A New Approach to Garbled Circuits

Anasuya Acharaya[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] TU Eindhoven, Eindhoven, the Netherlands
t.ashur@tue.nl
[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. In this work we revisit the foundations of garbled circuits to propose a novel approach for garbling where the truth table of each gate is represented *as a whole* using a single encoding of *parameterized length* rather than being encrypted in a row-wise manner. We prove the scheme's security for the case where each gate encoding has the same length ℓ as the labels. This scheme improves over the state-of-the-art in both gate size compression—improving over the work of Rosulek and Roy (CRYPTO'21) which required 1.5ℓ bits to represent a garbled gate—and in the adversary model, yielding statistical security for arbitrary circuits, and against adversaries that are computationally unbounded and are allowed an exponential number of random oracle queries.

1 Introduction

The theory and practice of garbled circuits has been the focus of a long line of research, starting from the seminal work of [Yao86]. Garbled circuits have been widely used for constant round secure two-party computation protocol (2PC) [LP09]. Such a protocol proceeds by designating one party as a garbler and the other as the evaluator. The garbler generates an encoded version of the circuit of the function to be computed, using its private randomness. This is referred to as a *garbled circuit* (GC). The GC is then sent to the evaluator along with input labels corresponding to the garbler's input bits. Next, the garbler and evaluator engage in a constant round sub-protocol (*e.g.*, oblivious transfer (OT)) through which the evaluator receives the input labels corresponding only to its own input to the function. Finally, the evaluator uses all the input labels and the GC to compute the function output which it can share with the garbler.

Bellare *et al.* presented in [BHR12] an abstraction for garbling that captures [LP09] and all its subsequent optimizations. Subsequent works published after [BHR12] have followed the same line of thought as [LP09] and also describe themselves in terms of [BHR12].

1.1 Our Contributions

In this work we suspend the use of [BHR12]'s formalism and deviate from the line of work starting in [LP09]. Concretely, 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 random oracle model wherein both the garbler and the evaluator are given access to a common random oracle (RO). This approach provides a twofold benefit in simultaneously improving the communication complexity and guaranteeing security against a stronger adversary.

Communication Complexity. Our garbling approach requires fresh randomness only while sampling circuit input labels. Taking advantage of the random oracle, all other labels and the rest of the GC are derived deterministically from this. Such an approach effectively decouples the size of the circuit from the amount of randomness used and the level of security achieved. In our scheme, we represent each garbled gate using ℓ bits, where ℓ is also the length of each wire label. This is an improvement over the state of the art, cf. [RR21]. [RR21] represents each AND gate using 1.5κ bits where κ is a computational security parameter that can be set to 128. In our notation, ℓ is a parameter independent of our *statistical security parameter* ρ. The parameter[7] ℓ can be set to 128 as in [RR21]. We leave further optimization effort to future research.

Adversary Model. Although we require randomness only for sampling circuit input labels, the use of a random oracle in garbling breaks the view of the mapping between input and (random) output pairs. Whereas previous work used this randomness to encrypt a predetermined output label—adding exploitable redundancy at the gate level—we use it to derive the output labels directly. As a result, in contrast with previous work that assumes PPT adversaries, our scheme remains secure even against adversaries that are unbounded in computation and memory.

We define the privacy notion that our scheme satisfies: *input privacy.* This is a slight modification of the *privacy* in [BHR12] that is tailored to be a meaningful definition against the adversaries we consider. Informally, it requires that for a function and two inputs that give the same output, if the adversary is given the garbled circuit and active input labels for one of these inputs, it cannot distinguish between which input actually corresponds to the circuit and labels

[7] We bound the number of random oracle queries that the adversary can make by an exponent in ℓ. As such, the running time of the adversary (even though unbounded) can be represented in terms of ℓ.

it got. We prove input privacy against unbounded adversaries and our proof holds in the random oracle model where the adversary is allowed to make an exponential number of queries to the random oracle. Furthermore, our proof does not require the random oracle to be programmable.

1.2 Vision for Future Research

To the best of our knowledge, ours is the first garbling scheme of its kind for garbling arbitrary circuits against computationally unbounded adversaries in a way that is also communication efficient. This is to be considered in contrast with existing schemes that give information-theoretic garbling of formulas that grow exponentially with the formula-depth [Kol05]. As a consequence, our scheme does not fit into the standard [BHR12] formalism and security games for garbled circuits. We conjecture that pursuing a new formalism would require generalizing standard garbling. Noting that such a new formalism may pave the way to further optimizations we leave spelling out its definition for future work.

In particular, we conjecture that random oracles may not be necessary for the security of the scheme and that it remains secure also in the standard model. This effort of finding suitable primitives to replace a random oracle is also a direction that we leave for future work.

Furthermore, our scheme currently requires six primitive calls to the random oracles for garbling a gate where standard constructions based on [LP09] have been able to do so using only four primitive calls and only for AND gates. We leave it for future work to be able to obtain similar improvements.

Moreover, since we prove, *inter alia*, that the scheme is statistically secure, we conjecture that it remains so against quantum adversaries and can find applications in the post-quantum world. Formally studying its implications in these directions is left for future work.

1.3 Related Work

Secure garbling of circuits and corresponding ways of succinctly representing its 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 where for each gate in the circuit, each line in the truth table of the gate functionality is encrypted separately (this is also known as the 'gate-by-gate paradigm'). The underlying primitive for encryption is a symmetric-key algorithm (*e.g.*, a pseudorandom function (PRF), a correlation robust hash function) 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].

The state-of-the-art in garbled circuit compression is [RR21] where the size of each garbled gate is compressed to 1.5 ciphertexts; and [HK20] where the communication for the circuit as a whole is reduced to the size of the longest *branch* of computation. Both these works are in the computational setting and guarantee security against Probabilistic Polynomial Time (PPT) adversaries. In this work, we present a garbling algorithm in the random oracle model that garbles circuits in the gate-by-gate paradigm, but capturing the truth table as a whole rather than row-wise. Furthermore, our scheme provides statistical security against computationally and memory unbounded adversaries. The adversary we consider is allowed access to the random oracle, but is limited to a number of invocations that is exponential in a statistical security parameter.

The subject of garbling in the presence of computationally and memory unbounded adversaries has already been broached in the literature. [Kil88] presents a scheme for information-theoretic randomization of permutation branching programs (PBPs), in the plain model. However, this scheme is restricted in the classes of functions it can represent efficiently (NC^1 functions only). [Kol05] presents a scheme for information theoretic garbling of formulas in the plain model. However, since the size of their garbling grows exponentially with the depth of the function formula, the construction is only efficient for functions admitting low-depth formulas. In comparison, our scheme operates in the random oracle model but can size-efficiently garble circuits of arbitrary functions. In addition, in order to facilitate switching to the plain model in future work, our current construction does not require the random oracle to be programmable.

An extended line of works is the study of *randomized encodings* [IK00,Ish13]. 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. In particular, [IK02,CFIK03] explore information-theoretic randomized encodings. The best known information-theoretic decomposable randomized encoding for general functions corresponds to [Kol05] and has size exponential in the formula depth.

1.4 Technical Overview

Our garbling scheme operates in the random oracle model and both the garbler and evaluator get access to a random oracle RO. In order to provide security against unbounded adversaries, the garbling scheme deviates from traditional garbling in many aspects. We will discuss them below.

The Garbling Algorithm. 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) $\mathsf{Init}(C) \to e$; (2) $\mathsf{Circuit}(C, e) = (F, Y)$; (3) $\mathsf{DecodingInfo}(Y) \to d$.

Input Label Sampling. The first subroutine in the garbling algorithm takes the circuit $\mathbf{C}$ and creates the input encoding set e. In the garbling algorithm, this subroutine, $\mathsf{Init}(\cdot)$ uses the garbler's randomness. Although the circuit $\mathbf{C}$ is given as input, this algorithm only uses $n \in \mathbf{C}$, the number of input wires, allowing to generate the generate e ahead of knowing the function f. Note that similar to other traditional garbling schemes, this is also a projective garbling scheme. So e contains a set of input wire labels. In our construction, for the n input wires, and a length parameter ℓ, an ℓ-length label is sampled uniformly at random corresponding to the 0 and 1 bit for each wire. The only constraint applied is 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 Y. In order to extend the existing randomness in a way that does not introduce redundancy, $\mathsf{Circuit}(\cdot)$ makes black box calls to random oracles.

This subroutine garbles a circuit by garbling separately each gate in topological order. To this effect, for the q total gates in the circuit C, each gate is assigned an index i in this ordering. There are two random oracles employed throughout the gate-by-gate garbling process: RO_0 and RO_1. These differ in the size of their domains. Both these random oracles are tweakable: they take as an additional input the gate index i so that they behave independently for each gate.

Garbling a Gate. For a gate i, let A and B be its input wires, C be its output wire and $g(\cdot)$ be its functionality (*e.g.*, AND, XOR). When garbling within a gate, our methods deviate significantly from traditional garbling techniques. At its core, we make the following key observations: (1) each gate is a binary gate so there are 4 combinations of input values, but only two possible output values; (2) in order to garble against an unbounded adversary, one cannot afford to leak information regarding correctness of attempted decryptions.

Stemming from the first observation, in our construction, we make a distinction between *labels* and *keys*. In the garbling, we attribute *labels* as representing a wire value. That is, for a circuit wire A, L_0^A and L_1^A are its labels (similarly L_1^B, L_0^B for B, and L_1^C, L_0^C for C). However, contrary to traditional garbling, we do not use labels and keys interchangeably. We attribute *keys* to a gate, with each key being used to *derive* an output wire label (rather than hiding it as in previous work). Note that since there are only two output wire labels (L_0^C, L_1^C), we only need two keys (K_0^i, K_1^i), one to derive each label. Therefore, our construction employs a mechanism to convert the four possible combinations of input wire labels into two keys.

