Efficient Garbled Pseudorandom Functions and Lookup Tables from Minimal Assumption

Wei-Kai Lin
University of Virginia

Zhenghao Lu
Shanghai Jiao Tong University

Hong-Sheng Zhou
Virginia Commonwealth University

November 30, 2025

Abstract

Yao's garbled circuits have received huge attention in both theory and practice. While garbled circuits can be constructed using minimal assumption (i.e., the existence of pseudorandom functions or one-way functions), the state-of-the-art constructions (e.g., Rosulek-Roy, Crypto 2021) are based on stronger assumptions. In particular, the "Free-XOR" technique (Kolesnikov-Schneider, ICALP 2008) is essential in these state-of-the-art constructions, and their security can only be proven in the random oracle model, or rely on the "circular-correlation robust hash" assumption.

In this paper, we aim to develop new techniques to construct efficient garbling schemes using minimal assumptions. Instead of generically replacing the Free-XOR technique, we focus on garbling schemes for specific functionalities. We successfully eliminated the need for Free-XOR in several state-of-the-art schemes, including the one-hot garbling (Heath and Kolesnikov, CCS 2021) and the garbled pseudorandom functions, and the garbled lookup tables (Heath, Kolesnikov and Ng, Eurocrypt 2024). Our schemes are based on minimal assumptions, i.e., standard pseudorandom functions (PRFs)—we resolved the need for circular security. The performance of our scheme is almost as efficient as the best results except for a small constant factor. Namely, for any lookup table $\{0,1\}^n \to \{0,1\}^m$, our scheme takes $n + (5n+9)m\lambda + 2^n \cdot m$ bits of communication, where λ is the security parameter of PRF.

Contents

1 Introduction

Since first being introduced in the 1980s [Yao86], Yao's garbled circuits have undergone extensive research in both theory and practice. Garbled circuits have been used for constructing constant-round secure multiparty computation [BMR90, LP09, BLO16] and many primitives (e.g., [GGP10, BHHI10, JKO13, FNO15, GKP+13], to name a few). Non-trivial new abstractions have been developed: Applebaum, Ishai and Kushilevitz [IK00, AIK04, App17] view garbled circuits as *randomized encodings* of functions; later, Bellare, Hoang, and Rogaway [BHR12b] formalize garbled circuits as a standalone cryptographic primitive, called *garbling schemes*. In this paper, we will follow Bellare et al's formalization and use garbled circuits and garbling schemes exchangeably.

Garbling with short ciphertext. Over the past decades, immense research efforts have been made to reduce the concrete size of garbled circuits [BMR90, NPS99, KS08, PSSW09, KMR14, GLNP15, ZRE15, RR21, AAC+23]. To achieve lower communication complexity in garbling, multiple novel techniques have been introduced, including row reduction technique [NPS99, PSSW09], pointer-and-permute technique [BMR90], free-XOR technique [KS08], and many more [GLNP15, ZRE15, RR21, HK21]. Elaborately, a garbling scheme consists of a *Garbler* and an *Evaluator*; the Garbler "encrypts" an input x and a function f into X and F correspondingly, and then the Evaluator can compute on (X, F) to learn $f(x)$ but no additional information about (x, f).[1] The major efficiency metric is the communication cost of the garbled functionality, i.e., $|F|$ (since the garbled input X is typically shorter).

Boolean circuits and Free-XOR. Since Yao's seminal work [Yao86], boolean circuits are the "mainstream" model of garbling schemes. In this approach, we consider any f that is computable by a (poly-size) boolean circuit, which is composed of constant-sized *universal boolean gates*, such as $\{\mathsf{AND}, \mathsf{XOR}\}$. We devise a garbling scheme for the universal gates, and then we compose the garbled gates to obtain the garbled circuit F, analogous to the composition of f. In this approach, the communication cost is proportional to the number of boolean gates of f and proportional to a *per-gate* cost, which depends on a computational security parameter λ. The state-of-the-art garbled boolean circuits is devised by Rosulek and Roy [RR21], it is based on the "Free-XOR" technique [KS08], and it takes only 1.5λ bits per AND, and it takes no communication for XOR gates!

The per-gate garbling is fast for boolean circuits, however, it is limited by the per-gate cost. Even with the state-of-the-art, the total cost is still $O(|f| \cdot \lambda)$ when we garble a function f that takes $|f|$ boolean gates. Also, there are lower bounds on the per-gate cost, showing that $\Omega(\lambda)$ bits are necessary [ZRE15, AHS24, FLZ24, Kim24, BK24, XHX24, LGW24].[2]

Garbled lookup tables. To achieve an overall cost that is lower than the garbled boolean circuit, many garbling schemes are proposed for functionalities in other computation models [AIK11, LO13, HK20, HK21, HKO23, PLS23, HKN24]. In this work, we focus on the *lookup tables* (LUT for short). That is, we consider functions $f : \{0,1\}^n \to \{0,1\}^m$, described by the truth table of $2^n \cdot m$ bits. The plaintext computation $f(x)$ is simply looking up the xth row in the table. In a garbling scenario, we assume that the table size is polynomial in the security parameter λ, that is, $n = O(\log \lambda)$ and $m = \mathrm{poly}(\lambda)$. To garble a lookup table, a naive approach is to hardwire the truth table into a boolean circuit and then garble the circuit (using a per-gate garbling). The cost would be $O(2^n \cdot m \cdot \lambda)$, incurring an $O(\lambda)$ overhead compared to the description length $|f|$.

Since lookup tables are widely used, garbled LUTs are applicable in many scenarios, e.g., secure machine learning with high numerical precision [RRG+21]. Note that in some applications (e.g., [RRG+21]), the function f is public and pre-determined (while only the input x is secret), but garbling the LUT remains non-trivial. Moreover in such applications, we often want to compose the result of a lookup into subsequent computation, e.g., another boolean circuit.

[1] Certain leakage of f exists, depending on the scenarios and constructions.

[2] The lower bounds assume a restricted "linear garbling" framework [ZRE15] or its extensions [AHS24, FLZ24, Kim24, BK24, XHX24, LGW24], which model almost all natural schemes.

The recent work of Heath, Kolesnikov and Ng [HKN24] proposes a more efficient garbled LUT, which takes only $(n-1+nm)\lambda + 2^n \cdot m$ bits! Especially when n is large, that is faster than the naive approach by almost a λ factor, which could be a more than 100x speedup in practice. Also, their scheme is composable with subsequent boolean circuits, as wanted.

Garbling based on minimal assumption. It is always desirable to base cryptographic schemes on weaker assumptions. In the line of garbling schemes, Lindell and Pinkas [LP09] proved that Yao's garbled circuits can be based on the standard CPA-secure symmetric key encryption, and then Gueron et al. [GLNP15] developed a scheme that takes 2λ bits per AND and λ bits per XOR and is based on pseudorandom functions (PRFs). Hence, efficient garbled boolean circuits can be based on the minimal assumption, the existence of one-way functions [HILL99].

However, the results of garbled LUTs [HKN24] and the state-of-the-art garbled boolean circuits [RR21] relied on the "Free-XOR" technique, which requires stronger assumptions. Free-XOR is proposed by Kolesnikov and Schneider [KS08]. They assumed that there is a random oracle $H(\cdot)$ that is accessible by both Garbler and Evaluator. Roughly speaking, Garbler samples a global λ-bit secret key Δ, and for two wires (a, b), Garbler samples two λ-bit wire keys (A, B) and garbles (a, b) into input labels $A \oplus a\Delta$ and $B \oplus b\Delta$. The garbled XOR is evaluated by performing $(A \oplus a\Delta) \oplus (B \oplus b\Delta) = (A \oplus B) \oplus (a \oplus b)\Delta$, which is indeed a garbling of the wire $a \oplus b$, and it takes *no communication*. The garbled AND is more involved, but all recent schemes use H and the global Δ as follows: Garbler encrypts some ciphertext using the random oracle H and then sends the ciphertext to Evaluator, who decrypts the ciphertext using H and the input labels and then obtains the desired output label, such as $C \oplus (a \wedge b)\Delta$ for some C. The Free-XOR incurs many challenges when we want to replace the random oracle with weaker assumptions. Specifically, as in the garbled AND, the global Δ is used as part of the secret key meanwhile being part of the message in the same ciphertext—that requires *circularly secure cryptographic primitives* had we removed the random oracle.

Indeed, there are two approaches substituted the random oracle (RO) in the Free-XOR. Choi et al. [CKKZ12] proved that Kolesnikov-Schneider's scheme is secure without using RO. Instead, their scheme is based on a new assumption, called "Circular-Correlation Robust Hash" (CCRH). As suggested by its name, CCRH is by definition circular, and it is still unclear if CCRH can be based on other weaker and non-circular assumptions. The other approach is initiated by Applebaum [App13], which replaced RO with a special symmetric key encryption that is secure under a combined form of "related-key" and "key-dependent message" attacks. Then, Applebaum proved that such special encryption can be constructed, using the "learning parity with noise (LPN)" assumption. This approach is subsequently extended [BDH14] to be based on other computational assumptions, including LWE and DDH.

Notice that the cost is different when the schemes are based on different assumptions. For boolean gates, it takes *a small constant factor* in cost to replace the assumption of RO with PRF.[3] However, looking at garbled LUTs, there is a huge gap in efficiency when the scheme is based on weaker assumptions. Eliminating RO from garbled LUTs increases the cost by roughtly λ times. Of course, as proved in [HKN24], RO can be replaced with CCRH without increasing cost. That is unsatisfactory because CCRH is a circular and stronger assumption. Alternatively via Applebaum's approach, we might be able to base garbled LUTs on LPN (or other assumptions), but still, that would be far from minimal assumptions.

In this work, we aim for a garbling scheme of lookup tables that is as efficient as the RO-based scheme, up to a small constant factor. We raise the following question:

Is it possible to develop an efficient garbling scheme for lookup tables, based on minimal assumptions?

1.1 Our Results

We answer the above question affirmatively.

[3] Strictly, it depends on the ratio of XOR gates in the circuit. For circuits with 50% XOR, the factor is 2.

Theorem 1.1 (Garbled LUTs based on PRF—informal)**.** *Assume the existence of pseudorandom functions (PRFs). There exists a garbling scheme such that for any $n = O(\log \lambda)$ and $m = \mathrm{poly}(\lambda)$, for any function $f : \{0,1\}^n \to \{0,1\}^m$, the communication of the garbled function f takes $n + (5n+9)m\lambda + 2^n \cdot m$ bits, where λ is the PRF security parameter.*

Compared to the previous result, i.e., $(n - 1 + nm)\lambda + 2^n \cdot m$ bits [HKN24], we eliminated the need for RO nor CCRH, at the cost of a small constant. Observe that when 2^n dominates λ, our scheme only increases the cost by a $1 + o(1)$ factor, compared to [HKN24]. Our scheme is composable—the input and output wire labels are in the same format as in garbled boolean circuits, such as [GLNP15] (so that they can be connected to the precedent or the subsequent gates). In our theorem, λ is exactly the key-length of the PRF, and the cost is exact (rather than asymptotic). Since the PRF can be instantiated with block ciphers, our scheme is practical. Theoretically, our scheme can be based on the minimal assumption, the existence of one-way functions (which is impractical).

Our construction followed the beautiful framework of Heath, Kolesnikov, and Ng [HKN24]. Indeed, in hindsight, our scheme can be obtained by intricately modifying their scheme. However, at first glance, their scheme and proof relied heavily on circular security (of CCRH or RO), and it was totally unclear whether the circularity could be eliminated. Hence, a conceptual yet important contribution of this work is a *novel approach* to eliminating circularity in Free-XOR-based schemes (see Section 2.5).

The garbled LUTs of [HKN24] consist of two major building blocks, called one-hot garbling and garbled PRF.[4] While we constructed both building blocks, we highlight garbled PRF here.

Theorem 1.2 (Garbled PRF, informal.)**.** *Assume the existence of pseudorandom functions (PRFs). There exists a garbled PRF scheme such that for any $n = O(\log \lambda)$ and $m = \mathrm{poly}(\lambda)$, the scheme samples the seed s and garbles the seeded function $r_s : \{0,1\}^n \to \{0,1\}^m$ into a garbled functionality F. Moreover, the seed $|s|$ takes $m(1 + 2n\lambda)$ bits, and $|F|$ takes $n + (3n+4)m\lambda$ bits, where λ is the security parameter of the underlying PRF.*

Roughly speaking, the security of a garbled PRF aims to hide the seed s from Evaluator, yet Evaluator can compute the garbled labels of $r_s(x)$. A naive way is to garble (the boolean circuit of) a PRF using garbled boolean circuit, which uses the (circuit of) PRF in a non-black-box way and incurs a huge overhead, $\mathrm{poly}(\lambda)$. Heath, Kolesnikov, and Ng [HKN24] constructed an efficient garbled PRF based on CCRH (or RO). Our scheme achieves the same efficiency, up to a small factor, but is based on standard and black-box PRFs. We believe garbled PRFs are of independent interest. For example, we can obtain an oblivious PRF [FIPR05] by composing a standard 1-out-of-2 oblivious transfer with a garbled PRF.

Concurrent work. Liu, Liu, and Peng [LLP25] independently proposed a garbled lookup table with communication only linear in n, the input size of the lookup table. That is, their result eliminates the additive term 2^n in the work of Heath, Kolesnikov, and Ng [HKN24]. Their result is based on the decisional composite residuosity (DCR) assumption, and the lookup table is public in their setting. In comparison, our result minimizes the assumption (i.e., using PRF) while maintaining the same communication cost, whereas the concurrent work reduces the communication cost through stronger assumptions. It is open whether we can get the best of "both worlds," that is, achieving $\ll 2^n$ communication meanwhile basing on weaker assumptions, either RO or OWFs.

Due to space constraints, we defer additional related works to Section 5.

2 Technical Overview

Our goal is to garble a lookup table (LUT), denoted by a function $f : \{0,1\}^n \to \{0,1\}^m$. We will follow the framework of Heath, Kolesnikov, and Ng [HKN24], which involves three main steps: *(i)* construct a one-hot garbling scheme (Section 2.2), *(ii)* construct a garbled pseudorandom function (PRF) (Section 2.3), and *(iii)* obtain the garbled lookup table by composing the one-hot and PRF (Section 2.1).

[4] Garbled PRF is called "evaluating random function" [HKN24, Section 4.2].

We will use standard pseudorandom generators (PRGs) and pseudorandom functions (PRFs), which rely only on the existence of one-way functions [HILL99]. For readability, we will focus on garbling binary functions, $f : \{0,1\}^n \to \{0,1\}$, and we will show it takes $O(n\lambda + 2^n)$ bits of communication. To garble m-bit-output functions, we concatenate m instances, taking $O(nm\lambda + 2^n m)$-bit communication.

In Section 2.1, we will revisit and use the strategy of [HKN24] as well as recap the abstractions of one-hot garbling and garbled PRF. In Sections 2.2 and 2.3, we will construct one-hot garbling and garbled PRF and then put them together into LUT in Section 2.4 While Section 2.1 summarizes [HKN24], our constructions in Sections 2.2 and 2.3 are novel. In Section 2.5, we will compare with the previous schemes and discuss how we eliminate circular security.

Notation. We use $\lambda \in \mathbb{N}$ to represent the computational security parameter. For $n \in \mathbb{N}$, the set $\{0, 1, \ldots, n-1\}$ is denoted by $[n]$. Vectors and equivalently arrays are often denoted by bold letters with their elements in parenthesis, e.g., $\mathbf{x} = (x_0, \ldots, x_n) = (x_i)_{i\in[n]}$. Elements are also indexed from 0 using square brackets, i.e., $\mathbf{x}[i] = x_i$. For an array $\mathbf{x}$, we use $\mathbf{x}[i:j]$ to denote the sub-array $(\mathbf{x}[i], \ldots, \mathbf{x}[j])$. The inner product of two arrays $\mathbf{x}$ and $\mathbf{y}$ of equal length can be represented as $\langle \mathbf{x}, \mathbf{y} \rangle$ (where arithmetica is modulo reduce 2). We use $\|$ to denote the concatenation of two vectors or strings. For any n-bit string $\mathbf{x} \in \{0,1\}^n$, we sometimes abuse $\mathbf{x}$ to denote the integer in $[2^n]$ using the standard binary representation. For a finite set S, $x \leftarrow S$ denotes that the random variable x is sampled uniformly at random from S. For possibly randomized algorithm A, $x \leftarrow A()$ or $x := A()$ denotes that the variable x is assigned to the output of A.

