, GC techniques are often formalized as *garbling schemes* [6]. A garbling scheme factors evaluation of program $\mathcal{P}$ on joint secret input x into four steps:

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

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

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

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

Security against a corrupted E is proved by considering this interaction and constructing a simulator that – from the program output y alone – can forge a 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.

[1] [35] uses less communication than [38], but it uses significantly more computation. We consider both techniques state-of-the-art.

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 $\tilde{x}$ 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.

Constructing garbling schemes that provably achieve this second notion is *notoriously difficult*. This difficulty arises from the fact that E's input can depend on the garbled program itself. Proving this secure is a significant challenge.

Cost accounting for adaptive GC schemes

When constructing adaptively-secure GC, we consider the cost of the offline and the online phases separately. The crucial metrics are the offline and online communication costs. Ideally, offline phase communication should be close 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, multiplied by 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 $\tilde{\mathcal{P}}$ was first observed by [5]. 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 [22]. 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 [5]. This both requires modification to the selectively secure scheme, and it roughly doubles computational cost. It would be far preferable to obtain adaptive security without increasing computation and, indeed, without modifying implementation. In addition the [5] proof requires that the simulator *program* the RO. This is a strong model, which cannot be met by any fixed function, and which is widely seen as much stronger than a non-programmable RO.

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 one 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 [35], and even arithmetic techniques [3, 19]; see Appendix C in the full version. 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 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, 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 [38] in terms of both computation and communication. [38] is a state-of-the-art Boolean scheme[2], 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 [21].

We emphasize that while RO-based proofs are sometimes straightforward, proving unmodified GC schemes adaptively secure remains surprisingly difficult. Our work proves many such schemes secure, and it provides a framework by which to show security for future schemes. To our knowledge, no other works show adaptive GC from RO, except see Section 1.3's discussion of [5, 18].

1.2 The Practical Importance of Adaptive Security

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; 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. Envisioning larger systems incorporating MPC/GC, even implemented

[2] The more recent [35] scheme uses less communication than [38], but it uses more computation.

and maintained by a 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.

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 [32, 33]. Following discussion at the MPTS workshops, it seems impractical to standardize many possible combinations of GC and OT. Rather, GC, OT, and other related primitives are likely to be standardized separately. A more robust, versatile, and resilient GC primitive would be much preferable for standardization than a brittle one, subject to execution order constraints.

Option 2: Use a programmable random oracle (PRO) to mask GCs

Namely, use the PRO-based technique of [5]. This option roughly doubles the total computation cost, both for G and E, compared to selectively-secure GC, and to our solution. Perhaps worse than this increased cost is the fact that it requires modification to the selectively-secure GC scheme. Far more attractive would be to simply show that existing schemes are, in fact, adaptively secure, if the hash function is sufficiently strong; this is what we achieve here.

The approach of [5] also assumes programmability of the RO (PRO). PRO is an unusually strong model that violates several impossibility results, e.g., enabling non-interactive non-committing encryption [31] and adaptive GC with online phase independent of the program output size [5]. Both are impossible in the standard model [2, 31]. In contrast, NPRO is a milder model, which is widely used in practical cryptography. For instance, the standardized RSA-OAEP encryption scheme uses NPRO [7, 9, 10, 30]. Notably, in the GC setting, the widely used Free-XOR key homomorphism relies on circular correlation robustness [8], which can be understood as a weakening of the NPRO model.

Option 3: Implement adaptive GC in the standard model

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

Improving 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 [28] 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 [28]'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

[5] were the first to thoroughly investigate the problem of adaptive GC. They pointed out that existing schemes do not seem to admit a proof of adaptive security, and they gave two constructions that *are* 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 GC.

This said, [5]'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 [28]. In this way, the one-time-pad-based scheme admits a natural proof of adaptive security.

[5] 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 [2]. In particular, [2] showed that any standard model adaptive garbling scheme must have an online phase scaling at least with the program's output, but [5]'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 [2] lower bound.[3]

In concurrent work, [18] 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 random permutation model [17], 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 framework that may be useful in large class of GC constructions. Our simulation methods are less customized and more general. Our framework applies to garbling of TSCs, and thus our results immediately imply an adaptively secure garbled RAM scheme in the NPROM, which was not shown by [18].

Other Works on Adaptive GC

A number of other works made progress on adaptive GC, including by giving surprising, but expensive, solutions that assume only the existence of one-way functions [22], showing security of schemes for low-depth circuits [24, 23], lower-bounding the security of certain techniques in the adaptive setting [25, 1], and achieving adaptive security from heavy-weight public-key-style methods [14, 22]. The prior work section in the full version discusses these at greater length.