A key design technique for this conversion is inspired from the second observation stated above. We require that for the gate i, label combinations $\{(L_0^A, L_0^B), (L_0^A, L_1^B), (L_1^A, L_0^B), (L_1^A, L_1^B)\}$ be mapped to keys (K_0^i, K_1^i) in such a way that the gate functionality $g(\cdot)$ is preserved. For instance, if the gate is an AND gate, $\{(L_0^A, L_0^B), (L_0^A, L_1^B), (L_1^A, L_0^B)\}$ should be mapped to K_0^i, and (L_1^A, L_1^B) to K_1^i. Furthermore, this mapping is exactly what the garbled gate itself constitutes and therefore an encoding of it needs to be made public. In traditional garbling, each such input label pair (L^A, L^B) would be separately used to *encrypt* an L^C, yielding the four ciphertexts that make up the garbled gate. However, in doing so, additional information needs to be provided to the evaluator to indicate a successful decryption. Such an indicator would leak information to an unbounded adversary that is capable of brute-forcing the entire space, regardless of the type of primitive that was used for encryption (*e.g.*, PRF; or in the random oracle model).

The solution we employ encodes all four pairs into *one encoding* ∇. This is such that ∇ in its entirety, applied to the correct input label pair will give the correct key. This eliminates the need for the validity indicator altogether.

Finally, note that the entire gate garbling process needs to be a result of deterministic steps starting from the input label values. Therefore, for gate i, let L_1^A, L_0^A and L_1^B, L_0^B be the input labels. First, in order to eliminate redundancy, for $a, b \in \{0, 1\}$ each pair is input to a random oracle: $RO_0^i(L_a^A, L_b^B) \to X_{ab}^i$. RO_0, along with tweak i, takes as input two labels with total length 2ℓ and outputs an ℓ-length string X^i. Next, the intermediate values $(X_{00}^i, X_{01}^i, X_{10}^i, X_{11}^i)$ are encoded according to the gate functionality into a single ℓ-bit string ∇^i. This string has the properties that given ∇^i and any one X_{ab}^i it is mapped to an $K_{g(a,b)}^i$. Also, the pair ∇^i and X_{ab}^i do not reveal information sufficient to determine the other K^i or other values X^i. Next, for the values K_0^i and K_1^i so derived[8], since they no longer have full entropy, they are input to another random oracle to derive the output label: $RO_1^i(K_c^i) \to L_c^C$ for $c \in \{0, 1\}$. This random oracle has ℓ-bit strings as both its domain and range. The garbled gate itself can be represented using only ∇^i.

Decoding Information. Once all the garbled gates and output wire labels are derived in F, it remains to generate the output decoding information d. Here, again, if we employ a decoding mechanism that allows distinguishing between a correct and incorrect decoding, this reveals information that an unbounded adversary can take advantage of. Therefore, we need to be able to decode in such a way that for all output labels, valid or invalid, it yields some plausible decoding, but with the constraint that for valid output labels, the decoding is additionally also *correct*.

In our construction, we employ another random oracle RO_2 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 wire labels (L_0^j, L_1^j), it holds that $\mathsf{lsb}(RO_2(L_0^j, d^j)) = 0$ and $\mathsf{lsb}(RO_2(L_1^j, d^j)) = 1$. Note that such a decoding will always yield some output even for arbitrary output labels. The subroutine $\mathsf{DecodingInfo}(Y) \to d$ generates this decoding information given the output wire labels set.

Evaluating the Garbled Circuit. For completeness we outline the evaluation procedure. 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 the random oracles RO_0, RO_1, RO_2 and knows the indices of each gate in the circuit.

[8] Care is taken in the construction that both these values are unique.

Starting with the labels $L \in X$ we describe each label and key in its view during an honest evaluation as *active*. For each gate i, with active input labels L_a^A, L_b^B, the evaluator works by first accessing $RO_0^i(L_a^A, L_b^B) = X_{ab}^i$. Then using X_{ab}^i and $\nabla^i \in F$, it computes $K_{g(a,b)}^i$. Next it derives the gate output wire label $RO_1^i(K^i) = L^C$.

Finally, during decoding, for an output wire label L_b^j, using $d^j \in d$, it computes $\mathsf{lsb}(RO_2(L_b^j, d^j)) = b$ as the function output.

Security. We guarantee statistical security against a computationally and memory unbounded adversary that is allowed an exponentially bounded number of random oracle queries. Here, again, we deviate from the treatment of [BHR12] and define a different indistinguishability based security game. A formal definition and analysis is provided in Section 3.4. We prove that our scheme satisfies our privacy notion using two key arguments: (1) the adversary gains no distinguishing advantage when no random oracle queries are made; (2) we bound the advantage gained upon a single oracle query, and use this to argue that even when an exponential (in ℓ) number of queries are made, the advantage doesn't exceed $2^{-\rho}$, where ρ is the statistical security parameter.

2 Preliminaries

We provide here the definitions and notations we use throughout the paper.

2.1 Circuit Syntax

A Boolean circuit $\mathbf{C} : \{0,1\}^n \to \{0,1\}^m$ has n input wires enumerated by the indices $1, \ldots, n$, and m output wires enumerated by $n+q-m+1, \ldots, n+q$, where $q = |\mathbf{C}|$ is the number Boolean gates.[9] The output wire of gate i (also denoted by g^i) is $n+i$, which also implies that the gates satisfy a topological order, allowing to speak of $g^i > g^j$ when $i > j$. We say that a gate $i \in [q'] \subseteq [q]$ is *at the first level* if one or both of its input wires is also an input wire for the circuit.

On occasion, we abuse notation and use g as a synonym for the binary function described by this gate. Namely, $g^i(a,b)$ is the result of the binary function of gate g^i on the binary inputs a and b. For example, if g^i is an XOR gate then $g^i(a,b) = a \oplus b$. The interpretation would always be clear from the context.

2.2 Garbling Schemes

We briefly recall garbling schemes as described in [BHR12] along with the properties of interest.

[9] The convention is to comprise the circuit of XOR and AND gates with fan-in 2.

Definition 1 (Garbling scheme [BHR12]). *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function with a circuit representation $\mathbf{C}$. A garbling scheme* $(\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev})$ *consists of four polynomial-time algorithms defined as follows:*

- $(F, e, d) \leftarrow \mathsf{Gb}(1^\rho, \mathbf{C})$*: is a probabilistic algorithm that takes as input a circuit $\mathbf{C}$ with n input wires and m output wires; and returns a garbled circuit F, input encoding set e, and output decoding set d.*
- $X := \mathsf{En}(e, x)$ *is a deterministic algorithm that takes as input an input encoding set e and a plain value x; and returns the value's encoding X.*
- $Y := \mathsf{Ev}(F, X)$*: is a deterministic algorithm that takes as input a garbled circuit F and a set of input labels X; and returns the output labels Y where $|Y| = m$.*
- $\{\perp, y\} := \mathsf{De}(Y, d)$*: is a deterministic algorithm that takes as input the output decoding information d and output labels Y; and returns either the failure symbol $\perp$ or a plain value $y = f(x)$.*

These algorithms must satisfy the following properties:

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

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

- ***Privacy**: For all circuits $\mathbf{C}_0, \mathbf{C}_1$ such that the leakage $\phi(\mathbf{C}_0) = \phi(\mathbf{C}_1)$, and every x_0, x_1 such that $\mathbf{C}_0(x_0) = \mathbf{C}_1(x_1)$,*

$$\{F_0, X_0, d_0\}_{(F_0,e_0,d_0)\leftarrow\mathsf{Gb}(f_0), X_0=\mathsf{En}(e_0,x_0)} \overset{c}{\approx} \{F_1, X_1, d_1\}_{(F_1,e_1,d_1)\leftarrow\mathsf{Gb}(f_1), X_1=\mathsf{En}(e_1,x_1)}$$

2.3 Other Basic Notations

We denote by ρ a statistical security parameter. In our garbling scheme, for each gate indexed i, we denote by ∇^i the garbled gate information. There are three random oracles used in the scheme: RO_0, RO_1 and RO_2. Out of these, RO_0 and RO_1 are used to garble each gate, and they are tweakable: they take the gate index i as an additional input. RO_2 is used to generate decoding information.

For each wire W, its 0-label is denoted by L_0^W and its 1-label by L_1^W. For each gate i, the outputs of RO_0 are denoted as X_{ab}^i, for $a, b \in \{0,1\}$. The keys in the garbling are denoted as K_0^i and K_1^i. We denote by $\mathsf{lsb}(\cdot)$ a function that outputs the least significant bit of the string that is its input.

2.4 Random Oracles

The security proof of our scheme holds in the non-programmable 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. More Precisely,

Definition 2 (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 $\{0,1\}^b$.*

We model honest parties as PPT algorithms whereas the adversary is computationally and memory unbounded. All parties have limited access to the random oracle, namely, the number of queries is bounded. Note that the bound on the number of queries is not necessarily polynomial. Concretely, our scheme allows the adversary to make an exponential number of queries, but this number is not enough to fully determine the complete map of the function that was sampled in Definition 2.

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 unbounded adversary.

Lemma 1. *Let $RO : \{0,1\}^\ell \to \{0,1\}^k$ be a random oracle with fixed sized inputs of length ℓ. 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 $r \in \{0,1\}^k$,*

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

Proof: The random oracle RO works by seeing a query $q \in \{0,1\}^\ell$, and if q has not been queried before, it samples a fresh element $r \in \{0,1\}^k$ from the range, and then maps q to r as its response. This mapping is stored in a list of prior queries. If q had been queried before, the random oracle will find it on this list and return the response in its mapping. Letting Q be the list of all previous queries, over all random choices of r,

$$\Pr[RO(q) = r | q \notin Q] = \frac{1}{2^k}.$$

Note that for the case that a query q is not in the list of previously made queries, its response is freshly sampled and independently of all previous query responses. Therefore,

$$\Pr[RO(q) = r | \mathcal{V}(Q), q \notin Q] = \frac{1}{2^k}.$$

If, by contradiction, $\mathcal{V}(Q) \supset \{Q, R\}$, this would mean that more information about the random oracle is revealed. Then, there would exist queries $q \notin Q$ for which,

$$\Pr[RO(q) = r | \mathcal{V}(Q), q \notin Q] \neq \frac{1}{2^k}$$

This would contradict the fact that the response is sampled independently. Therefore, the lemma follows. □

2.5 The Adversary

We denote by $\mathcal{A}$ a computationally and memory unbounded adversary. This means that it can execute any algorithm that runs in finite time and keep the output in memory for a later use. This definition not only includes all PPT adversaries, but also adversaries that run in super-polynomial time and generate an arbitrary-sized outputs that they store in memory.