We denote the garbler and the evaluator as Garbler and Evaluator. We often number input wires by $i \in [n]$ and denote the two garbled labels as (k_i^0, k_i^1), where the superscripts represent actual bits. That is, the actual value 0 is garbled by label k_i^0, while the label k_i^1 garbles 1. We refer to k_i^0 and k_i^1 as the 0-label and 1-label for wire i, respectively. Both labels are independently and randomly sampled (except for one-hot garbling). For readability, we omit permute / color bits of wire labels in this overview.

2.1 Revisiting Garbled Lookup Tables [HKN24]

One-hot garbling. We begin with recalling one-hot garbling. For any $n \in \mathbb{N}$, one-hot encoding is a mapping $\mathcal{H} : \{0,1\}^n \to \{0,1\}^{2^n}$ such that for any input $\mathbf{x} \in \{0,1\}^n$, $\mathcal{H}(\mathbf{x})$ is defined to be the vector

$$\mathcal{H}(\mathbf{x}) := (\underbrace{0, \cdots, 0}_{\mathbf{x}}, 1, \underbrace{0, \cdots, 0}_{2^n - \mathbf{x} - 1}).$$

(We abuse $\mathbf{x}$ to denote both the string and the integer represented by $\mathbf{x}$.) The one-hot *garbling* is a scheme that garbles $\mathcal{H}(\cdot)$. A beautiful observation of Heath and Kolesnikov [HK21] is that, we only need $2^n + 1$ labels to encode the vector $\mathcal{H}(\mathbf{x})$. That is, for each instance of one-hot garbling, Garbler samples $\hat{\mathbf{h}} := (\hat{h}_z)_{z\in\{0,1\}^n}$ to be a vector of 2^n labels to encode 0's and $\Delta \in \{0,1\}^\lambda$ be an offset to encode 1, where $\hat{h}_z \in \{0,1\}^\lambda$; Next, for any $\mathbf{x}$, we can garble $\mathcal{H}(\mathbf{x})$ into the vector $\mathbf{h}$ such that by definition,

$$\mathbf{h} = (h_z)_{z\in\{0,1\}^n} := \left(\hat{h}_0, \cdots, \hat{h}_{\mathbf{x}-1}, \hat{h}_{\mathbf{x}} \oplus \Delta_g, \hat{h}_{\mathbf{x}+1}, \cdots, \hat{h}_{2^n-1}\right). \tag{1}$$

We stress that Δ is sampled independently at random for each instance of one-hot garbling. To avoid confusion, henceforth we denote the offset by Δ_g where g is the unique "gate numbering" of the one-hot instance. We defer its construction to Section 2.2.

Such one-hot garbling facilitates LUT. Let $\mathcal{T}_f \in \{0,1\}^{2^n}$ be the truth table of $f : \{0,1\}^n \to \{0,1\}$. Suppose that Evaluator obtains both $\mathcal{T}_f$ and the above one-hot encoding $\mathbf{h}$. Then, the inner product $\langle \mathcal{T}_f, \mathbf{h} \rangle$ yields a garbling of $f(\mathbf{x})$ since

$$\langle \mathcal{T}_f, \mathbf{h} \rangle = \bigoplus_{z\in\{0,1\}^n} (f(z) \cdot h_z) = (f(\mathbf{x}) \cdot \Delta_g) \oplus \bigoplus_{z\in\{0,1\}^n} (f(z) \cdot \hat{h}_z).$$

That is a correct output when we define in advance the output labels to be $k^0 := \bigoplus_z (f(z) \cdot \hat{h}_z)$ and $k^1 := k^0 \oplus \Delta_g$. It appeared that we had achieved the garbling of the lookup table directly from one-hot garbling. However, Evaluator needs to know the actual value $\mathbf{x}$ to efficiently obtain one-hot garbling (see Section 2.2), and in the above, Evaluator knows the truth table $\mathcal{T}_f$. Both compromised privacy, and they are addressed next.

Building garbled lookup table from garbled pseudorandom function. To hide both $\mathbf{x}$ and f, [HKN24] employs a randomization technique. Specifically, Garbler selects a (pseudo)random function $r : \{0,1\}^n \to \{0,1\}$ and a random masking string $\mathbf{a} \in \{0,1\}^n$, and defines a new function f' by

$$f'(z) := f(z \oplus \mathbf{a}) \oplus r(z).$$

With f' prepared, Garbler sends to Evaluator the truth table $\mathcal{T}_{f'}$ of the function f' as well as the value $\mathbf{y} := \mathbf{x} \oplus \mathbf{a}$. Since $\mathbf{x}$ is masked by $\mathbf{a}$ and f is masked by r, knowing $\mathbf{y}$ and $\mathcal{T}_{f'}$ does not reveal any information about $\mathbf{x}, f$. Notice that the truth table of r must remain hidden from Evaluator. This is because $f' \oplus r$ is an offset of f—if Evaluator knows r in addition to $\mathcal{T}_{f'}$, she could derive f, $\mathbf{a}$, and $\mathbf{x}$.

Next, using one-hot garbling, Evaluator obtains $\mathbf{h}$ for $\mathcal{H}(\mathbf{y} = \mathbf{x} \oplus \mathbf{a})$. Through the inner product $\langle \mathcal{T}_{f'}, \mathbf{h} \rangle$, the output label of $f'(\mathbf{y})$ is derived, which equals to the output label of $f(\mathbf{x}) \oplus r(\mathbf{y})$. To obtain the desired label of $f(\mathbf{x})$, Evaluator needs to obtain the label of $r(\mathbf{y})$ and then to apply a simple garbled XOR. Therefore, we want Evaluator to derive the label of $r(\mathbf{y})$, while keeping the truth table of r hidden from Evaluator. That is exactly garbling a PRF; we will show an efficient construction in Section 2.3.

2.2 One-Hot Garbling

Recall that, a one-hot garbling scheme aims to garble the one-hot encoding $\mathcal{H}(\cdot)$ efficiently. Let $(k_0^0, k_0^1), \cdots, (k_{n-1}^0, k_{n-1}^1)$ be the input labels. Garbler shall compute output labels $\hat{\mathbf{h}} := (\hat{h}_j)_{j \in \{0,1\}^n}$ and Δ_g as well as a short ciphertext c^{hot}. Then, given plaintext input $\mathbf{x}$, its labels $(k_0^{\mathbf{x}[0]}, \cdots, k_{n-1}^{\mathbf{x}[n-1]})$, and ciphertext c^{hot}, Evaluator shall compute the labels of $\mathcal{H}(\mathbf{x})$, denoted by $\mathbf{h} = (h_j)_{j \in \{0,1\}^n}$ (Equation (1)). A naive way would be using a generic garbling scheme to garble $\mathcal{H}(\cdot)$, which would take $O(2^n \lambda)$ communication, which is proportional to λ. Fortunately, it is allowed to reveal input $\mathbf{x}$ to Evaluator, and the 0-labels $\hat{\mathbf{h}}$ can be pseudorandom and generated using a short random seed. They are harnessed to achieve short communication.

To reduce the communication, we will use a binary tree, similar to Goldreich-Goldwasser-Micali (GGM) construction [GGM84], briefly recalled using the following notation. It is a complete binary tree of 2^n leaves, where the root is level 0 and the leaves are in level n. For each level i, each node in level i is indexed by the tuple (i, j), where $j \in [2^i - 1]$ is also viewed as an i-bit binary string. The root is associated with a random seed s of λ bits. Then, each internal node expands its the random seed to its two children using a PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. Namely, letting $s_{0,0} = s$ be the root seed, for each seed $s_{i,j}$, we call G inductively by assigning $(s_{i+1,j\|0}, s_{i+1,j\|1}) \leftarrow \mathsf{G}(s_{i,j})$, until $i = n - 1$.

Our one-hot garbling modifies GGM tree as follows. We ignore the root s but sample its two children $s_{1,0}, s_{1,1} \leftarrow \{0,1\}^\lambda$ uniformly at random, and then we continue with the inductive expansion using G. With the modification, we define the 0-labels $\hat{\mathbf{h}}$ to be the leaf values, that is,

$$\text{let} \quad \hat{h}_j := s_{n,j} \quad \text{for all } j \quad \text{and let} \quad \hat{\mathbf{h}} := (\hat{h}_j)_{j \in \{0,1\}^n}.$$

Next, we want Evaluator to learn $\mathbf{h}$, which is identical to $\hat{\mathbf{h}}$ except for the entry $h_{\mathbf{x}}$. Observe that in the GGM tree, Garbler can reveal a half of $\hat{\mathbf{h}}$ (i.e., leaves) to Evaluator by sending the ciphertexts[5]

$$c_{0,0} := \mathsf{Enc}_{k_0^1}(s_{1,0}) \quad \text{and} \quad c_{0,1} := \mathsf{Enc}_{k_0^0}(s_{1,1}).$$

[5]Later in technical sections, we will instantiate the encryption with a PRF.

This way, when Evaluator gets k_0^0 (so that $\mathbf{x}[0] = 0$ and the second half of $\hat{\mathbf{h}}$ is identical to $\mathbf{h}$), she decrypts $c_{0,1}$ and obtains the second half $\mathbf{h}$ through GGM using $s_{1,1}$. Otherwise, $\mathbf{x}[0] = 1$, Evaluator obtains the first half $\mathbf{h}$ symmetrically. Notice that Garbler prepared the ciphertext without knowing input $\mathbf{x}$. Moreover, by the inductive construction of GGM, such "half revealing" can be proceeded inductively using the following invariant:

Invariant: For each $1 \leq i \leq n$, Evaluator learns seed $s_{i,j}$ for all $j \in \{0,1\}^i$ except for $j = \mathbf{x}[0 : i-1]$.

Clearly, the invariant holds for $i = 1$ using the ciphertext $(c_{0,0}, c_{0,1})$. We next show how Garbler prepares more ciphertext and then maintains the invariant. Suppose the invariant holds for $1 \leq i < n$. Then following GGM, Evaluator computes $(s_{i+1,j\|0},\ s_{i+1,j\|1}) \leftarrow \mathsf{G}(s_{i,j})$ for all $j \in \{0,1\}^i$ except for $j = \mathbf{x}[0 : i-1]$. The only "missing" seed in level $i+1$ is $s_{i+1,\mathbf{x}[0:i-1]\|(1-\mathbf{x}[i])}$, where $1 - \mathbf{x}[i]$ denotes the negation of bit $\mathbf{x}[i]$. To this end, Garbler supplies the missing seed by sending ciphertexts

$$c_{i,0} := \mathsf{Enc}_{k_i^1}\Big(\bigoplus_{j\in\{0,1\}^i} s_{i+1,j\|0}\Big) \text{ and } c_{i,1} := \mathsf{Enc}_{k_i^0}\Big(\bigoplus_{j\in\{0,1\}^i} s_{i+1,j\|1}\Big).$$

With label $k_i^{\mathbf{x}[i]}$, Evaluator decrypts the corresponding ciphertext $c_{i,1-\mathbf{x}[i]}$ and then obtains the missing seed by XORing with other seeds she already computed. This maintains the invariant for level $i+1$, and then inductively Garbler and Evaluator arrive at level n. Notice that Evaluator needs $\mathbf{x}$ to decrpyt the correct ciphertext.

When Evaluator obtained $s_{n,j} = \hat{h}_j = h_j$ for all $j \neq \mathbf{x}$, Evaluator needs $h_{\mathbf{x}} = \hat{h}_{\mathbf{x}} \oplus \Delta_g$ to complete $\mathbf{h} = (h_j)_{j\in\{0,1\}^n}$. Garbler samples Δ_g uniformly at random and then sends ciphertext $c_n := \Delta_g \oplus \bigoplus_{j\in\{0,1\}^n} \hat{h}_j$. Evaluator then XORs and obtains $c_n \oplus \bigoplus_{j\neq\mathbf{x}} h_j = \hat{h}_{\mathbf{x}} \oplus \Delta_g = h_{\mathbf{x}}$, as wanted.

Communication cost. The communication of our one-hot garbling is $(2n+1)$ ciphertexts, each is λ bits (assuming the encryption is length preserving). Hence, it takes $(2n+1)\lambda$ bits. Revealing input $\mathbf{x}$ takes additional n bit.

2.2.1 Illustrating Example

For better understanding of our construction, we consider a simple yet illustrative example where the size of $\mathbf{x}$ is 3, i.e. $n = 3$. Assuming the labels for 3 input wires are denoted as $(k_0^0, k_0^1), (k_1^0, k_1^1), (k_2^0, k_2^1)$, and Evaluator is aware of the plaintext input $\mathbf{x}$.

Generating 0-labels $\hat{\mathbf{h}}$ in the garbling process. Following the idea in GGM, Garbler selects two independently random seeds $s_{1,0}, s_{1,1} \leftarrow \{0,1\}^\lambda$ at level 1. Using the PRG G, Garbler expands the tree through 2 iterations to obtain 8 leaf values $(s_{3,000}, s_{3,001}, s_{3,010}, s_{3,011}, s_{3,100}, s_{3,101}, s_{3,110}, s_{3,111})$, each of length λ bits, as depicted in Figure 1a. The value $s_{3,j}$ of the jth leaf represents the 0-label $\hat{h}_j$, where $j \in \{0,1\}^3$. Next, to determine the 1-label for each wire, Garbler randomly selects a Δ_g from $\{0,1\}^\lambda$.

Determining labels of $\mathcal{H}(\mathbf{x})$ in the evaluation process. The remaining challenge is to enable Evaluator to obtain labels for $\mathcal{H}(\mathbf{x})$ (denoted by $\mathbf{h}$) during evaluation, given the plaintext input $\mathbf{x}$ and labels $k_0^{\mathbf{x}[0]}, k_1^{\mathbf{x}[1]}, k_2^{\mathbf{x}[2]}$.

<u>*Level 1.*</u> Consider the level 1 of the binary tree, if $\mathbf{x}[0] = 0$, the last four items of the one-hot encoding $\mathcal{H}(\mathbf{x})$ are all 0, so Evaluator should obtain the 0-labels $(\hat{h}_{3,100}, \hat{h}_{3,101}, \hat{h}_{3,110}, \hat{h}_{3,111})$, which are equal to $(s_{3,100}, s_{3,101}, s_{3,110}, s_{3,111})$. If Evaluator is provided with the seed $s_{1,1}$, she can reproduce the entire right subtree, thus obtaining these four 0-labels. Conversely, if $\mathbf{x}[0] = 1$, Evaluator should receive the seed $s_{1,0}$, enabling her to generate the left subtree and obtain $(s_{3,000}, s_{3,001}, s_{3,010}, s_{3,011})$, which are the 0-labels $(\hat{h}_{3,000}, \hat{h}_{3,001}, \hat{h}_{3,010}, \hat{h}_{3,011})$. Therefore, Garbler can use labels k_0^0 and k_0^1 to respectively encrypt $s_{1,1}$ and $s_{1,0}$. If $\mathbf{x}[0] = 0$, then Evaluator can obtain $s_{1,1}$; otherwise, she obtains $s_{1,0}$. The two ciphertexts are:

$$c_{0,0} := \mathsf{Enc}_{k_0^1}(s_{1,0}) \quad \text{and} \quad c_{0,1} := \mathsf{Enc}_{k_0^0}(s_{1,1}).$$

Level 2. Next, we move to level 2. If previously $\mathbf{x}[0] = 0$, then Evaluator has already obtained $s_{2,10}$ and $s_{2,11}$. If the next bit $\mathbf{x}[1]$ is 0, then the third and fourth items of the one-hot encoding $\mathcal{H}(\mathbf{x})$ are both 0, so Evaluator needs to receive $s_{2,01}$ to generate 0-labels $(\hat{h}_{3,010}, \hat{h}_{3,011})$. Else, $\mathbf{x}[1] = 1$, she would need $s_{2,00}$ to produce $(\hat{h}_{3,000}, \hat{h}_{3,001})$. Therefore, Garbler prepares the following two ciphertexts:

$$c_{1,0} := \mathsf{Enc}_{k_1^1}(s_{2,00} \oplus s_{2,10}) \quad \text{and} \quad c_{1,1} := \mathsf{Enc}_{k_1^0}(s_{2,01} \oplus s_{2,11}).$$

If the previous bit $\mathbf{x}[0] = 1$, Evaluator has already obtained $s_{2,00}$ and $s_{2,01}$, as shown in Figure 1b. She needs to retrieve one of the seeds, $s_{2,10}$ or $s_{2,11}$, depending on the value of $\mathbf{x}[1]$ (i.e. the label $k_1^{\mathbf{x}[1]}$). Analogously, this can be done using the same two ciphertexts, $c_{1,0}$ and $c_{1,1}$. That concludes this level.