[3] If the decoding table is given in the online phase in the adaptive scheme from [5], then one can show that their scheme is adaptively secure in the NPROM. We emphasize that [5] still prescribes an RO mask on every ciphertext, which our analysis avoids.

Garbled RAM (GRAM)

GRAM [29] 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. [12, 13, 15, 29], demonstrated feasibility results, but they were not concerned with polylog performance factors, so their constructions are expensive. More recent constructions [20, 21, 34, 16] target performance. The best symmetric-key result [21] garbles only $O(\log^3 n \cdot \log\log n)$ fan-in-two gates per access.

More interesting than [21]'s cost is its formalism. [21] 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 TSCs [21]. Our result leverages the TSC model to achieve adaptively secure GRAM. We review the TSC model in Section 3.

2 Technical Overview

This section explains our approach at a high level, providing sufficient detail for informal understanding. Section 4 in this work and Section 5 in the full version of the paper formalize the ideas explained here.

Note that we prove adaptive security of garbled tri-state circuits (TSCs) because the TSC model is more general (and useful – it supports efficient RAM implementation) than Boolean circuits. For those unfamiliar, we explain the TSC model in Section 3.2. Our proofs and proof intuition apply and can be read and understood in the narrower context of Boolean circuits.

2.1 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 3.2.

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 [26]. 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:

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

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

Gate handling

The function of each TSC gate-type was explained in Section 3.2. Roughly, our garbling of gates is as follows:

- **XOR gates** are handled simply by XORing the input labels. This is the Free XOR optimization [26].
- **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:

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

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

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

2.2 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 3.3 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 hybrid worlds, 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 question is to consider whether or not this use of "programming" should be allowed in the NPROM.

We emphasize that this use of "programming" is *formally* in the NPROM [9]. The model simply constrains that the *simulator* cannot program the oracle. As a proof technique, we introduce hybrid worlds, which – like the real and ideal worlds – are simply descriptions of the output of some program (the adversary). We introduce these hybrid thought experiments only as a means by which to compare and relate two objects – the output of $\mathcal{A}$ in the real and ideal experiments. The tools we use to compare these objects are immaterial. 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.

Indeed, the NPROM can be understood as insisting that we can heuristically instantiate the RO in *both* the real world and the ideal world, and it is in this sense in which the RO is non-programmable – if the RO were programmed, it would not be instantiable by any pre-defined hash function. But of course we can *in a thought experiment* reason about $\mathcal{A}$'s

output in the context of that (heuristically assumed random) truth table; a part of that reasoning is considering what would happen if we were to replace some truth table rows by fresh randomness.

This style of using programming inside hybrids is analogous techniques in the universal composability (UC) framework. In the UC framework, one cannot "rewind" the adversary (aka environment), but some works have found value in rewinding the adversary in the context of hybrid experiments [11, 27].

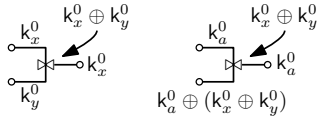
Appendix A in the full version provides more discussion regarding NPRO.

Rekeying a Garbled Circuit

In a standard model GC proof of *selective* security, 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. [28]. 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.

We next show how to rekey a TSC join gate (defined in Section 2.1), a gate that is more primitive than Boolean AND/OR. In particular, a Boolean AND gate can be efficiently implemented from a small number of TSC gates, including two joins [21]. In the resulting garbled AND, G and E take the same actions as if they had used the half-gates AND gate construction [38]. The rekeying of a single join gate can be understood as a subprocedure of a rekeying of the more complex Boolean AND gate. (In TSC-based RAM, joins are also used outside the construction of Boolean AND gates.) Consider the following join:



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 [6] 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.

Proof intuition

In short, our security proof combines query hiding and rekeying. By applying query hiding, we show that $\mathcal{A}$'s chosen input x cannot depend on the content of the RO. However, it still might depend on the garbled circuit itself. Rekeyability then decouples the garbled circuit from its wire keys and 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.

3 Preliminaries

3.1 Notation

- λ is a security parameter and can be understood as the length of GC labels.
- $x \approx y$ denotes distributions x and y are computationally indistinguishable.
- $x \overset{s}{\approx} y$ denotes 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.

3.2 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.

[21] 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 [29]. Our presented garbling scheme handles TSCs. Thus, we review relevant definitions. All definitions in this section are adapted from [21].