The definition above allows the adversary to reconstruct the full map of any known function with a finite domain by simply iterating all inputs (sometimes in exponential time), computing the function locally, and storing the result in memory. This applies to both deterministic and randomized functions where the difference between the two is that reconstructing the former involves iterating over the input domain whereas reconstructing the latter requires iterating over the input domain for all possible sequences of coin-tosses.

A random oracle however is a special case of a probabilistic function. Proofs in the random-oracle model can exploit the fact that $\mathcal{R}$ is chosen at random, and that the oracle interface allows to evaluate it, inevitably leaking some of the randomness. However the adversary is prevented from learning the full description of $\mathcal{R}$ since it is limited in the number of queries that it can make to the oracle. Definition 3 formalizes the resulting adversary in our context.

Definition 3 (Oracle Adversary). *An oracle adversary $\mathcal{A}^{\mathcal{O}}$ is a computationally unbounded algorithm in the random oracle model where the number of queries made to the oracle during an attack is bounded by $2^{f(\rho,\ell)}$ where ρ is the security parameter and ℓ is a parameter determining the labels length for circuit input wires.*

We continue on to describe our garbling scheme in Sections 3.1–3.2. In Section 3.4 we show that this scheme defined achieves input privacy against the adversary from Definition 3 for any $f(\rho,\ell) \leq \ell - \rho - 1$.

3 The Scheme

In this section we present our scheme and prove its security. The scheme consists of different algorithms for the garbler (Section 3.1) and for the evaluator (Section 3.2). In Section 3.3 we discuss the correctness of the scheme and give an intuition about its security. Then, in Section 3.4 we prove input privacy against an information theoretic adversary.

3.1 Garbler

Given a circuit $\mathbf{C}$ for the function, the garbler employs the following algorithms: (1) $\mathsf{Init}(\mathbf{C},\ell) \rightarrow e$; (2) $\mathsf{Circuit}(\mathbf{C},e) = (F,Y)$; (3) $\mathsf{DecodingInfo}(Y) \rightarrow d$.

Input Encoding Generation. The garbler starts by executing an algorithm: $\mathsf{Init}(\mathbf{C}, \ell) \rightarrow e$. It is formally described in Algorithm 1. Let n be the number of input wires in $\mathbf{C}$ and ℓ be a length parameter. This algorithm uses the garbler's randomness to sample ℓ-length labels to represent the 0 and 1 value 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. Looking ahead, this is the only step in the garbled circuit construction that involves any randomness.

Algorithm 1 Algorithm $\mathsf{Init}(\mathbf{C}, \ell)$

```
1: extract $n$ from $\mathbf{C}$
2: $e = []$
3: for input wire $W \in [n]$ do
4:     Sample $L_0^W \leftarrow \{0,1\}^\ell$ uniformly at random
5:     Sample $L_1^W \leftarrow \{0,1\}^\ell - \{L_0^W\}$ uniformly at random
6:     Set $e[W] = e_W = (L_0^W, L_1^W)$
7: end for
8: Return $e$
```

Garbled Circuit Generation. After creating the input encoding set e, the garbler runs a deterministic algorithm to generate the garbled circuit: $\mathsf{Circuit}(e, \mathbf{C}) = (F, Y)$. 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 Y, a set of labels (representing both the 0 and 1 value) for the m output wires of the garbled circuit. A description for it is given in Algorithm 2.

This algorithm works gate-by-gate where, for each of the q gates in the circuit, it creates a garbled gate by calling a subroutine as described in Algorithm 3. The garbled circuit so produced is,

$$F = \nabla_1, \ldots, \nabla_q.$$

Note that unlike previous work the intermediate labels are not predetermined and are instead derived from e by means of queries to the random oracle (details are discussed in the text ahead).

Gate Garbling. We discuss now the subroutine that the garbling algorithm uses to garble each gate of the circuit: $(\mathsf{L_0^C}, \mathsf{L_1^C}, \nabla^\mathsf{i}) \leftarrow \mathsf{Gate}(\mathsf{L_0^A}, \mathsf{L_1^A}, \mathsf{L_0^B}, \mathsf{L_1^B}, \mathsf{i}, \mathsf{type})$. This subroutine receives as input the gate index i, its input labels set $(L_0^A, L_1^A, L_0^B, L_1^B)$ and an indicator, *type*, that says whether this is an AND gate or an XOR gate. The subroutine outputs a gate garbling ∇^i and a set of labels for the gate output wire (L_0^C, L_1^C). The details of this subroutine are formally described in Algorithm 3.

Algorithm 2 Circuit$(e, \mathbf{C})$

```
1: initialize the wire label set $W = [W_1, \ldots, W_q]$ where each $W_i = []$
2: for each circuit input wire $A$ do    ▷ set circuit input wires as the input wires for appropriate gates
3:     for each gate indexed $i$ for which $A$ is an input wire do
4:         $W_i = W_i + (L_0^A, L_1^A)$
5:         $W[i] \leftarrow W_i$
6:     end for
7: end for
8: initialize $F = []$
9: initialize $Y = []$
10: for each gate $i$ in $\mathbf{C}$ in topological order do
11:     extract from $\mathbf{C}$, the input wire indices: $A, B$
12:     extract $W_i = (L_0^A, L_1^A, L_0^B, L_1^B) \in W$
13:     if $g = \wedge$ then
14:         $(\mathsf{L_0^C}, \mathsf{L_1^C}, \nabla^\mathsf{i}) \leftarrow \mathsf{Gate}(\mathsf{L_0^A}, \mathsf{L_1^A}, \mathsf{L_0^B}, \mathsf{L_1^B}, \mathsf{i}, \mathsf{AND})$
15:     else
16:         $(\mathsf{L_0^C}, \mathsf{L_1^C}, \nabla^\mathsf{i}) \leftarrow \mathsf{Gate}(\mathsf{L_0^A}, \mathsf{L_1^A}, \mathsf{L_0^B}, \mathsf{L_1^B}, \mathsf{i}, \mathsf{XOR})$
17:     end if
18:     set $F[i] \leftarrow \nabla^i$
19:     for each gate indexed $j$ for which $C$ is an input wire do
20:         $W_j = W_j + (L_0^C, L_1^C)$
21:         $W[j] \leftarrow W_j$
22:     end for
23:     if $i$ is an output gate then
24:         $Y[i] \leftarrow (L_0^C, L_1^C)$
25:     end if
26: end for
27: Return $(F, Y)$
```

This is a deterministic function but with access to two random oracles: RO_0 and RO_1. Both of these are tweakable random oracles that take the gate index i as an additional input. The oracle $RO_0^i(\cdot)$ takes 2ℓ-bit strings as inputs and outputs an ℓ-bit string uniformly at random for each input. The oracle $RO_1^i(\cdot)$ takes an ℓ-bit string and outputs a random string of the same length.

A gate is garbled in the following stages. First, given the set of input labels $(L_0^A, L_1^A, L_0^B, L_1^B)$, each of the combinations, $((L_0^A, L_0^B), (L_0^A, L_1^B), (L_1^A, L_0^B), (L_1^A, L_1^B))$ is a 2ℓ-bit string where at least ℓ bits are common with at least one other combination. To unlink the pairs, the input label combinations are passed into a random oracle RO_0. In order for the random oracle to sample fresh outputs for different gates which may have potentially the same input wires, RO_0 is initialized with a *tweak*, as $RO_0^i(\cdot)$. For bits $a, b \in \{0, 1\}$, this step creates $RO_0^i(L_a^A, L_b^B) \rightarrow X_{ab}^i$. The values $(X_{00}^i, X_{01}^i, X_{10}^i, X_{11}^i)$ are intermediate garbling values, each ℓ-bit long, that are the outputs of the random oracle.

Algorithm 3 $\mathsf{Gate}((L_0^A, L_1^A), (L_0^B, L_1^B), i, type)$

```
1: $X_{00}^i = RO_0^i(L_0^A, L_0^B)$
2: $X_{01}^i = RO_0^i(L_0^A, L_1^B)$
3: $X_{10}^i = RO_0^i(L_1^A, L_0^B)$
4: $X_{11}^i = RO_0^i(L_1^A, L_1^B)$
5: initialize $\nabla^i \leftarrow 0^\ell$
6: if $type = AND$ then
7:     for index $j \in [\ell]$ do
8:         Slice $\leftarrow X_{00}^i[j]||X_{01}^i[j]||X_{10}^i[j]||X_{11}^i[j]$
9:         if Slice $\in \{0000, 0001, 1110, 1111\}$ then          ▷ See Table 1
10:            $\nabla^i[j] \leftarrow 1$
11:        end if
12:    end for
13:    $K_0^i = X_{00}^i \& \nabla^i$
14:    $K_1^i = X_{11}^i \& \nabla^i$
15: else if $type = XOR$ then
16:    for index $j \in [\ell]$ do
17:        Slice $\leftarrow X_{00}^i[j]||X_{01}^i[j]||X_{10}^i[j]||X_{11}^i[j]$
18:        if Slice $\in \{0000, 0110, 1001, 1111\}$ then          ▷ See Table 2
19:            $\nabla^i[j] \leftarrow 1$
20:        end if
21:    end for
22:    $K_0^i = X_{00}^i \& \nabla^i$
23:    $K_1^i = X_{01}^i \& \nabla^i$
24: end if
25: $L_0^C \leftarrow RO_1^i(K_0^i)$
26: $L_1^C \leftarrow RO_1^i(K_1^i)$
27: Return $(L_0^C, L_1^C, \nabla^i)$
```

Next, the set $(X_{00}^i, X_{01}^i, X_{10}^i, X_{11}^i)$ is used to create a gate garbling ∇^i of size ℓ bits. This lies in the heart of our construction and is one of our key contributions. The details on how exactly ∇^i is created are given in Tables 1–2 and depends on the gate type. This garbling ∇^i is such that for any intermediate value X_{ab}^i, a *key* can be derived as $\nabla^i \& X_{ab}^i = K_{g(a,b)}^i$. This key is used for garbling and $g(\cdot, \cdot)$ here represents the gate functionality: AND or XOR. An essential property that ∇^i satisfies is that on its application with any of the X_{ab}^i, it produces one of two distinct values K_0^i and K_1^i and that too according to the gate functionality.

