Oblivious Garbling and its Applications

Tomer Ashur[1], Carmit Hazay[2], and Rahul Satish[3]

[1] 3MI Labs, Leuven, Belgium tomer@3milabs.tech
[2] Bar-Ilan University, Ramat Gan, Israel carmit.hazay@biu.ac.il
[3] IT University of Copenhagen rahs@itu.dk

Abstract. A garbling scheme encodes a function and an input into two independent artifacts from which the output can be recovered, but nothing else is revealed. This clean separation between function and input has made garbling one of the most versatile primitives in cryptography. Yet it hides an asymmetry that has gone largely unexamined: while the input is cryptographically protected, the function is fully exposed to whoever performs the garbling. As garbling is increasingly deployed in settings where garbling is delegated to untrusted infrastructure, published on public ledgers, or distributed among multiple parties, this asymmetry becomes a fundamental barrier. The function, which may encode proprietary models, confidential policies, or sensitive decision logic, is leaked unconditionally to the garbling server.

We introduce oblivious garbling, a new paradigm that closes this gap. In our framework, the garbler receives only a designated leakage of the circuit and remains oblivious to everything else. We present the first construction instantiating this notion where the leakage is the circuit topology alone, achieving linear complexity with no blow-up in the size of the garbled circuit. The construction extends to the malicious setting with no asymptotic overhead. Beyond its theoretical contribution, oblivious garbling has immediate practical consequences: it enables outsourced garbling without function exposure, garbling on untrusted hardware without leaking proprietary logic, and a multi-party garbling protocol in which no garbling party learns the function, all without resorting to universal circuits.

1 Introduction

Over the last few decades, garbling schemes have emerged as one of the most fundamental objects in cryptography. Since their introduction by Yao [34], they have been studied extensively from both theoretical and practical perspectives, leading to a deep understanding of their security properties [6] and a long line of efficiency improvements [18, 23, 26, 35]. Today, garbling schemes serve as the backbone of constant-round secure computation and appear as a building block across a wide range of cryptographic constructions, including verifiable computation [4, 10], functional encryption [12, 28], and software protection [5, 11, 13].

Garbling schemes can be viewed as a cryptographic object that, given a function f and an input x, produces two corresponding encodings $\hat{F}$ and $\hat{X}$ respectively.

One can in turn use these encodings to evaluate to the function output $f(x)$, revealing nothing beyond it. This makes them a natural and efficient tool for secure two-party computation: both parties hold private inputs to a publicly agreed-upon function, one acts as the garbler and the other as the evaluator, and the garbling scheme ensures that neither party learns anything about the other's input beyond what the output itself reveals.

Beyond this classical setting, the garbler can naturally be thought of as a computationally powerful server that encodes functions on behalf of clients, and the evaluator as a client who holds private data and wishes to learn the result of a computation without exposing it. A natural instantiation of this is outsourced garbling [5, 8, 15], where a function owner delegates the garbling workload to a powerful server to avoid bearing the computational burden of garbling large circuits locally. In more advanced deployments, such as blockchain-based secure computation [1–3], this is taken further to support public verifiability and regulatory compliance. A server publishes the garbled circuit $\hat{F}$ on-chain, where everyone can observe it and any leakage it carries. Clients publish their garbled inputs, and any assisting entity can evaluate and publish the encoded output, either on-chain or off-chain as required. The client then retrieves $f(x)$ without any further interaction, and the on-chain record provides a publicly auditable trace of the computation. In each of these settings, a third entity performs the garbling on behalf of the function owner: it may be a cloud server engaged for its computational resources, a hardware accelerator offering garbling at line rate, or a public ledger infrastructure. The function owner may resort to such *third-party* services because it lacks the cryptographic infrastructure to garble large circuits itself, or because compliance requirements mandate that its computation be publicly auditable on-chain. In either case, the function being garbled may be highly proprietary, and the function owner would strongly prefer that the server performing the garbling learns nothing about it.

For many applications the function encodes highly sensitive information that the server should not see. A proprietary machine learning model encodes its architecture, weight comparisons, and decision thresholds, constituting the core intellectual property of the company that owns it. A financial risk assessment circuit encodes confidential business rules that a regulated institution cannot expose to third parties. A medical diagnosis protocol encodes clinical decision logic derived from years of proprietary research, which a hospital or pharmaceutical company cannot afford to expose to external infrastructure. A legal compliance circuit encodes regulatory rules or contractual conditions that must remain confidential even from the parties executing the computation. A biometric matching protocol encodes sensitive policy parameters that must remain private even from the infrastructure provider processing the garbling. Moreover, the function and the input are not always interchangeable objects: a function encoding a large machine learning model may contain millions of gates, whereas the client's input may be a modest sized vector of values. Hiding the function is therefore a qualitatively different and harder problem than hiding the input.

A natural attempt towards a solution to this notion of function hiding is to rely on universal circuits [17, 20, 21, 29]. A universal circuit is a fixed-topology graph that can simulate any circuit up to a given size bound, with the actual circuit description provided as an additional encoded input by the function owner. Since whoever processes the computation sees only the fixed universal topology, the target function is hidden. Crucially, this transformation converts the function hiding problem into an input hiding problem: the circuit description is now just another input, and any scheme that protects inputs, including garbling schemes, can be directly applied. However, simulating a circuit of size n requires a universal circuit of size $\Omega(n \log n)$ [32], and the best known constructions achieve a concrete size of $3n \log_2 n$ [21]. This means the cost of any computation now scales with both the function size and the circuit size together, and for a circuit with one million gates this corresponds to a $60\times$ blowup.

In blockchain-based computation, a $60\times$ blowup translates directly into a $60\times$ increase in on-chain gas cost. In outsourced settings, it means processing and transmitting a circuit sixty times larger. Beyond these concrete costs, routing a function through a universal circuit destroys all circuit-level structure: practitioners routinely tailor circuit representations to optimize the AND-to-XOR gate ratio and circuit depth, and these optimizations become impossible once the function is compiled into a fixed universal skeleton. The approach is thus expensive in most settings where one would actually want to use it, and raises the natural question of whether function hiding can be achieved without this overhead.

We note that existing techniques for *function-hiding* against an evaluator cannot be repurposed here. A rich line of work on private function evaluation [19, 22, 31] has developed methods to hide gate types from the evaluator during evaluation, using uniform output encodings or oblivious routing. These techniques are valuable and address an important problem, but they are orthogonal to ours. They still require the garbler to have complete knowledge of the function in order to construct the truth-table-based encodings. They achieve asymmetric privacy: the circuit is hidden from the evaluator, but fully exposed to the garbler. Our goal is precisely the complement, and this objective is entirely absent from the existing literature.

In this work, we show that the garbling paradigm can be reconceived to support *function-hiding* against a garbler without resorting to universal circuits and without any blowup in the circuit size. We introduce the notion of *oblivious garbling*, a garbling paradigm in which the garbler receives only what a designated leakage function reveals about the circuit and remains completely oblivious to everything else. We present a concrete and efficient instantiation targeting the notion of *semi-private* function hiding [25], where the leakage is the circuit topology alone. This precisely captures the sensitivity structure of the motivating scenarios above: in outsourced garbling, hardware deployment, and blockchain-based computation, the circuit topology is typically either public knowledge, dictated by hardware constraints, or reconstructible from agreed-upon size bounds and

connectivity requirements. The genuinely sensitive information is the gate logic itself, the specific threshold comparisons, multiplexer conditions, decision rules, and model parameters that encode the function owner's proprietary intellectual property. Protecting exactly this, while tolerating topology leakage, is what semi-private function hiding achieves, and it is sufficient for a variety of existing applications.