Level 3. At level 3, Garbler constructs the following two ciphertexts in the same manner as level 2:

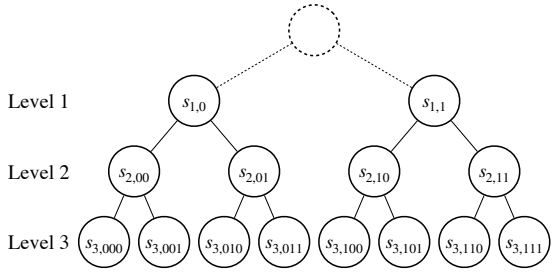
$$c_{2,0} := \mathsf{Enc}_{k_2^1}(s_{3,000} \oplus s_{3,010} \oplus s_{3,100} \oplus s_{3,110}) \quad \text{and}$$
$$c_{2,1} := \mathsf{Enc}_{k_2^0}(s_{3,001} \oplus s_{3,011} \oplus s_{3,101} \oplus s_{3,111}).$$

Based on the label $k_2^{\mathbf{x}[2]}$ and the already available 6 leaf values (0-labels), Evaluator decrypts and obtains the last 0-label. Up to this point, Evaluator has obtained 7 out of the 8 labels of $\mathcal{H}(\mathbf{x})$, all of which are 0-labels, leaving only the final 1-label yet to be acquired.

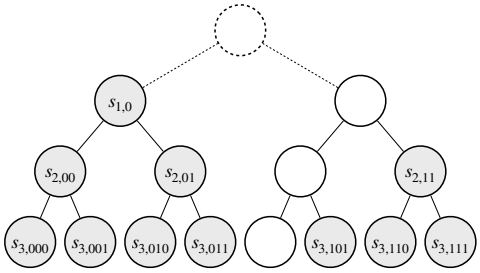
Finally, Garbler sends a single ciphertext to facilitate this process:

$$c_3 := s_{3,000} \oplus s_{3,001} \oplus s_{3,010} \oplus s_{3,011} \oplus s_{3,100} \oplus s_{3,101} \oplus s_{3,110} \oplus s_{3,111} \oplus \Delta_g.$$

Evaluator can XOR the 7 labels she already possesses with c_3 to obtain the final 1-label, namely $h_{\mathbf{x}} = \hat{h}_{\mathbf{x}} \oplus \Delta_g = s_{3,\mathbf{x}} \oplus \Delta_g$. In Figure 1b, we provide a concrete example of the evaluation.



(a) The binary tree for generating 0-labels $\hat{\mathbf{h}}$.



(b) A toy example of the evaluation, where $\mathbf{x} = [1, 0, 0]$, and $\mathcal{H}(\mathbf{x}) = [0, 0, 0, 0, 1, 0, 0, 0]$. Grey nodes denote that Evaluator is aware of the values of those nodes.

Figure 1: Illustration of the garbled PRF construction and evaluation.

2.3 Garbled Pseudorandom Functions (PRFs)

As mentioned in Section 2.1, we want to garble a function $r : \{0,1\}^n \to \{0,1\}$ such that

1. the truth table of r is pseudorandom, the description length of garbled r is shorter than 2^n bits, and
2. given the garbled r and a garbled input $\tilde{\mathbf{y}}$, Evaluator can compute the garbled output that can be decoded to $r(\mathbf{y})$, yet Evaluator learned nothing about r.
3. Evaluator may learn the plaintext $\mathbf{y}$ and use $\mathbf{y}$ to facilitate the evaluation since we already revealed $\mathbf{y}$ in Section 2.1. This assumption does not trivialize the problem, and it is only for presentation and can be removed (see Remark 4.4).

We observed that r is exactly a pseudorandom function (PRF) and garbling r is essentially garbling the seed of a PRF. A naive way could be applying a standard garbling scheme on a standard PRF, but that would be inefficient and use the underlying PRF in a non-black-box manner. We next show a black-box and efficient garbled PRF. We will begin with the intuition and construction of our PRF r and then present the garbling scheme for r. Our construction is inspired by [HKN24], but we will compare it with their scheme in Section 2.5.

2.3.1 Intuition and Construction of Our PRF

Our PRF construction is aimed to facilitate the garbling later. That is, Garbler must ensure the truth table $\mathcal{T}_r$ of r remains pseudorandom while Evaluator can compute the garbled output of $r(\mathbf{y})$. To do so, the key idea is to sample n independent "sub-seeds", which yield n independent "sub-PRFs." Each sub-seed is further split into two "half-seeds." By revealing exactly one half-seed (out of the two), later in the garbling scheme, for each input bit $\mathbf{y}[i]$, Evaluator can compute the corresponding sub-PRF on half of the input domain $\{0,1\}^n$. We will appropriately partition the input domain into n pairs of halves, so that putting together n sub-PRFs, Evaluator can only compute the PRF on exactly one point, i.e., the input $\mathbf{y}$. The above idea is elaborated below.

The PRF r is constructed by the XOR of $n+1$ functions $r_0, r_1, \cdots, r_n : \{0,1\}^n \to \{0,1\}$. Namely,

$$r(z) := r_0(z) \oplus r_1(z) \oplus \cdots r_n(z).$$

Except for r_n, each r_i is constructed as follows. Let F^* be an underlying PRF that outputs 1 bit. For each i, the domain $\{0,1\}^n$ is partitioned by the ith bit, named as Even and Odd halves:

$$\mathsf{Even}(n,i) := \{z \in \{0,1\}^n : z[i] = 0\} \quad \text{and}$$
$$\mathsf{Odd}(n,i) := \{z \in \{0,1\}^n : z[i] = 1\}.$$

For each even/odd domain, r_i is evaluated using an independent and randomly sampled half-seed, $s_i^{\mathrm{even}}/s_i^{\mathrm{odd}}$, with an underlying F^*. That is,

$$r_i(z) := \mathsf{F}^*_{s_i^{\mathrm{even}}}(z) \text{ if } z \in \mathsf{Even}(n,i) \quad \text{and}$$
$$r_i(z) := \mathsf{F}^*_{s_i^{\mathrm{odd}}}(z) \text{ if } z \in \mathsf{Odd}(n,i).$$

Lastly, r_n is a constant and random function: we sample $b \leftarrow \{0,1\}$ uniformly at random and let $r_n(z) := b$. We abuse notation and write r_n instead of $r_n(z)$. Overall, the seed of r is $\mathbf{s} := \left((s_i^{\mathrm{even}}, s_i^{\mathrm{odd}})_{i\in[n]}, r_n\right)$, which is $2n\lambda + 1$ bits. Clearly, r is pseudorandom, and for $n = \Omega(\log \lambda)$, the seed is shorter than the truth table. Notice that, even when a half-seed is revealed, the truth table $\mathcal{T}_r$ remains pseudorandom, which facilitates garbling next.

2.3.2 Garbling PRF using One-Hot Encoding

With the PRF r constructed, we present the garbling scheme of r below.

Recall that Evaluator gets both the garbled input $\tilde{\mathbf{y}}$ and its plaintext $\mathbf{y}$, and then she wants to compute a garbling of $r(\mathbf{y})$. To hide the seed $\mathbf{s}$ of r from Evaluator, we intentionally split sub-seeds into halves so that half-seeds can be revealed. Also, recall that we have an efficient one-hot garbling scheme: Garbler prepared the 0-labels $\hat{\mathbf{h}} = (\hat{h}_z)_{z\in\{0,1\}^n}$ and the offset Δ_g, and Evaluator can obtain the one-hot garbling $\mathbf{h} = (h_z)_{z\in\{0,1\}^n}$ of $\mathcal{H}(\mathbf{y})$.

We begin from the view of Evaluator, who already obtained $\mathbf{h}$. Suppose that we could give Evaluator the truth table $\mathcal{T}_r$ of r, then Evaluator could simply compute the desired garbled output by inner product $\langle \mathcal{T}_r, \mathbf{h}\rangle = \widetilde{r(\mathbf{y})}$. However, that would reveal $\mathcal{T}_r$ and violate security. Instead, the idea is to reveal *half* of the sub-PRF r_i depending on the input bit $\mathbf{y}[i]$ for each i. That is not enough to completely compute

$\langle \mathcal{T}_r, \mathbf{h} \rangle$. The next idea is, Garbler can encrypt and send both halves of the inner products *in advance* (without knowing $\mathbf{y}$), and then Evaluator can recover the whole inner product (which depends on $\mathbf{y}$) by decrypting the half-seed and the half inner product. Namely, for each i,

- Garbler knows input encoding $(k_i^0, k_i^1)_{i\in[n]}$, the one-hot labels $((\hat{h}_z)_{z\in\{0,1\}^n}, \Delta_g)$, and the PRF seed $\mathbf{s} = \big((s_i^{\mathrm{even}}, s_i^{\mathrm{odd}})_{i\in[n]}, r_n\big)$. Garbler prepares half inner products

$$t_i^{\mathrm{even}} := \bigoplus_{z\in \mathsf{Even}(n,i)} r_i(z)\cdot \hat{h}_z \quad \text{and} \quad t_i^{\mathrm{odd}} := \bigoplus_{z\in \mathsf{Odd}(n,i)} r_i(z)\cdot \hat{h}_z. \tag{2}$$

  Moreover, Garbler sends the following pair of ciphertexts, which encrypts the half-seed together with the *opposite half inner product*, using input labels as the encryption keys:

$$c_i^{\mathrm{even}} := \mathsf{Enc}_{k_i^0}(s_i^{\mathrm{even}} \| t_i^{\mathrm{odd}}) \quad \text{and} \quad c_i^{\mathrm{odd}} := \mathsf{Enc}_{k_i^1}(s_i^{\mathrm{odd}} \| t_i^{\mathrm{even}}). \tag{3}$$

- Evaluator receives both ciphertexts $(c_i^{\mathrm{even}}, c_i^{\mathrm{odd}})$ but only one key $k_i^{\mathbf{y}[i]}$ and decrypts only one ciphertext. Yet, that is sufficient to compute the whole inner product. To see why, consider the case $\mathbf{y}[i] = 0$. Evaluator decrypts c_i^{even} and gets $s_i^{\mathrm{even}} \| t_i^{\mathrm{odd}}$ and then computes

$$t_i^{\mathrm{odd}} \oplus \bigoplus_{z\in \mathsf{Even}(n,i)} \mathsf{F}^*_{s_i^{\mathrm{even}}}(z)\cdot h_z = \left(\bigoplus_{z\in \mathsf{Odd}(n,i)} r_i(z)\cdot \hat{h}_z\right) \oplus \bigoplus_{z\in \mathsf{Even}(n,i)} r_i(z)\cdot h_z = \bigoplus_{z\in\{0,1\}^n} r_i(z)\cdot h_z,$$

  where the first equality follows by construction of r_i and t_i^{odd}, and the second by $\hat{h}_z = h_z$ when $z \neq \mathbf{y}$, which holds as $\mathbf{y}[i] = 0$ and thus $\mathbf{y} \in \mathsf{Even}(n,i)$ but $z \in \mathsf{Odd}(n,i)$. The other case, $\mathbf{y}[i] = 1$, is symmetric and omitted. The result is the desired inner product $\langle \mathcal{T}_{r_i}, \mathbf{h} \rangle$.

The above inner products $\langle \mathcal{T}_{r_i}, \mathbf{h} \rangle$ are calculated using the same $\mathbf{h}$ and thus the same Δ_g for all i. Hence, Evaluator can next compute $\langle \mathcal{T}_{r_0}, \mathbf{h} \rangle \oplus \cdots \oplus \langle \mathcal{T}_{r_{n-1}}, \mathbf{h} \rangle$. That is almost equal to $\langle \mathcal{T}_r, \mathbf{h} \rangle$, except for the last random bit, r_n. This is intentional: the above allows Evaluator to learn $r_i(\mathbf{y})$ for all $i \in \{0, \cdots, n-1\}$, but that could reveal *one output* of r, Thus we randomize r using r_n so that Evaluator learns nothing about $\mathcal{T}_r$. Since r_n is a constant function, $c_n := \langle \mathcal{T}_{r_n}, \mathbf{h} \rangle$ is the same for all $\mathbf{y}$, so Garbler can prepare and send c_n to Evaluator in advance. Now, Evaluator can compute the garbled output by

$$\widetilde{r(\mathbf{y})} := \langle \mathcal{T}_{r_0}, \mathbf{h} \rangle \oplus \cdots \langle \mathcal{T}_{r_{n-1}}, \mathbf{h} \rangle \oplus c_n = \langle \mathcal{T}_r, \mathbf{h} \rangle.$$

Loose ends. The above construction is simplified for exposition, There are some loose ends. Firstly, revealing the half inner product (such as t_i^{odd}) or the last inner product c_n are *actually insecure*—for example, if $r_n = 0$, then $c_n = \langle \mathcal{T}_{r_n}, \mathbf{h} \rangle$ is a zero string, which totally reveals r_n! Hence, it is necessary (for Garbler) to mask each $t_i^{\mathrm{even}}, t_i^{\mathrm{odd}}$, and c_n appropriately. The masking is simple and only shifts the resulting output labels, which can be prepared by Garbler. Secondly, the garbled output $\langle \mathcal{T}_r, \mathbf{h} \rangle$ is still using labels that depends on Δ_g, which was sampled by the one-hot garbling internally. This can be eliminated by the standard "soldering" technique [NO09] (i.e., using the labels to encrypt and decrypt independent labels).

Communication cost. The total communication of the garbled PRF is $(6n+2)\lambda$ bits: the one-hot garbling takes $(2n+1)\lambda$ bits, and the ciphertexts of half-seeds and half inner products take $(4n+1)\lambda$ bits.

2.3.3 Illustrating Example

Continuing with our previous example where $n = 3$, we assume Garbler has generated 0-labels $\hat{\mathbf{h}} = (\hat{h}_z)_{z\in\{0,1\}^3}$ and the offset Δ_g, and Evaluator has obtained the one-hot garbling $\mathbf{h} = (h_z)_{z\in\{0,1\}^3}$ of $\mathcal{H}(\mathbf{y})$. In addition, the labels for $\mathbf{y}$ are denoted as $k_0^{\mathbf{y}[0]}, k_1^{\mathbf{y}[1]}$, and $k_2^{\mathbf{y}[2]}$.

Evaluator needs to obtain the garbled output $\widetilde{r(\mathbf{y})}$ without learning the truth table $\mathcal{T}_r$ of r. As mentioned, to achieve this, Garbler can construct 4 functions $r_0, r_1, r_2, r_3 : \{0,1\}^3 \to \{0,1\}$, and define the PRF r as $r(z) := r_0(z) \oplus r_1(z) \oplus r_2(z) \oplus r_3(z)$. The truth table for each function r_i only contains 8 bits, where $i \in \{0,1,2\}$. For simplicity, in our example, instead of sampling half-seed $s_i^{\text{even}}/s_i^{\text{odd}}$ to generate the truth table for each r_i, Garbler directly selects 8 random bits to serve as the truth table of r_i.

We use four steps to describe how Garbler processes each function r_j ($j \in \{0,1,2,3\}$) and prepares the corresponding ciphertexts. In Figure 2, we provide a high-level description of these four steps.