Finally, note that the ℓ-length keys K_0^i and K_1^i derived in the previous stage need not preserve full entropy. Therefore, these are passed as input to another random oracle RO_1 with tweak i that gives ℓ-bit strings as outputs. The output of this random oracle, (L_0^C, L_1^C) are designated as the labels of the output wire C of this gate. These output labels, along with the gate garbling ∇^i are the outputs of this subroutine.

Decoding Information. The last of the Garbler's algorithms is a randomized algorithm: $\mathsf{DecodingInfo}(Y) \to d$. It takes the labels set for the output wires, Y,

Table 1. For a gate index i and $j \in [\ell]$, this table defines $\nabla^i_\wedge[j]$ as a function of $X^i_{00}[j], X^i_{01}[j], X^i_{10}[j], X^i_{11}[j]$. In addition, the right side demonstrates how combining $X^i_{ab}[j] \& \nabla^i[j]$ collapses into only two distinct key values $K^i_0 = K_{00} = K_{01} = K_{10}$ and $K^i_1 = K_{11}$. Each row in the table corresponds to one bit-slice of the values $X^i_{ab}[j]$ for $a, b \in \{0,1\}$.

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

and returns a sequence d that would allow the evaluator to map them back from the encrypted domain to the plain domain; see Algorithm 4. Here we denote by $\mathsf{lsb}(\cdot)$ the least significant bit of a string. This function also employs a random oracle $RO_2(\cdot)$ that takes a 2ℓ-bit input. We do not fix its output domain, though it could be of any length greater than 1.

Algorithm 4 $\mathsf{DecodingInfo}(Y)$

```
1: initialize $d = []$
2: **for** output wire $j \in [n+q-m+1, n+q]$ **do**
3:     $L^j_0 = Y[j][0]$
4:     $L^j_1 = Y[j][1]$
5:     **repeat**
6:         Sample $d^j \in_R \{0,1\}^\ell$
7:     **until** $\mathsf{lsb}(RO_2(L^j_0, d^j)) = 0$ and $\mathsf{lsb}(RO_2(L^j_1, d^j)) = 1$
8:     $d \leftarrow d^j$
9: **end for**
10: **Return** d
```

3.2 Evaluator

Evaluating a garbled circuit that was created as in Algorithms 1-4 is straightforward. For completeness, we describe the three algorithms that the evaluator employs for this purpose: Encode, $\mathsf{Evaluate}$, and Decode. The interfaces and

Table 2. For a gate index i and $j \in [\ell]$, this table defines $\nabla^i_{\oplus}[j]$ as a function in $X^i_{00}[j], X^i_{01}[j], X^i_{10}[j], X^i_{11}[j]$. In addition, the right side demonstrates how combining $X^i_{ab}[j] \& \nabla^i[j]$ collapses into only two distinct key values $K^i_0 = K_{00} = K_{11}$ and $K^i_1 = K_{01} = K_{10}$. Each row in the table corresponds to one bit-slice of the values $X^i_{ab}[j]$ for $a, b \in \{0, 1\}$.

	X^i_{00}	X^i_{01}	X^i_{10}	X^i_{11}	$\nabla_{\oplus}$	K_{00}	K_{01}	K_{10}	K_{11}
0	0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0
5	0	1	0	1	0	0	0	0	0
6	0	1	1	0	1	0	1	1	0
7	0	1	1	1	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0
9	1	0	0	1	1	1	0	0	1
10	1	0	1	0	0	0	0	0	0
11	1	0	1	1	0	0	0	0	0
12	1	1	0	0	0	0	0	0	0
13	1	1	0	1	0	0	0	0	0
14	1	1	1	0	0	0	0	0	0
15	1	1	1	1	1	1	1	1	1

purpose of these are respectively the same as En, Ev, and De in standard garbling [BHR12]. For brevity we only describe them in algorithmic form in Algorithms 5–7. Note that the evaluator also has access to the same random oracles RO_0, RO_1, and RO_2 as in the garbling algorithms and at the time of evaluation, their input to output mapping for the queries made by the garbler are fixed. This is what yields correct evaluation.

Algorithm 5 Algorithm $\mathsf{En}(e, x)$

```
1: initialize $X = []$
2: **for** every $j \in [n]$ **do**
3:     extract $L^j_{x_j} = e_j[x_j]$
4:     set $X[j] = L^j_{x_j}$
5: **end for**
6: **Return** $X$
```

3.3 Properties

Correctness. The correctness of the scheme follows from Algorithms 2–4 and the definition of $\nabla_{\wedge}$ and $\nabla_{\oplus}$ in Tables 1–2.

In Algorithm 2 the garbler iterates over the gates. For each gate i, it calls Algorithm 3 with two pairs of labels corresponding the the gate's input labels, and the gate's type. Algorithm 3 calls the random oracle with pairs of input

Algorithm 6 Algorithm $\mathsf{Ev}(F, X)$

```
1: initialize $Y = []$
2: for each gate $i \in [q]$ in a topological order do
3:     $L^A, L^B \leftarrow$ active labels associated with the input wires of gate $i$
4:     extract $\nabla^i \leftarrow F[i]$
5:     $L^C \leftarrow RO_1^i(RO_0^i(L^A, L^B)\&\nabla^i)$
6:     if $i$ is a circuit output wire then
7:         $Y[i] \leftarrow L^C$
8:     end if
9: end for
10: Return $Y$
```

Algorithm 7 Algorithm $\mathsf{De}(Y, d)$

```
1: initialize $y = []$
2: for $j \in [n + q - m + 1, n + q]$ do
3:     $y[j - (n + q - m + 1)] \leftarrow \mathsf{lsb}(RO_2(Y[j], d^j))$
4: end for
5: Return $y$
```

labels such that

$$\begin{aligned} RO_0^i(L_0^A, L_0^B) &\mapsto X_{00}^i; \\ RO_0^i(L_0^A, L_1^B) &\mapsto X_{01}^i; \\ RO_0^i(L_1^A, L_0^B) &\mapsto X_{10}^i; \\ RO_0^i(L_1^A, L_1^B) &\mapsto X_{11}^i. \end{aligned} \tag{1}$$

Then, it uses either Table 1 or Table 2 to form a ∇^i capturing the bitwise difference between the X_{ab}^i values, such that

$$\begin{aligned} X_{00}^i\&\nabla^i &\mapsto K_{00}^i; \\ X_{01}^i\&\nabla^i &\mapsto K_{01}^i; \\ X_{10}^i\&\nabla^i &\mapsto K_{10}^i; \\ X_{11}^i\&\nabla^i &\mapsto K_{11}^i, \end{aligned} \tag{2}$$

preserves the plain gate's logic. Consequently,

$$|\{K_{00}^i, K_{01}^i, K_{10}^i, K_{11}^i\}| = 2.$$

Composing (1)–(2) we obtain for an AND gate

$$\begin{aligned} RO_0^i(L_0^A, L_0^B)\&\nabla^i &\mapsto K_0^i; \\ RO_0^i(L_0^A, L_1^B)\&\nabla^i &\mapsto K_0^i; \\ RO_0^i(L_1^A, L_0^B)\&\nabla^i &\mapsto K_0^i; \\ RO_0^i(L_1^A, L_1^B)\&\nabla^i &\mapsto K_1^i, \end{aligned}$$

and for an XOR gate

$$\begin{aligned}
RO_0^i(L_0^A, L_0^B)\&\nabla^i &\mapsto K_0^i;\\
RO_0^i(L_0^A, L_1^B)\&\nabla^i &\mapsto K_1^i;\\
RO_0^i(L_1^A, L_0^B)\&\nabla^i &\mapsto K_1^i;\\
RO_0^i(L_1^A, L_1^B)\&\nabla^i &\mapsto K_0^i.
\end{aligned}$$

Correctness can be verified in a bit-wise manner.

Complexity. Compared to a classical Yao's garbled circuit [Yao86], which implies four calls to the cryptographic primitive for each gate, our scheme requires six primitive calls per gate in Algorithm 3. On the other hand, the amount of randomness required by our scheme is linear in the the label length ℓ and the number of circuit input wires n whereas all follow ups from [LP09] require randomness that is linear in ℓ and the overall number of wires $q + n$.

As an interesting consequence of using lesser randomness and deriving subsequent values deterministically is that the garbled circuit has fewer redundancies, enabling in Section 3.4 a privacy proof against a computationally and memory unbounded adversary, *i.e.*, *statistical security*, that is stronger than the standard computational security. Note that since that statistical distance is measured only in terms of e irrespective of F and d, $|\nabla^i|$ and therefore $|F|$ can be made arbitrarily small without affecting security. For brevity we set $|\nabla| = \ell$ for our proof in Section 3.4 (thus improving over the state of the art, cf. [RR21]) and leave further investigation of how small F and d can be made to future research.

3.4 Privacy

Adversary. For our proof, we consider an adversary $\mathcal{A}$ that is computationally unbounded, but can only make an (exponentially) bounded number of queries to the random oracles (RO). Letting ρ be a statistical security parameter, and ℓ be the length of each wire label, the number of queries that $\mathcal{A}$ is allowed to make to the RO is bounded by an exponent in ρ and ℓ. Our main claim, which we formalize later in Theorem 1 is stated informally in Proposition 1.