1.1 Our Contributions

In this work, we present the first garbling scheme that achieves semi-private function privacy from the garbler. Our construction enables the garbling entity to securely garble a circuit using only its topology, completely oblivious to the underlying gate functionalities, and entirely without using universal circuits. Our approach operates as a black-box over any symmetric-key garbling scheme that supports AND gate garbling and the free-XOR optimization. It perfectly preserves the original circuit topology, imposing zero asymptotic garbling overhead compared to the underlying scheme. We summarize our contributions below.

- **New notion.** We introduce and formally define oblivious garbling (Definition 3), extending the BHR framework [6] to capture garbler-side function privacy. The definition introduces a new algorithm SelectGate that conveys gate-type information to the evaluator without revealing it to the garbler.
- **First linear-overhead construction.** We present the first oblivious garbling scheme for circuits with AND and XOR gates achieving $\mathcal{O}(|f|)$ computation and communication, with garbling cost per gate matching that of the underlying AND-gate scheme and no blowup in circuit size.
- **Malicious security.** We extend our construction to the malicious setting via authenticated garbling [16, 30], preserving $\mathcal{O}(|f|)$ complexity with no additional assumptions beyond those of the underlying scheme.

Our construction is black-box in any symmetric-key garbling scheme supporting free-XOR and is fully compatible with state-of-the-art optimizations. Instantiated with the three-halves scheme [26], it achieves a garbling size of 1.5κ bits per gate. At its core, our technique implies a universal gadget that allows the evaluator to program it to either AND or XOR gate at evaluation time, through the GateSelect mechanism. Since the evaluator must choose between the two possible gate types for every gate in the circuit, this translates to $\mathcal{O}(|f|)$ communication for SelectGate, an inherent price for obliviousness. In the context of two-party computation, this is realised via OT, one per gate, which can be batched together with the input encoding OTs and instantiated using OT extension [14] to reduce the public-key overhead to a small constant. Since these OTs are batched with input encoding, the construction introduces no extra rounds. Our construction extends seamlessly to the malicious setting through the authenticated garbling [16,30] technique. As this extension already requires $\mathcal{O}(|f|)$ OTs as part of its preprocessing, the gate selection mechanism adds no additional asymptotic overhead to the scheme.

Applications. Oblivious garbling enables a range of deployment models in which the function must remain private from the garbler, a guarantee that no prior garbling scheme can provide without resorting to universal circuits.

1. Multi-party garbling with full function privacy. In the classical scenario of MPC, the circuit is public, and only the inputs are protected. When the circuit itself is sensitive, existing multi-party garbling protocols offer no relief: they require all garbling parties to know the complete circuit, hiding it only from the designated evaluator. Oblivious garbling resolves this cleanly. All garbling parties work from the public topology, collaboratively producing a garbled circuit of uniform $\mathcal{G}_{\text{tri}}$ gadgets without any party learning what any gate computes. Gate semantics are deferred entirely to evaluation time, when the function owner conveys gate types to the evaluator via OT. This yields the first practical framework for function-private multi-party garbling in which no garbling party learns the function, while maintaining constant-round complexity. Achieving this previously required universal circuits and their attendant logarithmic blowup.

2. Outsourced garbling. In many settings, a device may lack the computational resources or infrastructure to locally garble a large circuit. Offloading this work to a cloud server is natural, but under any existing scheme, it requires transmitting the complete circuit, immediately exposing the function. With oblivious garbling, the device transmits only the wiring topology. The server garbles the circuit without ever seeing gate logic, and gate semantics are activated later during evaluation. This extends naturally to blockchain-based settings [1–3], where the server publishes the garbled circuit on-chain for public verifiability and regulatory compliance, while the function remains entirely hidden from the server and from the public ledger. Because every gate is garbled identically from the server's perspective, this architecture also supports streaming: the circuit can be processed and transmitted layer by layer without the server accumulating any gate-type information, keeping the computational and on-chain footprint minimal.

3. Hardware-accelerated garbling. Specialized hardware, FPGAs, ASICs, or secure enclaves, can garble circuits at line rate and at a fraction of the energy cost of general-purpose processors, making them attractive platforms for high-throughput secure computation. However, such hardware is frequently manufactured by partially trusted or potentially compromised supply chains, and even a trusted enclave exposes the full circuit to its runtime. Oblivious garbling acts as a privacy-preserving compilation layer: the hardware processes only the circuit topology, never ingesting the sensitive gate logic. When the circuit encodes proprietary trading algorithms, biometric matching policies, or classified decision procedures, the intellectual property remains cryptographically isolated from the execution hardware, regardless of whether the hardware itself is adversarially manufactured.

1.2 Technical Overview

Our goal is to design a garbling scheme that, in addition to the standard privacy guarantees of any garbling scheme, also ensures that the garbler learns nothing about the logic of individual gates beyond the circuit topology. The central observation driving our construction is that the gadget $\mathcal{G}_{\text{tri}}(x, y, z) = x \wedge (y \oplus z)$ is a universal programmable gate: fixing $z = 0$ reduces it to AND on (x, y), and fixing $x = 1$ reduces it to XOR on (y, z). Since $\mathcal{G}_{\text{tri}}$ is structurally equivalent to a single AND gate with a free-XOR preprocessing step on one input, it can be garbled using any standard AND-gate garbling scheme at exactly the cost of one AND gate. We replace every gate in the circuit with a $\mathcal{G}_{\text{tri}}$ gadget and have the garbler process all gadgets identically, without knowing which will be programmed as an AND gate and which as an XOR gate. Crucially, the gadgets are not garbled obliviously. The obliviousness of the scheme emerges not from how gadgets are garbled but from how they are used: because every gadget is identical from the garbler's perspective, no information about gate types is leaked during garbling. Since each gadget is independent, the entire garbled circuit can be produced in parallel from the topology alone.

Garbling gadgets independently creates a connectivity challenge: the output label of one gadget must feed into the correct input terminal of the next, but which terminal is correct depends on the gate type, which the garbler does not know. We resolve this via a technique we call *soldering*, building on label-translation ideas from [9, 24]. For each wire connection, the garbler computes two soldering values, one for the AND programming and one for the XOR programming, as XOR offsets between the 0-labels of adjacent wires under the global free-XOR offset Δ. These values are information-theoretically compact and require no cryptographic assumptions beyond the underlying garbling scheme. The free-XOR structure is essential to our construction in two distinct ways: it allows the internal XOR computation inside $\mathcal{G}_{\text{tri}}$ to be processed for free, and it enables soldering to be computed as simple XOR operations on wire labels with no additional ciphertexts, since label translations under Δ preserve the algebraic structure that correctness relies on.

During evaluation, the evaluator selects the correct soldering value for each gate via a 1-out-of-2 oblivious transfer, with the gate type as the selection bit. The garbler, acting as OT sender, learns nothing about which path was taken, and the evaluator learns only the active soldering value. All $\mathcal{O}(|f|)$ such OTs are batched in a single round via OT extension [14], so the amortized per-gate cost reduces to a small number of symmetric-key operations plus the message payload. The resulting scheme is fully black-box with respect to any symmetric-key garbling scheme supporting free-XOR. The garbling cost per gadget is exactly that of one AND gate, the total gadget count matches the original circuit size, and the circuit topology is preserved exactly, giving $\mathcal{O}(|f|)$ computation and communication with no blowup.