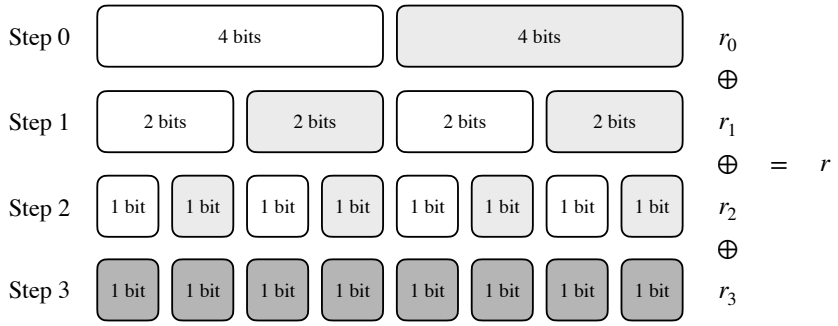


Figure 2: A toy example of the garbling for the PRF r. The white and gray blocks in the first row together form the truth table $\mathcal{T}_{r_0}$, while the second and third rows correspond to the truth tables for r_1 and r_2, respectively. In steps 0-2, Evaluator learns half of the truth table. Specifically, in step 0, Evaluator learns either the white or the gray block. The same applies to steps 1 and 2, where Evaluator can only obtain one set of blocks, either white or gray. As can be seen, for the function $r_0 \oplus r_1 \oplus r_2$, Evaluator can only learn one bit of the truth table. By XORing with a random bit r_3 at step 3, the truth table $\mathcal{T}_r$ can be fully hidden.

Step 0 (r_0). Garbler randomly selects 8 bits $\mathcal{T}_{r_0}$, which serves as the truth table of r_0. The garbled output $\widetilde{r_0(\mathbf{y})}$ is given by the inner product $\langle \mathcal{T}_{r_0}, \mathbf{h} \rangle$, which can be split into two parts:

$$\langle \mathcal{T}_{r_0}, \mathbf{h} \rangle = \langle \mathcal{T}_{r_0}[0:3], \mathbf{h}[0:3] \rangle \oplus \langle \mathcal{T}_{r_0}[4:7], \mathbf{h}[4:7] \rangle.$$

Also, Garbler precomputes the following half inner products:

$$t_0^{\text{even}} := \langle \mathcal{T}_{r_0}[0:3], \hat{\mathbf{h}}[0:3] \rangle \quad \text{and} \quad t_0^{\text{odd}} := \langle \mathcal{T}_{r_0}[4:7], \hat{\mathbf{h}}[4:7] \rangle.$$

If $\mathbf{y}[0] = 0$, then the only 1-label is definitely in the left half of $\mathbf{h}$, meaning the right half consists entirely of 0-labels, i.e. $\mathbf{h}[4:7] = \hat{\mathbf{h}}[4:7]$. If Evaluator can obtain $\mathcal{T}_{r_0}[0:3]$ and the value of t_0^{odd}, then she can directly compute the garbled output $\widetilde{r_0(\mathbf{y})}$:

$$\widetilde{r_0(\mathbf{y})} := \langle \mathcal{T}_{r_0}[0:3], \mathbf{h}[0:3] \rangle \oplus t_0^{\text{odd}}.$$

Conversely, if $\mathbf{y}[0] = 1$, then the 1-label will definitely be in the right half of $\mathbf{h}$, meaning the left half consists entirely of 0-labels. If Evaluator is aware of $\mathcal{T}_{r_0}[4:7]$ and t_0^{even}, then she can compute $\widetilde{r_0(\mathbf{y})}$ as follows:

$$\widetilde{r_0(\mathbf{y})} := \langle \mathcal{T}_{r_0}[4:7], \mathbf{h}[4:7] \rangle \oplus t_0^{\text{even}}.$$

Therefore, Garbler can send the following ciphertexts to Evaluator:

$$c_0^{\text{even}} := \mathsf{Enc}_{k_0^0}(\mathcal{T}_{r_0}[0:3] \| t_0^{\text{odd}}),\ c_0^{\text{odd}} := \mathsf{Enc}_{k_0^1}(\mathcal{T}_{r_0}[4:7] \| t_0^{\text{even}}).$$

We remark that, in the general case, the length of $\mathcal{T}_{r_0}$ is 2^n. If $2^n > 2\lambda$, then $\mathcal{T}_{r_0}[0:3]$ and $\mathcal{T}_{r_0}[4:7]$ should be generated by half seeds s_0^{even} and s_0^{odd} respectively (the length of each seed is λ). Therefore, the

maximum length of the ciphertexts in this step is 4λ, while in our example, the length of the ciphertexts is $8 + 2\lambda$.

Step 1 (r_1). As observed in step 0, if Evaluator possesses k_0^1, she can decrypt c_0^{odd} to obtain $\mathcal{T}_{r_0}[4:7]$, thus learning the right half of the truth table. If Evaluator possesses k_0^0, she can access the left half of the truth table. In either case, Evaluator can obtain half of $\mathcal{T}_{r_0}$. However, our goal is to keep the truth table of r completely hidden from Evaluator. Therefore, in step 1, we introduce a new random function r_1.

Specifically, Garbler selects 8 random bits as the truth table $\mathcal{T}_{r_1}$. The garbled output $\widetilde{r_1(\mathbf{y})}$ is given by the inner product $\langle \mathcal{T}_{r_1}, \mathbf{h} \rangle$, which can also be decomposed into four parts: $\langle \mathcal{T}_{r_1}, \mathbf{h} \rangle = \langle \mathcal{T}_{r_1}[0:1], \mathbf{h}[0:1] \rangle \oplus \langle \mathcal{T}_{r_1}[2:3], \mathbf{h}[2:3] \rangle \oplus \langle \mathcal{T}_{r_1}[4:5], \mathbf{h}[4:5] \rangle \oplus \langle \mathcal{T}_{r_1}[6:7], \mathbf{h}[6:7] \rangle$.

Similar to step 0, Garbler prepares two half inner products:

$$t_1^{\text{even}} := \langle \mathcal{T}_{r_1}[0:1], \hat{\mathbf{h}}[0:1] \rangle \oplus \langle \mathcal{T}_{r_1}[4:5], \hat{\mathbf{h}}[4:5] \rangle \quad \text{and}$$
$$t_1^{\text{odd}} := \langle \mathcal{T}_{r_1}[2:3], \hat{\mathbf{h}}[2:3] \rangle \oplus \langle \mathcal{T}_{r_1}[6:7], \hat{\mathbf{h}}[6:7] \rangle.$$

If $\mathbf{y}[1] = 0$, then both $\mathbf{h}[2:3]$ and $\mathbf{h}[6:7]$ consist of 0-labels, namely $\mathbf{h}[2:3] = \hat{\mathbf{h}}[2:3]$ and $\mathbf{h}[6:7] = \hat{\mathbf{h}}[6:7]$. Since Evaluator knows the values of $\mathcal{T}_{r_1}[0:1]$ and $\mathcal{T}_{r_1}[4:5]$, as well as the value of t_1^{odd}, she can compute the garbled output $\widetilde{r_1(\mathbf{y})}$ as follows:

$$\widetilde{r_1(\mathbf{y})} := \langle \mathcal{T}_{r_1}[0:1], \mathbf{h}[0:1] \rangle \oplus \langle \mathcal{T}_{r_1}[4:5], \mathbf{h}[4:5] \rangle \oplus t_1^{\text{odd}}.$$

If $\mathbf{y}[1] = 1$, then both $\mathbf{h}[0:1]$ and $\mathbf{h}[4:5]$ are 0-labels. Since Evaluator knows the values of $\mathcal{T}_{r_1}[2:3]$ and $\mathcal{T}_{r_1}[6:7]$, along with t_1^{even}, she can compute the garbled output $\widetilde{r_1(\mathbf{y})}$

$$\widetilde{r_1(\mathbf{y})} := \langle \mathcal{T}_{r_1}[2:3], \mathbf{h}[2:3] \rangle \oplus \langle \mathcal{T}_{r_1}[6:7], \mathbf{h}[6:7] \rangle \oplus t_1^{\text{even}}.$$

Therefore, Garbler will send the following two ciphertexts:

$$c_1^{\text{even}} := \mathsf{Enc}_{k_1^0}(\mathcal{T}_{r_1}[0:1] \| \mathcal{T}_{r_1}[4:5] \| t_1^{\text{odd}}) \quad \text{and}$$
$$c_1^{\text{odd}} := \mathsf{Enc}_{k_1^1}(\mathcal{T}_{r_1}[2:3] \| \mathcal{T}_{r_1}[6:7] \| t_1^{\text{even}}).$$

Step 2 (r_2). To obtain $\widetilde{r_1(\mathbf{y})}$, Evaluator needs to know half of the truth table $\mathcal{T}_{r_1}$. If we consider $r_0 \oplus r_1$ as the final PRF r, then Evaluator only has knowledge of one-quarter of the truth table of r (as can be easily seen from Figure 2). Toward our goal, in step 2, we introduce another random function r_2.

Garbler selects 8 random bits as the truth table $\mathcal{T}_{r_2}$. Again, the key observation is that if $\mathbf{y}[2] = 0$, then $\mathbf{h}[1]$, $\mathbf{h}[3]$, $\mathbf{h}[5]$, and $\mathbf{h}[7]$ will definitely be 0-labels. Conversely, if $\mathbf{y}[2] = 1$, then $\mathbf{h}[0]$, $\mathbf{h}[2]$, $\mathbf{h}[4]$, and $\mathbf{h}[6]$ will definitely be 0-labels. The garbled output $\widetilde{r_2(\mathbf{y})}$ is given by $\langle \mathcal{T}_{r_2}, \mathbf{h} \rangle$.

Therefore, Garbler precomputes the following half inner products:

$$t_2^{\text{even}} := \langle \mathcal{T}_{r_2}[0], \hat{\mathbf{h}}[0] \rangle \oplus \langle \mathcal{T}_{r_2}[2], \hat{\mathbf{h}}[2] \rangle \oplus \langle \mathcal{T}_{r_2}[4], \hat{\mathbf{h}}[4] \rangle \oplus \langle \mathcal{T}_{r_2}[6], \hat{\mathbf{h}}[6] \rangle,$$
$$t_2^{\text{odd}} := \langle \mathcal{T}_{r_2}[1], \hat{\mathbf{h}}[1] \rangle \oplus \langle \mathcal{T}_{r_2}[3], \hat{\mathbf{h}}[3] \rangle \oplus \langle \mathcal{T}_{r_2}[5], \hat{\mathbf{h}}[5] \rangle \oplus \langle \mathcal{T}_{r_2}[7], \hat{\mathbf{h}}[7] \rangle.$$

The two ciphertexts for this step are:

$$c_2^{\text{even}} := \mathsf{Enc}_{k_2^0}(\mathcal{T}_{r_2}[0] \| \mathcal{T}_{r_2}[2] \| \mathcal{T}_{r_2}[4] \| \mathcal{T}_{r_2}[6] \| t_2^{\text{odd}}) \quad \text{and}$$
$$c_2^{\text{odd}} := \mathsf{Enc}_{k_2^1}(\mathcal{T}_{r_2}[1] \| \mathcal{T}_{r_2}[3] \| \mathcal{T}_{r_2}[5] \| \mathcal{T}_{r_2}[7] \| t_2^{\text{even}}).$$

Step 3 (r_3). Up to this point, if we define $r = r_0 \oplus r_1 \oplus r_2$, then we can observe that Evaluator will only learn a *single* bit of r's truth table $\mathcal{T}_r$. Therefore, Garbler can inject a random bit r_3 to hide this revealed bit by sending ciphertext $c_3 := \langle \mathcal{T}_{r_3}, \mathbf{h} \rangle$.

Evaluator computes the final output label $\widetilde{r(\mathbf{y})} := \widetilde{r_0(\mathbf{y})} \oplus \widetilde{r_1(\mathbf{y})} \oplus \widetilde{r_2(\mathbf{y})} \oplus c_3$.

2.4 Cost of Garbled Lookup Table

The total communication for garbling LUT $f : \{0,1\}^n \to \{0,1\}$ consists of three parts: one-hot garbling, garbled PRF, and the masked truth table $\mathcal{T}_{f'}$. Their costs are $(2n+1)\lambda$, $(6n+2)\lambda$, and 2^n bits correspondingly. Additionally, there are minor steps, such as XORing the partial results and revealing the masked input $\mathbf{y}$, which take $O(\lambda)$ and $O(n)$ bits. Hence, the total cost is $8n\lambda + O(n+\lambda) + 2^n$ bits. For functions that outputs m bits, the total cost takes a multiplicative factor m, $8nm\lambda + mO(n+\lambda) + 2^n m$ bits. We will elaborate on detailed constants in Section 4.4 and explain optimizations in Remark 4.6.

Remark 2.1 (Insecurity of reusing one-hot garbling.)**.** There are two instances of one-hot garbling in the garbled LUT, and both instance are evaluated on the same input $\mathbf{y}$, but we have to garble them independently for security. It is because a reused one-hot garbling reuses the same Δ_g, and later that yields two correlated PRF keys (both depend on Δ_g).

2.5 Our Conceptual Contribution: Eliminating Circularity

As claimed, we achieve the same efficiency compared to the previous schemes [HK21, HKN24] up to a small constant factor, but we relied only on one-way functions. Our main contribution is eliminating the need for circular-secure assumptions, such as random oracle or circular correlation robust hash (CCRH). At the first glance, our scheme looks similar to the previous schemes. However, we took a novel and modular approach to eliminate the circularity. Our approach is a conceptual contribution, and it is potentially applicable to future schemes that rely on circular security. In hindsight, we demonstrate how to modify the previous schemes into our scheme, step by step.

One-hot garbling: sample independent wire labels. We started by looking closely at the previous one-hot garbling [HK21]. The scheme instantiated a GGM tree, but it relied on the Free-XOR technique [KS08]. Namely, all the input wire labels (k_i^0, k_i^1) shared the same global offset Δ (so that $k_i^1 = k_i^0 \oplus \Delta$ for all wire i). Moreover, the seed of the GGM tree was derived from the input labels, k_0^0 and $k_0^1 = k_0^0 \oplus \Delta$. That incurred circular security: the resulting GGM tree depended on Δ, and the labels k_i^1 depended on Δ, meanwhile we encrypted (part of) the tree using k_i^1. Clearly in the encryption, the plaintext is key-dependent, and we needed a circular-secure encryption scheme. Inspired by the "FleXOR" technique [KMR14], we observe that the circularity can be resolved *using a different offset Δ_i for each pair of wire labels* (k_i^0, k_i^1). As a result, that makes the GGM tree depends only on Δ_0, but we encrypt the tree using $k_i^1 = k_i^0 \oplus \Delta_i$, which is independent of the plaintext, i.e., GGM tree. Adopting different Δ_i's is equivalent to sample each pair (k_i^0, k_i^1) independently. That gives a one-hot garbling based on one-way functions.

Garbled PRF: introduce independent random seeds. We next looked at the garbled PRF of [HKN24] (which was called as "random function evaluation"). We adopt the above technique of independent wire labels, but there is another challenge. In the previous scheme, the PRF was seeded using the input wire labels (k_i^0, k_i^1). That is, the seeds were set to be

$$(s_i^{\text{even}}, s_i^{\text{odd}}) := (k_i^0, k_i^1).$$

Such seeds incurred circular security because later the wire labels were used again as secret keys to encrypt messages, which depended on the seeds. Namely, instead of Equation (3) of our scheme, their encryption was roughly

$$c_i^{\text{even}} := \mathsf{Enc}_{k_i^0}(k_i^0 \| t_i^{\text{odd}}) \quad \text{and} \quad c_i^{\text{odd}} := \mathsf{Enc}_{k_i^1}(k_i^1 \| t_i^{\text{even}}).$$

Of course, since Evaluator already obtained the label k_i^b for a bit b, there was no need to encrypt k_i^0 nor k_i^1 (with key k_i^b itself). Hence, the previous scheme skipped k_i^b and performed

$$c_i^{\text{even}} := \mathsf{Enc}_{k_i^0}(t_i^{\text{odd}}) \quad \text{and} \quad c_i^{\text{odd}} := \mathsf{Enc}_{k_i^1}(t_i^{\text{even}}).$$

The above seemed good until we looked at the preparation of the messages $(t_i^{\text{even}}, t_i^{\text{odd}})$, which was derived from $(s_i^{\text{even}}, s_i^{\text{odd}}) = (k_i^0, k_i^1)$ correspondingly (as per Equation (2)). That is, t_i^{odd} depended on k_i^1 while t_i^{even} depended on k_i^0. Then, sending the above two ciphertexts incurred a two-hop circularity—particularly in the proof, when k_i^1 was given and thus t_i^{even} was known to Evaluator, we could not argue that c_i^{even} was secure because t_i^{even} could leak information about k_i^0 (and symmetrically for the opposite case).

