A New Approach to Garbled Circuits

Anasuya Acharya[1], Tomer Ashur[2,3], Efrat Cohen[4], Carmit Hazay[5], and Avishay Yanai[6]

[1] Bar-Ilan University, Ramat Gan, Israel
acharya@biu.ac.il
[2] imec-COSIC, KU Leuven, Leuven, Belgium
tomer.ashur@esat.kuleuven.be
[3] Cryptomeria, Leuven, Belgium
tomer@cryptomeria.tech
[4] Bar-Ilan University, Ramat Gan, Israel
efrat.choen@biu.ac.il
[5] Bar-Ilan University, Ramat Gan, Israel
carmit.hazay@biu.ac.il
[6] VMware Research, Herzliya, Israel
yanaia@vmware.com

Abstract. A garbling scheme is a fundamental cryptographic building block with a long list of applications. The study of different techniques for garbling a function, towards optimizing computation and communication complexity, has been an area of active research. Most common garbling techniques work by representing each gate in the circuit as a set of ciphertexts that encrypt its truth table row-by-row.

In this work we present a new garbling scheme in the random oracle (RO) model that garbles circuits in the gate-by-gate paradigm by capturing the gate functionality ($\mathsf{AND}, \mathsf{XOR}$) *as a whole* rather than as a set of ciphertexts. The final gate garbling requires 4κ bits of communication in expectation, 4 RO calls for garbling and 1 RO call for evaluation. We prove that the scheme satisfies privacy in the non-programmable random oracle model and against PPT adversaries. We also show how this scheme can be extended to support free-XOR and garble *any* gate functionality over binary inputs.

Keywords: Garbled Circuits · Gate-by-Gate Garbling · Random Oracles

1 Introduction

The theory and practice of garbling circuits has been the focus of a long line of research, starting from the seminal work of [Yao86], and further optimized in the works of [LP09,PSSW09,BHR12,ZRE15,HK20,RR21], to name a few. Garbled circuits (GCs) are a fundamental building block that represents a function and a secret input in such a way that evaluating the garbled circuit on the input representation reveals nothing beyond the function output. GCs have a long list of applications like constant round secure two-party computation

(2PC) [LP09], constant round multiparty computation [BMR90,BLO16], zero-knowledge proofs [FNO15,GKPS18], bootstrapping obfuscators [App13], functional encryption [GKP^{+}13], and verifiable computation [GGP10].

Owing to its wide range of applications, Bellare *et al.* presented an abstraction for garbling in [BHR12], viewing it as a fundamental building-block for use in cryptographic protocols. This abstraction is termed as a garbling scheme and is a framework defining four algorithms. A garbling algorithm takes a function representation, i.e. a circuit, and uses it to create a garbled circuit (GC). Depending on the scheme, the GC may have certain function hiding properties: given a GC, the actual functionality garbled remains hidden. The garbling algorithm also creates an input encoding function. Next, an input encoding algorithm takes any valid input to the circuit garbled and uses the input encoding function to give 'input labels' that correspond to the GC. The input labels typically have the property that, when looked at in isolation, they do not reveal the input represented. An evaluation algorithm takes a GC and a set of input labels for a certain input, and derives a representation of the function output. Finally, an output decoding algorithm derives the function output from its representation output by the evaluation algorithm. It is required that nothing beyond the function output is revealed. [BHR12] also gives various definitions of desirable properties for garbling schemes like correctness, privacy, authenticity and obliviousness.

A scheme for garbling circuits was first proposed in [Yao86] and its security was formalized in [LP09]. The formalism of [BHR12] captures this construction and many subsequent works in garbling published after [BHR12] have followed the same line of thought as [LP09], also describing themselves in terms of [BHR12]. [LP09] garbles a circuit in a gate-by-gate manner where each gate is garbled by encoding its truth table row-by-row, creating a set of ciphertexts. Subsequent optimizations reduce the size of garbled gates by either reducing ciphertext sizes, allowing certain ciphertexts to not require communication [PSSW09], or re-writing the gate functionality so that its truth table has fewer rows [ZRE15].

1.1 Our Contributions

We propose a novel scheme for garbling circuits in the gate-by-gate paradigm that captures the gate's truth table as a whole in one encoding, rather than as a set of encrypted rows. We operate in the non-programmable random oracle (RO) model wherein both the garbler and the evaluator are given access to a common random oracle. Our garbling approach requires 4 RO queries to garble *any* binary gate functionality and 1 RO query for evaluation. For a computational security parameter κ, letting the length of each input label be κ, the expected length of each garbled gate is 4κ bits. We also describe how this scheme can be modified to support free-XOR at the cost of increasing the size of other garbled gates.

Although this scheme does not improve upon the current state-of-the-art in garbling size, it produces a garbling with size that is comparable. It also has certain advantages over schemes that produce garblings. For instance, the garbling scheme in [ZRE15] produces gate garblings of size 2κ while providing free-XOR compatibility. However, evaluating their GC requires 2 calls to the underlying cryptographic primitive for a gate, while our scheme requires only 1 RO call. The garbling scheme in [PSSW09] also produces gate garblings of size 2κ while making 4 calls to the underlying cryptographic primitive for garbling and 1 call for evaluation. However, it does not support free-XOR and nor can it be extended to support it like our scheme allows. We also have an advantage in computation complexity over [RR21] that produces a gate garbling of size 1.5κ at the cost of up to 6 primitive calls for garbling and 3 for evaluation. Further novelty of our scheme lies in the new approach employed for garbling that opens up a variety of avenues for future work.

Our scheme satisfies *correctness* and *privacy* [BHR12] against a PPT adversary with access to $t(\kappa)$ queries to the random oracle for any polynomial $t(\cdot)$. Informally, the privacy-by-indistinguishability property requires that for two circuits $\mathbf{C}_0$ and $\mathbf{C}_1$ that have the same topology, and for two inputs x_0 and x_1 such that $\mathbf{C}_0(x_0) = \mathbf{C}_1(x_1)$, a garbling of $\mathbf{C}_0$ along with input labels corresponding to x_0 should be indistinguishable from a garbling of $\mathbf{C}_1$ and input labels for x_1.

[BHR12] also contains a result stating that if the *leakage function* for a garbling scheme is invertible, the definitions of privacy-by-indistinguishability and privacy-by-simulation are equivalent. For our garbling scheme, the leakage function – information about the function revealed by the garbling – is the topology of the circuit garbled. This is indeed an invertible leakage – given a circuit topology, one can construct a circuit that has that topology. Therefore, it holds that our garbling scheme also satisfies privacy-by-simulation.

1.2 Related Work

Secure garbling of circuits and corresponding ways of succinctly representing the garbling has been the aim of a long line of research [BMR90,NPS99,KS08,LP09]. The most common paradigm for garbling a circuit operates at the gate level (also known as the 'gate-by-gate paradigm') where for each gate in the circuit, each line in the truth table of the gate functionality is encrypted separately. The underlying primitive for encryption is a symmetric-key algorithm (*e.g.*, a pseudorandom function (PRF), a circular-correlation robust hash function (CCR), a CPA-secure dual-key cipher (DKC)) which yields extremely fast algorithms. This paradigm led to a long sequence of successful optimizations in computation and communication, that established garbled circuits as a practical tool for achieving 2PC [PSSW09,KMR14,ZRE15].

Minimizing the size of garbled circuit representation so as to reduce the communication complexity is a widely studied research area. To this effect, [KS08] proposes a garbling technique that allows for 'free-XOR' – an XOR gate need not be represented in the garbling at all. Following [LP09], [PSSW09] proposes schemes that garble each gate in a circuit by garbling its truth-table row wise, but in a way that certain garbled rows need not be communicated. For a computational security parameter κ, one such scheme (GRR3) produces a gate garbling of size of 3κ, while still remaining compatible with free-XOR. Another scheme (GRR2) garbles each gate with 2κ-bits, at the cost of forfeiting free-XOR compatibility. Both of these are improvements over the 4κ-bits required in [LP09].

Another work, [ZRE15] takes this further by proposing a garbling technique that garbles each gate using 2κ-bits, while remaining compatible with free-XOR. [KKS16] shows a scheme in which 2κ bits can be used to garble internal gates of a circuit, while gates with circuit input wires as input can be garbled using κ bits. For certain classes of circuits, formulas in particular, their construction requires between κ and 1.5κ bits per garbled gates on average.

Table 1. Comparison of our scheme with related work. We compare the garbling size and the number of primitive calls for garbling and evaluation for an AND gate. κ is the computational security parameter.

Garbling Scheme	Gate Garbling Size	Number of RO Calls	
		Garbling	Evaluation
[ZRE15](Free-XOR)	2κ	4	2
[RR21](Free-XOR)	$1.5\kappa + 8$	6	3
Our Work	4κ	4	1
Our Work (Free-XOR)	8κ	4	1

The state-of-the-art in garbled gate size optimization today is [RR21] where the size of each garbled gate is 1.5κ bits. Pursuing a different line of garbling size optimization, [HK20] proposes a scheme that reduces the size of the circuit as a whole to the size of the longest *branch* of computation.

An extended line of works the generalizes garbling is the study of *randomized encodings* [IK00,Ish13,App17]. Given a function f and an input x, a randomized encoding is a representation $\hat{f}(x, r)$ generated using randomness r such that no information beyond $f(x)$ can be derived from it. A garbling can be viewed as a special case of a randomized encoding. Specifically, a projective garbling such as ours is a case of a *decomposable* randomized encoding, where given the garbling and the active input labels only, nothing beyond the function output is revealed.

1.3 Vision for Future Research

Our garbling scheme garbles binary gates *as a whole* in an efficient non-linear manner, as opposed to encrypting the truth-table of the gate row-by-row. This opens up avenues in multiple new directions. One future direction would be to try to extend this scheme to support computationally unbounded adversaries. The security proof for our scheme does not make any assumptions on the strategy of the adversary (or its running time) except that the number of RO queries it is limited to making is polynomial in κ - the security parameter. It remains to consider if the scheme remains secure when the number of queries permitted is relaxed to be sub-exponential in κ, or even a small exponent in κ. It also remains to consider if slight modifications to the scheme would provide statistical security.

For simplicity of explanation and proof, we make use of a random oracle (RO) while describing our scheme. However, the security proof does not rely on properties like programmability or extractability of the RO. So we conjecture that random oracles may not be necessary for the security of the scheme. Further, the extension of our scheme to obtain free-XOR does not depend on the properties of the RO. So, owing to our garbling technique, we also conjecture that the RO can be replaced with a simpler primitive that need not satisfy circular security. This would imply that the resulting scheme achieves free-XOR without the circular security assumption inherent in all prior works achieving free-XOR. This effort of finding suitable primitives to replace a random oracle is also a direction that we leave for future work.

Our garbling technique can garble a gate as a whole into one encoding, and the resulting encoding can be compressed in a lossless manner – due to the resulting encoding not having full entropy. We defer to future empirical analysis to determine the achievable compression rate.[7] Furthermore, an interesting direction to explore would be if two or more gate functionalities can be condensed into one encoding whose compressed representation is smaller than the size of individually garbling the gates. Coupled with free-XOR, this would drastically reduce the size of the resulting garbled circuit. We leave studying the trade-off between the number of gates in one encoding and compression rate to future work.

Finally, we also conjecture that this scheme can be shown as secure in the adaptive garbling setting – where the adversary can choose inputs after it receives the garbled circuit from the challenger. Proving that a garbling scheme is adaptive secure has remained an open question for Yao's garbled circuits, while subsequent works prove that certain modifications to it is adaptive secure [KKPW21]. These either incur a security loss or use expensive cryptographic primitives. We conjecture that our scheme can be shown as adaptive secure as-it-is using slight modification to our existing proof technique. On replacing the RO with a cheaper primitive or operation in the standard model, we would have an adaptive garbling scheme that is much more efficient than the current state-of-the-art.

2 Technical Overview

Our garbling scheme operates in the random oracle model where both the garbler and evaluator get access to a random oracle (RO). Below we discuss the key design aspects of the core scheme. Discussion about the free-XOR extension is deferred to Section 5.