Proposition 1. *The garbling scheme described in Algorithms 1–4 achieves input privacy (Algorithm 8) against any oracle adversary $\mathcal{A}$ with $f(\rho, \ell) = \ell - \rho - 1$ (Definition 3).*

When an adversary is given a garbling (F, d) and input labels X, we make a separation between the notion of honest and adversarial queries. We say that an RO query is *honest* when it is necessary for evaluating F on inputs X. Let H be this set of honest queries that the adversary is allowed. For a garbled circuit with q gates and m output wires, $|H| = 2q + m$. We term any additional queries made to the RO as *adversarial*. Let Q be the set of adversarial queries that $\mathcal{A}$ makes to the RO.

Definition 4. *A set of adversarial queries Q that an adversary $\mathcal{A}$ makes to the random oracle is* ***permissible*** *if it holds that for a statistical security parameter ρ, and label length ℓ,*

$$|Q| < 2^{\ell-\rho-1}$$

Security Game. We re-define privacy to a game better suited in the presence of the adversary described above. However, before looking at the game itself, it would be instructive to first understand some relevant terminology:

For a function f, let its input domain be $\{0,1\}^n$. We denote by $x^0, x^1 \in \{0,1\}^n$ two valid inputs to f, of the choice of $\mathcal{A}$, under the constraint that $f(x^0) = f(x^1)$. We denote by F a garbled circuit for f; by d its decoding information; and by X the active input labels for F. Let e be the input encoding set for F. Let us denote by $\mathsf{Circuit}(f, e)$ the garbling subroutine that given an input encoding set e derives the complete garbling (F, d) for a function f (*i.e.*, Algorithm 2 and Algorithm 4). For $b \in \{0,1\}$, we denote by $\varepsilon(X, x^b)$ the set of all e such that $X = En(e, x^b)$.

The security game takes place between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ and breaks the usual order in which garbling is done. First, $\mathcal{A}$ samples (f, x^0, x^1) and gives it to $\mathcal{C}$. Then, $\mathcal{C}$ samples X from a uniform distribution. Next, it samples a bit $b \in \{0,1\}$ uniformly at random and samples the input encoding set e^b uniformly from $\varepsilon(X, x^b)$. This is used to derive the garbling F, d. Finally, $\mathcal{C}$ sends the challenge (F, X, d) to the adversary $\mathcal{A}$ and the latter is tasked with guessing the bit b used internally by $\mathcal{C}$. Algorithm 8 describes formally the actions of the challenger $\mathcal{C}$.

Algorithm 8 Input Privacy

```
1: proc $Garble(f, x^0, x^1)$
2:     if $x^0, x^1 \notin \{0,1\}^n$ return $\bot$
3:     if $f(x^0) \neq f(x^1)$ return $\bot$
4:     $X \leftarrow \{0,1\}^{n\ell}$
5:     $b \leftarrow \{0,1\}$
6:     $e^b \leftarrow \varepsilon(X, x^b)$
7:     $(F, d) \leftarrow \mathsf{Circuit}(f, e^b)$
8:     Return $(F, X, d)$
```

Distinguishing Advantage. In the privacy game, $\mathcal{A}$ has in its view (f, x^1, x^0) of its own choice, (F, X, d) that it receives as the challenge, the set of honest queries H and Q, the set of adversarial RO query responses that it receives. 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[10].

[10] We omit writing (f, x^0, x^1) for brevity but it is assumed to be always included.

Definition 5. *For any unbounded adversary $\mathcal{A}$, for any (f, x^0, x^1) of its choice, (F, X, d) as output from Algorithm 8, honest evaluation RO queries H, and a permissible set of adversarial queries Q (Definition 4), we define the adversary's* ***advantage*** *to be,*

$$\mathsf{Adv} = \left| \Pr[\mathcal{A}(\mathcal{V}(F, X, d, H, Q)) = 0] - \Pr[\mathcal{A}(\mathcal{V}(F, X, d, H, Q)) = 1] \right|$$

The probability distribution is taken over the secrets of the challenger $\mathcal{C}$ (random choices of $X \leftarrow \{0,1\}^{n\ell}$, $b \leftarrow \{0,1\}$, and $e^b \leftarrow \varepsilon(X, x^b)$), and the choice of the adversarial query set Q. The values (F, d) are determined by e^b and the contents of the honest query set H are fixed for a choice of (F, X, d).

The adversary's task is to distinguish between the case that the challenger chooses $b = 0$ and $b = 1$ given $\mathcal{V}(F, X, d, H, Q)$. This boils down to the probability that e^b comes from $\varepsilon(X, x^0)$ or from $\varepsilon(X, x^1)$. Therefore, the advantage as in Definition 5 can be rewritten as,

$$\mathsf{Adv} = \left| \Pr[e^b \in \varepsilon(X, x^0) | \mathcal{V}(F, X, d, H, Q)] - \Pr[e^b \in \varepsilon(X, x^1) | \mathcal{V}(F, X, d, H, Q)] \right| \tag{3}$$

Proof Outline. The aim is to show that for an execution of Algorithm 8, over all possible *permissible* Q, the adversary $\mathcal{A}$'s advantage is bounded by $\mathsf{Adv} < 2^{-\rho}$. We open by first arguing that $\mathcal{A}$'s a-priori advantage when it chooses (f, x^0, x^1) is 0.
Next, we show that since X is chosen by the challenger independent of b, $\mathcal{A}$'s advantage on knowing (f, x^0, x^1, X) is also 0. We do this by first showing in Lemma 2 that for all (X, x), the set $\varepsilon(X, x)$ has the same size. Next, through Lemma 3 we show that for all X, the sets $\varepsilon(X, x^0)$ and $\varepsilon(X, x^1)$ are disjoint. Note that for x^0 and x^1, a computationally unbounded $\mathcal{A}$ can reconstruct the sets $\varepsilon(X, x^0)$ and $\varepsilon(X, x^1)$. We use the lemmas to argue that for every input encoding set $e^0 \in \varepsilon(X, x^0)$, there exists an $e^1 \in \varepsilon(X, x^1)$. Therefore knowing (f, x^0, x^1, X) yields no advantage.
Now we consider the case that the adversary sees the complete challenge (F, X, d) but doesn't make any random oracle queries. We show through Lemma 4 that $\mathcal{A}$'s advantage given $\mathcal{V}(F, X, d)$ is 0. This is because there is no way to link (F, d) to an input encoding set e without making random oracle queries.
We then consider the case where RO queries are made. Note that certain query and response mappings in the RO have already been determined when the challenger $\mathcal{C}$ creates the garbling in Algorithm 8. In fact, to avoid confusion, we consider the random oracle to be fixed from this point onward.

For smooth treatment of this case, we first prove the following relevant results.

First, looking at the RO in isolation, we recall in Lemma 1 that when an RO is queried, it's response reveals no additional information about values that are not yet queried. Therefore, limiting the size of Q to that of *permissible* (Definition 8) sets, no further information about the random oracle can be learnt. Also, all the queries in H reveal no more information than the *active path* in the circuit F.

Next, in the context of garbling, Lemma 5 bounds the probbility that any input encoding set e, that is not the e^b used by the challenger $\mathcal{C}$, would be compatible with (F, d) (Definition 12) for a random choice of the RO. This helps us determine the a-priori statistical distance between an $e \in \varepsilon(X, x^b)$ being compatible with (F, d) and $e \in \varepsilon(X, x^{1-b})$ being compatible.

Now we consider the case where all honest queries in H are made but $Q = \phi$. For both cases $b = 0$ and $b = 1$ the active labels X are the same. Starting from X, all RO queries would reveal the *active path* in both cases. Also, since $f(x^0) = f(x^1)$, decoding gives the same result. So we argue that $\mathcal{A}$'s advantage given (F, X, d, H) is also 0.

Finally, we get down to the case where $\mathcal{A}$ makes dishonest queries: a query that is not an honest query. It is here that we use the proof of Lemma 5 to bound the information revealed about b, when a single query is made. This is extended to the case of multiple queries using Lemma 1 to conclude that for any permissible Q, $\mathcal{A}$'s advantage cannot exceed $2^{-\rho}$. This concludes the proof.

We next define certain terms and notations that are used throughout the proof before laying out the proof itself.

3.4.1 Preliminaries In this section, we define certain terms that we use within our proof. In particular, these definitions also enumerate the possible checks that an adversary can perform as part of its strategy. Note that the adversary's goal given (F, X, d) is to distinguish whether (F, d) were created from an $e^0 \leftarrow \varepsilon(X, x^0)$ or from an $e^1 \leftarrow \varepsilon(X, x^1)$.

We begin by defining **input encoding compatibility** for projective garbling schemes.

Definition 6 **((x, X, e)-compatibility).** *For a projective garbling scheme, we say that a tuple $(x, X, e) \in \{0,1\}^n \times \{0,1\}^{n\ell} \times \{0,1\}^{2n\ell}$ is encoding compatible if and only if $X = En(e, x)$.*

Our garbling scheme is indeed a projective garbling scheme. Typically, during garbling, the input encoding set e and the function input x are first fixed before $En(e, x) = X$ gives the active input labels. However, in Algorithm 8, the challenger $\mathcal{C}$ first samples X and then e^b is sampled from $\varepsilon(X, x^b)$, that is the set of all e such that (x^b, X, e) are compatible (Definition 6). Looking ahead, our proof will show that given (X, f, x^0, x^1), when no RO queries are made, for every encoding compatible triplet (x^0, X, e^0) there exists an equally likely compatible triplet (x^1, X, e^1). As a consequence, for a secret e^b, it becomes impossible to say from the given information $\mathcal{V}(F, X, d)$ whether it is e^1 or e^0.

Our next definition is about **random oracle (RO) compatibility** for our construction for a single garbled gate.