To this end, we find that *introducing independent random seed* helps. Particularly, the seed of the PRF (without garbling) should be independent of the (randomness of) garbling scheme. This technique costs only the extra encryption of the seeds, a small factor.

Putting together: look for global dependency. We examined the above techniques using proofs, but we found a remaining circular dependency. Observe that, t_i^{even} also depended on (half of) $\hat{\mathbf{h}}$ as per Equation (2), but $\hat{\mathbf{h}}$ is the GGM tree leaves, which were in turn derived from the wire labels (k_0^0, k_0^1) as per the above "one-hot garbling" paragraph. The dependency occurred similarly for the odd case. When composed in the garbled PRF, it suffered circularity—we encrypted t_0^{even} using key k_0^1 as per Equation (3). Fortunately, it is easy to resolve the issue. We apply the idea "independent random seed" one more time—just use independent random seed in the GGM tree (instead of seeded by labels). The cost is only an extra encrypted seed, a minor and additive cost.

Notice that in this case, the standalone one-hot garbling did not suffer from the circularity issue, but the composition of garbled PRF did. Because it is easy to incur new circularity by composing modules, we believe that *globally checking the proof is necessary*. We use the above approach and standard garbling techniques to obtain our scheme based on one-way functions.

3 Preliminaries

Notations. We use $X \approx_c Y$ to denote that distributions X and Y are computationally indistinguishable. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive polynomial p and all sufficiently large λ, the condition $\mathsf{negl}(\lambda) < 1/p(\lambda)$ holds.

3.1 One Hot Encoding

Definition 3.1 (One-hot encoding)**.** *Consider an array $\mathbf{a} \in \{0,1\}^n$ of length n. The one-hot encoding of $\mathbf{a}$ is a length-2^n array, denoted by $\mathcal{H}(\mathbf{a})$, such that for all $i \in [2^n]$:*

$$\mathcal{H}(\mathbf{a})[i] := \begin{cases} 1 & \text{if } i = \mathbf{a}, \\ 0 & \text{otherwise}. \end{cases}$$

Definition 3.2 (Truth table)**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a function. The truth table of f, denoted by $\mathcal{T}_f$, is an array of length 2^n defined as follows:*

$$\mathcal{T}_f[i] := f(i).$$

Thus, the i-th element of $\mathcal{T}_f$ is the output $f(i)$.

Lemma 3.3 (Evaluation by truth table)**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a function, and let $\mathbf{a} \in \{0,1\}^n$ be an array of length n, then*

$$\langle \mathcal{T}_f, \mathcal{H}(\mathbf{a}) \rangle = f(\mathbf{a}).$$

3.2 Garbling Schemes

In this paper, we will use a (slightly) modified version of the abstraction for garbling schemes by Bellare, Hoang, and Rogaway [BHR12b]. Concretely, we will "decouple" Bellare et al's garbling algorithm into

two parts, *key garbling* and *functionality garbling*: in the *key garbling* algorithm, the encoding function description e will be generated; this e along with f, i.e., the functionality to be garbled, will then be used as input of the *functionality garbling* algorithm, to produce output (F, d). Note that, the original garbling algorithm by Bellare et al, takes f as input, and generate e and (F, d) as the output. We remark that, our slightly modified version can be viewed as a special case of Bellare et al's. In addition, existing natural constructions can be based on our modified version.

Definition 3.4. *For a function f with n-bit input and m-bit output, a garbling scheme, denoted as $\mathcal{GC} = (\mathsf{KG}, \mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$, consists of the following algorithms:*

- (Key) Garbling algorithm $e \leftarrow \mathsf{KG}(1^\lambda, 1^n)$: *On input $(1^\lambda, 1^n)$, outputs e, where e describes an encoding function.*
- (Functionality) Garbling algorithm $(F, d) \leftarrow \mathsf{Gb}(f, e)$: *On input (f, e), outputs (F, d), where F is a* **garbled functionality***, and d describes a decoding function.*
- Encoding algorithm $X := \mathsf{En}(e, x)$: *On input (e, x), outputs X,*
  where x is an **initial input** *for f, and X is a* **garbled input***.*
- Evaluation algorithm $Y := \mathsf{Ev}(F, X)$: *On input (F, X), outputs Y, where Y is a* **garbled output***.*
- Decoding algorithm $y := \mathsf{De}(d, Y)$: *On input (d, Y), outputs y, where y is a* **final output***.*

The correctness property is defined as follows:

- **Correctness:** For any function f and any input x, if $e \leftarrow \mathsf{KG}(1^\lambda, 1^n)$ and $(F, d) \leftarrow \mathsf{Gb}(f, e)$, then it holds that
$$\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x)$$
with all but negligible probability.

Due to space limitations, the definitions of the garbling scheme's security and other preliminaries are provided in Appendix A.

4 Our Construction

In this section, we describe the full construction of our garbled lookup table. Specifically, in Section 4.1, we provide constructions for some basic gadgets, all based on existing garbling techniques. In Section 4.2, we will formally describe our one-hot garbling approach. Section 4.3 will present the construction of the garbled PRF. Finally, in Section 4.4, we will formally describe the garbled lookup table, which utilizes the tools developed in the preceding three sections.

Recall that for wire i, its two labels are (k_i^0, k_i^1). The label k_i^0 garbles the actual value 0, while k_i^1 garbles 1. We use the least significant bit to represent the color bit of each label, i.e., the color bits for k_i^0 and k_i^1 are $\mathsf{lsb}(k_i^0)$ and $\mathsf{lsb}(k_i^1)$ respectively, and they satisfy $\mathsf{lsb}(k_i^0) \oplus \mathsf{lsb}(k_i^1) = 1$. The permute bit of this wire is $\mathsf{lsb}(k_i^0)$. The permute bit, color bit, and actual value x satisfy $x = \mathsf{lsb}(k_i^0) \oplus \mathsf{lsb}(k_i^x)$. Throughout the construction in this section, we let g be a fixed and public integer, which is unique for each instance of the garbled table.

4.1 Basic Gadgets

Before formally describing our main construction, we first briefly introduce some basic gadgets required for our construction. They are classic and well-established garbling techniques (and provided for completeness). We use a PRF $\hat{\mathsf{F}}_k : \{0,1\}^* \to \{0,1\}^\lambda$ to encrypt messages, where $k \in \{0,1\}^\lambda$.

First, we present the classical garbling scheme $\mathsf{Gate}.\{\mathsf{Gb}, \mathsf{Ev}\}$ for a binary gate with two inputs and one output; please see Procedure 1 and Procedure 2 for details. The input parameters of the $\mathsf{Gate}.\mathsf{Gb}$ procedure include gate, which specifies the type of gate, such as XOR or AND.

Procedure 1. $\mathsf{Gate}.\mathsf{Gb}\left(\mathsf{gate}, (k_0^0, k_0^1), (k_1^0, k_1^1)\right)$:

1. Choose labels for the output wire: $k^0, k^1 \leftarrow \{0,1\}^\lambda$ such that $\mathsf{lsb}(k^0) \oplus \mathsf{lsb}(k^1) = 1$.
2. Define the permute bits: $p_0 := \mathsf{lsb}(k_0^0)$, $p_1 := \mathsf{lsb}(k_1^0)$.
3. Compute four cipertexts: for $i \in \{0,1\}$ and $j \in \{0,1\}$,
$$c_{i,j} := \hat{\mathsf{F}}_{k_0^{i \oplus p_0}}(g \| i \| j) \oplus \hat{\mathsf{F}}_{k_1^{j \oplus p_1}}(g \| i \| j) \oplus k^{\mathsf{gate}(i \oplus p_0, j \oplus p_1)}.$$
4. Output $\left((k^0, k^1), c\right)$, where $c := (c_{0,0}, c_{0,1}, c_{1,0}, c_{1,1})$.

Procedure 2. $\mathsf{Gate}.\mathsf{Ev}\,(k_0, k_1, c)$:

(0) Interpret $c = (c_{0,0}, c_{0,1}, c_{1,0}, c_{1,1})$.

1. Compute the output label $k := \hat{\mathsf{F}}_{k_0}(g \| \mathsf{lsb}(k_0) \| \mathsf{lsb}(k_1)) \oplus \hat{\mathsf{F}}_{k_1}(g \| \mathsf{lsb}(k_0) \| \mathsf{lsb}(k_1)) \oplus c_{\mathsf{lsb}(k_0), \mathsf{lsb}(k_1)}$.
2. Output k.

In our main construction, we will also use the single-bit identity gate, denoted and defined by $\mathsf{Id}(b) := b$. We will use the standard construction to garble this identity gate. The scheme is denoted by $\mathsf{IDF}.\{\mathsf{Gb}, \mathsf{Ev}\}$; please refer to Procedures 3 and 4.

Procedure 3. $\mathsf{IDF}.\mathsf{Gb}(k^0, k^1)$:

1. Choose labels for the output wire: $k^{0,\mathrm{out}}, k^{1,\mathrm{out}} \leftarrow \{0,1\}^\lambda$ such that $\mathsf{lsb}(k^{0,\mathrm{out}}) \oplus \mathsf{lsb}(k^{1,\mathrm{out}}) = 1$.
2. Compute the ciphertexts: $c_0 := \hat{\mathsf{F}}_{k^{\mathsf{lsb}(k^0)}}(g) \oplus k^{\mathsf{lsb}(k^0),\mathrm{out}}$ and $c_1 := \hat{\mathsf{F}}_{k^{\mathsf{lsb}(k^1)}}(g) \oplus k^{\mathsf{lsb}(k^1),\mathrm{out}}$.
3. Output $(k^{0,\mathrm{out}}, k^{1,\mathrm{out}}, c)$, where $c := (c_0, c_1)$.

Procedure 4. $\mathsf{IDF}.\mathsf{Ev}\,(k, (c_0, c_1))$:

1. Compute $k^{\mathrm{out}} := \hat{\mathsf{F}}_k(g) \oplus c_{\mathsf{lsb}(k)}$.
2. Output k^{out}.

Remark 4.1. Currently, based on the standard assumption, garbling an AND gate requires ciphertexts of 2λ bits, whereas garbling an XOR gate requires λ bits [GLNP15]. Therefore, we can introduce these optimal results in this subsection. We can also apply row reduction technique [NPS99] to reduce the ciphertext length for the identity gate garbling.

4.2 One-Hot Garbling

In this subsection, we describe our construction of one-hot garbling scheme $\mathsf{OneHot}.\{\mathsf{Gb}, \mathsf{Ev}\}$; please see Procedure 5 and Procedure 6 for details. Our construction will utilize a PRF and a PRG as building blocks. Specifically, we use PRF $\hat{\mathsf{F}}_k : \{0,1\}^* \to \{0,1\}^\lambda$ to encrypt messages, where $k \in \{0,1\}^\lambda$ is the secret key, and we use PRG $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ to generate the GGM tree.

Procedure 5. $\mathsf{OneHot.Gb}\left((k_i^0, k_i^1)_{i\in[n]}\right)$:

1. Extract the permute bits: for all $i \in [n]$, $p_i := \mathsf{lsb}(k_i^0)$.[6]
2. Sample random seeds $s_{1,0}, s_{1,1} \leftarrow \{0,1\}^\lambda$, and $\Delta_g \leftarrow \{0,1\}^{\lambda-1}\|1$.
3. Extend the GGM tree and prepare ciphertexts for each level. That is, for each i from 0 to $n-1$:
   (a) Compute ciphertexts for level $i+1$:
   $$c_{i,0} := \hat{\mathsf{F}}_{k_i^1}(g) \oplus \left(\bigoplus_{j\in[2^i]} s_{i+1,2j}\right) \quad \text{and}$$
   $$c_{i,1} := \hat{\mathsf{F}}_{k_i^0}(g) \oplus \left(\bigoplus_{j\in[2^i]} s_{i+1,2j+1}\right).$$
   (b) If $i \neq n-1$, then for $j \in [2^{i+1}]$: compute $s_{i+2,2j}\|s_{i+2,2j+1} := \mathsf{G}(s_{i+1,j})$.
4. Compute the last ciphertext $c_n := \left(\bigoplus_{i\in[2^n]} s_{n,i}\right) \oplus \Delta_g$.
5. Output the following:
   (a) One hot garbling (all 0-labels) $\hat{\mathbf{h}} = (\hat{h}_z)_{z\in\{0,1\}^n}$, where $\hat{h}_z := s_{n,z}$,
   (b) Offset Δ_g,
   (c) Ciphertexts $c := \left((c_{i,0}, c_{i,1})_{i\in[n]}, c_n, (p_0, \cdots, p_{n-1})\right)$.

Procedure 6. $\mathsf{OneHot.Ev}\left((k_i)_{i\in[n]}, c\right)$:

(0) Interpret $c = \left((c_{i,0}, c_{i,1})_{i\in[n]}, c_n, p := (p_0, \cdots, p_{n-1})\right)$.

1. Reveal the input wire values: for all $i \in [n]$, let
   $$y_i := \begin{cases} 0 & \text{if } \mathsf{lsb}(k_i) = p_i, \\ 1 & \text{otherwise.} \end{cases}$$
2. Initially, set $sum := 0$. // This variable indicates the active path in the GGM tree.
3. Evaluate the GGM tree, i.e., for i from 0 to $n-1$, perform:
   (a) Compute the sibling seed of the active node:
   $$s_{i+1,sum+1-y_i} := \hat{\mathsf{F}}_{k_i}(g) \oplus c_{i,1-y_i} \oplus \left(\bigoplus_{j\in[2^i] \text{ and } 2j\neq sum} s_{i+1,2j+1-y_i}\right).$$

[6] Looking forward, later in our garbled lookup table (Procedure 11), we will optimize by reusing the color bits as the mask, which will yield $p_i = 0$ for all i. Here we aim for a simple yet generic construction. Jumping ahead, the same situation occurs in Step 1 of Procedure 8.

(b) For $j \in [2^{i+1}]$: if $i \neq n-1$ and $s_{i+1,j} \neq \perp$, compute $s_{i+2,2j} \| s_{i+2,2j+1} := \mathsf{G}(s_{i+1,j})$.

(c) Update $sum := 2(sum + y_i)$.

4. Compute the 1-label $k := c_n \oplus (\bigoplus_{i \in [2^n] \text{ and } i \neq \mathbf{y}} s_{n,i})$, where $\mathbf{y} \in [2^n]$ is the integer $y_0 \| y_1 \cdots \| y_{n-1}$ in binary representation.

5. Define the output labels: for all $z \in \{0,1\}^n$, let

$$h_z := \begin{cases} s_{n,z} & \text{if } z \neq \mathbf{y}, \\ k & \text{otherwise.} \end{cases}$$

6. Output $\mathbf{h} := (h_z)_{z \in \{0,1\}^n}$.

4.3 Garbled PRF

In this subsection, we describe our construction of the garbled PRF scheme.

4.3.1 Notions and Building Blocks

In the following construction, we consider PRFs that computes on n-bit strings. The resulting PRF is a composition of $2n$ instances of PRFs (i.e., $2n$ seeds). For each input string, we will apply a different set of PRF instances. Hence, we will describe the composition using the following even-odd sets.