In the malicious setting, a corrupt garbler might attempt to inject an incorrect soldering value to steer evaluation toward a wrong output. Our construction

extends naturally to this setting via authenticated garbling [16, 30]: each wire label carries an authentication tag under the IT-MAC structure of authenticated garbling, and an incorrect soldering value produces a translated label whose tag fails verification immediately, causing the evaluator to abort. The correctness of the garbled gadgets themselves follows directly from the correctness of the underlying authenticated garbling scheme. The authentication of soldering values reduces to local operations on existing tag material, adding no overhead beyond standard authenticated garbling. Since authenticated garbling already requires $\mathcal{O}(|f|)$ preprocessing OTs for its own purposes, the $\mathsf{SelectGate}$ OTs batch into the same OT extension call [7, 14, 27, 33] at no asymptotic overhead, preserving the $\mathcal{O}(|f|)$ complexity of the semi-honest scheme.

2 Preliminaries

Circuits. We define a boolean circuit $\mathbf{C} : \{0,1\}^{\mathsf{n}} \to \{0,1\}^{\mathsf{m}}$ as the following tuple: $(\mathsf{n}, \mathsf{m}, \mathsf{q}, \{A_g, B_g, C_g, f_g\}_{g \in [\mathsf{q}]})$, where f_g represents the functionality of the gate, $\mathsf{n} \geq 2$ is the number of inputs, $\mathsf{m} \geq 1$ is the number of outputs, and $\mathsf{q} \geq 1$ is the number of gates. For a given gadget $g \in \mathbf{C}$, the function $\mathsf{GadgetIn}(g)$ returns the indices of all the input wires, while the function $\mathsf{GadgetOut}(g)$ outputs the index of the output wire. Additionally, the functions $\mathsf{CircuitIn}(\mathbf{C}), \mathsf{CircuitOut}(\mathbf{C})$ outputs the list of all input and output gates of the circuit $\mathbf{C}$ respectively.

2.1 Definition for Garbling Schemes

We use and state the definition in [26], a slightly modified definition of garbling schemes to that of [6]. We rewrite it here for the sake of completeness.

Definition 1 *A garbling scheme consists of four algorithms:*

1. $\mathsf{Garble}(1^\kappa, f) \to (F, e, d)$ *which takes in as input the security parameter* κ*, and the functionality* f *in the form of a circuit and returns the garbled circuit* F*, encoding information* e *for the wire labels and the decoding information* d *for the output wire labels.*
2. $X := \mathsf{Encode}(e, x)$ *which takes in as input the encoding information* e*, input* x *and gives the Garbled input* X*.*
3. $Y := \mathsf{Eval}(F, X)$ *which takes in as input the garbled circuit* F*, the garbled input* X*, and returns the garbled evaluation of* f *on* x*, as* Y*.*
4. $y := \mathsf{Decode}(d, Y)$ *which takes in as input the decoding information* d*, garbled output* Y *and gives the output* y*.*

We want the following properties from a garbling scheme:

Correctness: *For any circuit* f *and input* x*, while* $(F, e, d) \leftarrow \mathsf{Garble}(1^\kappa, f)$*, we have* $f(x) = \mathsf{Decode}(d, \mathsf{Eval}(\mathsf{Encode}(e, x)))$ *except for negligible probability.*

Privacy: *There is a simulator* Sim *such that for any circuit* f *with leakage information* $\Phi(f)$ *and input* x*, the following distributions are indistinguishable:*

$(F, e, d) \leftarrow \mathsf{Garble}(1^\kappa, f)$
$X := \mathsf{Encode}(e, x)$
return (F, X, d)

$(F, X, d) \leftarrow \mathsf{Sim}(1^\kappa, \Phi(f), f(x))$
return (F, X, d)

Obliviousness: *There is a simulator* Sim *such that for any circuit f with leakage information $\Phi(f)$ and input x, the following distributions are indistinguishable:*

$(F, e, d) \leftarrow \mathsf{Garble}(1^\kappa, f)$
$X := \mathsf{Encode}(e, x)$
return (F, X)

$(F, X) \leftarrow \mathsf{Sim}(1^\kappa, \Phi(f))$
return (F, X)

Authenticity: *For any circuit f and input x, no PPT adversary* Adv *can make the following distribution output* TRUE *except with negligible probability:*

$(F, e, d) \leftarrow \mathsf{Garble}(1^\kappa, f)$
$X := \mathsf{Encode}(e, x)$
$Y \leftarrow \mathsf{Adv}(F, d, X)$
return $\mathsf{Decode}(d, Y) \notin \{f(x), \perp\}$

2.2 Circular Correlation Robustness

We use the definition of circular correlation robustness from [35]. Given a hash function H, we define two oracles:

- $\mathsf{Circ}_R(x, i, b) = H(x \oplus R, i) \oplus bR$, where $R \in \{0,1\}^{\kappa-1}\|1$
- $\mathsf{Rand}(x, i, b)$: a random function with κ-bit output.

Definition 2 (Circular Correlation Robustness) *A sequence of oracle queries of the form (x, i, b) is* legal *if the same value of (x, i) is never queried with different values of b. A hash function H is* circular correlation robust *if for all polynomial-time adversaries $\mathcal{A}$ making only legal queries,*

$$\left| \Pr_R \left[\mathcal{A}^{\mathsf{Circ}_R}(1^\kappa) = 1 \right] - \Pr_{\mathsf{Rand}} \left[\mathcal{A}^{\mathsf{Rand}}(1^\kappa) = 1 \right] \right|$$

is negligible.

The legality restriction prevents the adversary from trivially recovering R. Note that for the single-key version used here, no extra parameter a is needed to produce values of the form $H(x \oplus aR, i) \oplus bR$, since setting $a = 0$ would be illegal under the definition anyway. We also note that the adversary is allowed unrestricted access to H itself, in addition to the oracle in the experiment.

3 Oblivious Garbling

In this section, we introduce the notion of oblivious garbling, where the garbler remains oblivious to the circuit's functionality, knowing only its topology. We discuss and present a scheme for arbitrary circuits consisting of 2-input AND and XOR gates with a communication cost of $O(|f|)$ bits and $O(|f|)$ OTs, where $|f|$ is the size of the circuit. To accomplish this, we first replace all gates in the circuit with the $\mathcal{G}_{\text{tri}}$ gadget, defined as $\mathcal{G}_{\text{tri}} = x \wedge (y \oplus z)$. We observe that this gadget can be adapted into an AND gate by setting $z = 0$ and utilizing x and y as input lines. Similarly, the gadget can be reduced to an XOR gate by setting $x = 1$ and using y and z as inputs. We use this observation to devise a garbling scheme for circuits containing AND and XOR gates, without disclosing the gate type during the garbling process.