The Garbling Algorithm. Conforming to the [BHR12] formalism, the input to the garbling algorithm is a circuit $\mathbf{C}$; and it outputs a garbled circuit F, an input encoding set e, and an output decoding set d. The algorithm itself can be separated into the following subroutines that are executed sequentially: (1) $\mathsf{Init}(\mathbf{C}) \to e$; (2) $\mathsf{Circuit}(\mathbf{C}, e) = (F, D)$; (3) $\mathsf{DecodingInfo}(D) \to d$.

[7] A previous version of this paper contained an incorrect claim about the achievable compression rate.

Input Label Sampling. The first subroutine in the garbling algorithm takes the circuit $\mathbf{C}$ and creates the input encoding set e. This subroutine $\mathsf{Init}(\cdot)$ is a randomized algorithm. From within $\mathbf{C}$, this algorithm only uses n, the number of input wires. This allows the generation of e potentially ahead of knowing the function f. Similar to other traditional garbling schemes, the scheme we design is also a projective garbling scheme. So e contains a set of input wire labels. In our construction, for each of the n input wires, for an 'external length parameter' ℓ, an ℓ-length label is sampled uniformly at random to represent the 0 and 1 bit, under the constraint that both labels for the same wire cannot be the same.

Gate-by-Gate Garbling. The next subroutine $\mathsf{Circuit}(\cdot)$ is a deterministic function. It takes the input encoding set e with all the randomness it entails, and extends it to create the complete garbled circuit F and output wire labels D. In order to extend the existing randomness in a way that lets the garbling preserve its privacy, $\mathsf{Circuit}(\cdot)$ makes black box calls to a random oracle RO.

Each gate in the circuit is garbled separately and in a topological order. To this effect, for the q total gates in the circuit $\mathbf{C}$, each gate is assigned an index g in this ordering. The random oracle RO employed throughout the gate-by-gate garbling process is tweakable: it takes as an additional input the gate index g so that it behaves independently for each gate.

Garbling a Gate. For a gate g, let A and B be its input wires, g be its output wire index, and f_g be its functionality (*e.g.*, AND, XOR). When garbling a gate, our methods deviate significantly from traditional garbling techniques. At its core, we make the following observation: each gate is a binary gate so there are 4 combinations of input values, but only two possible output values corresponding to one output wire.

Therefore, at its core, a gate garbling is a means to *convert* a pair of input labels into an output label. For a wire A, L_0^A and L_1^A are its labels (similarly $\mathsf{L}_1^B, \mathsf{L}_0^B$ for B, and $\mathsf{L}_1^g, \mathsf{L}_0^g$ for output wire g). We require that for the gate g, the input label combinations be mapped to $(\mathsf{L}_0^g, \mathsf{L}_1^g)$ in such a way that the gate functionality f_g is preserved. For instance, if the gate is an AND gate, $\{(\mathsf{L}_0^A, \mathsf{L}_0^B)$, $(\mathsf{L}_0^A, \mathsf{L}_1^B)$, $(\mathsf{L}_1^A, \mathsf{L}_0^B)\}$ should be mapped to L_0^g, and $(\mathsf{L}_1^A, \mathsf{L}_1^B)$ to L_1^g.

We encode all four input label pairs into *one encoding* ∇^g such that, given one label from each input wire, ∇^g can be used to *convert* these into the correct output label. The details on how ∇^g is generated can be found in Appendix A where Table 3 indicates how the garbling for the AND functionality is generated and Table 4 indicates the same for the XOR functionality. These tables are part of the description of the garbling scheme: that is, they are predetermined and remain the same regardless of the circuit garbled or the randomness used.

The entire gate garbling process is a result of deterministic steps starting from the input label values. For gate g with input labels $\mathsf{L}_1^A, \mathsf{L}_0^A$ and $\mathsf{L}_1^B, \mathsf{L}_0^B$, first, in order to eliminate redundancy, for each pair of input bits $(a, b) \in \{0, 1\}^2$ the input labels is input to a random oracle: $\mathsf{RO}^g(\mathsf{L}_a^A, \mathsf{L}_b^B) \to X_{ab}^g$. The random oracle RO takes as input the tweak g and two labels with total length 2ℓ, and outputs an ℓ'-length string. The output length ℓ' is much larger than that of the input and each RO output string is sampled uniformly at random and independently of the responses of other queries to RO^g.

Next, the random oracle outputs $(X^g_{00}, X^g_{01}, X^g_{10}, X^g_{11})$ are used to derive a single ℓ^g-bit string ∇^g (that is padded by 0s to make its length equal to ℓ') that encodes the gate functionality. ∇^g has the properties that given any one X^g_{ab}, it maps it to an ℓ-bit uniformly random binary string $\mathsf{L}^g_{f_g(a,b)}$. The gate garbling ∇^g has Hamming weight ℓ and the positions in this string that contain '1' are termed as 'effective key positions'. Each bit in ∇^g is set independently until its hamming weight becomes ℓ. We denote the length of the garbling up to this point as ℓ^g bits. It follows that ℓ^g varies for each gate g, but it still holds that $\ell^g = O(\kappa)$. The mapping of X^g_{ab} to an output label is done by projecting the bits in X^g_{ab} over the effective key positions in ∇^g. The resulting output label is of length ℓ and is independently and identically distributed (i.i.d.) over all bit positions. It also holds that the pair ∇^g and X^g_{ab} do not reveal any information about the inactive output label or the other random oracle outputs.

Decoding Information. Once all the garbled gates and output wire labels are derived in F, it remains to generate the output decoding information d. Following the same principle as we used for garbling gates, we want to avoid, to the extent possible, an adversary distinguishing between a valid decoding and an invalid decoding. Therefore, we need to be able to decode in such a way that for all label values, valid or invalid, it yields some plausible decoding, but with the constraint that for valid output labels - labels used within the garbling - the decoding is additionally also *correct.* In our construction, we employ another random oracle RO' for this. In the subroutine that creates the decoding information, for every output wire j, we sample an ℓ-bit string d^j. This string has the property that, given output wire labels $(\mathsf{L}^j_0, \mathsf{L}^j_1)$, it holds that $\mathsf{RO}'(\mathsf{L}^j_0, d^j) = 0$ and $\mathsf{RO}'(\mathsf{L}^j_1, d^j) = 1$. Note that such a decoding will always yield some output even for arbitrary ℓ-bit strings that are not output labels. The subroutine $\mathsf{DecodingInfo}(D) \rightarrow d$ generates this decoding information given the output wire labels set.

Evaluating the Garbled Circuit. An evaluator, given the garbled circuit F, a set of input wire labels X, and the decoding information d, works gate-by-gate. It has access to RO and RO' and knows the indices of each gate. Starting with the input labels $\mathsf{L} \in X$ we term each value in its view during an honest evaluation as *active.* For each gate g, with active input labels $\mathsf{L}^A_a, \mathsf{L}^B_b$, the evaluator works by first deriving $\mathsf{RO}^g(\mathsf{L}^A_a, \mathsf{L}^B_b) = X^g_{ab}$. Then using X^g_{ab} and $\nabla^g \in F$, it computes $\mathsf{L}^g_{f_g(a,b)} = X^g_{ab} \circ \nabla^g$ where $\circ$ is the operation selecting the bits in X^g_{ab} over the effective key positions in ∇^g. For an output wire label L^j_b, using $d^j \in d$, it computes $\mathsf{RO}'(\mathsf{L}^j_b, d^j) = b$ as the function output.

Security Intuition. Our scheme satisfies *privacy* against a PPT adversary. This notion is modeled as a game between the adversary and a challenger where the adversary first picks two circuits $\mathbf{C}_0$ and $\mathbf{C}_1$ of its choice such that they have the same topology. That is, letting Φ denote the leakage function revealing the topology of a circuit, it needs to hold that $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$. The adversary also chooses two inputs x^0 and x^1 such that $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$. The challenger picks a bit $b \in \{0,1\}$, garbles $\mathbf{C}_b$ and encodes the input x^b. It sends the resulting (F, X, d) to the adversary and then the adversary, making up to a polynomial number of queries to the random oracles, needs to output which bit b the challenger chose.

In order to understand why our scheme satisfies this notion of privacy, first note that the garbling (F, d) in the challenge, in isolation does not reveal any information about the circuits $\mathbf{C}_0$ or $\mathbf{C}_1$, beyond Φ, the topology that is identical. This is because the garbling hides the gate functionality as each gate garbling originates from the same distribution regardless of the functionality.

Given the complete challenge (F, X, d), an honest evaluation already reveals the complete 'active path' in the garbling. Our proof follows by proving that the knowledge of the active path gives the adversary no advantage at all in distinguishing. That is, given the complete challenge (F, X, d) and all honest queries, they are distributed independently of the bit b. Hence, we identify that learning elements in the 'inactive paths' is a prerequisite to privacy violation.

We formalize the notion of "learning a label" as making a random oracle query leading eventually to an inactive label. We identify what kind of queries lead to this and term them as bad events[8]. There are three bad events that are triggered by a query to RO. In 'Bad Event 1', the adversary "guesses" a candidate input label to the gate input wire A and queries it to RO together with the active input label of wire B. 'Bad Event 2' is defined symmetrically for the input wire B of a gate and 'Bad Event 3' is defined when inactive candidates are queried on both input wires. They are all analysed similarly.

'Bad Event 1' is triggered by two sub-events. First, when the candidate is the *inactive input label* and the query outputs either the *active output label* or the *inactive output label.* Second, when the output from the query is a *valid output label* - the active or inactive output wire label used in the garbling - but the input label tested is *not the inactive input label.* This case is possible since the RO maps each input to an output value independently and uniformly at random. So a value that is not the inactive input label is mapped to a gate output label.

Intuitively, the resistance against this event comes due to the size of the set of candidate labels to be tested. Stemming from the fact that ℓ is appropriately set, and there is a unique active label L^g, it follows that there are $2^\ell - 1$ candidate labels for which the output of RO is unknown to the adversary. Beyond this information, any two labels within the set of possible labels are uniform and independent. This holds by construction because the labels of a wire are either sampled uniformly at random (input wire labels) or derived as projections of random oracle outputs (internal wire labels). The latter also results in a random ℓ-bit string owing to the fact that the random oracle outputs are sampled freshly and uniformly at random for each distinct domain value, and the nature of the

[8] The proof technique we use is similar to [BHKR13] except that our adversary is PPT rather than query bounded. Since we do not assume anything about the adversarial strategy, this implies only that the bound on the number of queries is polynomial.

gate garbling ∇^g used to select a subset of these bits. Within the set of candidate labels for a wire, there is always the inactive label - used in the garbling - triggering the bad event. However, possibly other 'false positive' label values may also trigger the bad event owing to the nature of the random oracle outputs. Considering multiple RO queries amounts to sampling without replacement. In essence, the security argument boils down to the fact that the set of candidate labels is sufficiently large so that the adversary cannot cover a non-negligible portion of it within their query budget. The advantage gained by making RO queries increases linearly in the number of queries and decreases exponentially in ℓ (Theorem 4). Thus, the adversary's advantage is always negligible.

It remains to argue that when adversarial queries are made across different random oracles in different gates the bound on the probability of bad events remains unchanged. As a special case, let us consider two gates such that the output wire of one feeds into the other gate as input. First, note that the co-domain (all possible RO outputs) of the random oracle is $\{0,1\}^{\ell'}$. However the domain is much smaller: $\{0,1\}^{2\ell}$. Due to this, the size of its range (the subset of the co-domain that is the set of actual RO outputs corresponding to all the RO inputs) is also upper-bounded by $2^{2\ell}$. However, due to the properties of the random oracle, it is not possible to distinguish between its co-domain and range without querying the domain set. If a label in the range of RO is the inactive input label to the next gate, it will have triggered a 'Bad Event' as the inactive output label of the previous gate, ending the game. If the label is not the inactive input label, the adversary learns that further queries using this label as input are unnecessary, but gains no insight how to choose the candidate for the next query among the other $2^\ell - 1$ candidate input labels. Learning not to query this label to RO has cost the adversary a query in the previous gate hence it remains the case that "one query → one discarded value".