Definition 4.2. *For any $n \in \mathbb{N}$ and $i \in [n]$, sets $\mathsf{Even}(n,i)$ and $\mathsf{Odd}(n,i)$ are defined to be*

$$\mathsf{Even}(n,i) := \{z \in \{0,1\}^n : z[i] = 0\},$$
$$\mathsf{Odd}(n,i) := \{z \in \{0,1\}^n : z[i] = 1\}.$$

Different PRF lengths. In this section, PRFs are applied for two different output lengths: for any key $k \in \{0,1\}^\lambda$, we denote by $\mathsf{F}_k : \{0,1\}^n \to \{0,1\}$ to be the PRF that outputs only 1 bit, while $\mathsf{F}'_k : \{0,1\}^* \to \{0,1\}^{2\lambda}$ outputs 2λ bits.

Generating seeds. Before we describe the garbling algorithm for the PRF, we first present the sampling of the seed and the (plaintext) evaluation of the PRF, refer to Procedure 7.

Procedure 7. $\mathsf{SeedGen}\left(1^\lambda, 1^n\right)$:

1. For each $i \in [n]$, compute the following (seeds of) PRFs:

(a) Sample PRF seeds $s_i^{\text{even}}, s_i^{\text{odd}} \leftarrow \{0,1\}^\lambda$.

(b) Define $r_i : \{0,1\}^n \to \{0,1\}$ to be

$$r_i(z) := \begin{cases} \mathsf{F}_{s_i^{\text{even}}}(z) & z \in \mathsf{Even}(n,i), \\ \mathsf{F}_{s_i^{\text{odd}}}(z) & z \in \mathsf{Odd}(n,i). \end{cases}$$

2. Sample a mask bit $r_n \leftarrow \{0,1\}$.

3. Compute the truth table $\mathcal{T}_r$ of the function $r : \{0,1\}^n \to \{0,1\}$ that is defined by $r(z) := r_n \oplus \bigoplus_{i \in [n]} r_i(z)$.

4. Output $\left((s_i^{\text{even}}, s_i^{\text{odd}})_{i \in [n]}, r_n, \mathcal{T}_r\right)$.

4.3.2 Construction of Garbled PRF Scheme

Our garbled PRF scheme is denoted by $\mathsf{PRF}.\{\mathsf{Gb}, \mathsf{Ev}\}$; please see Procedure 8 and Procedure 9 for details.

Procedure 8. $\mathsf{PRF.Gb}\left((k_i^0, k_i^1)_{i\in[n]}\right)$:

1. Extract the permute bits: for all $i \in [n]$, $p_i := \mathsf{lsb}(k_i^0)$.
2. Prepare one-hot garbling (Procedure 5):
$$\left(\hat{\mathbf{h}} := (\hat{h}_z)_{z\in\{0,1\}^n}, \Delta_g, c^{\mathrm{hot}}\right) \leftarrow \mathsf{OneHot.Gb}\left((k_i^0, k_i^1)_{i\in[n]}\right).$$
3. Sample the PRF seeds and truth table (Procedure 7):
$$\left((s_i^{\mathrm{even}}, s_i^{\mathrm{odd}})_{i\in[n]}, r_n, \mathcal{T}_r\right) \leftarrow \mathsf{SeedGen}\left(1^\lambda, 1^n\right).$$
4. Prepare garbled PRF for each $i \in [n]$ as below:
   (a) Prepare "half" inner products:
$$t_i^{\mathrm{even}} := \bigoplus_{z\in\mathsf{Even}(n,i)} \mathsf{F}_{s_i^{\mathrm{even}}}(z)\cdot\hat{h}_z \quad \text{and}$$
$$t_i^{\mathrm{odd}} := \bigoplus_{z\in\mathsf{Odd}(n,i)} \mathsf{F}_{s_i^{\mathrm{odd}}}(z)\cdot\hat{h}_z.$$
   (b) Sample label $k_i^{\mathrm{msk}} \leftarrow \{0,1\}^\lambda$.
   (c) Compute $\begin{cases} c_i^{\mathrm{even}} := \mathsf{F}'_{k_i^0}(g) \oplus \left(s_i^{\mathrm{even}} \| (k_i^{\mathrm{msk}} \oplus t_i^{\mathrm{odd}})\right), \\ c_i^{\mathrm{odd}} := \mathsf{F}'_{k_i^1}(g) \oplus \left(s_i^{\mathrm{odd}} \| (k_i^{\mathrm{msk}} \oplus t_i^{\mathrm{even}})\right). \end{cases}$
5. Garbled the bit r_n: sample $k_n^{\mathrm{msk}} \leftarrow \{0,1\}^\lambda$, and then compute $c_n := k_n^{\mathrm{msk}} \oplus (r_n \cdot \Delta_g)$.
6. Solder the output wires: Compute
$$k^0 := \langle \mathcal{T}_r, \hat{\mathbf{h}}\rangle \oplus k_n^{\mathrm{msk}} \oplus \bigoplus_{i\in[n]} k_i^{\mathrm{msk}}, \quad \text{and} \quad k^1 := k^0 \oplus \Delta_g,$$
   and then garble the identity function $(k^{0,\mathrm{out}}, k^{1,\mathrm{out}}, c^{\mathrm{out}}) \leftarrow \mathsf{IDF.Gb}(k^0, k^1)$.
7. Output the following:
   - Truth table $\mathcal{T}_r$,
   - Garbled ciphertexts $c := \left(c^{\mathrm{hot}}, (c_i^{\mathrm{even}}, c_i^{\mathrm{odd}})_{i\in[n]}, c_n, c^{\mathrm{out}}, p := (p_0, \cdots, p_{n-1})\right)$,
   - Output labels $(k^{0,\mathrm{out}}, k^{1,\mathrm{out}})$.

Procedure 9. $\mathsf{PRF.Ev}\left((k_i)_{i\in[n]}, c\right)$:

(0) Interpret $c = \left(c^{\mathrm{hot}}, (c_i^{\mathrm{even}}, c_i^{\mathrm{odd}})_{i\in[n]}, c_n, c^{\mathrm{out}}, (p_0, \cdots, p_{n-1})\right)$.

1. Reveal the input wire values: for all $i \in [n]$, let

$$y_i := \begin{cases} 0 & \text{if } \mathsf{lsb}(k_i) = p_i, \\ 1 & \text{otherwise.} \end{cases}$$

2. Obtain one-hot garbling $\mathbf{h}$ (Procedure 6):

$$\left(\mathbf{h} := (h_z)_{z\in\{0,1\}^n}\right) := \mathsf{OneHot.Ev}\left((k_i)_{i\in[n]}, c^{\mathrm{hot}}\right).$$

3. Decrypt the "half" seeds and "half" inner products: for all $i \in [n]$,

$$\begin{cases} \left(s_i^{\mathrm{even}} \| u_i^{\mathrm{odd}}\right) := \mathsf{F}'_{k_i}(g) \oplus c_i^{\mathrm{even}} & \text{if } y_i = 0, \\ \left(s_i^{\mathrm{odd}} \| u_i^{\mathrm{even}}\right) := \mathsf{F}'_{k_i}(g) \oplus c_i^{\mathrm{odd}} & \text{if } y_i = 1. \end{cases}$$

4. Compute the remaining "half" inner products from the seeds: for all $i \in [n]$,

$$\begin{cases} u_i^{\mathrm{even}} := \bigoplus_{z\in\mathsf{Even}(n,i)} \mathsf{F}_{s_i^{\mathrm{even}}}(z) \cdot h_z & \text{if } y_i = 0, \\ u_i^{\mathrm{odd}} := \bigoplus_{z\in\mathsf{Odd}(n,i)} \mathsf{F}_{s_i^{\mathrm{odd}}}(z) \cdot h_z & \text{if } y_i = 1. \end{cases}$$

5. Compose the inner products:

$$k := c_n \oplus \bigoplus_{i\in[n]} \left(u_i^{\mathrm{even}} \oplus u_i^{\mathrm{odd}}\right).$$

6. Solder the output wire: $k^{\mathrm{out}} := \mathsf{IDF.Ev}\left(k, c^{\mathrm{out}}\right)$.

7. Output k^{out}.

Remark 4.3 (Standalone garbled PRF)**.** The procedure SeedGen samples seeds and generates the truth table $\mathcal{T}_r$. Notice that, assuming F is a PRF, SeedGen samples a seed $s' = \left((s_i^{\mathrm{even}}, s_i^{\mathrm{odd}})_{i\in[n]}, r_n\right)$ and defines the PRF $r'_{s'} := r$ seeded by s'. Hence, $\mathcal{T}_r$ is exactly the truth table of $r'_{s'}$. Our procedures $(\mathsf{PRF.Gb}, \mathsf{PRF.Ev})$ can be augmented with proper encoding and decoding algorithms $(\mathsf{En}, \mathsf{De})$ in a direct way. Then, we would obtain a standalone garbling scheme for the PRF $r'_{s'}$.

Notice that in such garbled PRF, Evaluator decrypts either c_i^{even} or c_i^{odd} and learns part of s' yet $r'_{s'}$ (and thus $\mathcal{T}_r$) remains pseudorandom (this will be proved in our simulator for garbled LUTs, Section B). Hence, it would require a non-standard security game to define the security of the standalone garbled PRF. We leave the definition in future work.

Remark 4.4 (Hidding input of garbled PRF)**.** In the Garbler's procedure PRF.Gb, the input $\mathbf{y} = (y_0, \cdots, y_{n-1})$ is revealed to Evaluator by sending the permute bits p_i. Strictly speaking, the input should not be revealed. There is a standard black-box composition to hide the input: Garbler samples an additional mask $\mathbf{a}$ uniformly at random and defines the PRF to be $r'(z) := r(z \oplus \mathbf{a})$, and then Garbler reveals to Evaluator only $\mathbf{y} \oplus \mathbf{a}$. Evaluator will obtain the garbled output $\widetilde{r'(\mathbf{y} \oplus \mathbf{a})} = \widetilde{r(\mathbf{y})}$, as wanted. Actually, we will use the same technique later in the garbled lookup table (Section 4.4).

4.4 Garbled Lookup Table

In this subsection, we will describe the final garbled lookup table scheme $\mathsf{LUT}.\{\mathsf{KG}, \mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De}\}$ in the following Procedures 10, 11, 12, 13, and 14. The scheme adheres to Definition 3.4 (which we will

prove in Section B).

Procedure 10. $\mathsf{LUT.KG}(1^\lambda, 1^n)$:

1. For each input wire $i \in [n]$:
   (a) Choose two random keys $k_i^0, k_i^1 \leftarrow \{0,1\}^\lambda$ such that $\mathsf{lsb}(k_i^0) \oplus \mathsf{lsb}(k_i^1) = 1$.
   (b) Prepare the encoding function: $e[i,0] := k_i^0$, and $e[i,1] := k_i^1$.
2. Output e.

Procedure 11. $\mathsf{LUT.Gb}\left(f, (k_i^0, k_i^1)_{i\in[n]}\right)$:

1. Compute a mask $\mathbf{a} := \mathsf{lsb}(k_0^0) \| \mathsf{lsb}(k_1^0) \| \cdots \| \mathsf{lsb}(k_{n-1}^0)$.
2. Prepare one hot garbling (Procedure 5):
$$\left(\hat{\mathbf{h}}, \Delta_g, c^{\text{hot}}\right) \leftarrow \mathsf{OneHot.Gb}\left((k_i^{\mathbf{a}[i]}, k_i^{1-\mathbf{a}[i]})_{i\in[n]}\right).$$
3. Prepare a random lookup table (Procedure 8):
$$\left(\mathcal{T}_r, c^{\text{prf}}, (k^{0,\text{out}}, k^{1,\text{out}})\right) \leftarrow \mathsf{PRF.Gb}\left((k_i^{\mathbf{a}[i]}, k_i^{1-\mathbf{a}[i]})_{i\in[n]}\right).$$
4. Define a new function $f'(z) := f(z \oplus \mathbf{a}) \oplus r(z)$ and compute its truth table $\mathcal{T}_{f'}$ using $(\mathbf{a}, \mathcal{T}_r)$.
5. Compute $w^0 := \langle \mathcal{T}_{f'}, \hat{\mathbf{h}} \rangle$, and $w^1 := w^0 \oplus \Delta_g$.
6. Compute the output labels (Procedure 1):
$$\left((k^0, k^1), c^{\text{bin}}\right) \leftarrow \mathsf{Gate.Gb}\left((k^{0,\text{out}}, k^{1,\text{out}}), (w^0, w^1), \mathsf{XOR}\right).$$
7. Prepare the decoding function d: $d[0] := \hat{\mathsf{F}}_{k^0}(g)$, and $d[1] := \hat{\mathsf{F}}_{k^1}(g)$.
8. Output the following:
   (a) Decoding function d,
   (b) Garbled ciphertexts $F := (c^{\text{hot}}, c^{\text{prf}}, c^{\text{bin}}, \mathcal{T}_{f'})$.

Procedure 12. $\mathsf{LUT.En}(e, x)$:

1. For $i \in [n]$: $X[i] = e[i, x[i]]$.
2. Output X.

Procedure 13. $\mathsf{LUT.Ev}\left(F, (k_i)_{i\in[n]}\right)$:

(0) Interpret $F = (c^{\text{hot}}, c^{\text{prf}}, c^{\text{bin}}, \mathcal{T}_{f'})$.

1. Obtain one-hot garbling $\mathbf{h}$ (Procedure 6): $\mathbf{h} := \mathsf{OneHot.Ev}\left((k_i)_{i\in[n]}, c^{\text{hot}}\right)$.

2. Obtain the output label of the random lookup table (Procedure 9): $k^{\text{out}} := \mathsf{PRF.Ev}\left((k_i)_{i\in[n]}, c^{\text{prf}}\right)$.
3. Compute the output label for f': $w := \langle \mathcal{T}_{f'}, \mathbf{h} \rangle$.
4. Obtain the output label (Procedure 2): $k := \mathsf{Gate.Ev}(k^{\text{out}}, w, c^{\text{bin}})$.
5. Output $Y := \hat{\mathsf{F}}_k(g)$.

Procedure 14. $\mathsf{LUT.De}(d, Y)$:

1. Compute the output $y := \begin{cases} 0 & \text{if } d[0] = Y, \\ 1 & \text{if } d[1] = Y. \end{cases}$
2. Output y.

Lemma 4.5 (Communication costs.)**.** *The communication cost of our scheme is the following.*

- *One-hot garbling, Procedure 5:* $n + (2n+1)\lambda$
- *Garbled PRF, Procedure 8:* $2n + (6n+4)\lambda$.
- *Garbled LUT, Procedure 11, for any binary function* $f : \{0,1\}^n \to \{0,1\}$*:* $3n + (8n+9)\lambda + 2^n$.

All costs are measured in the number of bits communicated.

The security proof of our construction is in Appendix B.

Remark 4.6 (Optimizations)**.** The above cost can be slightly reduced by lightly modify the scheme. First, using the classic garbled row reduction technique[NPS99], we can further reduce the length of ciphertexts. Specifically, in Step 4b of Procedure 8, we can set $k_i^{\text{msk}} := \mathsf{F}'_{k_i^0}(g)[\lambda+1 : 2\lambda] \oplus t_i^{\text{odd}}$. As a result, the right half of c_i^{even} is guaranteed to be zero and does not need to be transmitted. This approach can save a total of $n\lambda$ bits of ciphertexts.

Second, as mentioned in Footnote 6 of Procedure 5, the permute bits (p_i) are instanciated 3 times, but they all reveal the same masked input $x \oplus \mathbf{a}$. Moreover, when f outputs m bits, the permute bits can be reused throughout m instances.

Third, we derive the half seeds $(s_i^{\text{even}}, s_i^{\text{odd}})$ from wire labels (k_i^0, k_i^1) using PRF instead of encrypting them (Step 4c of Procedure 8). That is, instead of encrypting s_i^{even} by $\mathsf{F}'_{k_i^0}(g) \oplus (s_i^{\text{even}} \| \ldots)$, we set s_i^{even} to the appropriate prefix of $\mathsf{F}'_{k_i^0}(g)$ directly; Symmetrically, we set s_i^{odd} to $\mathsf{F}'_{k_i^1}(g)$. There is no circular security: supposing that the label k_i^0 is given, both s_i^{even} and $k_i^{\text{msk}} \oplus t_i^{\text{odd}}$ are obtained, but k_i^1 remains independently random except for being a PRF key, and thus the security of F' holds. With that, we do not send the seeds, and it saves $2n\lambda$ bits.