Definition 7 **((W^i, t^i)-compatibility).** *For a gate indexed i, let the tuple of labels $W^i = ((L_0^A, L_1^A), (L_0^B, L_1^B)) \in \{0,1\}^{2\ell} \times \{0,1\}^{2\ell}$ be its input labels. Then for a tuple of intermediate values $t^i = (X_{00}^i, X_{01}^i, X_{10}^i, X_{11}^i) \in \{0,1\}^{4\ell}$ we say that W^i and t^i are RO-compatible for a fixed random oracle RO_0 if and only if* *it holds that,*

$$\begin{aligned} X^i_{00} &= RO^i_0(L^A_0, L^B_0) \\ X^i_{01} &= RO^i_0(L^A_0, L^B_1) \\ X^i_{10} &= RO^i_0(L^A_1, L^B_0) \\ X^i_{11} &= RO^i_0(L^A_1, L^B_1) \end{aligned}$$

Definition 7 points to the necessity of making RO calls in order to check if certain wire labels were indeed used to create the garbled gate. In order to extend Definition 7 from the case of a single gate, to that of all the gates of a complete garbled circuit, it is first necessary to define what would be a set of **compatible internal wires**.

Definition 8 **((e, W)-compatibility).** *For a garbled circuit, let $e \in \{0,1\}^{2n\ell}$ be the set of labels for the input wires. Letting q be the number of gates, let $W \in \{0,1\}^{4q\ell}$ be the set of labels for wires that are input to all the gates of the circuit where $W = [W^1, \ldots, W^q]$ and each $W^i \in \{0,1\}^{2\ell} \times \{0,1\}^{2\ell}$ is the set of input labels to the gate indexed i. We say that e and W are compatible if for a fixed choice of random oracles RO_0 and RO_1 for each gate, it holds that W is derived from running* $\mathsf{Circuit}(f, e)$ *as in Algorithm 8. For this set of random oracles (RO_0, RO_1), we say that e* ***extends*** *to W.*

We now use this to define **circuit RO compatibility** for all the gates in a circuit.

Definition 9 **((e, t)-compatibility).** *For a garbled circuit, let $e \in \{0,1\}^{2n\ell}$ be the set of labels for the input wires. Letting q be the number of gates, for a sequence of tuples $t = [t^1, \ldots, t^q]$ we say that e and t are compatible if and only if there exists a set of wire labels $W = [W^1, \ldots, W^q]$ such that e and W are wire compatible as in Definition 8, and for every gate $i \in [q]$ the pair (W^i, t^i) is RO compatible according to Definition 7.*

Note that for any set (e, t) with the correct dimensions, there exists some random oracles RO_0 and RO_1 that make these compatible as in Definition 9. However, when running $\mathsf{Circuit}(\cdot)$ (Algorithm 8) the random oracle has been fixed to exactly one instance by the garbler. This post-hoc nature is an important aspect of Definitions 7–9. Our proof will show that the adversary cannot exploit the fact that the random oracle has been fixed, and that information can only be learnt through querying it as if it were not.

The next definition is about **gate logic compatibility**.

Definition 10 **((t^i, ∇^i)-compatibility).** *For a gate indexed i, a tuple of intermediate values $t^i = (X^i_{00}, X^i_{01}, X^i_{10}, X^i_{11}) \in \{0,1\}^{4\ell}$ and a string $\nabla^i \in \{0,1\}^\ell$ are said to be $\wedge$-compatible (resp., $\oplus$-compatible) if and only if for every bit position $j \in [\ell]$, the string $(X^i_{00}[j], X^i_{01}[j], X^i_{10}[j], X^i_{11}[j], \nabla^i[j]) \in \{0,1\}^5$ is consistent with a row in (the first 5 columns of) Table 1 (resp., Table 2).*

Recall that the random oracles are fixed by the challenger during garbling. We show in our proofs that from the adversary's point of view, while examining a particular gate, the only way for it to create a compatible (W^i, t^i) pair (Definition 7) is by querying the RO. Then, the only way for it to verify if this candidate for t^i is valid, is to check for (t^i, ∇^i) compatibility (Definition 10). Therefore, after the challenger fixes a garbled circuit F, for an adversary to narrow down an e from which it was created, the above checks become necessary.

Definition 10 can be extended from a single gate case to a definition for **logic compatibility**.

Definition 11 (**(t, F)-compatibility**). *Letting q be the number of gates in the garbled circuit F, for a sequence of tuples $t = [t^1, \ldots, t^q]$, and the sequence of garbled gates $F = [\nabla^1, \ldots, \nabla^q]$, we say that (t, F) are logically compatible if and only if for every gate $i \in [q]$ it holds that (t^i, ∇^i) are compatible according to Definition 10.*

Finally, Definitions 8-11 can be compiled to the definition below that characterizes a **compatible garbling** and allows for correct decoding of the function output.

Definition 12 (**(e, F, d)-compatibility**). *For a garbled circuit F, let e be a sequence of input wire labels and q be the number of gates. Let $F = [\nabla^1, \ldots, \nabla^q]$ be the sequence of garbled gates and let $d \in \{0,1\}^{m\ell}$ be a sequence of output decoding labels. We say that (e, F, d) are compatible if and only if on fixing random oracles (RO_0, RO_1, RO_2) there exists a set of tuples $t = [t^1, \ldots, t^q]$ for which (e, t) are compatible (Definition 9) and (t, F) are logically compatible (Definition 11). The output decoding labels in $d = [d^1, \ldots, d^m]$ are of the form $d^j \in \{0,1\}^\ell$. It additionally holds that for every output bit index j, and output wire labels (L_0^j, L_1^j),*

$$\mathsf{lsb}(RO_2(L_0^j, d_j)) = 0 \text{ and } \mathsf{lsb}(RO_2(L_1^j, d_j)) = 1$$

The end goal of the adversary in the privacy game is to distinguish for a given (F, d), whether the e^b compatible (Definition 12) with it comes from the set $\varepsilon(X, x^0)$ or $\varepsilon(X, x^1)$. We now proceed towards detailing the proof.

3.4.2 Proof We open with the observation that the only values depending on b are e^b, F, and d. Since (F, d) are computed deterministically from e^b, it is enough to consider how b affects the distribution of e^b as a random variable. Definition 5 and Equation 3 formalize this.

Adv with (f, x^0, x^1). We first look at the a-priori advantage of the adversary $\mathcal{A}$ before the execution of Algorithm 8. Before analyzing the concrete advantage, we show through Lemma 2 that for any $(X, x^0, x^1) \in \{0,1\}^{n\ell} \times \{0,1\}^n \times \{0,1\}^n$, $|\varepsilon(X, x^0)| = |\varepsilon(X, x^1)|$.

Lemma 2. *For any (X, x), $|\varepsilon(X, x)| = (2^\ell - 1)^n$.*

Proof: Consider $e \in \varepsilon(X, x) \in \{0,1\}^{2n\ell}$. In projective schemes we can decompose e into its components

$$e = e^1, \dots, e^n$$

such that e^j is associated with an input wire $j \in [n]$ and this can be further decomposed into $e^j = [L_0^j, L_1^j]$ where L_0^j is understood to be the label encoding the 0-semantics of wire j and L_1^j the wire's 1-semantics. For fixed X and x, this fixes n labels and their positions in e, one for each e^j. Therefore, there remain n labels to choose. Each label $L_{1-x_j}^j \in e^j$ can be a string in $\{0,1\}^\ell$. However, it cannot be equal to the label $L_{x_j}^j \in X$. This gives $2^\ell - 1$ possibilities.

On considering the full set of n wires, this gives $(2^\ell - 1)^n$ possibilities. □

Therefore, for any choice of x^0 and x^1 of the adversary, $\mathcal{A}$ cannot bias the distribution from which e^b is selected.

Adv given (f, x^0, x^1, X). Now, consider the case that $\mathcal{A}$ is additionally given X. Note that given this information, an unbounded adversary can enumerate over all the elements of the set $\varepsilon(X, x^0)$ and $\varepsilon(X, x^1)$. From Lemma 2 it follows that $\varepsilon(X, x^0)$ and $\varepsilon(X, x^1)$ are of the same size. We show in Lemma 3 below that for any X, both these sets are disjoint.

Lemma 3. *For any X and $x^1 \neq x^0$, $\varepsilon(X, x^1) \cap \varepsilon(X, x^0) = \phi$.*

Proof: Consider for the sake of contradiction that this is not the case and $\varepsilon(X, x^1) \cap \varepsilon(X, x^0) \neq \phi$. Then there exists some input wire labels set e such that $e \in \varepsilon(X, x^1)$ and $e \in \varepsilon(X, x^0)$. Note that $x^1, x^0 \in \{0,1\}^m$ and $x^1 \neq x^0$. So there exists an index $i \in [m]$ such that $x^1[i] \neq x^0[i]$. Without loss of generality let $x^1[i] = 1$. For the set of active labels X, let $L^i \in X$ be the label in the i^{th} position. Then, in e, for the i^{th} input wire, both the 0-label (from x^0) and the 1-label (from x^1) would be L^i. This yields a contradiction since both labels for the same wire cannot have the same value. □

From Lemma 2 and Lemma 3 it follows that for a fixed X, x^0, x^1, and every e^0 compatible with (X, x^0), there exists an e^1 compatible with (X, x^1).

$$\Pr[e^b \in \varepsilon(X, x^0) | X, x^0, x^1, f] = \Pr[e^b \in \varepsilon(X, x^1) | X, x^0, x^1, f] = \frac{1}{2} \tag{4}$$

This holds since $b \in \{0,1\}$ is uniformly distributed, and $\varepsilon(X, x^0)$ and $\varepsilon(X, x^1)$ are disjoint and of equal size.

Note that (x^b, X, e^b) are by construction encoding compatible (Definition 6). It is also independent of the function f. Therefore, rewriting Equation (4) in terms of Definition 5, we have that,

$$\mathsf{Adv} = \left| \Pr[e^b \in \varepsilon(X, x^0) | X, x^0, x^1, f] - \Pr[e^b \in \varepsilon(X, x^1) | X, x^0, x^1, f] \right| = 0$$

Adv *given* (f, x^0, x^1, F, X, d). Now consider the situation where the adversary $\mathcal{A}$ is given the full challenge set (F, X, d) but is yet to make any queries to the random oracle. The following lemma shows that when $Q \cup H = \emptyset$ the advantage is 0.