Security model. We prove security in the *semi-honest* model with two roles: a *garbler* (who may also act as the function owner's delegate) and an *evaluator*. The garbler receives only the circuit topology $\Phi(f)$ and learns nothing about the type of any individual gate. The function owner (which may be a separate entity, or the evaluator itself) holds the full circuit description f and provides gate-type information to the evaluator via the SelectGate procedure, implemented as OT (see Section 3.1). All parties are assumed to be semi-honest: they follow the protocol specification but may attempt to infer information from their transcripts. We note that no additional cryptographic assumptions (such as commitment schemes) are required beyond the underlying garbling scheme and OT, since the soldering values are information-theoretically compact XOR offsets under the free-XOR structure; their correctness does not require authentication in the semi-honest model.

3.1 Obliviously Garbling a Single Gate

We continue by illustrating the challenges involved in achieving oblivious garbling for an entire circuit, and explaining how we address them. As suggested above, the goal is to devise a technique that takes advantage of the fact that inputs to the $\mathcal{G}_{\text{tri}}$ gadgets can be programmed to reduce them to either an XOR or an AND gate. While programming inputs for input gates where the evaluator can control the inputs is straightforward, programming intermediate gates where the evaluator does not have access to the wire values is more challenging. Using the $\mathcal{G}_{\text{tri}}$ gadget to implement certain gates requires programming its inputs, effectively turning it into an input gadget itself. However, this reveals information about the type of gate: the wire used for programming (x for XOR or z for AND) reveals the functionality of the gate. Therefore, we need a solution that can let an evaluator reduce the gadget to an XOR gate or an AND gate, while hiding this reduction from the garbler.

We take a two-step approach to solving this problem. First, we use any black-box garbling scheme that can garble an AND gate with its labels adhering to free-XOR property to garble each gadget in the circuit independently. This preserves

obliviousness during garbling, since the only leakage is the number of $\mathcal{G}_{\text{tri}}$ gadgets. However, it is incomplete, since the evaluation still requires revealing the input labels for subsequent gadgets in topological order. To solve this problem, we use the "soldering" technique of [9, 24]. This technique allows the dynamic construction of circuits from independently garbled gates during evaluation, facilitating the connection of the output wires of each gadget to the corresponding input wires of the next, effectively constructing the desired circuit without revealing any gate information. We will first discuss how the soldering technique works, and then show how it can be used to obliviously garble a non-input gate.

Soldering Technique: The soldering technique, as the name suggests, allows one to connect independently generated gates. It was introduced in [24] to allow the transformation of the key representing bit b on one wire into the key representing bit b on another wire. This is done by computing some auxiliary information called the *soldering*. More formally, let's consider a scenario where we need to solder the output wire a of gate g_i to the first input wire b of gate g_{i+1}. Additionally, lets assume that we are in the free-XOR setting of [18] and denote by Δ the global free-XOR offset. To solder, we calculate the soldering value $S_{i,i+1} = A_0 \oplus B_0$ where A_0, B_0 are the 0-labels for wires a, b, respectively. Given the label representing bit-j for wire a as A_j, one can derive the label representing bit-j for wire b (denoted as B_j) using the solder value $S_{i,i+1}$ through the following computation: $A_j \oplus S_{i,i+1} = A_0 \oplus j\Delta \oplus A_0 \oplus B_0 = B_0 \oplus j\Delta = B_j$.

Garbling a single non-input gate: Consider the i-th non-input gate g'_i in the original circuit, with input wires x', y'. If we convert this to a $\mathcal{G}_{\text{tri}}$ gate g_i, say with wires x, y, z, the values these wires need to take depend on what g'_i needs to be programmed to do. The following illustrates the options:

- When g'_i is an XOR gate, the inputs of g should be $x = 1$, $y = y'$, $z = x'$.
- When g'_i is an AND gate, the inputs of g should be $x = x'$, $y = y'$, $z = 0$.

As discussed earlier, if only one of x, z is set to input in the topology, it becomes obvious which gate the gadget reduces to. We know that this is not a problem for input gates, since in this case all wires are input wires. Therefore, even in the case of non-input gates, we garble the gadget as if all wires were input wires, i.e., we garble each gadget independently, without considering that these wires might be the output wires of a gadget in the previous layer. As mentioned earlier, one can use any garbling scheme in the free-XOR setting to achieve this.

Let's assume that the inputs to gate g'_i in the original circuit were the outputs p, q of gates g'_{i-1} and g'_{i-2}, respectively. Since g_{i-1} and g_{i-2} were also garbled independently, evaluating them would only reveal the output labels P_{b_1}, Q_{b_2}, where b_1, b_2 are the values taken by wires p, q and P_{b_1}, Q_{b_2} are their corresponding labels. Having obtained these labels, if we wish to evaluate the gadget g_i as an AND gate, we need the labels X_{b_1}, Y_{b_2}, Z_0 for the wires x, y, z respectively. Similarly, to evaluate the gadget as an XOR gate, we will need the labels X_1, Y_{b_2}, Z_{b_1}. To achieve this, we use the soldering technique to transfer the bits

b_1 and b_2 from P_{b_1} and Q_{b_2} to the corresponding labels of g_i. The garbling algorithm calculates these soldering values for both AND and XOR scenarios. We then transfer one set of values to the evaluator using a 1-out-of-2 OT protocol.

OT instantiation of SelectGate. We make explicit the interactive protocol realizing SelectGate. For each gate l in the circuit (other than circuit input gates), the garbler holds both soldering pairs $(\mathsf{aux}_{\mathsf{xor},l}, \mathsf{aux}_{\mathsf{and},l})$ and the evaluator holds the gate type $\mathsf{type}_l \in \{0,1\}$ (where 0 denotes AND and 1 denotes XOR). The selection is implemented via OT as follows:

- The garbler acts as the OT *sender* with messages $m_0 = \mathsf{aux}_{\mathsf{and},l}$ and $m_1 = \mathsf{aux}_{\mathsf{xor},l}$.
- The evaluator acts as the OT *receiver* with selection bit $b = \mathsf{type}_l$.
- The evaluator receives $m_b = \mathsf{aux}_{\mathsf{type}_l,l}$ and learns nothing about the other soldering pair.
- The garbler learns nothing about type_l.

Observe that the value Y_{b_2} appears identically in both $\mathsf{aux}_{\mathsf{xor},l}$ and $\mathsf{aux}_{\mathsf{and},l}$ (see Figure 1), so it can be sent in plaintext, reducing the OT payload from 3κ to 2κ bits per gate. All $|f|$ gate OTs can be batched and run in parallel using standard OT extension [14], after a single setup phase of κ base OTs. This interaction is the only communication beyond the garbled circuit transmission. We note that this interaction does not break the desirable properties of garbled circuits: the garbling phase remains non-interactive, and the OT for SelectGate can be pipelined with the OT for input encoding.