Therefore no advantage beyond what was learned directly by the query propagates between gates. This analysis holds without loss of generality when considering any number of gates in the circuit. We use this to bound the probability of encountering any of the three bad events, given $t(\kappa)$ queries to the random oracles and conclude that this is negligible in κ in our main result (Theorem 1).

3 Preliminaries

Table 2 contains a list of all the parameters with respect to which our garbling scheme is constructed. We denote by $\{0,1\}$ the set containing 0 and 1, and by $\{0,1\}^n$ the set of vectors of length n with each position containing 0 or 1. We use [] to denote an empty array. For a vector V and $i \in \mathbb{N}$, we denote by $V[i]$ the element in the i^{th} position in the vector. When $[i]$ is used in isolation, it signifies the set of elements $1, \ldots, i$.

Circuit Notation. For a function $f : \{0,1\}^n \to \{0,1\}^m$, let $\mathbf{C}$ be its circuit representation. Let q be the number of gates in $\mathbf{C}$. Each gate $g \in [q]$ is defined by a gate functionality $f_g \in \{\mathsf{AND}, \mathsf{XOR}\}$, two input wires A, B and an output wire g where, $A, B, g \in [n+q]$ and topological ordering holds: $A, B < g$.

Table 2. Table of Parameters

Parameter	Information
n	number of circuit input wires
m	number of circuit output wires
q	number of gates in the circuit
ℓ	(external length parameter) length of a wire label
ℓ'	(internal length parameter) length of approximate keys
ℓ^g	length of garbled gate ∇^g
κ	computational security parameter
s	number of adversarial random oracle queries

Random Oracles. The security proof of our scheme holds in the random oracle model, which abstracts a truly random function. Following the notation from [KL14], the random-oracle model posits the existence of a public, random function $\mathcal{R}$ that can be evaluated only by "querying" an oracle – which can be thought of as a "black box" – returning $\mathcal{R}(x)$ when given input x.

Definition 1 (Random Oracle). *A random oracle* RO *is an interface for an oracle function* $\mathcal{R} : \{0,1\}^a \to \{0,1\}^b$ *that is sampled uniformly from the family of functions that map the domain of binary strings* $\{0,1\}^a$ *into* $\{0,1\}^b$*.*

Garbling Scheme. [BHR12] abstracts garbling as a primitive containing four algorithms as given in Definition 2. In the definition, a function f is represented as a circuit $\mathbf{C}$. We also denote by $\Phi(\mathbf{C}) = (n, m, q, \{A, B, g\}_{g\in[q]})$ the topology of the circuit $\mathbf{C}$. Finally, $x \in \{0,1\}^n$ denotes the function input and $y \in \{0,1\}^m$ denotes the function output.

Definition 2 (Garbling Scheme [BHR12]). *Let* $f : \{0,1\}^n \to \{0,1\}^m$ *be a function with circuit representation* $\mathbf{C}$ *and* κ *be a computational security parameter. A garbling scheme* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev})$ *has four PPT algorithms:*

- $\mathsf{Gb}(1^\kappa, \mathbf{C}) \to (F, e, d)$*: returns a garbling* F*, input encoding set* e*, and output decoding set* d*.*
- $\mathsf{En}(e, x) := X$*: returns the encoding* X *for function input* x*.*
- $\mathsf{Ev}(F, X) := Y$*: returns the output labels* Y *by evaluating* F *on* X*.*
- $\mathsf{De}(Y, d) := \{\perp, y\}$*: returns either the failure symbol* $\perp$ *or a value* $y = f(x)$*.*

These algorithms must satisfy the following properties:

- ***Correctness****: For every* κ*, circuit* $\mathbf{C}$ *and input* x*,*

$$\Pr[y = \mathbf{C}(x) : (F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathbf{C}),\ X = \mathsf{En}(e, x),\ Y = \mathsf{Ev}(F, X),\ y = \mathsf{De}(d, Y)] = 1$$

- ***Privacy****: Let Algorithm 1 denote the actions of the challenger in an indistinguishability game. Let* Φ *be a leakage function representing the topology of* *a circuit. For every PPT adversary $\mathcal{A}$ (with access to* RO*), for all circuits* $\mathbf{C}_0, \mathbf{C}_1$ *s.t.* $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$ *and every* x^0, x^1 *s.t.* $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$ *of the choice of* $\mathcal{A}$*, there exists a negligible function* μ *such that* $\mathcal{A}$*'s advantage is,*

$$\mathsf{Adv}(\kappa) = \left| \Pr[\mathcal{A}^{\mathsf{RO}}(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1, F, X, d) = b] - \frac{1}{2} \right| < \mu(\kappa)$$

Algorithm 1 Privacy

```
proc $\mathsf{Challenger}(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1)$
    if $x^0, x^1 \notin \{0,1\}^n$ or $\Phi(\mathbf{C}_0) \neq \Phi(\mathbf{C}_1)$ or $\mathbf{C}_0(x^0) \neq \mathbf{C}_1(x^1)$ return $\perp$
    $b \leftarrow \{0,1\}$
    $(F, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathbf{C}_b)$
    $X = \mathsf{En}(e, x^b)$
    Return $(F, X, d)$
```

4 The Scheme

In this section we present our garbling scheme. The scheme itself is presented in Section 4.1. We present in Section 4.2 the intuition behind why the scheme is correct. In Section 4.3, we discuss the security guarantee and outline the proof of security. A full proof is presented in Appendix B.

4.1 Garbling Algorithm

For a function $f : \{0,1\}^n \to \{0,1\}^m$, let $\mathbf{C}$ be its circuit representation. The garbling algorithm has the following form:

Algorithm 2 Algorithm $\mathsf{Gb}(1^\kappa, \mathbf{C})$

```
set the external length parameter $\ell = \kappa$ and internal length parameter $\ell' = 8\ell$
$\mathsf{Init}(\mathbf{C}, \ell) \to e$
$\mathsf{Circuit}(e, \mathbf{C}, \ell, \ell') = (F, D)$
$\mathsf{DecodingInfo}(D, \ell) \to d$
Return $F, e, d$
```

The garbling algorithm as above begins by setting the variables ℓ and ℓ' defined in Table 2. These parameterize the lengths of the inputs and outputs of the random oracles employed in the construction. The 'external length parameter' ℓ parameterizes the length of all wire labels throughout the circuit. The additional 'internal length parameter' ℓ' parameterizes the length of the intermediate values in the gate garbling – the outputs of RO – and serves as a loose upper bound on the length of each gate garbling ℓ^g. The actual length of the gate garbings are variable and much smaller than ℓ' bits. Since the intermediate garbling values never have to be communicated, and so do not contribute to the communication complexity, ℓ' can be arbitrarily larger than ℓ. We refer the reader to Appendix A.2 for details as to why ℓ' is set to 8ℓ. Finally, ℓ is also the Hamming weight of ∇^g and parameterizes the effective length of the gate garbling. The complete garbling algorithm employs two random oracles of the following forms: (1) $\mathsf{RO}^g : \{0,1\}^{2\ell} \to \{0,1\}^{\ell'}$; and (2) $\mathsf{RO}' : \{0,1\}^{2\ell} \to \{0,1\}$. The first is used in each gate and so it uses the gate number g as a tweak. The latter is used for circuit output decoding.

Input Encoding Generation. The garbler starts by executing $\mathsf{Init}(\mathbf{C}, \ell) \to e$, formally described in Algorithm 3. Let n be the number of input wires in $\mathbf{C}$ and ℓ be the external length parameter. This algorithm uses the garbler's randomness to sample ℓ-length labels to represent the 0 and 1 values for each input wire. These labels are sampled uniformly at random, under the constraint that two labels for the same wire cannot take the same value. This resulting set of input wire labels is the input encoding set e.

Algorithm 3 $\mathsf{Init}(\mathbf{C}, \ell)$

```
extract $n$ from $\mathbf{C}$ and initialize $e = []$
for input wire $W \in [n]$ do
    Sample $\mathsf{L}_0^W \leftarrow \{0,1\}^\ell$ uniformly at random
    Sample $\mathsf{L}_1^W \leftarrow \{0,1\}^\ell - \{\mathsf{L}_0^W\}$ uniformly at random
    Set $e[W] = e_W = (\mathsf{L}_0^W, \mathsf{L}_1^W)$
end for
Return $e$
```

Garbled Circuit Generation. The garbler now runs a deterministic algorithm to generate the garbled circuit: $\mathsf{Circuit}(e, \mathbf{C}, \ell, \ell') = (F, D)$. This algorithm receives as input a circuit $\mathbf{C}$ with q gates and a projective input encoding set e with labels for all n input wires. The output of this algorithm is a garbled circuit F, and a set D of pairs of labels for the m output wires of the garbled circuit. This is described in Algorithm 4. This algorithm works gate-by-gate where it creates a garbled gate by calling a subroutine described in Algorithm 5. The garbled circuit so produced is $F = (\nabla_1, \ldots, \nabla_q)$.

Gate Garbling. We discuss now the subroutine that the garbling algorithm uses to garble each gate of the circuit: $(\mathsf{L}_0^g, \mathsf{L}_1^g, \nabla^g) \leftarrow \mathsf{Gate}(\mathsf{L}_0^A, \mathsf{L}_1^A, \mathsf{L}_0^B, \mathsf{L}_1^B, g, f_g, \ell)$. This subroutine receives the gate index g, input labels set $(\mathsf{L}_0^A, \mathsf{L}_1^A, \mathsf{L}_0^B, \mathsf{L}_1^B)$ and a gate functionality indicator, $f_g \in \{\mathsf{AND}, \mathsf{XOR}\}$. For simplicity and completeness, we only discuss these functionalities although we can encode any gate functionality over binary inputs. The subroutine outputs a gate garbling ∇^g (with

Algorithm 4 $\mathsf{Circuit}(e, \mathbf{C}, \ell, \ell')$

```
$\forall g \in [q]$, initialize the random oracle $\mathsf{RO}^g[2\ell, \ell']$
initialize the wire label set $W = [W_1, \ldots, W_{n+q}]$
for each circuit input wire $A$ do
    $W_A = (\mathsf{L}_0^A, \mathsf{L}_1^A) \in e$
end for
initialize $F = []$, $D = []$
for each gate $g = (f_g, A, B, g)$ in $\mathbf{C}$ in topological order do
    extract input wire labels $\mathsf{L}_0^A, \mathsf{L}_1^A, \mathsf{L}_0^B, \mathsf{L}_1^B \in W$
    compute $(\mathsf{L}_0^g, \mathsf{L}_1^g, \nabla^g) \leftarrow \mathsf{Gate}(\mathsf{L}_0^A, \mathsf{L}_1^A, \mathsf{L}_0^B, \mathsf{L}_1^B, g, f_g, \ell)$
    set $F[g] \leftarrow \nabla^g$
    set $W_g = (\mathsf{L}_0^g, \mathsf{L}_1^g) \in W$
    if $g$ is an output gate then
        $D[g] \leftarrow (\mathsf{L}_0^g, \mathsf{L}_1^g)$
    end if
end for
Return $(F, D)$
```

Hamming weight ℓ) and a set of labels for the gate output wire $(\mathsf{L}_0^g, \mathsf{L}_1^g)$, each of ℓ-bit length. The details of this subroutine are formally described in Algorithm 5. This is a deterministic function but with access to random oracle RO^g.

A gate is garbled in the following stages. First, given the set of input labels $(\mathsf{L}_0^A, \mathsf{L}_1^A,\ \mathsf{L}_0^B, \mathsf{L}_1^B)$, note that each of the combinations in $((\mathsf{L}_0^A, \mathsf{L}_0^B), (\mathsf{L}_0^A, \mathsf{L}_1^B), (\mathsf{L}_1^A, \mathsf{L}_0^B), (\mathsf{L}_1^A, \mathsf{L}_1^B))$ is a 2ℓ-bit string where ℓ bits are common with any other combination. To unlink the pairs, the input label combinations are passed into a random oracle RO^g. In order for this function to sample fresh outputs for different gates which may have potentially the same input wires, the input to RO^g also includes the gate id g as a *tweak*. For bits $a, b \in \{0, 1\}$, this step creates $\mathsf{RO}^g(\mathsf{L}_a^A, \mathsf{L}_b^B) = X_{ab}^g$. The values $(X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$ are intermediate garbling values termed 'approximate key', each ℓ'-bit long, that are the outputs of the random oracle. Note that since ℓ' is an internal length parameter and, as all internal variables are not communicated, it can be arbitrarily long without effecting the communication complexity of the garbling scheme.