Lemma 4. *Given Garble*$(f, x^0, x^1) \rightarrow (F, X, d)$ *as in Algorithm 8, it holds that*

$$\left| \Pr[e^b \in \varepsilon(X, x^0) | \mathcal{V}(F, X, d)] - \Pr[e^b \in \varepsilon(X, x^1) | \mathcal{V}(F, X, d)] \right| = 0$$

Proof: For each gate, recall that by Definition 10 the pair of W^i and t^i values are separated by a random oracle RO_0. By the definition of the random oracle, the only way to determine if W^i and t^i are compatible is by querying it. Since no queries were made by assumption, the adversary has no way to identify the event (e^b, F, d) *are compatible* (Definition 12) for any choice of e^b. The lemma follows since without querying the random oracle (F, d) and e are independent and we already had that the a-priori advantage is 0. □

Fixing the Random Oracles. Note that when the challenger $\mathcal{C}$ was garbling using $\mathsf{Circuit}(\cdot)$ in Algorithm 8, the random oracles (RO_0, RO_1, RO_2) were fixed. Let Q be the set of adversarial random oracle queries and without loss of generality, let us consider queries to RO_0. When we wish to discuss information the adversary learns from queries made to the random oracle, we write $\mathcal{V}(Q)$.

We already know from Lemma 1 that when the random oracle is looked at in isolation, a set of queries reveals no more information to the adversary beyond the set of responses. Lemma 1 holds for both honest queries in H and adversarial queries in Q alike. Let W be the wire labels set compatible with e^b (Definition 8). For a candidate label $L \in Q$ that is queried, either $L \in W$ or $L \notin W$. If $L \in W$, it follows that information about e^b, beyond (F, d), is revealed. If, in the worst case, $e^b \in Q$ then all the randomness used in garbling $(F, d) \in \mathcal{V}(Q)$ along with the challenger's bit b. In contrast, querying labels $L \notin W$ eliminates all wire sets W' that contain L, and their corresponding e, from being possible candidates for e^b.

We examine the a-priori statistical distance of the distribution of compatible (e, F, X, d) when $e \in \varepsilon(X, x^b)$ and when $e \in \varepsilon(X, x^{1-b})$ over a random choice of the random oracles. Going forward, we term as a *false positive*, the event that (e, F, d) is compatible (Definition 12) given that $e \neq e^b$, where e^b is the input encoding set that the challenger used to compute (F, d). The following lemmas facilitate this calculation.

Lemma 5. *In Algorithm 8, for every input encoding set e such that $e \neq e^b$ and over a random choice of the RO,*

$$Pr[(e, F, d) \text{ is compatible (Definition 12)}] \leq 0.75^{q\ell}$$

Proof: By definition, $e \neq e^b$. However, it could be the case that for some input encoding set $e \in \varepsilon(X, x^b) \cup \varepsilon(X, x^{1-b})$, the random oracle mapping is such that (F, d) is also compatible with e even though $e \neq e^b$.

Let the circuit have q gates. For a particular garbling $F = \{\nabla^1, \ldots, \nabla^q\}$, let us first consider one gate with the index i. Recall that a gate garbling ∇^i is derived according to either Table 1 or Table 2 given a set $t^i = (X^i_{00}, X^i_{01}, X^i_{10}, X^i_{11})$.

Given ∇^i, we count the number of possibles tuples $t^i = (X^i_{00}, X^i_{01}, X^i_{10}, X^i_{11}) \in \{0,1\}^{4\ell}$ that can be compatible with it (Definition 10). This corresponds to a uniformly random choice of random oracle outputs from RO^i_0.

∇^i is an ℓ-bit string and each bit position $\nabla^i[j]$ is created as a function of the bits $t^i[j] = (X^i_{00}[j], X^i_{01}[j], X^i_{10}[j], X^i_{11}[j])$. From the 16 columns in Tables 1 and 2, there are 4 possible combinations of $t^i[j]$ for which $\nabla^i[j] = 1$, and 12 combinations for which $\nabla^i[j] = 0$. Let us denote by $Hw(\nabla^i)$ the hamming weight of ∇^i (the number of 1s in ∇^i). Then the number of possible compatible choices of t^i would be upper bounded as,

$$4^{Hw(\nabla^i)} \cdot 12^{\ell - Hw(\nabla^i)} = 4^\ell \cdot 3^{\ell - Hw(\nabla^i)} \leq 12^\ell$$

Now, we count the number of $t = \{t^1, \ldots, t^q\}$ that are compatible (Definition 11) with the whole garbled circuit $F = \{\nabla^1, \ldots, \nabla^q\}$. This is bounded by $12^{q\ell}$. Note that for each of the q gates, the random oracles behave independently for each gate due to the tweaks and we count over all possible RO_0 responses. The total number of possible sets, compatible and incompatible, $t \in \{0,1\}^{4q\ell}$ is $2^{4q\ell}$. So, for any input encoding set $e \neq e^b$ (that extends to a unique W (Definition 8) that gives a unique t), the probability that e is compatible is upper bounded by $\frac{12^{q\ell}}{16^{q\ell}} = 0.75^{q\ell}$. Ergo,

$$\Pr[(e, F, d) \text{ is compatible} | e \neq e^b \text{ chosen by challenger}] \leq 0.75^{q\ell},$$

which completes the proof of this lemma.

□

Putting in plain words, the above corollary analyses the probability that some input encoding set e is compatible with (F, d) when the challenger garbles (F, d) from an $e^b \neq e$.

We now show that the number of false positives in $\varepsilon(X, x^b)$ and $\varepsilon(X, x^{1-b})$ is normally distributed and statistically close.

Lemma 6. *In Algorithm 8, for $b \in \{0,1\}$, the number of false positives in $\varepsilon(X, x^{1-b})$ is normally distributed with mean $\mu^{fp}_{\varepsilon(X,x^{1-b})} = 0.75^{q\ell}(2^\ell - 1)^n$ and variance $\sigma^{fp^2}_{\varepsilon(X,x^{1-b})} = 0.75^{q\ell}(1 - 0.75^{q\ell})(2^\ell - 1)^n$, i.e.,*

$$b^{1-b}_{fp} \sim \mathcal{N}\Big(0.75^{q\ell}(2^\ell - 1)^n, 0.75^{q\ell}(1 - 0.75^{q\ell})(2^\ell - 1)^n\Big)$$

Proof: By construction, $e \neq e^b$ thus any compatible e is a false positive by definition and we know from Lemma 5 that

$$\Pr[(e, F, d) \text{ is compatible}] \leq 0.75^{q\ell}$$

where q is the number of gates in the circuit and ℓ is the length of the label. From Lemma 2 there are $N = (2^\ell - 1)^n$ candidates $e \in \varepsilon(X, x^1)$. As the candidates are independent due to the properties of the random oracle, we can model the distribution of false positives in $\varepsilon(X, x^{1-b})$ as a binomial random variable

$$b_{\text{fp}} \sim \mathcal{B}((2^\ell - 1)^n, 0.75^{q\ell})$$

Using the normal approximation to the binomial distribution we obtain

$$b_{\text{fp}} \approx \mathcal{N}\Big(0.75^{q\ell}(2^\ell - 1)^n, 0.75^{q\ell}(1 - 0.75^{q\ell})(2^\ell - 1)^n\Big)$$

and the lemma follows. □

Lemma 7. *In Algorithm 8, for $b \in \{0, 1\}$, the number of false positives in $\varepsilon(X, x^b)$ is normally distributed with mean*

$$\mu^{fp}_{\varepsilon(X,x^b)} = 0.75^{q\ell}((2^\ell - 1)^n - 1) = \mu^{fp}_{\varepsilon(X,x^{1-b})} - 0.75^{q\ell}$$

and variance

$$\sigma^{fp^2}_{\varepsilon(X,x^b)} = 0.75^{q\ell}(1 - 0.75^{q\ell})((2^\ell - 1)^n - 1) = \sigma^{fp^2}_{\varepsilon(X,x^{1-b})} - 0.75^{q\ell}(1 - 0.75^{q\ell})$$

i.e.,

$$b^b_{fp} \sim \mathcal{N}\Big(0.75^{q\ell}((2^\ell - 1)^n - 1), 0.75^{q\ell}(1 - 0.75^{q\ell})((2^\ell - 1)^n - 1)\Big)$$

Proof: The proof follows the same analysis as that of Lemma 6. □

Focusing only on false positive values, we see that the statistical distance between b^0_{fp} and b^1_{fp} is rather small. Formally, it is bounded by the difference between the two distributions,

$$b_{\text{fp}} = \Big|b^b_{\text{fp}} - b^{1-b}_{\text{fp}}\Big|$$

This variable is also normally distributed with mean,

$$\begin{aligned}\mu &= \mu^{\text{fp}}_{\varepsilon(X,x^b)} - \mu^{\text{fp}}_{\varepsilon(X,x^{1-b})} \\ &= \mu^{\text{fp}}_{\varepsilon(X,x^{1-b})} - 0.75^{q\ell} - \mu^{\text{fp}}_{\varepsilon(X,x^{1-b})} \\ &= -0.75^{q\ell}\end{aligned}$$

and variance,

$$\begin{aligned}\sigma^2 &= {\sigma^{\mathrm{fp}}_{\varepsilon(X,x^b)}}^2 + {\sigma^{\mathrm{fp}}_{\varepsilon(X,x^{1-b})}}^2 \\ &= {\sigma^{\mathrm{fp}}_{\varepsilon(X,x^{1-b})}}^2 - 0.75^{q\ell}(1-0.75^{q\ell}) + {\sigma^{\mathrm{fp}}_{\varepsilon(X,x^{1-b})}}^2 \\ &= 2{\sigma^{\mathrm{fp}}_{\varepsilon(X,x^{1-b})}}^2 - 0.75^{q\ell}(1-0.75^{q\ell})\end{aligned}$$