3.2 Defining Oblivious Garbling Schemes

Definition 3 *An oblivious garbling scheme consists of five algorithms:*

1. $\mathsf{Garble}(1^\kappa, \Phi_g(f)) \to (F, e, d, \mathsf{aux}_g)$ *which takes in as input the security parameter κ, and a leakage function Φ_g for the functionality f and returns the garbled circuit F, encoding information e for the wire labels and the decoding information d for the output wire labels, and some auxiliary information* aux_g.
2. $\mathsf{aux_e} := \mathsf{SelectGate}(\mathsf{aux}_g, f)$ *which takes as input the auxiliary information computed by* Garble *and outputs the auxiliary information needed for evaluation for a given circuit f.*
3. $X := \mathsf{Encode}(e, x)$ *which takes in as input the encoding information e, input x and gives the Garbled input X.*
4. $Y := \mathsf{Eval}(F, X, \mathsf{aux}_e)$ *which takes in as input the garbled circuit F, the garbled input X, the auxiliary information* aux_e *computed by* SelectGate *for f and returns the garbled evaluation of f on x, as Y.*

5. $y := \mathsf{Decode}(d, Y)$ *which takes in as input the decoding information* d*, garbled output* Y *and gives the output* y*.*

We want the following properties from a garbling scheme:

Correctness: *For any circuit* f *and input* x*, while* $(F, e, d, \mathsf{aux_g}) \leftarrow \mathsf{Garble}(1^\kappa, \Phi_g(f))$*, we have* $f(x) = \mathsf{Decode}(d, \mathsf{Eval}(F, \mathsf{Encode}(e, x), \mathsf{SelectGate}(\mathsf{aux}_g, f)))$ *except for negligible probability.*

Privacy: *There is a simulator* Sim *such that for any circuit* f *with leakage* $\Phi_g(f)$ *with respect to* Garble *and for any input* x*, the following distributions are indistinguishable:*

$(F, e, d, \mathsf{aux}_g) \leftarrow \mathsf{Garble}(1^\kappa, \Phi_g(f))$
$X := \mathsf{Encode}(e, x)$
$\mathsf{aux}_e := \mathsf{SelectGate}(\mathsf{aux_g}, f)$
return $(F, X, d, \mathsf{aux}_e)$

$(F, X, d, \mathsf{aux}_e) \leftarrow \mathsf{Sim}(1^\kappa, f, f(x))$
return $(F, X, d, \mathsf{aux}_e)$

Obliviousness: *There is a simulator* Sim *such that for any circuit* f *with leakage* $\Phi_g(f)$ *with respect to* Garble *and for any input* x*, the following distributions are indistinguishable:*

$(F, e, d, \mathsf{aux}_g) \leftarrow \mathsf{Garble}(1^\kappa, \Phi_g(f))$
$X := \mathsf{Encode}(e, x)$
$\mathsf{aux}_e := \mathsf{SelectGate}(\mathsf{aux_g}, f)$
return (F, X, aux_e)

$(F, X, \mathsf{aux}_e) \leftarrow \mathsf{Sim}(1^\kappa, f)$
return (F, X, aux_e)

Authenticity: *For any circuit* f *and input* x*, no PPT adversary* Adv *can make the following distribution output* TRUE *except with negligible probability:*

$(F, e, d, \mathsf{aux}_g) \leftarrow \mathsf{Garble}(1^\kappa, \Phi_g(f))$
$X := \mathsf{Encode}(e, x)$
$\mathsf{aux}_e := \mathsf{SelectGate}(\mathsf{aux_g}, f)$
$Y \leftarrow \mathsf{Adv}(F, d, X, \mathsf{aux}_e)$
return $\mathsf{Decode}(d, Y) \notin \{f(x), \perp\}$

3.3 Obliviously Garbling an Arbitrary Circuit

Formalising the techniques mentioned in Section 3.1, Figure 1,2,3 describes an oblivious garbling scheme that uses the GTri gadget to instantiate every gate in any circuit comprised of 2-input AND and XOR gates.

Instantiating Gb3GADGET and Eval3GADGET To enable a concrete security proof, we instantiate Gb3GADGET and Eval3GADGET using the half-gates scheme of Zahur, Rosulek, and Evans [35], which achieves 2κ bits per AND gate and supports free-XOR. Since $\mathcal{G}_{\text{tri}}(x, y, z) = x \wedge (y \oplus z)$, the free-XOR label for the intermediate value $y \oplus z$ is $W_y^0 \oplus W_z^0$ at no cost, reducing Gb3GADGET to a single call to the half-gates AND procedure on inputs W_x^0 and $W_y^0 \oplus W_z^0$.

procedure Garble($1^\kappa, \Phi_g(f) = \mathsf{Topology(f)}$)**:**

- $\Delta :=_\$ \{0,1\}^{\kappa-1}||1$;
- **for** $i \in \mathsf{GadgetIn}(f)$ **do**
  - $W_i^0 :=_\$ \{0,1\}^\kappa$;
  - $W_i^1 := W_i^0 \oplus \Delta$;
  - **if** $i \in \mathsf{CircuitIn}(f)$ **then**
    - $e_i := W_i^0$;
- **for** $i \in \mathsf{GadgetOut}(f)\{in\ topo.order\}$ **do**
  - $\{a,b,c\} := \mathsf{GadgetInputs}(f,i)$;
  - $(W_i^0, \vec{G}_i, \vec{z}_i) := \mathsf{Gb3GADGET}(W_a^0, W_b^0, W_c^0)$;
  - $W_i^1 := W_i^0 \oplus \Delta$;
  - $F_i := (\vec{G}_i, \vec{z}_i)$;
- **for** $l \in \mathsf{GadgetOut}(f)$ **do**
  - $\{i,j,k\} := \mathsf{GadgetInputs}(f,l)$;
  - **if** $i,k \in \mathsf{CircuitIn}(f)$ **then**
    - continue;
  - **if** $j \in \mathsf{CircuitIn}(f)$ **then**
    - $\{a\} := \mathsf{IncomingWires}(f,i,j,k)$;
    - $\mathsf{aux}_{\mathsf{xor}} := \{W_i^1, W_j^0, W_a^0 \oplus W_k^0\}$;
    - $\mathsf{aux}_{\mathsf{and}} := \{W_a^0 \oplus W_i^0, W_j^0, W_k^0\}$;
    - $\mathsf{aux}_{g,l} := \{\mathsf{aux}_{\mathsf{xor}}, \mathsf{aux}_{\mathsf{and}}\}$
  - **if** $j \notin \mathsf{CircuitIn}(f)$ **then**
    - $\{a,b\} := \mathsf{IncomingWires}(f,i,j,k)$;
    - $\mathsf{aux}_{\mathsf{xor}} := \{W_i^1, W_b^0 \oplus W_j^0, W_a^0 \oplus W_k^0\}$;
    - $\mathsf{aux}_{\mathsf{and}} := \{W_a^0 \oplus W_i^0, W_b^0 \oplus W_j^0, W_k^0\}$;
    - $\mathsf{aux}_{g,l} := \{\mathsf{aux}_{\mathsf{xor}}, \mathsf{aux}_{\mathsf{and}}\}$
- **for** $i \in \mathsf{CircuitOut}(f)$ **do**
  - $D_i := \mathrm{lsb}(W_i^0)$;
- return $(\hat{F}, \hat{e}, \hat{D}, \hat{\mathsf{aux}}_g)$;

Figure 1: Our Oblivious Garbling scheme - Garbling

procedure Encode($\hat{e}, \hat{x}$)**:**

- **for** $e_i \in \hat{e}$ **do**
  - $X_i := e_i \oplus x_i \Delta$;
- return $\hat{X}$;

procedure Decode($\hat{D}, \hat{Y}$)**:**

- **for** $D_i \in \hat{D}$ **do**
  - $y_i := j \oplus \mathrm{lsb}(Y_i)$;
- return $\hat{y}$;

procedure SelectGate($\hat{\mathsf{aux}}_g, f$)**:**

- **for** $l \in \mathsf{GadgetOut}(f)$ **do**
  - $\mathsf{aux}_{g,l} := \{\mathsf{aux}_{\mathsf{xor}}, \mathsf{aux}_{\mathsf{and}}\}$;
  - **if** $l \in \mathsf{XORGates(f)}$ **then**
    - $\mathsf{aux}_{e,l} = \mathsf{aux}_{\mathsf{xor}}$;
  - **else**
    - $\mathsf{aux}_{e,l} = \mathsf{aux}_{\mathsf{and}}$;
- return $\hat{\mathsf{aux}}_e$;