Next, the set $(X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$ is used to create a gate garbling ∇^g. This lies in the heart of our construction and is one of our key contributions. The length of ∇^g is $\ell^g \leq \ell'$ and it varies for each garbled gate. Depending on the gate type, Tables 3–4 contain truth-tables that indicate how a single index of ∇^g is set as a function of the bits in the same index in each of $(X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$. These tables are part of the description of the garbling scheme and are fixed prior to running the garbling algorithm.

The garbling ∇^g is generated bit-by-bit until the Hamming weight comes to ℓ, the effective length. The gate garbling ∇^g is also made such that for any intermediate value X_{ab}^g, the output label can be derived as $\nabla^g \circ X_{ab}^g = \mathsf{L}_{f_g(a,b)}^g$ where $\circ$ is an operation that projects the bits in X_{ab}^g over all the positions where ∇^g is set to 1. An essential property that ∇^g satisfies is that on its application

Algorithm 5 $\mathsf{Gate}((\mathsf{L}_0^A, \mathsf{L}_1^A), (\mathsf{L}_0^B, \mathsf{L}_1^B), g, f_g, \ell)$

```
$X_{00}^g = \mathsf{RO}^g(\mathsf{L}_0^A, \mathsf{L}_0^B)$
$X_{01}^g = \mathsf{RO}^g(\mathsf{L}_0^A, \mathsf{L}_1^B)$
$X_{10}^g = \mathsf{RO}^g(\mathsf{L}_1^A, \mathsf{L}_0^B)$
$X_{11}^g = \mathsf{RO}^g(\mathsf{L}_1^A, \mathsf{L}_1^B)$
initialize $\nabla^g \leftarrow 0^{\ell'}$ and let $j = 1$
repeat
    Set $\mathsf{slice} \leftarrow X_{00}^g[j]||X_{01}^g[j]||X_{10}^g[j]||X_{11}^g[j]$
    if $f_g == \mathsf{AND} \wedge \mathsf{slice} \in \{0000, 0001, 1110, 1111\}$ then          ▷ See Table 3
        $\nabla^g[j] \leftarrow 1$
    else if $f_g == \mathsf{XOR} \wedge \mathsf{slice} \in \{0000, 1001, 0110, 1111\}$ then     ▷ See Table 4
        $\nabla^g[j] \leftarrow 1$
    end if
    increment $j = j + 1$
until $HW(\nabla^g) = \ell$ or $j = \ell'$
if $HW(\nabla^g) \neq \ell$ then
    ABORT the computation
end if
$\ell^g = j$
$\mathsf{L}_0^g = X_{00}^g \circ \nabla^g$          ▷ $A \circ B$ = projection of $A[i]$ for positions with $B[i] = 1$
if $f_g == \mathsf{AND}$ then
    $\mathsf{L}_1^g = X_{11}^g \circ \nabla^g$
else if $f_g == \mathsf{XOR}$ then
    $\mathsf{L}_1^g = X_{01}^g \circ \nabla^g$
end if
Return $(\mathsf{L}_0^g, \mathsf{L}_1^g, \nabla^g)$
```

with any of the X_{ab}^g, it produces one of two values L_0^g and L_1^g that are distributed uniformly at random in $\{0,1\}^\ell$ and, that too, according to the gate functionality. These, along with the gate garbling ∇^g are the outputs of this subroutine.

Decoding Information. The randomized algorithm $\mathsf{DecodingInfo}(D, \ell) \to d$ takes the labels set D for the output wires, and returns a sequence d that maps them back to their plain values; see Algorithm 6. This employs a random oracle RO'.

Completing the Garbling Scheme. It remains to describe, for completeness, the working of the input encoding algorithm En, the evaluation algorithm Ev and the output decoding algorithm De. The interfaces and purpose of these are respectively the same as in standard garbling [BHR12]. For brevity we only describe them in algorithmic form in Algorithm 7.

4.2 Motivating Our Scheme

The mainstream literature on garbled circuits has been operating under the gate-by-gate paradigm. Informally, binary gates are individually garbled in topological order. The garbling algorithm samples values for the two labels for each wire

Algorithm 6 $\mathsf{DecodingInfo}(D, \ell)$

```
initialize $\mathsf{RO}'[2\ell, 1]$ and $d = []$
for output wire $j \in [m]$ do
    extract $\mathsf{L}_0^j, \mathsf{L}_1^j \leftarrow D[j]$
    repeat
        sample $d^j \in_R \{0,1\}^\ell$
    until $\mathsf{RO}'(\mathsf{L}_0^j, d^j) = 0$ and $\mathsf{RO}'(\mathsf{L}_1^j, d^j) = 1$
    $d[j] \leftarrow d^j$
end for
Return d
```

(sometimes with additional constraints on their relations) and uses each pair of input labels as a key for encrypting the output label. This is the setting in which [LP09] proves the security of garbling schemes using a primitive that was later termed by [BHR12] a Dual-Key Cipher (DKC). Later, [ZRE15] termed this kind of garbling as 'linear'. They provided a model for linear garbling and showed that any scheme in their model that simultaneously achieves correctness and privacy requires at least two ciphertexts, thus providing a lower bound on the communication efficiency of such schemes. Our scheme deviates from [ZRE15]'s linear model in several key points.

Approximate Keys. Despite a syntactical similarity, a major difference from prior work is that we do not consider the input labels as keys. Instead, we consider them as an entropy source to an *Approximate-Key-Derivation Function.* This function, modeled as a random oracle and denoted by RO, converts each label pair into a uniformly distributed string of length ℓ'. The resulting tuple $t = (X_{00}, X_{01}, X_{10}, X_{11})$ is a set of approximate keys.

Gate Output Label Derivation. The tuple t contains approximate keys in the following sense: $t = (X_{00}, X_{01}, X_{10}, X_{11})$ can be viewed as a $4 \times \ell'$ binary matrix. The garbler scans for each $j \in [\ell']$, indices $\mathsf{slice}_j = (X_{00}[j], X_{01}[j], X_{10}[j], X_{11}[j])$. For an AND gate, the bits in the same column in X_{00}, X_{01} and X_{10} must agree on the same value. When they do, the respective position in L_0^g is set to this value and the respective position in L_1^g is set to the corresponding bit value from X_{11}. Otherwise (i.e., if they do not agree), the value in position j is not included in the construction of the output label.

Table 3 in Appendix A is a truth table according to which the indices slice_j are used to set the j^{th} index of ∇^g when the AND functionality is garbled. Note that the index in ∇^g is set to 1 only in the rows of the table where it holds that $X_{00}[j] = X_{01}[j] = X_{10}[j]$. Further, each index j of ∇^g is set independently, depending on a different slice_j. Each value in this slice is an RO output and so the slice is a uniformly random value in $\{0,1\}^4$. The right side of Table 3 contains the value in the output label that is a result of projecting the value of X_{ab}^g in the positions where ∇^g contains 1. One can see that L_{00}, L_{01}, and L_{10} always have the same value (and are therefore the same). If anywhere among

Algorithm 7 Algorithms to Evaluate the Garbling

```
procedure $\mathsf{En}(e, x)$
    initialize $X = []$
    for every $j \in [n]$ do
        set $X[j] = \mathsf{L}^j_{x_j} = e_j[x_j]$
    end for
    Return $X$
end procedure

procedure $\mathsf{Ev}(F, X)$
    initialize $Y = []$
    for each gate $g \in [q]$ in a topological order do
        $\mathsf{L}^A, \mathsf{L}^B \leftarrow$ active labels associated with the input wires of gate $g$
        extract $\nabla^g \leftarrow F[g]$ and compute $\mathsf{L}^g \leftarrow \mathsf{RO}(g, \mathsf{L}^A, \mathsf{L}^B) \circ \nabla^g$
        if $g$ is a circuit output wire then
            $Y[g] \leftarrow \mathsf{L}^g$
        end if
    end for
    Return $Y$
end procedure

procedure $\mathsf{De}(Y, d)$
    initialize $y = []$
    for $j \in [m]$ do
        $y[j] \leftarrow \mathsf{lsb}(\mathsf{RO}'(Y[j], d^j))$
    end for
    Return $y$
end procedure
```

the ℓ' positions we have $\mathsf{L}_{11} \neq (\mathsf{L}_{00} = \mathsf{L}_{01} = \mathsf{L}_{10})$ (i.e., Lines 1 and 14 in Table 3) then, $\{\mathsf{L}^A_0, \mathsf{L}^A_1\} \times \{\mathsf{L}^B_0, \mathsf{L}^B_1\} \mapsto \{\mathsf{L}^g_0, \mathsf{L}^g_1\}$ preserves the structure of a binary AND.

The case for an XOR gate is similar except that the agreement is sought between L_{00} and L_{11}, as well as between L_{01} and L_{10} (see the right side of Table 4). Then if at least once in the ℓ' positions $(\mathsf{L}_{00} = \mathsf{L}_{11}) \neq (\mathsf{L}_{01} = \mathsf{L}_{10})$, the structure of a binary XOR is preserved. Table 4 in Appendix A is a truth table according to which the indices slice_j are used to set the j^{th} index of ∇^g when the gate functionality is XOR. Note that the index in ∇^g is set to 1 only in the rows of the table where it holds that $X_{00}[j] = X_{11}[j]$ and $X_{01}[j] = X_{10}[j]$.

Both Table 3 and Table 4 are part of the description of the garbling scheme: that is, they are predetermined and remain the same regardless of the function garbled or the randomness used.

Garbling other gate functionalities. Generalizing the above technique lets us garble an n-input binary gate computing *any* functionality f_g. A gate g with n input wires and one output wire, with each wire holding binary values would have two ℓ-length labels for each wire. For each input wire indexed $i \in [n]$, let these labels be $\mathsf{L}^i_0, \mathsf{L}^i_1 \in \{0,1\}^\ell$. Garbling such a gate would require a random oracle of the form $\mathsf{RO}^g : \{0,1\}^{n\ell} \to \{0,1\}^{\ell'}$. Let $a = \{a_i\}_{i\in[n]} \in \{0,1\}^n$ be a possible input value to this gate. The garbling proceeds by first making 2^n calls to the random oracle of the form $\mathsf{RO}^g(\{\mathsf{L}_i^{a_i}\}_{i\in[n]}) \to X_a^g$, for all $a \in \{0,1\}^n$. Letting $t = \{X_a^g\}_{a\in\{0,1\}^n}$ be the set of approximate keys, we need that t be partitioned into two sets: $t^0 = \{X_a^g\}_{f_g(a)=0}$ containing all approximate keys that need to be mapped to the gate output label L_0^g, and $t^1 = \{X_a^g\}_{f_g(a)=1}$ with those that are mapped to L_1^g. Now, a table similar to Table 3,4 needs to be generated for the functionality f_g. On the left of the column with $\nabla_{f_g}^g$, this would contain $m = 2^n$ columns, one corresponding to each X_a^g. The table would contain 2^m rows corresponding to all possible values one slice of t can take. Out of these, the index in $\nabla_{f_g}^g$ is set to 1 only when the values in the slice of t^0 are equal and that of t^1 are equal. So, out of the 2^m rows, only 4 rows set $\nabla_{f_g}^g$ to 1: the row with 0^m, 1^m, the row equaling the transpose vector of the truth table of f_g, and the complement thereof. In effect, only knowing these 4 combinations suffices for gate garbling.