Therefore we have that, the random variable b_{fp} counting the difference between the distributions, $\big((e,F,d)\text{ compatible}|e \neq e^b, e \in \varepsilon(X,x^0)\big)$ and $\big((e,F,d)\text{ compatible}|e \neq e^b, e \in \varepsilon(X,x^1)\big)$, is distributed as,

$$b_{\mathrm{fp}} \sim \mathcal{N}\Big(-0.75^{q\ell}, 2\big(0.75^{q\ell}(1-0.75^{q\ell})(2^\ell-1)^n\big) - 0.75^{q\ell}(1-0.75^{q\ell})\Big)$$

Now, when we take into account that $e^b \in \varepsilon(X,x^b)$ chosen by the challenger is compatible with (F,d), and that this is just one element in $(2^\ell-1)^n$ total, we have the random variable diff that counts the difference between the distributions, $\big((e,F,d)\text{ compatible}|e \in \varepsilon(X,x^0)\big)$ and $\big((e,F,d)\text{ compatible}|e \in \varepsilon(X,x^1)\big)$. This is distributed as,

$$\mathsf{diff} \sim \mathcal{N}\Big((2^\ell-1)^{-n} - 0.75^{q\ell}, 2\big(0.75^{q\ell}(1-0.75^{q\ell})(2^\ell-1)^n\big) - 0.75^{q\ell}(1-0.75^{q\ell})\Big) \tag{5}$$

Note again that the goal of the adversary is to identify which set, $\varepsilon(X,x^0)$ or $\varepsilon(X,x^1)$ the challenger's secret input encoding set e^b came from, given only (F,X,d). We calculate the a-priori statistical distance between the distributions before any RO query is made. Over a random choice of the RO, given the challenge (F,X,d), and an arbitrary choice of e, for the secret choice bit b of the challenger, the statistical distance can be compactly described by the mean of the above random variable,

$$\Big|\Pr[(e,F,d)\text{ compatible}|e \in \varepsilon(X,x^b), \mathcal{V}(F,X,d)] - \Pr[(e,F,d)\text{ compatible}|e \in \varepsilon(X,x^{1-b}), \mathcal{V}(F,X,d)]\Big| \leq (2^\ell-1)^{-n} - 0.75^{q\ell}$$

Now, we need to show that if the access to the RO is limited only to honest queries H and *permissible* sets Q (Definition 4), then the adversary's advantage in distinguishing between these distributions, $e \in \varepsilon(X,x^0)$ and $e \in \varepsilon(X,x^1)$, is still bounded by $2^{-\rho}$ for a statistical security parameter ρ. We require that this bound hold for all choices of the set Q that the adversary can possibly make. Let $|Q| = s$ be the bound on the number of queries.

Honest Queries. Recall that H is the set of all honest queries. For m output bits and q being the number of gates in F, $|H| = 2q + m$. It is easy to see that given (F, X, d) making all the queries in H reveals the *active path* in F to the adversary. That is, with the input labels in X, one can evaluate the garbled circuit to find $f(x_b)$. Note that $f(x_0) = f(x_1)$ so the output itself reveals no information on b. This evaluation reveals for every wire w^j, an active label L^j. Let the set of wire labels $W_X \subset W$ be the ones that are revealed. Further, for each gate j, one value $X^j \in t^j$ is revealed. Recall that we already establish from Lemma 4 that given $\mathcal{V}(F, X, d)$ the adversary has 0 advantage. For both x^0 and x^1, the same set X is the active input labels set. Given F, during the garbling, the active path revealed was generated deterministically depending only on X. Since for the cases $b = 0$ and $b = 1$, the garbling F is different, the active paths will contain different values.

However, for any gate i, the adversary having two active input labels L_X^A and L_X^B can query RO_0^i to derive only one active X^i. We know from the property of the RO (Lemma 1) that this reveals no information about other query to response mappings and so nothing about other candidate inactive labels. Given X^i and $\nabla^i \in F$, it is a property of how ∇ is constructed that it reveals no information about the other $X^{i'}$ values in the tuple t^i. Although one can compute the active key K^i from this, it reveals nothing about the inactive $K^{i'}$. Further, we know (Lemma 1) that RO_1^i when queried with the active key reveals nothing beyond the active output label L_X^C. This holds, by induction, for every gate in the circuit.

We know from Lemma 1 that the queries in H will reveal no information beyond this active path. So, the adversary's view does not give it any advantage in finding b.

$$\left| \Pr[e^b \in \varepsilon(X, x^0) | \mathcal{V}(F, X, d, H)] - \Pr[e^b \in \varepsilon(X, x^1) | \mathcal{V}(F, X, d, H)] \right| = 0$$

Adversarial Queries. We are now considering the case where all honest queries in H are already made and it only remains to make *adversarial queries*. This is without loss of generality since making adversarial queries *without* making the honest queries first gives the adversary strictly lesser information than making them after all honest queries are made. Going forward, we analyse the advantage of the adversary given it's complete view,

$$\mathsf{Adv} = \left| \Pr[e^b \in \varepsilon(X, x^0) | \mathcal{V}(F, X, d, H, Q)] - \Pr[e^b \in \varepsilon(X, x^1) | \mathcal{V}(F, X, d, H, Q)] \right|$$

Without loss of generality, consider the queries made to RO_0. For a gate indexed i with input wires A and B, let L_X^A and L_X^B be the active labels that are already derived from X due to queries in H. The adversary can now select *candidate* inactive labels $L^{A'}$ and $L^{B'}$ and test these with the active labels using the random oracle. On compiling the random oracle responses, this would form a *candidate* t^i that can be checked for compatibility with $\nabla^i \in F$ (Definition 10). We already know from the analysis of the proof for Lemma 5 that up to 12^ℓ out of the total 16^ℓ possible t^i values are compatible.

We bound the adversary's advantage on a single query. Consider just the wire A. An adversary chooses a candidate inactive label $L^{A'}$ for it. It performs *one* *random oracle query* $RO_0^i(L^{A'}, L_X^B) \rightarrow X^{i'}$. This $X^{i'}$ is part of a candidate t^i that has probability up to $\frac{12^\ell}{16^\ell} = 0.75^\ell$ of being compatible with ∇^i. Therefore, the probability that the label $L^{A'}$ leads to a compatibility is bounded by,

$$\Pr[L^{A'} \text{ is compatible}] \leq 0.75^\ell$$

Next, we bound the information revealed when the candidate $L^{A'}$ *for an input wire label* is found to be incompatible. We know from Lemma 2 that the total number of input encoding sets e fixed at the active labels X is $(2^\ell - 1)^n$. Disqualifying a candidate inactive label $L^{A'}$ for the A^{th} input bit effectively disqualifies all the input encoding sets e' with $L^{A'}$ at position A. Since the other labels could take any value, the total number of disqualified input encoding sets is $(2^\ell - 1)^{n-1}$. Therefore, the advantage would be,

$$(\mathsf{Adv}|L^{A'} \text{ is incompatible}) = \frac{(2^\ell - 1)^{n-1}}{(2^\ell - 1)^n} = \frac{1}{2^\ell - 1}$$

Note that considering the input wire labels only is good enough to hold without loss of generality for any wire of the circuit. This is because any internal wire is separated from the input encoding set by additional random oracle calls. An adversary cannot query the RO for the inverse mapping of a label. Hence, disqualifying candidate inactive labels for an internal wire without knowing which input wire label it extends (Definition 8) from reveals no information.

Therefore, we have that a single RO call corresponding to the adversary's candidate inactive label $L^{A'}$ for wire A bounds $\mathcal{A}$'s advantage by,

$$\begin{aligned}
\mathsf{Adv} &= (\mathsf{Adv}|L^{A'} \text{ is compatible}) \cdot \Pr[L^{A'} \text{ is compatible}] \\
&\quad + (\mathsf{Adv}|L^{A'} \text{ is not compatible}) \cdot \Pr[L^{A'} \text{ is not compatible}] \\
&\leq \Pr[L^{A'} \text{ is compatible}] + (\mathsf{Adv}|L^{A'} \text{ is not compatible}) \\
&\leq 0.75^\ell + \frac{1}{2^\ell - 1} \\
&\leq \left(\frac{3}{4}\right)^\ell + \left(\frac{1}{2}\right)^\ell = \frac{3^\ell + 2^\ell}{2^{2\ell}}
\end{aligned} \tag{6}$$

We are now ready to present our main theorem:

Theorem 1. *Let* Circuit *be as in Algorithm 2. Then every oracle adversary* $\mathcal{A}$ *with* $f(\rho, \ell) = \ell - \rho - 1$ *(Definition 3) has an advantage* $\mathsf{Adv} < 2^{-\rho}$ *in the Input Privacy game (Algorithm 8) where* ρ *is a statistical security parameter.*

Proof: Let s be the number of queries made in Q. For statistical security parameter ρ, keeping in mind that $\mathsf{Adv} < 2^{-\rho}$, we have from (6) that,

$$\begin{aligned} s \cdot \frac{3^\ell + 2^\ell}{2^{2\ell}} &< 2^{-\rho} \\ s(3^\ell + 2^\ell) &< 2^{2\ell-\rho} \\ s &< 2^{-\rho}\frac{2^{2\ell}}{(3^\ell + 2^\ell)} \\ &< 2^{-\rho}\frac{2^{2\ell}}{2^{\ell+1}} \\ &= 2^{\ell-\rho-1} \end{aligned}$$

This matches the number of queries allowed in a *permissible* set Q (Definition 4). Note that this number of queries allowed grows exponentially in ℓ. □

Acknowledgments Tomer Ashur is an FWO post-doctoral fellow under Grant Number 12ZH420N. Anasuya Acharya, Efrat Cohen and Carmit Hazay were 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.

References

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

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

CFIK03. Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In *EUROCRYPT*, pages 596–613, 2003.

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.

IK02. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

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.

Kil88. Joe Kilian. Founding cryptography on oblivious transfer. In *ACM*, pages 20–31, 1988.

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.

Kol05. Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *ASIACRYPT*, pages 136–155, 2005.

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.