Figure 2: Our Oblivious Garbling scheme - Gate and Input Encoding

procedure $\mathsf{Eval}(\hat{F}, \hat{X}, \hat{\mathsf{aux}}_e)$**:**
- **for** $i \in \mathsf{GadgetIn}(\hat{F})$ **do**
  - $W_i := X_i$;
- **for** $i \notin \mathsf{GadgetIn}(\hat{F})\{in\ topo.order\}$ **do**
  - $\{a, b, c\} := \mathsf{GadgetInputs}(\hat{F}, i)$;
  - **if** $(a, b, c) \in \mathsf{CircuitIn}(f)$ **then**
    - $E_i := \mathsf{Eval3GADGET}(F_i, W_a, W_b, W_c)$;
  - **else**
    - $\mathsf{aux}_{e,i} = \{\mathsf{aux}_a, \mathsf{aux}_b, \mathsf{aux}_c\}$;
    - **if** $i \in \mathsf{XORGates(f)}$ **then**
      - **if** $b \in \mathsf{CircuitIn}(f)$ **then**
        - $E_i := \mathsf{Eval3GADGET}(F_i, \mathsf{aux}_a, \mathsf{aux}_b, \mathsf{aux}_c \oplus W_c)$;
      - **else**
        - $E_i := \mathsf{Eval3GADGET}(F_i, \mathsf{aux}_a, \mathsf{aux}_b \oplus W_b, \mathsf{aux}_c \oplus W_c)$;
    - **else**
      - **if** $b \in \mathsf{CircuitIn}(f)$ **then**
        - $E_i := \mathsf{Eval3GADGET}(F_i, \mathsf{aux}_a \oplus W_a, \mathsf{aux}_b, \mathsf{aux}_c)$;
      - **else**
        - $E_i := \mathsf{Eval3GADGET}(F_i, \mathsf{aux}_a \oplus W_a, \mathsf{aux}_b \oplus W_b, \mathsf{aux}_c)$;
- Return $\hat{E}$;

Figure 3: Our Oblivious Garbling scheme – Evaluation

private procedure $\mathsf{GbAnd}(W_a^0, W_b^0)$**:**
- $p_a \leftarrow \mathrm{lsb}(W_a^0)$; $p_b \leftarrow \mathrm{lsb}(W_b^0)$; $j \leftarrow \mathsf{NextIndex}()$; $j' \leftarrow \mathsf{NextIndex}()$;
- `// First half gate`
- $T_G \leftarrow H(W_a^0, j) \oplus H(W_a^0 \oplus \Delta, j) \oplus p_b \cdot \Delta$;
- $W_G^0 \leftarrow H(W_a^0, j) \oplus p_a \cdot T_G$;
- `// Second half gate`
- $T_E \leftarrow H(W_b^0, j') \oplus H(W_b^0 \oplus \Delta, j') \oplus W_a^0$;
- $W_E^0 \leftarrow H(W_b^0, j') \oplus p_b \cdot (T_E \oplus W_a^0)$;
- **return** $(W_G^0 \oplus W_E^0,\ T_G,\ T_E)$;

procedure $\mathsf{Gb3GADGET}(W_x^0, W_y^0, W_z^0)$**:**
- $W_m^0 \leftarrow W_y^0 \oplus W_z^0$;
- **return** $\mathsf{GbAnd}(W_x^0,\ W_m^0)$;

Figure 4: Concrete instantiation of Gb3GADGET using half-gates [35]

The global offset $\Delta \in \{0,1\}^{\kappa-1}\|1$ is shared with the outer garbling procedure (Figure 1).The hash function $H : \{0,1\}^\kappa \times \mathbb{N} \to \{0,1\}^\kappa$ is assumed to be circular correlation robust (Definition 2). The evaluator procedure Eval3GADGET correspondingly computes $W_m \leftarrow W_y \oplus W_z$ for free and then runs the standard half-gates evaluation on (T_G, T_E, W_x, W_m), recovering the active output label.

We stress that the half-gates instantiation is chosen for clarity of exposition and concreteness of the security proof. Any garbling scheme that supports free-XOR and satisfies the privacy and authenticity properties of Definition 1 can be used as a drop-in replacement. In particular, the three-halves scheme [26] reduces the garbled gate size from 2κ to 1.5κ bits with no change to the outer oblivious garbling construction; we use this in the efficiency comparison of Section 3.5.

Theorem 1 *Let H be a circular correlation robust hash function (Definition 2). Instantiating* $(\mathsf{Gb3GADGET}, \mathsf{Eval3GADGET})$ *with the half-gates scheme of Figure 4, the garbling scheme described in Figures 1, 2, 3 meets Definition 3 for any circuit with XOR and AND gates, with a garbled gate size of 2κ bits.*

Proof Sketch. We prove all the properties of Definition 3 below.

Correctness. Follows directly from the correctness of the half-gates instantiation of $\mathsf{Gb3GADGET}$ [35] and the correctness of the soldering technique [24]: given the active label P_{b_1} on wire p, the soldering value $S = W_a^0 \oplus P_a^0$ enables the evaluator to compute $P_{b_1} \oplus S = W_a^0 \oplus b_1\Delta$, the active label on the target wire a, by the free-XOR structure.

Privacy. We build a simulator that proceeds as follows.

- For each $\mathcal{G}_{\text{tri}}$ gadget, the simulator samples fresh uniform labels for each input wire and produces a simulated garbled gate using the SimAND procedure of the half-gates scheme [35], which operates under a random oracle $\mathcal{O} = \mathsf{Rand}$.
- For each wire connection, the simulator computes the soldering value as the XOR of the simulated active output label of the producing gadget and the simulated active input label of the consuming gadget.
- The simulator sets aux_e according to the gate type in f, selecting either $\mathsf{aux}_{\mathsf{xor}}$ or $\mathsf{aux}_{\mathsf{and}}$ for each gate consistently with f.
- The output labels are computed from the simulated active labels, using $f(x)$ to determine which labels are active at the output wires.

We argue indistinguishability via the following hybrid sequence, proceeding from the simulator to the real execution. The hybrid structure follows that of the half-gates proof [35], extended with the soldering values.

Hybrid $\mathcal{G}_1$ (simulator): For each gadget, sample fresh uniform labels W_i^0 and set $W_i^1 = W_i^0 \oplus \Delta$ for a uniformly random $\Delta \in \{0,1\}^{\kappa-1}\|1$. Produce garbled gates via SimAND calls under $\mathcal{O} = \mathsf{Rand}$. Soldering values are computed as XOR offsets between simulated active labels, and $\widehat{\mathsf{aux}}_e$ is set according to f.

Hybrid $\mathcal{G}_2$ ($\mathcal{G}_1 \approx \mathcal{G}_2$): Change the oracle $\mathcal{O}$ from Rand to Circ_R. By the CCR assumption on H (Definition 2), these two hybrids are computationally indistinguishable.