▶ **Definition 1** (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. 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 2** (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[4]*. 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.

[4] i.e. if one input or output is boolean, propagate the boolean value to each other wire, even if the output propagates to the input wires.

[21] show that to make this work, it suffices to reveal to E every buffer control wire. These (plaintext) control bits allow E to determine which gates are ready for evaluation. G reveals control bits by including in the GC one extra bit per 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. [21] solve this by extending TSC input to additionally include randomness. The random part of the input can be used as masks on control bits. An oblivious TSC guarantees that if randomness r is properly sampled, the circuit correctly evaluates with overwhelming probability. When the oblivious TSC is garbled, randomness r is sampled locally by G and included as part of his input to the circuit. 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 3** (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:*

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

3.3 Definition of Non-Programmable Random Oracle Model

We consider security in the non-programmable RO model (NPROM), as defined by [9]. 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, $\mathcal{A}$'s tape is instead connected to an externally defined oracle $\mathcal{O}$, and $\mathcal{O}$ responds directly.

The informal rationale underlying the NPROM is that it more closely resembles the 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 the PROM because we 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. Our results also hold for an even weaker model called the Auxiliary Input ROM [36]. This model and proof that our scheme is compatible can be found in Appendix D in the full version.

3.4 Garbling Schemes

We consider security for *garbling schemes*, as defined by [6]:

▶ **Definition 4** (Garbling Scheme [6]). *A **garbling scheme** for a class of circuits $\mathcal{C}$ is a tuple of four procedures $(Garble, Encode, Eval, Decode)$:*

- $Garble(1^\lambda, C) \to (\tilde{C}, e, d)$*: 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:*

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

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

3.5 Definition of Adaptive Security

Our notion of adaptivity is based on definitions from [5] and [6]:

▶ **Definition 5** (Adaptive Privacy). *A garbling scheme is **adaptively private** if for all circuits C, there exists a simulator $\mathcal{S}$ such that for all stateful PPT adversaries $\mathcal{A}$ the following quantity is negligible in λ:*

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

where Real, Ideal *are as follows:*

$\mathrm{Real}_{\mathrm{prv}}^{\mathcal{A},C}(1^\lambda)$	$\mathrm{Ideal}_{\mathrm{prv}}^{\mathcal{A},\mathcal{S},C}(1^\lambda)$
1 : $(\tilde{C}, e, d) \leftarrow Garble^{\mathcal{O}}(1^\lambda, C)$	1 : $\tilde{C} \leftarrow \mathcal{S}^{\mathcal{O}}(1^\lambda, C)$
2 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$	2 : $x \leftarrow \mathcal{A}^{\mathcal{O}}(1^\lambda, \tilde{C})$
3 : $\tilde{x} \leftarrow Encode(e, x)$	3 : $(\tilde{x}, d) \leftarrow \mathcal{S}^{\mathcal{O}}(C(x))$
4 : **return** $\mathcal{A}^{\mathcal{O}}(\tilde{x}, d)$	4 : **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 B in the full version.

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 in the full version of the paper, we see that our garbling of TSCs is query-hiding and rekeyable; Appendix C in the full version 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 $Supp(Garble(1^\lambda, C))$ denote the **support** of Garble on input C. Formally, $Supp(Garble(1^\lambda, C))$ 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 $(\tilde{C}, e, d) \leftarrow Garble^{\mathcal{O}}(1^\lambda, C)$, 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:*

$QueryHiding^{\mathcal{A},C}(\lambda)$	
1 :	*Sample random oracle* $\mathcal{O}$
2 :	$\delta, (\tilde{C}, e, d) \leftarrow Garble^{\mathcal{O}}(1^\lambda, C)$
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*[5] *procedure Rekey:*

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

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:*

$Rekeyable^R(1^\lambda, C)$		$Rekeyable^I(1^\lambda, C)$	
1 :	*Sample random oracle* $\mathcal{O}$	1 :	*Sample random oracle* $\mathcal{O}$
2 :	$\delta, (\tilde{C}, e, d) \leftarrow Garble^{\mathcal{O}}(1^\lambda, C)$	2 :	$\delta, (\tilde{C}, e, d) \leftarrow Garble^{\mathcal{O}}(1^\lambda, C)$
3 :		3 :	$(\delta', e', d') \leftarrow Rekey(1^\lambda, C, \tilde{C})$
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. 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*, there exists a simulator* $\mathcal{S}$ *s.t. for all* x*, all GCs* $\tilde{C} \in Supp(Garble(1^\lambda, C))$*, all oracles* $\mathcal{O}$*, and all* PPT *adversaries* $\mathcal{A}$*, the following ensembles are statistically close in* λ*:*

[5] 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.