Let us consider the special case of 2-input binary gates for some functionality f_g. The random oracle is of the form $\mathsf{RO}^g : \{0,1\}^{2\ell} \to \{0,1\}^{\ell'}$ and there are 4 approximate keys in the tuple $t = (X_{00}^g, X_{01}^g, X_{10}^g, X_{11}^g)$. In the truth-table for setting the bits in $\nabla_{f_g}^g$, out of the $2^4 = 16$ rows, 4 rows would set the bit in $\nabla_{f_g}^g$ to 1. Therefore, when a garbling is generated for a 2-input binary gate, it has the same distribution regardless of the gate functionality. This is the basis of the gate functionality hiding property of the scheme.

No ciphertexts. In the evaluation algorithm Ev of our scheme, given two input labels, the evaluator can obtain from the random oracle exactly one approximate key whereas what they need is the output label. To enable this, the garbler records in ∇ the bit positions from which the output label is derived. We believe ∇ should not be considered a ciphertext since it does not encrypt any labels but instead encodes the relation between an approximate-key and the output label. Furthermore, whereas a ciphertext normally captures the relation between a pair of input labels and the output label, ∇ captures the relation between all input labels and both output labels. Finally, each bit of ∇ is zero with probability $\frac{3}{4}$ and one with probability $\frac{1}{4}$.

4.3 Security

We consider a PPT adversary $\mathcal{A}$ that runs for $t(\kappa)$ time steps for any polynomial $t(\cdot)$ in the security parameter κ. $\mathcal{A}$ has access to the random oracles RO^g and RO' but, owing to its running time, is restricted to making at most $t(\kappa)$ queries overall. Among these, we make a distinction between the set of honest queries H, and adversarial queries Q. Given the challenge (F, X, d) output from Algorithm 1, we term the set of queries made in $\mathsf{Ev}(F, X) = Y$ and $\mathsf{De}(Y, d)$ as the honest queries H. For a circuit $\mathbf{C}$ with q gates and m output wires, this includes q calls to RO, and m calls to RO'. Therefore, $|H| = q + m$ and its contents are determined completely by the challenge. Any other query $\mathcal{A}$ makes is an adversarial query in Q. We only consider sets Q where $|Q| < t(\kappa)$.

Definition 3. *A set of adversarial queries Q that an adversary $\mathcal{A}$ makes to the random oracles RO^g, and RO' is* ***permissible*** *if it holds that for a security parameter κ, and polynomial $t(\cdot)$,*

$$|Q| < t(\kappa)$$

Security Game. The security game for Privacy from Definition 2 is an interaction between the adversary $\mathcal{A}$ and the challenger. Let Φ be a leakage function denoting the topology of a circuit. That is, for a circuit $\mathbf{C}$, $\Phi(\mathbf{C})$ outputs everything except the gate functionality of each gate in the circuit. First $\mathcal{A}$ picks circuits $\mathbf{C}_0, \mathbf{C}_1$ of its choice such that $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$, and two inputs $x^0, x^1 \in \{0,1\}^n$ such that $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$. Then, $(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1)$ are given to the challenger. On receiving this, the challenger first samples a bit $b \leftarrow \{0,1\}$ uniformly at random. It then garbles $\mathbf{C}_b$, creating (F, e, d). It encodes x^b using e to get X. The challenge (F, X, d) is sent back to $\mathcal{A}$. Now $\mathcal{A}$ with polynomial running time, and access to the random oracles RO^g, and RO' to make honest queries H and adversarial queries Q, is tasked with guessing b that was used internally by the challenger.

The adversary's view. In the privacy game from Definition 2, $\mathcal{A}$ has in its view $(\mathbf{C}_0, \mathbf{C}_1, x^1, x^0)$ of its own choice, (F, X, d) that it receives as the challenge, the set of honest queries (and responses) H, and adversarial queries (and responses) Q. We define a function $\mathcal{V}(\cdot)$ that represents the information learnt by $\mathcal{A}$. For instance, $\mathcal{V}(F, X, d)$ refers to the information $\mathcal{A}$ can deduce from the challenge (F, X, d). In particular, by $\mathcal{V}(F, X, d, H, Q)$ we denote all the information learnt by the adversary[9]. The advantage Adv of $\mathcal{A}$ can be restated as

$$\mathsf{Adv} = \left| \Pr[\mathcal{A}(\mathcal{V}(F, X, d, H, Q)) = b] - \frac{1}{2} \right|$$

where the probability distribution is taken over the secrets of the challenger (i.e., random choice of $b \leftarrow \{0,1\}$, and the garbling randomness: $(F, e, d) \leftarrow \mathsf{Gb}(\mathbf{C}_b)$), and the choice of the adversarial query set Q. The adversary needs to distinguish between the cases that the challenger chooses $b = 0$ and $b = 1$.

Theorem 1. *Let $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ be a garbling scheme as in Algorithms 2–7. Let κ be a computational security parameter. For every PPT adversary $\mathcal{A}$ with running time $t(\kappa)$ having access to all random oracles $\mathsf{RO} \in (\mathsf{RO}^g, \mathsf{RO}')$ in the Privacy game (Definition 2), $\exists$ a negligible function μ s.t. $\mathcal{A}$'s advantage is,*

$$\mathit{Adv}(\kappa) = \left| \Pr[\mathcal{A}^{\mathsf{RO}}(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1, F, X, d) = b] - \frac{1}{2} \right| < \mu(\kappa)$$

Proof Outline. We prove that our garbling scheme preserves privacy against a PPT adversary. The privacy game (Algorithm 1) returns as a challenge

[9] We omit writing $(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1)$ and other garbling parameters like ℓ and ℓ' for brevity but it is assumed to be always included.

(F, X, d) and the adversary $\mathcal{A}$ is tasked with guessing the bit b such that $(F, e, d) \leftarrow \mathsf{Gb}(\mathbf{C}_b)$ and $X = \mathsf{En}(e, x^b)$. Note that the garbling (F, d) in isolation is distributed identically for both $\mathbf{C}_0$ and $\mathbf{C}_1$. This is because the garbling technique creates each gate garbling ∇^g in a way that it is distributed identically regardless of the gate functionality $f_g \in \{\mathsf{AND}, \mathsf{XOR}\}$ and $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$.

We denote by honest queries the RO queries that are necessary for evaluating the garbling F on X. Then our proof follows by proving in Theorem 2 that given the challenge (F, X, d) and the set of honest queries H only, the view of the adversary $\mathcal{A}$ is identically distributed for the cases where $b = 0$ and $b = 1$. In order to show this, we first prove that nothing beyond the *active path* P of the evaluation is revealed from the given information. Next, we show that the active path is identically distributed for both cases.

All queries that are not honest queries are referred to as adversarial queries. When an adversarial query is made, we make a distinction between the case where a response lies in the garbling F, and those that do not. We call the former a 'Bad Event'. When a 'Bad Event' occurs we assume an adversary can detect this, and that it gives it enough information to distinguish for b. Therefore, we bound the probability of a Bad Event for a single query in Theorem 3.

When a query does not lead to a bad event, this implies that its response is irrelevant to the construction of (F, d). Making such a query does not give the adversary any advantage over the case considered in Theorem 2. However, it restricts the domain of future queries to the random oracles. As a result, a future query to it may have a higher probability of incurring a bad event. In Theorem 4, we bound the advantage that the adversary would have on making s adversarial queries. This is done by first, calculating the probability that an i^{th} query leads to a bad event given that $i - 1$ previous queries have not triggered a bad event. This probability is an increasing function of i. Next, $\mathcal{A}$'s advantage is bounded as the complement of the probability that no bad event has occurred in s queries. The probability of no bad event occurring is calculated as the product of the complement of the individual probabilities for each round i that was previously calculated. This result also holds in the case where $\mathcal{A}$ makes $t(\kappa)$ adversarial queries, completing the proof for the main theorem: Theorem 1.

5 Supporting Free-XOR

The garbling scheme in Section 4 can be further extended to support free-XOR. The idea is similar to existing free-XOR schemes where the garbler samples a secret global offset $\Delta \in \{0,1\}^\ell$. For each input wire, the 0-label is sampled uniformly at random and the 1-label is set such that $\mathsf{L}_0 \oplus \mathsf{L}_1 = \Delta$. The XOR gate is evaluated by setting the output label as the bitwise XOR between the labels of the two input wires. This complies with the XOR gate functionality and maintains the invariant that for the output wire of the XOR gate, $\mathsf{L}_0 \oplus \mathsf{L}_1 = \Delta$. This gate itself has no garbling representation.

It now remains to show how other gate functionalities like AND are garbled so that the output wire labels maintain the same invariant. This is done by including

Algorithm 8 $\mathsf{Gate}((\mathsf{L}_0^A, \mathsf{L}_1^A), (\mathsf{L}_0^B, \mathsf{L}_1^B), g, \ell, \Delta)$

```
$X_{00}^g = \mathsf{RO}^g(\mathsf{L}_0^A, \mathsf{L}_0^B)$, $X_{01}^g = \mathsf{RO}^g(\mathsf{L}_0^A, \mathsf{L}_1^B)$, $X_{10}^g = \mathsf{RO}^g(\mathsf{L}_1^A, \mathsf{L}_0^B)$, $X_{11}^g = \mathsf{RO}^g(\mathsf{L}_1^A, \mathsf{L}_1^B)$
initialize $\nabla^g \leftarrow 0^{\ell'}$ and let $j = 1$, $j' = 1$
repeat
    $\mathsf{slice} \leftarrow \Delta[j'] || X_{00}^g[j] || X_{01}^g[j] || X_{10}^g[j] || X_{11}^g[j]$
    if $\mathsf{slice} \in \{00000, 10001, 11110, 01111\}$ then        ▷ See Table 5
        $\nabla^g[j] \leftarrow 1$ and $j' = j' + 1$
    end if
    $j = j + 1$
until $j' == \ell$
$\mathsf{L}_0^g = X_{00}^g \circ \nabla^g$
$\mathsf{L}_1^g = X_{11}^g \circ \nabla^g$
Return $(\mathsf{L}_0^g, \mathsf{L}_1^g, \nabla^g)$
```

Δ as one of the constraints, along with $t = (X_{00}, X_{01}, X_{10}, X_{11})$, that is used to create ∇^g. Table 5 in Appendix A indicates the new set of constraints. In this table, the index j in ∇^g is set to 1 only when the indices in $X_{00} = X_{01} = X_{10}$ and when for the desired index j' in Δ, it holds that $X_{00} \oplus X_{11} = \Delta$. Algorithm 8 details the gate garbling algorithm for the AND gate, supporting free-XOR. Note that while the index j is incremented in every iteration, going over all the indices in ∇^g, the index j' is incremented only when one bit in ∇^g is set to 1, so as to move to the next element in Δ. This continues until the ℓ bits in Δ are exhausted.

Note that out of the 32 different ways that $\nabla^g[j]$ can be set in Table 5, only $\frac{1}{8}$ of them sets it to 1 and the rest set the bit to 0. So in order to maintain a Hamming weight of ℓ in ∇^g, its size would become 8ℓ in expectation. Therefore, supporting free-XOR in this scheme incurs the cost of increasing the size of the gate garbling ∇^g by double in expectation.

Although this modification to the scheme for free-XOR compatibility increases the size of the garbled gates, the same security analysis as in the proof in Appendix B, with the exception that the leakage function Φ now not only reveals the topology of the circuit, but also the position of the XOR gates. This leakage function is also invertable and so the extension from indistinguishability based privacy to simulation based privacy still holds.

Our security proof follows a ‘Bad Event’ analysis and the advantage that the adversary gains is calculated in terms of the probability of encountering a bad event. Therefore the advantage in the privacy game for this scheme with free-XOR can be calculated as being the same as that for the scheme in Section 4. However, there remains a crucial difference between the modified scheme and the original. In the original scheme, we assume for simplicity that if a Bad Event is encountered, the adversary can distinguish and security is violated. However, this may not always be the case. In practice, encountering a bad event would only aid in violating privacy when it is encountered at certain favourable wires or gates, depending on the circuit topology. But, in the modified scheme, encountering a bad event means finding an inactive output label and along with the active value already known, this reveals the value of the global offset $\varDelta$. Then all the inactive labels throughout the garbling can be found and the privacy is violated.