With that, the cost is $n + (5n+9)m\lambda + 2^n m$ for $f : \{0,1\}^n \to \{0,1\}^m$.

5 Related Work

Since its introduction in 1980s by Yao [Yao86], garbled circuits have become a fundamental cryptographic primitive. The applications of garbled circuits include constant-round secure multiparty computation [BMR90, LP09, BLO16, BGH+23], verifiable computation [GGP10], key-dependent message security [BHHI10], zero-knowledge proofs [JKO13, FNO15], functional encryption [GKP+13], and more.

In [LP09], Lindell and Pinkas utilized garbled circuits to construct a secure two-party computation protocol and formally proved its security. Bellare et al., in [BHR12b], abstracted garbled circuits into a scheme and defined multiple security properties including privacy, obliviousness, and authenticity.

In the context of semi-honest adversaries, optimizing the ciphertexts overhead for individual gates is a significant research direction. Several important techniques have emerged, such as the garbled row reduction [NPS99, PSSW09, GLNP15], which reduces ciphertexts by setting some rows to zero—currently, using row reduction, an AND gate requires only 2λ bits of ciphertexts. In [KS08], Kolesnikov and Schneider introduced the Free-XOR technique, which eliminates the need for transmitting ciphertexts for XOR gates by using a global offset Δ. The state-of-the-art result, [RR21], Rosulek and Roy utilized "slicing and dicing" to garble an AND gate with just 1.5λ bits, while XOR gates remain free. However, optimizations for garbling single gates seem to have reached their limits, as evidenced by the numerous lower bounds that have emerged recently. For instance, in [ZRE15], Zahur et al. demonstrated that garbling an AND gate in the linear model requires at least 2λ bits of ciphertexts.

Many works have been dedicated to enhancing security, particularly against malicious or adaptive adversaries. In [LP07], Lindell and Pinkas employed the cut-and-choose technique to construct garbled circuits against malicious corruption. In [WRK17], Wang et al. introduced authenticated garbling, which significantly improved the efficiency in scenarios with malicious adversaries. Bellare et al. detailed garbled circuits capable of resisting adaptive adversaries within the programmable random oracle model in [BHR12a]. Furthermore, in [HJO+16], Hemenway et al. proposed a scheme that are resistant to adaptive attacks, based on somewhere equivocal encryption and dependent on the existence of one-way functions.

Additionally, many studies concentrate on garbling large circuits, rather than single binary gates, on achieving better amortized ciphertexts overhead. For example, in [AIK11], Applebaum et al. achieved garbling for arithmetic circuits based on the LWE assumption. In [LO13], Lu and Ostrovsky pioneered garbled circuits within the RAM model, catering particularly to programs that necessitate frequent read/write operations. In [HK20], Heath and Kolesnikov introduced stacked garbling, where the garbling overhead for circuits with multiple branches is proportional only to the length of the longest branch. Subsequently, in [HK21], Heath and Kolesnikov introduced one-hot garbling and reduced the ciphertexts overhead for a series of linear operations, such as matrix multiplication. Specifically, in a recent work [HKN24], Heath et al. introduced an efficient garbling approach, which only requires a logarithmic number of ciphertexts for garbling a lookup table.

Acknowledgements

We thank the anonymous reviewers for their constructive comments; in particular, the TCC 2024 reviewers for identifying a security issue and suggesting an optimization.

Hong-Sheng Zhou was supported in part by NSF grant CNS-1801470 and the VCU Quest Fund.

References

[AAC+23] Anasuya Acharya, Tomer Ashur, Efrat Cohen, Carmit Hazay, and Avishay Yanai. A new approach to garbled circuits. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23, Part II*, volume 13906 of *LNCS*, pages 611–641. Springer, Heidelberg, June 2023. 1

[AHS24] Tomer Ashur, Carmit Hazay, and Rahul Satish. On the feasibility of sliced garbling. Cryptology ePrint Archive, Paper 2024/389, 2024. https://eprint.iacr.org/2024/389. 1

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004. 1

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129. IEEE Computer Society Press, October 2011. 1, 22

[App13] Benny Applebaum. Garbling XOR gates "for free" in the standard model. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 162–181. Springer, Heidelberg, March 2013. 2

[App17] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 1–44. Springer International Publishing, 2017. 1

[BDH14] Florian Böhl, Gareth T. Davies, and Dennis Hofheinz. Encryption schemes secure under related-key and key-dependent message attacks. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 483–500. Springer, Heidelberg, March 2014. 2

[BGH+23] Gabrielle Beck, Aarushi Goel, Aditya Hegde, Abhishek Jain, Zhengzhong Jin, and Gabriel Kaptchuk. Scalable multiparty garbling. In *ACM CCS 2023*, pages 2158–2172. ACM Press, November 2023. 21

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 423–444. Springer, Heidelberg, May / June 2010. 1, 21

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, Heidelberg, December 2012. 22

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012. 1, 13, 22

[BK24] Chunghun Baek and Taechan Kim. Can We Beat Three Halves Lower Bound? (Im)Possibility of Reducing Communication Cost for Garbled Circuits. 2024. Manuscript. 1

[BLO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 578–590. ACM Press, October 2016. 1, 21

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990. 1, 21

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012. 2

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, February 2005. 3

[FLZ24] Lei Fan, Zhenghao Lu, and Hong-Sheng Zhou. Column-wise garbling, and how to go beyond the linear model. *IACR Cryptol. ePrint Arch.*, 2024. https://eprint.iacr.org/2024/415. 1

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, Heidelberg, April 2015. 1, 21

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984. 5

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, August 2010. 1, 21

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013. 1, 21

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 567–578. ACM Press, October 2015. 1, 2, 3, 15, 22

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. 2, 4

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016. 22

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 763–792. Springer, Heidelberg, August 2020. 1, 22

[HK21] David Heath and Vladimir Kolesnikov. One hot garbling. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 574–593. ACM Press, November 2021. 1, 4, 12, 22

[HKN24] David Heath, Vladimir Kolesnikov, and Lucien K. L. Ng. Garbled circuit lookup tables with logarithmic number of ciphertexts. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 185–215. Springer, Switzerland, May 2024. 2, 1, 3, 4, 5, 8, 12, 22

[HKO23] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. Tri-state circuits - A circuit model that captures RAM. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 128–160. Springer, Heidelberg, August 2023. 1

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, November 2000. 1

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013. 1, 21

[Kim24] Taechan Kim. Analysis on sliced garbling via algebraic approach. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VIII*, volume 15491 of *Lecture Notes in Computer Science*, pages 237–265. Springer, 2024. 1

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007. 27

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014. 1, 12

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008. 1, 2, 12, 22

[LGW24] Ruiyang Li, Chun Guo, and Xiao Wang. Towards optimal garbled circuits in the standard model. Cryptology ePrint Archive, Paper 2024/1907, 2024. 1

[LLP25] Liqiang Liu, Tianren Liu, and Bo Peng. Garbled lookup tables from homomorphic secret sharing. Cryptology ePrint Archive, Paper 2025/254, 2025. 3

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 719–734. Springer, Heidelberg, May 2013. 1, 22

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, Heidelberg, May 2007. 22

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009. 1, 2, 21, 22

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, Heidelberg, March 2009. 9

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999. 1, 15, 21, 22

[PLS23] Andrew Park, Wei-Kai Lin, and Elaine Shi. NanoGRAM: Garbled RAM with $\widetilde{O}(\log N)$ overhead. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 456–486. Springer, Heidelberg, April 2023. 1

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009. 1, 22

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*,

volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Heidelberg. 1, 2, 22

[RRG+21] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. SiRnn: A math library for secure RNN inference. In *2021 IEEE Symposium on Security and Privacy*, pages 1003–1020. IEEE Computer Society Press, May 2021. 1

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 21–37. ACM Press, October / November 2017. 22

[XHX24] Fei Xu, Honggang Hu, and Changhong Xu. Bitwise garbling schemes — a model with $\frac{3}{2}\kappa$-bit lower bound of ciphertexts. Cryptology ePrint Archive, Paper 2024/1532, 2024. https://eprint.iacr.org/2024/1532. 1

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. 1, 21

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015. 1, 22

A Additional Preliminaries

A.1 Garbling Schemes

The security properties of garbling schemes are defined as follows:

- **Privacy with respect to leakage** Φ: There must exist a simulator $\mathsf{Sim}_{\mathsf{priv}}$ such that for any functionality f and input x the following distributions are computationally indistinguishable.

The Privacy game with respect to leakage Φ

01 $e \leftarrow \mathsf{KG}(1^\lambda, 1^n)$ 02 $(F, d) \leftarrow \mathsf{Gb}(f, e)$ 03 $X := \mathsf{En}(e, x)$ 04 return (F, X, d)	01 $(F, X, d) \leftarrow \mathsf{Sim}_{\mathsf{priv}}(1^\lambda, \Phi(f), f(x))$ 02 return (F, X, d)

In our garbled LUTs, we consider only the leakage of input and output sizes, that is, for any $f : \{0,1\}^n \to \{0,1\}^m$, $\Phi(f) = (1^n, 1^m)$.

- **Obliviousness w.r.t. leakage** Φ: There must exist a simulator $\mathsf{Sim}_{\mathsf{obliv}}$ such that for any functionality f and input x the following distributions are computationally indistinguishable.

The Obliviousness game w.r.t. leakage Φ

01 $e \leftarrow \mathsf{KG}(1^\lambda, 1^n)$ 02 $(F, d) \leftarrow \mathsf{Gb}(f, e)$ 03 $X := \mathsf{En}(e, x)$ 04 return (F, X)	01 $(F, X) \leftarrow \mathsf{Sim}_{\mathsf{obliv}}(1^\lambda, \Phi(f))$ 02 return (F, X)

- **Authenticity**: For any functionality f and input x, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the following distribution outputs TRUE with all but negligible probability.

The Authenticity game

$(e) \leftarrow \mathsf{KG}(1^\lambda, 1^n)$
$(F, d) \leftarrow \mathsf{Gb}(f, e)$
$X := \mathsf{En}(e, x)$
$\tilde{Y} \leftarrow \mathcal{A}(F, X)$
return $\mathsf{De}(d, \tilde{Y}) \notin \{f(x), \perp\}$

A.2 Pseudorandom Generators and Functions

For completeness, below we describe the definitions for Pseudorandom Generators (PRG) and Pseudorandom Functions (PRF). We remark that these definitions are taken from the textbook by Katz and Lindell [KL07].

Definition A.1 (Pseudorandom generator)**.** *Let* G *be a deterministic polynomial-time algorithm such that for any* λ *and any input* $s \in \{0,1\}^\lambda$*, the result* $\mathsf{G}(s)$ *is a string of length* $\ell(\lambda)$*.* G *is a pseudorandom generator if the following conditions hold:*

1. *(Expansion.) For every* λ *it holds that* $\ell(\lambda) > \lambda$*.*
2. *(Pseudorandomness.) For any PPT algorithm* D*, there is a negligible function* negl *such that*

$$|\Pr[D(\mathsf{G}(s)) = 1] - \Pr[D(r) = 1]| \leq \mathsf{negl}(\lambda),$$

where the first probability is taken over uniform choice of $s \in \{0,1\}^\lambda$ *and the randomness of* D*, and the second probability is taken over uniform choice of* $r \in \{0,1\}^{\ell(\lambda)}$ *and the randomness of* D*.*

Definition A.2 (Pseudorandom function)**.** *An efficient, length preserving, keyed function* $\mathsf{F} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *is a pseudorandom function if for all probabilistic polynomial-time distinguishers* D*, there is a negligible function* negl *such that:*

$$|\Pr[D^{\mathsf{F}_k(\cdot)}(1^\lambda) = 1] - \Pr[D^{f(\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where the first probability is taken over uniform choice of $k \in \{0,1\}^\lambda$ *and the randomness of* D*, and the second probability is taken over uniform choice of* $f \in \mathsf{Func}_\lambda$ *and the randomness of* D*, where* Func_λ *denotes the set of all functions mapping* λ*-bit strings to* λ*-bit strings.*

B Security Proof

In the garbled lookup table $\mathsf{LUT} = (\mathsf{LUT.KG}, \mathsf{LUT.Gb}, \mathsf{LUT.En}, \mathsf{LUT.Ev}, \mathsf{LUT.De})$, we employed three different PRFs and one PRG. The three PRFs are $\hat{\mathsf{F}}_k : \{0,1\}^* \to \{0,1\}^\lambda$, $\mathsf{F}_k : \{0,1\}^n \to \{0,1\}$, and $\mathsf{F}'_k : \{0,1\}^* \to \{0,1\}^{2\lambda}$, where k is uniformly sampled from $\{0,1\}^\lambda$, while the PRG is $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$.

Theorem B.1. *If* $\hat{\mathsf{F}}$*,* F*, and* F' *are pseudorandom functions, and* G *is a pseudorandom generator, then the garbling scheme* LUT *of the lookup table* $f : \{0,1\}^n \to \{0,1\}$ *is secure.*

Proof. We will prove that our scheme satisfies privacy, obliviousness, and authenticity.

Privacy. To prove privacy, we need to construct a simulator $\mathsf{Sim}_{\mathsf{priv}}$ such that

$$\mathsf{Sim}_{\mathsf{priv}}(1^\lambda, 1^n, f(x)) \approx_c (F, X, d),$$

where $e \leftarrow \mathsf{KG}(1^\lambda, 1^n), (F, d) \leftarrow \mathsf{Gb}(f, e),$ and $X \leftarrow \mathsf{En}(e, x)$.

In other words, $\mathsf{Sim}_{\mathsf{priv}}$ needs to simulate the garbled ciphertexts F, the garbled input X, and the output decoding d using only the lookup table output $f(x)$ and the input length n. Note that the actual input x and the table f are hidden from $\mathsf{Sim}_{\mathsf{priv}}$.

Procedure $\mathsf{Sim}_{\mathsf{priv}}(1^\lambda, 1^n, f(x))$**:**

1. For each input wire $i \in [n]$ of the lookup table, choose a 0-label for wire i: $k_i^0 \leftarrow \{0,1\}^\lambda$, and define the garbled input $X[i] := k_i^0$.
2. Compute the mask $\mathbf{a} := \mathsf{lsb}(k_0^0) \| \mathsf{lsb}(k_1^0) \| \cdots \| \mathsf{lsb}(k_{n-1}^0)$.
3. Simulate the one-hot garbling ($\mathsf{OneHot.Gb}$, Procedure 5) as follows:
   (a) Set permute bits $p_i := 0$ for $i \in [n]$ (see Footnote 6).
   (b) Generate the GGM tree as in the real world.
   (c) Replace seeds to uniformly random on the path from root to leaf $\mathbf{a}$ in the GGM tree: For $i \in \{1, 2 \cdots, n-1\}$, sample random seeds $s_{i+1,\mathbf{a}[0:i]} \leftarrow \{0,1\}^\lambda$. (Initial seed $s_{1,\mathbf{a}[0]}$ is always randomly generated.)
   (d) Simulate two ciphertexts for each level: for $i \in [n]$,
   $$\begin{cases} c_{i,0} \leftarrow \{0,1\}^\lambda, \quad c_{i,1} := \hat{\mathsf{F}}_{k_i^0}(g) \oplus (\bigoplus_{j \in [2^i]} s_{i+1,2j+1}) & \text{if } \mathbf{a}[i] = 0, \\ c_{i,0} := \hat{\mathsf{F}}_{k_i^0}(g) \oplus (\bigoplus_{j \in [2^i]} s_{i+1,2j}), \quad c_{i,1} \leftarrow \{0,1\}^\lambda & \text{if } \mathbf{a}[i] = 1. \end{cases}$$
   (e) Compute the last ciphertext $c_n := \bigoplus_{i \in [2^n]} s_{n,i}$.
   (f) Define garbled ciphertexts
   $$c^{\mathrm{hot}} := \left((c_{i,0}, c_{i,1})_{i \in [n]}, c_n, d := (d_0, \cdots, d_{n-1}) \right).$$
4. Simulate the PRF garbling by modifying $\mathsf{PRF.Gb}$ (Procedure 8) as follows.
   (a) *Modify Step 1 of Procedure 8*: for all $i \in [n]$, set permute bit $p_i := 0$.
   (b) *Modify Step 2 of Procedure 8*: simulate the one-hot garbling using the above Step 2 of this simulator.
   (c) *Modify Step 4c of Procedure 8*: simulate two ciphertexts for $i \in [n]$,
   $$\begin{cases} c_i^{\mathrm{even}} := \mathsf{F}'_{k_i^0}(g) \oplus \left(s_i^{\mathrm{even}} \| k_i^{\mathrm{msk}} \right), \quad c_i^{\mathrm{odd}} \leftarrow \{0,1\}^\lambda & \text{if } \mathbf{a}[i] = 0, \\ c_i^{\mathrm{even}} \leftarrow \{0,1\}^\lambda, \quad c_i^{\mathrm{odd}} := \mathsf{F}'_{k_i^0}(g) \oplus \left(s_i^{\mathrm{odd}} \| k_i^{\mathrm{msk}} \right) & \text{if } \mathbf{a}[i] = 1. \end{cases}$$
   (d) Output $(\mathcal{T}_r, c^{\mathrm{prf}})$.
5. Randomly choose a truth table $\mathcal{T}_{f'} \leftarrow \{0,1\}^{2^n}$.
6. Run evaluation procedure for one hot garbling (Procedure 6): $\mathbf{h} := \mathsf{EvOneHot}\left((k_i^0)_{i \in [n]}, c^{\mathrm{hot}} \right)$.
7. Run evaluation procedure for PRF garbling (Procedure 9): $w_0 := \mathsf{EvPRF}\left((k_i)_{i \in [n]}, c^{\mathrm{prf}} \right)$.
8. Compute the output label for f': $w_1 := \langle \mathcal{T}_{f'}, \mathbf{h} \rangle$.