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

Roughly speaking, the privacy simulator $\mathcal{S}$ forges an output decoding table d' such that the GC correctly evaluates to $C(x)$, even though the input is 0.

4.3 Adaptive Security

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

▶ **Theorem 9** (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:

$\mathcal{S}^{\mathcal{O}}(1^\lambda, C)$
$(\tilde{C}, e, d) \leftarrow \mathrm{Garble}^{\mathcal{O}}(1^\lambda, C)$
return $\tilde{C}$
// Second call; receive $C(x)$ from caller
$\tilde{x} \leftarrow \mathrm{Encode}(e, \mathbf{0})$
$d' \leftarrow \mathcal{S}_p(1^\lambda, C(x), d)$
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 $C(x)$.

Now, we show that the real-world and ideal-world experiments are indistinguishable. In Figure 1, we restate adaptive privacy's real/ideal experiments, in-lining the definition of our simulator. Figure 1 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.

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.

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

Figure 1 Real, ideal, and hybrid distributions used in our proof of Theorem 9.

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 $\tilde{C}$, $\mathcal{O}$, 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. ◀

References

1 Anasuya Acharya, Karen Azari, and Chethan Kamath. On the adaptive security of free-xor-based garbling schemes in the plain model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 214–244. Springer, 2025. doi:10.1007/978-3-031-91095-1_8.

2 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. doi:10.1007/978-3-642-40084-1_10.

3 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. doi:10.1145/2976749.2978410.

4 Cruz Barnum, David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Adaptive garbled circuits and garbled RAM from non-programmable random oracles. Cryptology ePrint Archive, Paper 2023/1527, 2023. URL: https://eprint.iacr.org/2023/1527.

5 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. doi:10.1007/978-3-642-34961-4_10.

6 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. doi:10.1145/2382196.2382279.

7 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. doi:10.1007/BFb0053428.

8 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. doi:10.1007/978-3-642-28914-9_3.

9 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. doi:10.1007/978-3-642-17373-8_18.

10 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. doi:10.1007/3-540-44647-8_16.

11 Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable SNARKs. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 315–346. Springer, Cham, April 2023. doi:10.1007/978-3-031-30617-4_11.

12 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. doi:10.1109/FOCS.2015.22.

13 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. doi:10.1145/2746539.2746593.

14 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. doi:10.1007/978-3-319-96878-0_18.

15 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. doi:10.1007/978-3-642-55220-5_23.

16 Tianyao Gu, Afonso Tinoco, Sri Harish G. Rajan, and Elaine Shi. PicoGRAM: Practical garbled RAM from decisional diffie-hellman. In *CRYPTO 2025, Part VIII*, LNCS, pages 247–281. Springer, Cham, August 2025. doi:10.1007/978-3-032-01913-4_8.

17 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. doi:10.1007/978-3-030-56880-1_28.

18 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. URL: https://eprint.iacr.org/2023/1528.

19 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. doi:10.1007/978-3-031-58740-5_1.

20 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. doi:10.1007/978-3-031-06944-4_1.

21 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. doi:10.1007/978-3-031-38551-3_5.

22 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. doi:10.1007/978-3-662-53015-3_6.

23 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. doi:10.1007/978-3-030-65277-7_33.

24 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. doi:10.1007/978-3-662-53641-4_17.

25 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. doi:10.1007/978-3-030-84245-1_17.

26 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. doi:10.1007/978-3-540-70583-3_40.

27 Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 179–188. ACM Press, May / June 2009. doi:10.1145/1536414.1536441.

28 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. doi:10.1007/s00145-008-9036-8.

29 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. doi:10.1007/978-3-642-38348-9_42.

30 Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016. doi:10.17487/RFC8017.

31 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. doi:10.1007/3-540-45708-9_8.

32 NIST. NIST workshop on multi-party threshold schemes 2020, 2020. URL: https://csrc.nist.gov/Events/2020/mpts2020.

33 NIST. NIST workshop on multi-party threshold schemes 2023, 2023. URL: https://csrc.nist.gov/events/2023/mpts2023.

34 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. doi:10.1007/978-3-031-30545-0_16.

35 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. doi:10.1007/978-3-030-84242-0_5.

36 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. doi:10.1007/978-3-540-74143-5_12.

37 Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. doi:10.1109/SFCS.1986.25.

38 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. doi:10.1007/978-3-662-46803-6_8.