Summarizing, in the original scheme, in practice, encountering a Bad Event may not always violate privacy. But, when the scheme is modified for free-XOR, encountering a Bad Event would always violate privacy. Owing to our conservative approach, the proof in Appendix B accounts for the worst case scenario when analyzing the scheme in Section 4, and is therefore agnostic to whether free-XOR compatibility was leveraged or not. Therefore both schemes are secure.

Garbling Other Gates. Extending the discussion in Section 4.2, a general n-input binary gate computing any functionality f_g can be garbled in a free-XOR compatible way. Here again, a random oracle is used to generate the 2^n approximate keys $t = \{X^g_a\}_{a\in\{0,1\}^n}$. Out of these, let $t^0 = \{X^g_a\}_{f_g(a)=0}$, and $t^1 = \{X^g_a\}_{f_g(a)=1}$. Similar to Table 5, the table for creating $\nabla^g_{f_g}$ contains $m = 2^n + 1$ columns on the left where 2^n columns correspond to the approximate keys and one column is for the global offset $\varDelta$. The number of rows in the table would be 2^m out of which only 4 would set ∇^g to 1: the rows where each element in t^0 takes the same value b_0, elements in t^1 takes value b_1, and $b_0 \oplus b_1$ equals the value in $\varDelta$.

Acknowledgments. The authors would like to thank the anonymous reviewers for their useful comments. In addition, we would like to thank Mike Rosulek for spotting an omission in an early version.

Anasuya Acharya, Efrat Cohen and Carmit Hazay are supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by ISF grant No. 1316/18. Carmit Hazay is also supported by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Algorand Foundation.

References

App13. Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. *IACR Cryptol. ePrint Arch.*, page 699, 2013.

App17. Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In *Tutorials on the Foundations of Cryptography*, pages 1–44. 2017.

BHKR13. Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE SP*, pages 478–492, 2013.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM CCS*, pages 784–796, 2012.

BLO16. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *ACM SIGSAC*, pages 578–590, 2016.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *ACM*, pages 503–513, 1990.

FNO15. Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT*, pages 191–219, 2015.

GGP10. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

GKP$^+$13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564. ACM, 2013.

GKPS18. Chaya Ganesh, Yashvanth Kondi, Arpita Patra, and Pratik Sarkar. Efficient adaptively secure zero-knowledge from garbled circuits. In *PKC*, pages 499–529, 2018.

HK20. David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In *CRYPTO*, pages 763–792, 2020.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

Ish13. Yuval Ishai. Randomization techniques for secure computation. In *Secure Multi-Party Computation*, volume 10 of *Cryptology and Information Security Series*, pages 222–248. 2013.

KKPW21. Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Daniel Wichs. Limits on the adaptive security of yao's garbling. In *CRYPTO*, pages 486–515, 2021.

KKS16. Carmen Kempka, Ryo Kikuchi, and Koutarou Suzuki. How to circumvent the two-ciphertext lower bound for linear garbling schemes. In *ASIACRYPT*, pages 967–997, 2016.

KL14. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. 2014.

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for XOR gates that beats free-xor. In *CRYPTO*, pages 440–457, 2014.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, 2009.

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *ACM-EC*, pages 129–139, 1999.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, pages 250–267, 2009.

RR21. Mike Rosulek and Lawrence Roy. Three halves make a whole? beating the half-gates lower bound for garbled circuits. In *CRYPTO*, pages 94–124, 2021.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FoCS*, pages 162–167, 1986.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, pages 220–250, 2015.

A Additional Details of the Scheme

Table 3. For a gate index g and $j \in [\ell']$, this table defines $\nabla^g_\wedge[j]$ (where $f_g = \mathsf{AND}$) as a function in $X^g_{00}[j], X^g_{01}[j], X^g_{10}[j], X^g_{11}[j]$. The right side demonstrates how $X^g_{ab}[j] \circ \nabla^g[j]$ collapses into only two distinct values $\mathsf{L}^g_0 = \mathsf{L}_{00} = \mathsf{L}_{01} = \mathsf{L}_{10}$ and $\mathsf{L}^g_1 = \mathsf{L}_{11}$. Each row in the table corresponds to one bit-slice of the values $X^g_{ab}[j]$ for $a, b \in \{0, 1\}$.

	X^g_{00}	X^g_{01}	X^g_{10}	X^g_{11}	$\nabla^g_\wedge$	L_{00}	L_{01}	L_{10}	L_{11}
0	0	0	0	0	1	0	0	0	0
1	0	0	0	1	1	0	0	0	1
2	0	0	1	0	0	-	-	-	-
3	0	0	1	1	0	-	-	-	-
4	0	1	0	0	0	-	-	-	-
5	0	1	0	1	0	-	-	-	-
6	0	1	1	0	0	-	-	-	-
7	0	1	1	1	0	-	-	-	-
8	1	0	0	0	0	-	-	-	-
9	1	0	0	1	0	-	-	-	-
10	1	0	1	0	0	-	-	-	-
11	1	0	1	1	0	-	-	-	-
12	1	1	0	0	0	-	-	-	-
13	1	1	0	1	0	-	-	-	-
14	1	1	1	0	1	1	1	1	0
15	1	1	1	1	1	1	1	1	1

Table 4. For a gate index g and $j \in [\ell']$, this table defines $\nabla^g_\oplus[j]$ (where $f_g = \mathsf{XOR}$) as a function in $X^g_{00}[j], X^g_{01}[j], X^g_{10}[j], X^g_{11}[j]$. In addition, the right side demonstrates how $X^g_{ab}[j] \circ \nabla^g[j]$ collapses into only two distinct values $\mathsf{L}^g_0 = \mathsf{L}_{00} = \mathsf{L}_{11}$ and $\mathsf{L}^g_1 = \mathsf{L}_{01} = \mathsf{L}_{10}$. Each row in the table corresponds to one bit-slice of the values $X^g_{ab}[j]$ for $a, b \in \{0, 1\}$.

	X^g_{00}	X^g_{01}	X^g_{10}	X^g_{11}	$\nabla^g_\oplus$	L_{00}	L_{01}	L_{10}	L_{11}
0	0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	-	-	-	-
2	0	0	1	0	0	-	-	-	-
3	0	0	1	1	0	-	-	-	-
4	0	1	0	0	0	-	-	-	-
5	0	1	0	1	0	-	-	-	-
6	0	1	1	0	1	0	1	1	0
7	0	1	1	1	0	-	-	-	-
8	1	0	0	0	0	-	-	-	-
9	1	0	0	1	1	1	0	0	1
10	1	0	1	0	0	-	-	-	-
11	1	0	1	1	0	-	-	-	-
12	1	1	0	0	0	-	-	-	-
13	1	1	0	1	0	-	-	-	-
14	1	1	1	0	0	-	-	-	-
15	1	1	1	1	1	1	1	1	1

Table 5. For a gate index g, $j \in [\ell']$ and $j' \in [\ell]$, this table defines $\nabla^g_\wedge[j]$ (where $f_g = \mathsf{AND}$) as a function of $X^g_{00}[j], X^g_{01}[j], X^g_{10}[j], X^g_{11}[j]$ and $\Delta[j']$. In addition, the right side demonstrates how combining $X^g_{ab}[j] \circ \nabla^g[j]$ collapses into only two distinct values $\mathsf{L}^g_0 = \mathsf{L}_{00} = \mathsf{L}_{01} = \mathsf{L}_{10}$ and $\mathsf{L}^g_1 = \mathsf{L}_{11}$ such that $\mathsf{L}^g_0 \oplus \mathsf{L}^g_1 = \Delta$. Each row in the table corresponds to one bit-slice of the values $X^g_{ab}[j]$ for $a, b \in \{0,1\}$.

	Δ	X^g_{00}	X^g_{01}	X^g_{10}	X^g_{11}	$\nabla^g_\wedge$	L_{00}	L_{01}	L_{10}	L_{11}
0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	-	-	-	-
2	0	0	0	1	0	0	-	-	-	-
3	0	0	0	1	1	0	-	-	-	-
4	0	0	1	0	0	0	-	-	-	-
5	0	0	1	0	1	0	-	-	-	-
6	0	0	1	1	0	0	-	-	-	-
7	0	0	1	1	1	0	-	-	-	-
8	0	1	0	0	0	0	-	-	-	-
9	0	1	0	0	1	0	-	-	-	-
10	0	1	0	1	0	0	-	-	-	-
11	0	1	0	1	1	0	-	-	-	-
12	0	1	1	0	0	0	-	-	-	-
13	0	1	1	0	1	0	-	-	-	-
14	0	1	1	1	0	0	-	-	-	-
15	0	1	1	1	1	1	1	1	1	1
16	1	0	0	0	0	0	-	-	-	-
17	1	0	0	0	1	1	0	0	0	1
18	1	0	0	1	0	0	-	-	-	-
19	1	0	0	1	1	0	-	-	-	-
20	1	0	1	0	0	0	-	-	-	-
21	1	0	1	0	1	0	-	-	-	-
22	1	0	1	1	0	0	-	-	-	-
23	1	0	1	1	1	0	-	-	-	-
24	1	1	0	0	0	0	-	-	-	-
25	1	1	0	0	1	0	-	-	-	-
26	1	1	0	1	0	0	-	-	-	-
27	1	1	0	1	1	0	-	-	-	-
28	1	1	1	0	0	0	-	-	-	-
29	1	1	1	0	1	0	-	-	-	-
30	1	1	1	1	0	1	1	1	1	0
31	1	1	1	1	1	0	-	-	-	-

A.1 Random Oracle Query Independence

The following lemma characterizes a key property of the random oracle. Here we denote by RO the random oracle itself, and by $\mathcal{V}(\cdot)$ the information learnt by an adversary, even when potentially unbounded.

Lemma 1 (Query Independence). *Let* $\mathsf{RO} : \{0,1\}^a \to \{0,1\}^b$ *be a random oracle with fixed sized inputs of length* a*. Let* $Q = (q_1, \ldots, q_m)$ *be the queries made to the random oracle. Let* $R = (r_1, \ldots, r_m)$ *be the set of responses such that for each query* q_j*,* r_j *is its response. Then* $\mathcal{V}(Q) = \{Q, R\}$*. For a query* $q \notin Q$*, for all random choices of responses* $r \in \{0,1\}^b$*,*

$$\Pr[\mathsf{RO}(q) = r | \mathcal{V}(Q)] = \Pr[\mathsf{RO}(q) = r]$$

A.2 Setting the Length of RO Output

The random oracle RO is employed in each gate in the garbling to derive the *approximate keys* $t = (X_{00}, X_{01}, X_{10}, X_{11})$ from the gate input labels $\mathsf{L}_0^A, \mathsf{L}_1^A$ and $\mathsf{L}_0^B, \mathsf{L}_1^B$. This oracle RO takes as an input a gate id $g \in [q]$ as a tweak, and two ℓ-bit labels: one from each input wire A and B. It outputs an ℓ'-bit value X_{ab}. In the garbling scheme, for a security parameter κ, we set $\ell = \kappa$ and $\ell' = 8\ell = 8\kappa$. In this section we discuss the reason why ℓ' is set this way in terms of κ.

The primary reason stems from the nature of the algorithm used to create ∇^g. The gate garbling ∇^g is created bit-by-bit independently until it contains ℓ positions with 1. From Table 3,4 it is evident that a position j in ∇^g is set to 1 with probability $\frac{1}{4}$ over a random choice of $(X_{00}[j], X_{01}[j], X_{10}[j], X_{11}[j]) \in \{0,1\}^4$. As these bits originate from random oracle outputs, they are indeed distributed uniformly at random. Therefore, ℓ' needs to be set such that the probability of ∇^g having Hamming weight $< \ell$ is negligible in κ. Let us now examine this probability for $\ell' = 8\kappa$.