9. Simulate the XOR gate garbling procedure (refer to Procedure 1):
   (a) Choose a random label $k \leftarrow \{0,1\}^\lambda$.
   (b) Simulate four ciphertexts: for $\alpha \in \{0,1\}$ and $\beta \in \{0,1\}$,
   $$\begin{cases} \tilde{c}_{\alpha,\beta} := \hat{\mathsf{F}}_{w_0}(g\|\alpha\|\beta) \oplus \hat{\mathsf{F}}_{w_1}(g\|\alpha\|\beta) \oplus k & \text{if } (\alpha,\beta) = (\mathsf{lsb}(w_0), \mathsf{lsb}(w_1)), \\ \tilde{c}_{\alpha,\beta} \leftarrow \{0,1\}^\lambda & \text{otherwise.} \end{cases}$$
   (c) Define $c^{\text{bin}} := (\tilde{c}_{0,0}, \tilde{c}_{0,1}, \tilde{c}_{1,0}, \tilde{c}_{1,1})$.
10. Let $F := (c^{\text{hot}}, c^{\text{prf}}, c^{\text{bin}}, \mathcal{T}_{f'})$. Define the output decoding set $d[f(x)] = \hat{\mathsf{F}}_k(g)$ and $d[1-f(x)] \leftarrow \{0,1\}^\lambda$.
11. Finally, output (F, X, d).

We use hybrid arguments to prove that the real world and the ideal world are indistinguishable. Suppose w.l.o.g. that in the real world, the garbled input X consists of active labels $k_0^{\mathbf{y}[0]}, k_1^{\mathbf{y}[1]}, \cdots, k_{n-1}^{\mathbf{y}[n-1]}$, where $\mathbf{y} = \mathbf{x} \oplus \mathbf{a}$ is the masked input and revealed to Evaluator, while the inactive labels $k_0^{1-\mathbf{y}[0]}, k_1^{1-\mathbf{y}[1]}, \cdots, k_{n-1}^{1-\mathbf{y}[n-1]}$ are unknown to Evaluator. In the real world, the ciphertexts $c_{i,0}$, $c_{i,1}$, c_i^{even}, and c_i^{odd}, where $i \in [n]$, are encrypted using both labels (k_i^0, k_i^1). We first use n hybrids to prove that those ciphertexts encrypted with inactive labels can be replaced with random values.

HYBRID$_0$: This is exactly the real world.

HYBRID$_i$, $i \in \{1, \cdots, n\}$: In HYBRID$_i$, for all $0 \le j < i$, the ciphertexts $c_{j,0}$, $c_{j,1}$, c_j^{even}, and c_j^{odd} encrypted by inactive labels are replaced with random strings, while for $i \le j < n$, all ciphertexts are generated using pseudorandom functions as in the real world.

Next, we show the indistinguishability between HYBRID$_i$ and HYBRID$_{i+1}$, where $0 \le i < n$. In HYBRID$_i$, all ciphertexts $c_{i,0}$, $c_{i,1}$, c_i^{even}, and c_i^{odd} are encrypted using PRFs (such as $\hat{\mathsf{F}}_{k_i^0}(g) \oplus \mu$ or $\mathsf{F}'_{k_i^0}(g) \oplus \mu$ for some message μ). In HYBRID$_{i+1}$, the inactive ciphertexts (encrypted by inactive labels $k_i^{1-\mathbf{y}[i]}$ in HYBRID$_i$) are replaced with random strings. Notice that in HYBRID$_i$, the inactive label $k_i^{1-\mathbf{y}[i]}$ is only used as the PRF key in these ciphertexts, while everything else is independent of $k_i^{1-\mathbf{y}[i]}$.[7] Therefore, by the pseudorandomness of PRFs $\hat{\mathsf{F}}$ and F', HYBRID$_i$ and HYBRID$_{i+1}$ are computationally indistinguishable.

HYBRID$_{n+i}$, $i \in \{1, \cdots, n\}$: For each $0 \le i < n$, HYBRID$_{n+i+1}$ is similar to HYBRID$_{n+i}$ except for the following modification.

- If $\mathbf{y}[i] = 0$ (i.e., k_i^0 is active), then replace $\mathsf{F}_{s_i^{\text{odd}}}(\cdot)$ with a random $\{0,1\}^n \to \{0,1\}$ function (which is a 2^n-bit random string). This modifies Step 1b of Procedure 7 and Step 4a of Procedure 8 (while PRF.Ev is unchanged).
- Otherwise, if $\mathbf{y}[i] = 1$, then replace $\mathsf{F}_{s_i^{\text{even}}}(\cdot)$ with a random function symmetrically.

We need to prove that HYBRID$_{n+i}$ and HYBRID$_{n+i+1}$ are indistinguishable, which can be reduced to the security of F. Specifically, suppose that $\mathbf{y}[i] = 0$ without loss of generality. Then, $\mathsf{F}_{s_i^{\text{odd}}}(\cdot)$ in HYBRID$_{n+i}$ is replaced by a random function in HYBRID$_{n+i+1}$. Observe that in both hybrids HYBRID$_{n+i}$ and HYBRID$_{n+i+1}$, s_i^{odd} is only used as the key of F (recall that s_i^{odd} is already erased from c_i^{odd} in earlier hybrids since k_i^1 is inactive). Therefore, by the pseudorandomness of F, HYBRID$_{n+i}$ and HYBRID$_{n+i+1}$ are computationally indistinguishable.

[7] Actually, the least significant bit of $k_i^{1-\mathbf{y}[i]}$ should be different from $\mathsf{lsb}(k_i^{\mathbf{y}[i]})$, that is, we lose 1-bit security.

HYBRID_{2n+1}: We modify HYBRID_{2n} as follows. For $i \in \{0, 1, \cdots, n-1\}$,

- If $\mathbf{y}[i] = 0$ (i.e., k_i^0 is active), then in Step 4c of Procedure 8, replace $\mathsf{F}'_{k_i^0}(g) \oplus \left(s_i^{\text{even}} \| k_i^{\text{msk}} \oplus t_i^{\text{odd}}\right)$ with $\mathsf{F}'_{k_i^0}(g) \oplus \left(s_i^{\text{even}} \| k_i^{\text{msk}}\right)$.
- Otherwise, $\mathbf{y}[i] = 0$, replace $\mathsf{F}'_{k_i^1}(g) \oplus \left(s_i^{\text{odd}} \| k_i^{\text{msk}} \oplus t_i^{\text{even}}\right)$ with $\mathsf{F}'_{k_i^1}(g) \oplus \left(s_i^{\text{odd}} \| k_i^{\text{msk}}\right)$.

Since k_i^{msk} is a fresh random string, HYBRID_{2n} is identical to HYBRID_{2n+1}. Note that we cannot move this hybrid before HYBRID_n because c_i^{even} and c_i^{odd} share the same k_i^{msk}, we cannot replace both $k_i^{\text{msk}} \oplus t_i^{\text{odd}}$ and $k_i^{\text{msk}} \oplus t_i^{\text{even}}$ in the two ciphertexts with k_i^{msk}. However, if one of the ciphertexts has already been replaced with a random string, then we can safely complete the replacement of the other ciphertext (i.e., k_i^{msk} now serves as a one-time pad).

HYBRID_{2n+2}: In this hybrid, we replace the function r and its truth table $\mathcal{T}_r$ with a random table, sampled uniformly from $\{0,1\}^{2^n}$ (modifying Step 3 of sub-Procedure 7).

In HYBRID_{2n+1}, r is defined by $r(z) := r_n \oplus \bigoplus_{i \in [n]} r_i(z)$. For each r_i, where $i \in [n]$, Evaluator is aware of half of the truth table, while the other half is generated by a uniformly random function. We want to prove that the truth table $\mathcal{T}_r$, obtained in this manner, appears uniformly random to Evaluator. We prove it by enumerating all $z \in \{0,1\}^n$. If z equals the masked input $\mathbf{y}$, then Evaluator knows $r_i(\mathbf{y})$ for all $i \in [n]$ but not r_n, and thus $r(\mathbf{y})$ is identical to uniform. Otherwise, $z \neq \mathbf{y}$, then there exists $j \in [n]$ such that $z[j] = 1 - \mathbf{y}[j]$, and thus $r_j(z)$ is uniformly random (by the modification of the above HYBRID_{n+j}). The truth table $\mathcal{T}_r$ is identical to uniform, and thus HYBRID_{2n+1} and HYBRID_{2n+2} are identically distributed.

HYBRID_{2n+3}: In the previous hybrid, the function f' is defined as $f'(z) := f(z \oplus \mathbf{a}) \oplus r(z)$. Now, we randomly sample its truth table $\mathcal{T}_{f'}$ from $\{0,1\}^{2^n}$.

Since $\mathcal{T}_r$ is uniformly random, $\mathcal{T}_{f'} = \mathcal{T}_{f(z \oplus \mathbf{a})} \oplus \mathcal{T}_r$ is also uniform. Hence, HYBRID_{2n+2} and HYBRID_{2n+3} are identically distributed.

HYBRID_{2n+4+i}, $i \in \{0, \cdots, n-2\}$: We modify the generation process of the GGM tree from the real world to the ideal world. In the real world, the entire tree is generated based on the PRG G. In the ideal world, the seeds on the path from the root to the leaf $\mathbf{y}$ are replaced with random strings. We begin the replacement from level 2 to level n, corresponding to $n-1$ hybrids.

Next, we show the indistinguishability between HYBRID_{2n+3+i} and HYBRID_{2n+4+i}, where $0 \leq i < n-1$. In HYBRID_{2n+3+i}, the seed $s_{i+2,\mathbf{a}[0:i+1]}$ is generated by $\mathsf{G}(s_{i+1,\mathbf{a}[0:i]})$, while in HYBRID_{2n+4+i}, the seed is randomly sampled. Because $s_{i+1,\mathbf{a}[0:i]}$ is never used (recall that $c_{i,\mathbf{a}[i]}$ no longer use the seed $s_{i+1,\mathbf{a}[0:i]}$ after HYBRID_n), by the pseudorandomness of G, these two hybrids are computationally indistinguishable.

HYBRID_{3n+3}: In the one hot garbling procedure, we no longer generate Δ_g. We simply compute $\bigoplus_{i \in [2^n]} s_{n,i}$ as the ciphertext c_n.

Since the random seed $s_{n,\mathbf{a}[0:n-1]}$ remains hidden from Evaluator, the distributions of HYBRID_{3n+2} and HYBRID_{3n+3} are identical.

HYBRID_{3n+4}: In this hybrid, the four ciphertexts for the XOR gate will be generated using the method described in Step 5 of the simulator $\mathsf{Sim}_{\mathsf{priv}}$.

Assume Evaluator obtains the active labels w_0 and w_1 for the two wires, while the two inactive labels w'_0 and w'_1 are uniformly random to her (since the offset Δ_g does not appear in her view). Compared to HYBRID_{3n+3}, in HYBRID_{3n+4} we replace the three ciphertexts encrypted with inactive labels with

random strings. Specifically, we first use the security of $\hat{\mathsf{F}}_{w_0'}$ to replace two of the ciphertexts with random strings. Finally, we use the security of $\hat{\mathsf{F}}_{w_1'}$ to replace the remaining ciphertext with a random string.

HYBRID_{3n+5}**:** We replace the decoding function $d[1-f(x)]$ with a random value from $\{0,1\}^\lambda$.

Assume the inactive label of the XOR gate's output wire is k'. Since k' is hidden from Evaluator, we can perform this modification based on the security of the PRF $\hat{\mathsf{F}}$.

Finally, we argue that the simulator $\mathsf{Sim}_{\mathsf{priv}}$ can use a uniform $\mathbf{a}$ instead of $\mathbf{y} = \mathbf{x} \oplus \mathbf{a}$ as in HYBRID_{3n+4}. In the simulator $\mathsf{Sim}_{\mathsf{priv}}$, the garbled input X consists of all 0-labels. We consider a different construction where $\mathsf{Sim}_{\mathsf{priv}}$ is aware of the actual input x. Specifically, Step 1 of $\mathsf{Sim}_{\mathsf{priv}}$ chooses $k_i^{x[i]} \leftarrow \{0,1\}^\lambda$ and defines $X[i] := k_i^{x[i]}$ for $i \in [n]$. In other words, the simulator $\mathsf{Sim}_{\mathsf{priv}}$ tracks active input labels instead of 0-labels. The distribution after this replacement is identical to HYBRID_{3n+5}. This concludes the proof of privacy.

Obliviousness. To prove obliviousness, we need to construct a simulator $\mathsf{Sim}_{\mathsf{obliv}}$ such that

$$\mathsf{Sim}_{\mathsf{obliv}}(1^\lambda, 1^n) \approx_c (F, X),$$

where $e \leftarrow \mathsf{KG}(1^\lambda, 1^n), (F, d) \leftarrow \mathsf{Gb}(f, e),$ and $X \leftarrow \mathsf{En}(e, x)$.

The simulator $\mathsf{Sim}_{\mathsf{obliv}}$ is very similar to $\mathsf{Sim}_{\mathsf{priv}}$. Recall that $\mathsf{Sim}_{\mathsf{priv}}$ has an input $f(x)$, which is only used in Step 10. The generation of garbled ciphertexts F and garbled input X does not depend on the information of $f(x)$. Therefore, we can simply remove the generation of the decoding function from $\mathsf{Sim}_{\mathsf{priv}}$ to obtain $\mathsf{Sim}_{\mathsf{obliv}}$. The method to prove that (F, X) output by $\mathsf{Sim}_{\mathsf{obliv}}$ in the ideal world is indistinguishable from the real world is the same as the proof for privacy.

Authenticity. To prove authenticity, we need to show that for any PPT adversary $\mathcal{A}$, given (F, X), the probability of generating a $\tilde{Y}$ such that $\mathsf{De}(d, \tilde{Y}) \notin \{f(x), \perp\}$ is negligible. If the pair (F, X) given to $\mathcal{A}$ is not generated in the real world but by the simulator $\mathsf{Sim}_{\mathsf{priv}}$, then the adversary's success probability is $2^{-\lambda}$, since $d[1-f(x)]$ is randomly generated. If such an adversary $\mathcal{A}$ exists that can generate $\tilde{Y}$ with a non-negligible probability such that $\mathsf{De}(d, \tilde{Y}) \notin \{f(x), \perp\}$, then Evaluator can invoke this adversary $\mathcal{A}$ to break the privacy property. □