Hybrid $\mathcal{G}_3$ ($\mathcal{G}_2 \equiv \mathcal{G}_3$): Introduce the variable R from Circ_R explicitly. The soldering values, which were previously computed as XOR offsets between independently sampled simulated active labels, are now expressed equivalently in terms of the actual 0-labels of adjacent wires and R. This is a purely algebraic rewriting: the distribution of all output variables is unchanged, since the 0-labels are uniformly random and the relationships between them are preserved.

Hybrid $\mathcal{G}_4$ ($\mathcal{G}_3 \equiv \mathcal{G}_4$, real execution): Replace SimAND calls with the real GbAnd procedure under $\mathcal{O} = \mathsf{Circ}_R$. As shown in [35], this induces identical distributions on all variables $(W_i^0, W_i^1, T_{G_i}, T_{E_i})$. The soldering values, being XOR offsets between 0-labels, remain consistent with the free-XOR structure and are identically distributed to the real execution. This hybrid is therefore the real execution of Figure 1.

We note that the above argument is not specific to half-gates. Any instantiation of $(\mathsf{Gb3GADGET}, \mathsf{Eval3GADGET})$ satisfying the privacy and authenticity of Definition 1 admits an analogous hybrid argument, replacing SimAND with the corresponding simulator of the underlying scheme.

Obliviousness. Obliviousness follows by modifying the privacy simulator to not use $f(x)$: the output labels $\hat{d}$ are omitted, and $\hat{X}$ is encoded using simulated labels without conditioning on the output value. Since the garbled gadgets are simulated independently and the soldering values depend only on the active labels and not on $f(x)$, the simulator requires no knowledge of the output. The resulting distribution is computationally indistinguishable from the real execution by the same CCR argument.

Authenticity. Suppose a PPT adversary $\mathcal{A}$ receives $(\hat{F}, \hat{X}, \hat{D}, \mathsf{aux}_e)$ and produces Y such that $\mathsf{Decode}(\hat{D}, Y) \notin \{f(x), \perp\}$. Any such forgery must correspond to an inactive output label on some $\mathcal{G}_{\mathrm{tri}}$ gadget reaching the output. Since soldering values in aux_e are applied as XOR offsets, they preserve the semantic bit carried by a label: a forged output label must therefore originate from a forged active label on some gadget. This contradicts the authenticity of the half-gates scheme [35] under the CCR assumption. Hence the probability of $\mathcal{A}$ succeeding is negligible.

This concludes the proof. □

Remark 1 *The theorem holds for any* $(\mathsf{Gb3GADGET}, \mathsf{Eval3GADGET})$ *satisfying the privacy and authenticity of Definition 1, yielding a garbled gate size of* $|G|$ *bits. In particular, instantiating with three-halves [26] reduces the gate size to* 1.5κ *bits with no change to the outer construction; see Section 3.5.*

3.4 Extension to Malicious Security

Our construction extends naturally to the malicious setting. We instantiate the underlying gate garbling with authenticated half-gates [16, 30]. When used in a two-party computation protocol, all OTs, including those used for $\mathsf{SelectGate}$,

can be replaced with their maliciously-secure counterparts. The two components requiring authentication against a malicious garbler are the garbled $\mathcal{G}_{\text{tri}}$ gadgets and the soldering values; we address each in turn.

Authenticated gadgets. Following [16, 30], each wire label carries an IT-MAC binding it to the garbler's global key Δ_G, so that any tampered label fails verification immediately. Authenticated Gb3GADGET proceeds in two steps. First, the label for the intermediate wire $m = y \oplus z$ is computed via free XOR of authenticated shares, a local operation requiring no interaction. Second, the AND gate on (x, m) is evaluated using a standard authenticated AND gate with a preprocessing triple, following [16,30]. The selective abort problem is resolved as in [16]: the values revealed during the AND evaluation are masked by preprocessing randomness unknown to the garbler, making any selective abort independent of the gate inputs. Each authenticated $\mathcal{G}_{\text{tri}}$ gadget produces a garbled gate of size $2(\kappa + 1)$ bits, one per authenticated AND.

Authenticated soldering. A soldering value $S^{\mathsf{label}} = W_a^0 \oplus W_b^0$ translates the active label on wire a to the corresponding label on wire b by XOR. Under the IT-MAC structure, an incorrect soldering value produces a translated label whose MAC fails verification under Δ_G, causing the evaluator to abort immediately. To enable MAC verification after translation, the garbler includes alongside the label offset a MAC correction term $S^{\mathsf{mac}} = K_G[a, 0] \oplus K_G[b, 0]$. Given the active label $W_a^{b_1}$ with MAC $M_G[a, b_1]$, the evaluator computes $W_b^{b_1} = W_a^{b_1} \oplus S^{\mathsf{label}}$ and verifies using $M_G[b, b_1] = M_G[a, b_1] \oplus S^{\mathsf{mac}}$, which follows from the linearity of the MAC structure.

Complexity. Authenticated garbling already requires $O(|C|)$ preprocessing OTs for its multiplication triples [16, 30]. The SelectGate OTs are of identical type and batch into the same OT extension call [7, 14, 27, 33] at no asymptotic overhead. The resulting scheme preserves the $O(|C|)$ communication and computation complexity of the semi-honest construction.

3.5 Concrete Efficiency

We compare two constructions of oblivious garbling against the only previously known method for garbler-side function privacy, namely universal circuits. Construction A is our scheme based on the $\mathcal{G}_{\text{tri}}$ gadget. Construction B is an alternative instantiation of Definition 3, described below. Both are new and neither has appeared prior to this work.

Construction B. An alternative instantiation of Definition 3 introduces a selector wire $c \in \{0, 1\}$ encoding the gate type ($c = 0$ for XOR, $c = 1$ for AND) and replaces each gate with the subcircuit $f_{\text{sel}}(a, b, c) = c \wedge (a \wedge b) \oplus (1 \oplus c) \wedge (a \oplus b)$. Expanding over GF(2) gives $a \oplus b \oplus c \wedge (a \oplus b \oplus (a \wedge b))$, which with free-XOR requires exactly two AND gates per original gate.

	Univ. Circuits	**Construction A**	**Construction B**
Semi-honest (three-halves [26])			
GC size (bits/gate)	$4.5\kappa \log_2 n$	$\mathbf{1.5}\kappa$	3κ
AND gates / gate	$3 \log_2 n$	**1**	2
Malicious (authenticated half-gates [16])			
GC size (bits/gate)	$6(\kappa{+}1) \log_2 n$	$\mathbf{2}(\kappa{+}\mathbf{1})$	$4(\kappa{+}1)$
AND gates / gate	$3 \log_2 n$	**1**	2
Garbler learns topology	No	Yes	Yes

Table 1. Concrete efficiency per original gate. $n = |f|$ denotes the number of gates in the original circuit. The $\log_2 n$ factor for universal circuits reflects the $3n \log_2 n$ gate blowup of the best known UC construction [21]. Construction B is an alternative instantiation of Definition 3. The $2\times$ advantage of Construction A over Construction B is preserved across both security models.