For a gate g, let H be a random variable that denotes the Hamming weight of ∇^g derived from a random $t = (X_{00}, X_{01}, X_{10}, X_{11})$ where each $X_{ab} \in \{0,1\}^{\ell'}$.

$$H \sim \mathsf{Binomial}(\ell', \frac{1}{4}) = \mathsf{Binomial}(8\kappa, \frac{1}{4})$$

This random variable has a mean $\mu = np = \frac{8\kappa}{4} = 2\kappa$ and variance $\sigma^2 = npq = 1.5\kappa$. Then, using Hoeffding's inequality for Binomial Distributions,

$$\begin{aligned}\Pr[H < \kappa] \leq & e^{-2n(p-\frac{k}{n})^2} \\ = & e^{-16\kappa(\frac{1}{4}-\frac{\kappa}{8\kappa})^2} \\ = & e^{-\frac{\kappa}{4}}\end{aligned}$$

which is negligible in κ.

B Proof of Theorem 1

Before stating the proof itself, we define certain terms used within our proof.

B.1 Proof Setup

In the security game, the adversary's goal given (F, X, d) is to distinguish whether $(F, d) \leftarrow \mathsf{Gb}(C_0)$ and $X = \mathsf{En}(e, x^0)$, or $(F, d) \leftarrow \mathsf{Gb}(C_1)$ and $X = \mathsf{En}(e, x^1)$. Going forward, for a gate g, we denote by L^A and L^B the active input labels, and by L^g the active output label. These values are revealed during the evaluation of F on X. We show in our proof that the knowledge of these *active values only* gives the adversary $\mathcal{A}$ zero advantage in distinguishing b.

We denote by L^{A*} and L^{B*} the inactive input labels, and by L^{g*} the inactive output label. We term as a "Bad Event", the case where an adversary $\mathcal{A}$ learns an inactive label for any wire in F. These reveal additional information about the circuit and potentially its correlation to X, leading the adversary to gain advantage in distinguishing in the privacy game. For simplicity of analysis, we assume that when a random oracle query leads to a 'Bad Event', privacy is already violated, without needing further work/queries from the adversary. For any wire indexed i, we denote by $\mathsf{L}^{i'}$ a *candidate* for an inactive label. Such a candidate is queried to the random oracle in order to learn whether it is the inactive label or not. We bound the probability of a "Bad Event" by computing a bound on the number of such possible queries. We then argue that for each query to $\mathsf{RO} \in (\mathsf{RO}^g, \mathsf{RO}')$, the probability of encountering a bad event is negligible.

We denote by Q the set of *adversarial queries and responses.* Setting $|Q| = s$,

$$Q = \Big\{ [g_i, q_i, r_i] \Big\}_{i \in [s]}$$

where g_i is the gate index for when RO is queried, q_i is the value input during the i^{th} query, and r_i is its respective response.

We denote by H, the set of *honest queries and responses.* Given the challenge (F, X, d), this is the set of queries to $\mathsf{RO}^g, \mathsf{RO}'$ that are made within $\mathsf{Ev}(F, X) = Y$ and $\mathsf{De}(Y, d) = y$. The set H has the following form:

$$H = \begin{cases} \{[g, (\mathsf{L}^{A_g}, \mathsf{L}^{B_g}), X^{AB_g}]\}_{g \in [q]} \\ \{[-, (Y[j], d^j), y_j]\}_{j \in [m]} \end{cases}$$

We denote by P the corresponding *active path* in F. P contains all the values revealed when evaluating the garbling (F, d) on X. All the elements in P can be derived from the elements in H. P has the form:

$$P = \Big\{ \big\{ \mathsf{L}^{A_g}, \mathsf{L}^{B_g}, X^{AB_g}, \mathsf{L}^g \big\}_{g \in [q]} \bigcup \big\{ Y[j], d^j, y_j \big\}_{j \in [m]} \Big\}$$

Note that all the labels L^w for each wire $w \in [n + q]$ are of length ℓ-bits (Table 2). This is also the length of the output labels $Y[j]$ and decoding labels d^j for each output wire $j \in [m]$. For each gate $g \in [q]$, the values X^{AB_g} has length ℓ'-bits. This is an upper bound on the length of the gate garbling ∇^g. Each ∇^g has Hamming weight ℓ. This Hamming weight is the *effective key length* and the indices in ∇^g containing 1 are termed as the *effective key positions.*

Definition 4 (Bad Event 1). *For a gate g with a garbling ∇^g, let $\mathcal{L}^B$ be the set of candidate inactive labels for input wire B. Let L^{g*} be the inactive output label and L^g be the active output label. 'Bad Event 1' occurs when for $\mathsf{L}^{b'} \in \mathcal{L}^B$ that is queried by the adversary to RO, it holds that,*

$$\mathsf{RO}^g(\mathsf{L}^A, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\}$$

For simplicity, we treat the test for whether a candidate output label $\mathsf{L}^{g'}$ is the inactive label L^{g*} as requiring zero additional calls after the call to RO^g for Bad Event 1 (Definition 4).

Lemma 2. *In the same setting as in Definition 4, let $B_{\mathcal{L}^B} \subseteq \mathcal{L}^B$ be the set of candidates leading to 'Bad Event 1', $\mathsf{L}^{b'}$ be the candidate queried in the i-th query, and $\mathcal{L}_i \subseteq \mathcal{L}^B$ the set consisting of the previous $i-1$ queried candidates. For effective key length ℓ of ∇^g it holds that,*

$$\Pr[\mathsf{RO}^g(\mathsf{L}^A, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | B_{\mathcal{L}^B} \cap \mathcal{L}_i = \varnothing] \leq \frac{1}{2^\ell - i} + 2^{-\ell+1}$$

i.e., the probability that the i-th query triggers 'Bad Event 1' is upper bounded by $\frac{1}{2^\ell - i - 1} + 2^{-\ell+1}$ as long as none of the previous queries triggered the same.

Proof: Let $\mathcal{S} = \mathcal{L}^B - \mathcal{L}_i$ be the set of candidate input labels that have not yet been queried. Note that the size of the set $\mathcal{S} \geq 2^\ell - i - 1$. Let E be the event that $B_{\mathcal{L}^B} \cap \mathcal{L}_i = \varnothing \cap \mathsf{L}^{b'} \notin \mathcal{L}_i$, that is, a new label is being queried and none of the previous $i-1$ queries have triggered a Bad Event. We calculate the probability of 'Bad Event 1' by considering two cases. One case is when the inactive input label is chosen: $\mathsf{L}^{b'} = \mathsf{L}^{B*} \in \mathcal{S}$. Querying on this yields one of L^{g*} or L^g (according to the gate functionality) with probability 1. The other case is when any other candidate $\mathsf{L}_i^{b'} \in \mathcal{S}$ is picked. Since the output of RO^g is a truly random string in $\{0,1\}^\ell$, it can yield L^{g*} or L^g with probability $\frac{2}{2^\ell}$. Therefore,

$$\begin{aligned}
&\Pr[\mathsf{RO}^g(\mathsf{L}^A, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | E] \\
&\quad = \Pr[\mathsf{RO}^g(\mathsf{L}^A, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | E, \mathsf{L}^{b'} = \mathsf{L}^{B*}] \cdot \Pr[\mathsf{L}^{b'} = \mathsf{L}^{B*} \big| E] \\
&\quad + \Pr[\mathsf{RO}^g(\mathsf{L}^A, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | E, \mathsf{L}^{b'} \neq \mathsf{L}^{B*}] \cdot \Pr[\mathsf{L}^{b'} \neq \mathsf{L}^{B*} \big| E] \\
&\quad = 1 \cdot \frac{1}{2^\ell - i - 1} + \frac{2}{2^\ell} \cdot \frac{2^\ell - i - 2}{2^\ell - i - 1} \\
&\quad \approx \frac{1}{2^\ell - i - 1} + 2^{-\ell+1}
\end{aligned}$$

□

Symmetrically, we define Bad Event 2 and 3 as follows:

Definition 5 (Bad Event 2). *For a gate g with a garbling ∇^g, let $\mathcal{L}^A$ be the set of candidate inactive labels for input wire A. Let L^{g*} be the inactive output label and L^g be the active output label. 'Bad Event 2' occurs when for $\mathsf{L}^{a'} \in \mathcal{L}^A$, it holds that,*

$$\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^B) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\}$$

Definition 6 (Bad Event 3). *For a gate g with a garbling ∇^g, let $\mathcal{L}^B$ and $\mathcal{L}^A$ be the set of candidate inactive labels for input wire B and A respectively. Let L^{g*} be the inactive output label and L^g be the active output label. 'Bad Event 3' occurs when for $\mathsf{L}^{b'} \in \mathcal{L}^B$ and $\mathsf{L}^{a'} \in \mathcal{L}^A$, it holds that,*

$$\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\}$$

Corollary 1. *In the same setting as Definition 5, let $B_{\mathcal{L}^A} \subseteq \mathcal{L}^A$ the set of candidates leading to 'Bad Event 2', $\mathsf{L}^{a'}$ be the candidate queried in the i-th query, and $\mathcal{L}_i \subseteq \mathcal{L}^A$ the set consisting of the previous $i-1$ queried candidates. For effective key length ℓ of ∇^g it holds that,*

$$\Pr[\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^B) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | B_{\mathcal{L}^A} \cap \mathcal{L}_i = \varnothing] \leq \frac{1}{2^\ell - i - 1} + 2^{-\ell+1}$$

i.e., the probability that the i-th query triggers 'Bad Event 2' is upper bounded by $\frac{1}{2^\ell - i} + 2^{-\ell+1}$ as long as none of the previous queries triggered the same.

Corollary 2. *In the same setting as Definition 6, let $B_{\mathcal{L}^A, \mathcal{L}^B} \subseteq \mathcal{L}^A \times \mathcal{L}^B$ be the ordered set of candidates leading to 'Bad Event 3', $\mathsf{L}^{b'}$ and $\mathsf{L}^{a'}$ be the candidate queried in the i-th query, and $\mathcal{L}_i \subseteq \mathcal{L}^A \times \mathcal{L}^B$ be the set consisting of the previous $i-1$ queries. For effective key length ℓ of ∇^g it holds that,*

$$\Pr[\mathsf{RO}^g(\mathsf{L}^{a'}, \mathsf{L}^{b'}) \circ \nabla^g \in \{\mathsf{L}^g, \mathsf{L}^{g*}\} | B_{\mathcal{L}^A, \mathcal{L}^B} \cap \mathcal{L}_i = \varnothing] \leq \frac{1}{2^\ell - i - 1} + 2^{-\ell+1}$$

i.e., the probability that the i-th query triggers 'Bad Event 3' is upper bounded by $\frac{1}{2^\ell - i} + 2^{-\ell+1}$ as long as none of the previous queries triggered the same.

B.2 The Complete Proof

Theorem 2 (Honest-but-Curious Adversarial Behaviour). *Let $\mathcal{A}$ be a PPT adversary. In the privacy game as in Algorithm 1, given $(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1)$ of $\mathcal{A}$'s choice such that $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$ and $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$, the challenge (F, X, d), and H, the set of honest queries only, it holds that,*

$$\begin{aligned} &\Pr[F, d \leftarrow \mathsf{Gb}(\mathbf{C}_0), X = \mathsf{En}(e, x^0) | \mathcal{V}(F, X, d, H)] \\ =&\Pr[F, d \leftarrow \mathsf{Gb}(\mathbf{C}_1), X = \mathsf{En}(e, x^1) | \mathcal{V}(F, X, d, H)] \end{aligned}$$

Proof: Before proving the above theorem, consider the following lemmas:

Lemma 3 (Honest Queries reveal only the Active Path). *Let (F, X, d) be the challenge that is output from Algorithm 1. Let H be the set of honest queries and let P be the active path. Then,*

$$\mathcal{V}(F, X, d, H) = P$$

Informally, the proof follows in two stages. The first step is to show that $\mathcal{V}(F, X, d, H)$ does indeed include P. This can be shown by the construction of the garbling scheme. Next, it remains to show that nothing beyond P is revealed. In order to prove this, we show that given P, the tuple (F, X, d, H) can be constructed. This completes the proof.

Proof: The proof follows in two steps. First we need to show that P can indeed be derived from $\mathcal{V}(F, X, d, H)$. That is,

$$P \subseteq \mathcal{V}(F, X, d, H)$$

This holds by construction. The active path P can be derived from $\mathcal{V}(F, X, d, H)$ since all of its elements can be determined from H. Recall, H is the set of honest queries to the random oracles that are necessary for computing $Y = \mathsf{Ev}(F, X)$ and $y = \mathsf{De}(Y, d)$ from the challenge. By definition, it has the form,

$$H = \begin{cases} \{[g, (\mathsf{L}^{A_g}, \mathsf{L}^{B_g}), X^{AB_g}]\}_{g\in[q]} \\ \{[-, (Y[j], d^j), y_j]\}_{j\in[m]} \end{cases}$$

So (F, X, d, H) does indeed complete all the information in the active path:

$$P = \left\{ \{\mathsf{L}^{A_g}, \mathsf{L}^{B_g}, X^{AB_g}, \mathsf{L}^g\}_{g\in[q]} \bigcup \{Y[j], d^j, y_j\}_{j\in[m]} \right\}$$

In order to complete the proof of the theorem, it remains to show that nothing beyond P is revealed from $\mathcal{V}(F, X, d, H)$. That is,

$$P \supseteq \mathcal{V}(F, X, d, H)$$

We show that P alone can be used to recreate the tuple (F, X, d, H). First, note that X contains the set of active labels for all circuit input wires. This is contained within P. The set $d = \{d^j\}_{j\in[m]}$ is the decoding information, also contained within P. H can also be determined by P. For each gate g, the elements $(\mathsf{L}^{A_g}, \mathsf{L}^{B_g}, X^{AB_g}) \in P$ are the query and response for RO^g. The set $\{Y[j], d^j, y_j\}_{j\in[m]}$ is the set of RO' query and responses in H. Finally, F is a set of gate garblings, ∇^g. For each $g \in [q]$, this can be derived from examining X^{AB_g} and L^g: ∇^g is set to 1 for only those positions in X^{AB_g} whose projection gives L^g. This completes the proof. □

Lemma 4 (Active Paths are Identically Distributed). *For the garbling $(F_0, d_0, e) \leftarrow \mathsf{Gb}(\mathbf{C}_0)$, let $X_0 = \mathsf{En}(e, x^0)$ and let P_0 and H_0 be the corresponding active path and honest queries set. Similarly, For the garbling $(F_1, d_1, e) \leftarrow \mathsf{Gb}(\mathbf{C}_1)$, let $X_1 = \mathsf{En}(e, x^1)$ and let P_1 and H_1 be the active path and honest queries set. Then if $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$ and $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$, it holds that,*

$$\{F_0, d_0, X_0, P_0, H_0\} \equiv \{F_1, d_1, X_1, P_1, H_1\}$$

Proof: The proof for this considers the distribution $A_0 = \{F_0, d_0, X_0, P_0, H_0\}$ that is derived using $\mathbf{C}_0$ and x^0, and the distribution $A_1 = \{F_1, d_1, X_1, P_1, H_1\}$ that is derived using $\mathbf{C}_1$ and x^1. Let us examine these distributions:

- In both distributions, the garbling $(F_0, d_0) \in A_0$ and $(F_1, d_1) \in A_1$ are distributed the same way. The garbling F_0, d_0 are a garbling of $\mathbf{C}_0$, and F_1, d_1 are a garbling of $\mathbf{C}_1$ using the garbling scheme in Algorithms 2-6. It holds that their topology, $\Phi(\mathbf{C}_0) = \Phi(\mathbf{C}_1)$. The garbling produced does not reveal any information beyond Φ. This is because the gate garbling ∇^g is distributed the same way regardless of the functionality $f_g \in \{\mathsf{AND}, \mathsf{XOR}\}$ due to the nature of Algorithm 5 and Table 3,4.

- Considering the complete challenge $(F_0, d_0, X_0) \in A_0$ and $(F_1, d_1, X_1) \in A_1$, note that the active input labels sets contain labels that are sampled independently and uniformly at random from $\{0,1\}^\ell$. These distributions, without making any random oracle queries, is also identically distributed since X and F, d are independent when no RO queries are made.
- On evaluating the challenges, note that $\mathbf{C}_0(x^0) = \mathbf{C}_1(x^1)$ and so the distributions cannot be distinguished on the basis of the output of the evaluation. The honest queries in the set H_0 are determined by (F_0, X_0, d_0). The distribution of these queries is identical to that in H_1 that are determined by (F_1, X_1, d_1). This is because the probability that the random oracle query responses are distributed as in H_0 is the same as the probability of it being as in H_1. Therefore $(F_0, d_0, X_0, H_0) \in A_0$ and $(F_1, d_1, X_1, H_1) \in A_1$ are identically distributed.
- The active paths P_0 and P_1 are determined completely by H_0 and H_1.

Therefore,

$$\{F_0, d_0, X_0, P_0, H_0\} \equiv \{F_1, d_1, X_1, P_1, H_1\}$$

□

From Lemma 3, given (F, X, d) and the honest queries H only, nothing beyond the active path P is revealed. Lemma 4 shows that the active paths for any $\mathbf{C}_0, x^0$ and $\mathbf{C}_1, x^1$ is identically distributed. Therefore, the theorem follows. □

Theorem 3 shows a bound on the advantage from a single adversarial query.

Theorem 3 (Advantage of a single malicious query). *Let $\mathcal{A}$ be a PPT adversary and ℓ be the effective key length. Given the challenge (F, X, d) as in Algorithm 1, $\mathcal{A}$'s advantage on a single adversarial query is bounded by,*

$$\mathit{Adv}_{|Q|=1} \leq 2^{-\ell} + \frac{1}{2^\ell - 2}$$

Proof: From Lemma 2 and Corollary 1 and 2, we can conclude that when the adversary $\mathcal{A}$ makes one adversarial query, it can encounter at most one of the 3 'Bad Events'. The probability of the same happening can be bounded as,

$$\begin{aligned} \Pr[\text{Bad Event}] &\leq \max_{j\in[3]}(\Pr[\text{Bad Event j}]) \\ &\leq \frac{1}{2^\ell - 2} + 2^{-\ell} \end{aligned}$$

It can also only gain advantage if the query it makes corresponds to a 'Bad Event'. Therefore, $\mathcal{A}$'s advantage on a single adversarial query is,

$$\mathsf{Adv}_{|Q|=1} \leq \Pr[\text{Bad Event}] \leq \max_{j\in[3]}(\Pr[\text{Bad Event j}]) \leq \frac{1}{2^\ell - 2} + 2^{-\ell}$$

□

Theorem 4 extends the result above to provide a bound on the advantage gained from multiple adversarial queries.

Theorem 4 (Advantage in multiple malicious queries). *Let $\mathcal{A}$ be a PPT adversary and ℓ be the effective key length. Given (F, X, d) as in Algorithm 1, the honest queries set H, and adversarial queries Q s.t. $|Q| = s$, $\mathcal{A}$'s advantage is,*

$$\mathsf{Adv}_{|Q|=s} < \frac{s}{2^\ell - 2}$$

Proof: In order to prove the above theorem, note that a query made by an adversary $\mathcal{A}$ can be broadly classified under one of the following categories:

1. An *Honest Query* where the query and the response for the random oracle lies on the active path of the garbling in the challenge (F, X, d). Theorem 2 shows that given all the queries H in the active path, $\mathcal{A}$'s advantage is 0.
2. An *Adversarial Query yielding a 'Bad Event'* is a query other than an Honest Query for which the response of the random oracle lies within the garbling in the challenge. This may reveal information about the garbling beyond the active path. On such an event, without loss of generality, we consider privacy as violated. Our proof builds towards bounding the probability of this event.
3. An *Adversarial Query not yielding a 'Bad Event'* is a random oracle query and response that can evidently not be involved in the construction of the challenge garbling. Making queries to the RO that yield such responses do not help identify the inactive path and therefore give no advantage. That is, given the honest-query-set H, and adversarial queries that do not lead to a 'Bad Event', this will at most help narrow down the domain of the RO. This helps increase the probability of eventually encountering a 'Bad Event'. However, until the 'Bad Event' is encountered this gives $\mathcal{A}$ no advantage over possessing H.

Let q_i be the event that the i^{th} adversarial query takes place given that all $i - 1$ queries before it have not lead to any bad event. We have from Lemma 2, and Corollary 1 and 2 that each of the 'Bad Events' 1, 2 and 3 are bounded as,

$$\Pr[\text{ Bad Event } \in \{1, 2, 3\} \big| q_i] \approx \frac{1}{2^\ell - i - 1} + 2^{-\ell}$$

Note that the probability of the 'Bad Event' increases with the increase in the number of queries and in each query, the probability of encountering any 'Bad Event' at all is calculated as the maximum of these above probabilities. Let us now compute, the probability that a 'Bad Event' is encountered given $|Q| = s$ adversarial queries to the same random oracle:

$$\begin{aligned}
\Pr[\text{ Bad Event } \big| |Q| = s] &= 1 - \Pr[\neg \text{ Bad Event } \big| |Q| = s] \\
&= 1 - \mathit{\Pi}_{i=1}^{s} 1 - \Pr[\text{ Bad Event } \big| q_i] \\
&< 1 - \mathit{\Pi}_{i=1}^{s} \left(1 - \frac{1}{2^\ell - i - 1} - 2^{-\ell} \right) \\
&\approx 1 - \mathit{\Pi}_{i=1}^{s} \left(\frac{2^\ell - i - 2}{2^\ell - i - 1} \right) \\
&= 1 - \frac{2^\ell - s - 2}{2^\ell - 2} \qquad = \frac{s}{2^\ell - 2}
\end{aligned}$$

We have seen in the proof for Theorem 3 that one adversarial query can trigger at most 1 'Bad Event' and the adversary $\mathcal{A}$'s advantage is bounded by the probability of a 'Bad Event' occurring. Given an adversarial query, if the response leads to a 'Bad Event', we assume that privacy is violated. If it does not, the views of the adversary are still identical. We therefore need to calculate the probability of at least one 'Bad Event' among $|Q| = s$ adversarial queries.

The above is a bound on the probability of a 'Bad Event' on a particular random oracle $\mathsf{RO} \in (\mathsf{RO}^g, \mathsf{RO}')$. It remains to extend this result to the case where adversarial queries were made to different random oracles across different gate garblings in the circuit. Note that each random oracle used in the construction is independent. So the result of queries to one random oracle do not affect the result of making (even the same) queries to a different random oracle, except for possibly reusing the query space as a result of seeing a query output without triggering a bad event. So the above is an upper bound that also extends to the case where not all of the previous $i-1$ queries have been made to the same random oracle since all those cases are bounded by this case. Hence it follows again that when $|Q| = s$, $\mathcal{A}$'s advantage is bounded by:

$$\mathsf{Adv}_{|Q|=s} < \frac{s}{2^\ell - 2}$$

□

Summing up, the proof of our final theorem follows.

Theorem 1 (Overall advantage of a malicious adversary - Restated). *Let* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *be a garbling scheme as in Algorithms 2–7. Let* κ *be a computational security parameter. For every PPT adversary* $\mathcal{A}$ *with running time* $t(\kappa)$ *having access to all random oracles* $\mathsf{RO} \in (\mathsf{RO}^g, \mathsf{RO}')$*, participating in the Privacy game (Definition 2),* $\exists$ *negligible function* μ *s.t.* $\mathcal{A}$*'s advantage is,*

$$\mathit{Adv}(\kappa) = \left| \Pr[\mathcal{A}^{\mathsf{RO}}(\mathbf{C}_0, \mathbf{C}_1, x^0, x^1, F, X, d) = b] - \frac{1}{2} \right| < \mu(\kappa)$$

Proof: For a PPT adversary $\mathcal{A}$ running for $t(\kappa)$ time steps, $|Q| \le t(\kappa)$. Setting $\ell = \kappa$, we have from Theorem 4,

$$\mathsf{Adv}_{|Q|=t(\kappa)} < \frac{t(\kappa)}{2^\kappa - 2}$$

□