Efficiency comparison. In the semi-honest setting we instantiate with three-halves [26], achieving 1.5κ bits per AND gate. Construction A garbles exactly one $\mathcal{G}_{\text{tri}}$ gadget per original gate, reducing to a single AND gate (Section 3.3), giving a garbled circuit size of 1.5κ bits per gate. Construction B requires two AND gates per gate at 3κ bits per gate. Universal circuits require $3n \log_2 n$ gates [21], giving $4.5\kappa \log_2 n$ bits per original gate. In the malicious setting we instantiate with authenticated half-gates [16, 30], achieving $2(\kappa + 1)$ bits per AND gate. Construction A gives $2(\kappa+1)$ bits per gate and Construction B gives $4(\kappa+1)$ bits per gate. The $2\times$ advantage of Construction A over Construction B is preserved exactly across both security models. In CPU-bound implementations where AND-gate evaluation is the bottleneck, Construction A also achieves a $2\times$ reduction in garbling computation compared to Construction B, and a $3 \log_2 n$ reduction compared to universal circuits, in both settings. The full comparison is summarised in Table 1.

On the cost of SelectGate. The SelectGate interaction requires one OT per non-input gate, with the function owner acting as receiver on the gate-type bit and the garbler as sender on the two soldering pairs. This is structurally identical to the OT already required for private input encoding: gate types are simply additional private inputs held by the function owner, and an information-theoretic argument shows that any scheme hiding gate types from the garbler while enabling correct evaluation must incur equivalent work. In the semi-honest setting, all SelectGate OTs batch with the input encoding OTs in a single round via OT extension [7, 14, 27, 33]. In the malicious setting, authenticated garbling already requires $O(|C|)$ preprocessing OTs, into which the SelectGate OTs batch at no asymptotic overhead. We do not attribute this cost to either construction: it is the inherent price of function privacy.

Acknowledgements Carmit Hazay was supported by ISF grant No. 607/25. Rahul Satish was supported by the Independent Research Fund Denmark (IRFD) grant number 0165-00079B.

References

1. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: SCALES - MPC with small clients and larger ephemeral servers. In: TCC. pp. 502–531 (2022). https://doi.org/10.1007/978-3-031-22365-5_18
2. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: Malicious security for SCALES - outsourced computation with ephemeral servers. In: CRYPTO. pp. 3–38 (2024). https://doi.org/10.1007/978-3-031-68400-5_1
3. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: Towards building efficient SCALES protocols. In: ASIACRYPT. pp. 35–67 (2025). https://doi.org/10.1007/978-981-95-5116-3_2
4. Applebaum, B., Ishai, Y., Kushilevitz, E.: From secrecy to soundness: Efficient verification via secure computation. In: ICALP. pp. 152–163 (2010). https://doi.org/10.1007/978-3-642-14165-2_14
5. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: ASIACRYPT. pp. 134–153 (2012). https://doi.org/10.1007/978-3-642-34961-4_10
6. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: ACM CCS. pp. 784–796 (2012). https://doi.org/10.1145/2382196.2382279
7. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO. pp. 489–518 (2019). https://doi.org/10.1007/978-3-030-26954-8_16
8. Choi, S.G., Katz, J., Kumaresan, R., Cid, C.: Multi-client non-interactive verifiable computation. In: TCC. pp. 499–518 (2013). https://doi.org/10.1007/978-3-642-36594-2_28
9. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: Minilego: Efficient secure two-party computation from general assumptions. In: EUROCRYPT. pp. 537–556 (2013). https://doi.org/10.1007/978-3-642-38348-9_32
10. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: CRYPTO. pp. 465–482 (2010). https://doi.org/10.1007/978-3-642-14623-7_25
11. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: One-time programs. In: CRYPTO. pp. 39–56 (2008). https://doi.org/10.1007/978-3-540-85174-5_3
12. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Functional encryption with bounded collusions via multi-party computation. In: CRYPTO. pp. 162–179 (2012). https://doi.org/10.1007/978-3-642-32009-5_11
13. Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: TCC. pp. 308–326 (2010). https://doi.org/10.1007/978-3-642-11799-2_19
14. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO. pp. 145–161 (2003). https://doi.org/10.1007/978-3-540-45146-4_9
15. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. IACR Cryptol. ePrint Arch. p. 272 (2011), http://eprint.iacr.org/2011/272
16. Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing authenticated garbling for faster secure two-party computation. In: CRYPTO. pp. 365–391 (2018). https://doi.org/10.1007/978-3-319-96878-0_13
17. Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. In: EUROCRYPT. pp. 699–728 (2016). https://doi.org/10.1007/978-3-662-49890-3_27
18. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: ICALP. pp. 486–498 (2008). https://doi.org/10.1007/978-3-540-70583-3_40

19. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC. pp. 83–97 (2008). https://doi.org/10.1007/978-3-540-85230-8_7
20. Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant's universal circuit: Improvements, implementation, and applications. IACR Cryptol. ePrint Arch. p. 17 (2016), http://eprint.iacr.org/2016/017
21. Liu, H., Yu, Y., Zhao, S., Zhang, J., Liu, W., Hu, Z.: Pushing the limits of valiant's universal circuits: Simpler, tighter and more compact. In: CRYPTO. pp. 365–394 (2021). https://doi.org/10.1007/978-3-030-84245-1_13
22. Mohassel, P., Sadeghian, S.S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: EUROCRYPT. vol. 7881, pp. 557–574 (2013). https://doi.org/10.1007/978-3-642-38348-9_33
23. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: ACM-EC. pp. 129–139 (1999). https://doi.org/10.1145/336992.337028
24. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: TCC. pp. 368–386 (2009). https://doi.org/10.1007/978-3-642-00457-5_22
25. Paus, A., Sadeghi, A., Schneider, T.: Practical secure evaluation of semi-private functions. In: ACNS. pp. 89–106 (2009). https://doi.org/10.1007/978-3-642-01957-9_6
26. Rosulek, M., Roy, L.: Three halves make a whole? beating the half-gates lower bound for garbled circuits. In: CRYPTO. pp. 94–124 (2021). https://doi.org/10.1007/978-3-030-84242-0_5
27. Roy, L.: Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In: CRYPTO. pp. 657–687 (2022). https://doi.org/10.1007/978-3-031-15802-5_23
28. Sahai, A., Seyalioglu, H.: Worry-free encryption: functional encryption with public keys. In: ACM CCS. pp. 463–472 (2010). https://doi.org/10.1145/1866307.1866359
29. Valiant, L.G.: Universal circuits (preliminary report). In: ACM STOC. pp. 196–203 (1976). https://doi.org/10.1145/800113.803649
30. Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: CCS. pp. 21–37 (2017). https://doi.org/10.1145/3133956.3134053
31. Wang, Y., Malluhi, Q.M.: Reducing garbled circuit size while preserving circuit gate privacy. Cryptology ePrint Archive, Paper 2017/041 (2017), http://eprint.iacr.org/2017/041
32. Wegener, I.: The complexity of Boolean functions. Wiley-Teubner (1987), http://ls2-www.cs.uni-dortmund.de/monographs/bluebook/
33. Yang, K., Weng, C., Lan, X., Zhang, J., Wang, X.: Ferret: Fast extension for correlated OT with small communication. In: CCS. pp. 1607–1626 (2020). https://doi.org/10.1145/3372297.3417276
34. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: FoCS. pp. 162–167 (1986). https://doi.org/10.1109/SFCS.1986.25
35. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: EUROCRYPT. pp. 220–250 (2015). https://doi.org/10.1007/978-3-662-46803-6_8