Garbled Lookup Tables from Homomorphic Secret Sharing

Liqiang Liu*, Tianren Liu†, and Bo Peng‡

Peking University

February 16, 2025

Abstract

Garbled Circuit (GC) is a fundamental tool in cryptography, especially in secure multiparty computation. Most garbling schemes follow a gate-by-gate paradigm. The communication cost is proportional to the circuit size times the security parameter λ.

Recently, Heath, Kolesnikov and Ng (Eurocrypt 2024) partially transcend the circuit size barrier by considering large gates. To garble an arbitrary n-input m-output gate, their scheme requires $O(nm\lambda) + 2^n m$ bits of communication. The security relies on circular correlation robust hash functions (CCRH).

We further improve the communication cost to $O(n\lambda_{\mathsf{DCR}} + m\lambda)$, removing the exponential term. The computation cost is $O(2^n(\lambda_{\mathsf{DCR}})^2)$, dominated by $O(2^n n)$ exponentiations. Our construction is built upon recent progress in DCR-based Homomorphic Secret Sharing (HSS), so it additionally relies on the decisional composite residuosity (DCR) assumption.

As an intermediate step, we construct programmable distributed point functions with decomposable keys, relying on the DCR assumption. Previously, such primitive can only be constructed from multilinear maps or sub-exponential lattice assumptions.

1 Introduction

Garbled circuit (GC), introduced in the seminal work of Yao [Yao82], is one of the most important technique in cryptography. GC allows a *garbler* to efficiently transform a boolean circuit C into a *garbled circuit* $\tilde{C}$, along with a simple (usually linear) mapping that maps any input x into its corresponding label L. If an *evaluator* is given the garbled circuit $\tilde{C}$ and the label L, it can efficiently compute $C(x)$ and nothing else about x.

*lql@pku.edu.cn

†trl@pku.edu.cn

‡bo.peng@stu.pku.edu.cn

GC enables constant-round practical multiparty secure computation. The bottleneck is usually the communication cost, in particular, the size of the garbled circuit. The textbook Yao's GC requires $O(|C| \cdot \lambda)$ bits of communication, where $|C|$ denotes the Boolean circuit size and λ denotes the security parameter. Since then, there has been a considerable amount of works [BMR90, NPS99, KS08, PSSW09, KMR14, GLNP18, ZRE15, RR21] dedicated to optimize the *concrete* efficiency of Yao's GC construction. These works bind tightly with the Boolean circuits basing on 2-input 1-output gates. In the state-of-the-art construction of Rosulek and Roy [RR21], XOR and NOT gates are free, while every AND gate requires $1.5\lambda + 5$ bits of communication.

To get around the circuit size barrier, Heath, Kolesnikov and Ng [HKN24] initialize the study of directly garbling large gates. Their communication cost of garbling an arbitrary n-input m-output gate is $2^n m + O(nm\lambda)$, saving roughly a factor of λ compared with the traditional gate-by-gate garbling.

	Communication	**Computation**	**Assumpt.**	**Hide f**
Ours	$O(n\lambda_{\mathsf{DCR}} + m\lambda)$	$O(N\lambda_{\mathsf{DCR}} c_{\text{mult}} + Nm\lambda_{\mathsf{DCR}})$	CCR & DCR	
[HKN24]	$O(nm\lambda + Nm)$	$O((N(1+\frac{m}{\lambda}) + nm)c_{\text{hash}} + Nm\lambda)$	CCR	○
Yao + [BPP00]	$O(\sqrt{N}m\lambda)$	$O(Nmc_{\text{hash}})$	CCR	
SGC [HK21b]	$O(n^2\lambda + nm\lambda)$	$O(N^{2.389}mc_{\text{hash}})$	CCR	
GPIR [HHK+22]	$\tilde{O}(\sqrt{N}m\lambda)$	$\tilde{O}(Nmc_{\text{hash}})$	CCR	
GRAM [PLS23]	$\tilde{O}(nm\lambda + n^3\lambda)$ amortized	$\tilde{O}(nmc_{\text{hash}} + n^3 c_{\text{hash}})$ amortized	CCR	○

Table 1: Comparison of communication and computation complexities for different approaches for computing $[\![x]\!] \mapsto [\![f(x)]\!]$ inside GC where $f : \{0,1\}^n \to \{0,1\}^m$ is a function with $N = 2^n$ possible inputs. The GRAM approach amortizes the cost over $\Omega(N)$ function evaluation. c_{hash} denotes the cost of evaluating a hash function. c_{mult} denotes the cost of multiplying two λ_{DCR}-bit integers.

Our Contribution. We further unbind the communication cost from the circuit size. Our new scheme only requires $O(n\lambda_{\mathsf{DCR}}+m\lambda)$ bits to garble a n-input m-output gate, where λ_{DCR} denotes the required bit-length of the RSA modulus to achieve λ-bit security in the Decision Composite Residuosity (DCR) assumption. Compared to [HKN24], we require an additional computational assumption as we borrow technique from recent progresses [OSY21, RS21] in Homomorphic Secret Sharing.

Theorem 1 (Main theorem, informal)**.** *Assuming the decisional composite residuosity (DCR) assumption, there is a GC extension for garbling arbitrary n-bit-input m-bit-output gates in the random oracle model and is compatible with free-XOR. The communication cost per such gate is $O(n\lambda_{\mathsf{DCR}} + m\lambda)$. The computation cost is $O(N{\lambda_{\mathsf{DCR}}}^2 + Nm\lambda_{\mathsf{DCR}})$, including $O(2^n n)$ exponentiations* [1].

[1]The time of multiplying two λ_{DCR}-bit integers is $c_{\text{mult}} = \lambda_{\mathsf{DCR}}$ in the Word RAM model.

The key step of our construction is to garble "one-hot" gates. A n-input 2^n-output one-hot gate maps input x to a long output vector in which only the x-th bit is 1. The communication complexity to garble a one-hot gate is $O(n\lambda_{\mathsf{DCR}})$. Our garbling is not fully compatible with free-XOR, otherwise it would imply the garbling of any n-input m-output gate without any additional communication. Instead, we need $m\lambda$ extra bits to close the gap.

The garbling of one-hot gates is essentially a private puncturable PRF: The PRF $F(x)$ outputs the 0-label of the x-th output wire; and the punctured key F_{-x} is the evaluator's view. While the works of one-hot garbling [HK21a, Hea24] can roughly be viewed as a puncturable PRF – the evaluator must learn x, our work rebuild the privacy of x. Our core technique essentially turns a puncturable PRF (PPRF) into a Programmable Distributed Point Functions (PDPF) [BGIK22], which is a privately puncturable PRF along with an additional programmability property: when deriving a punctured key, one can specify the outputs the key yields at the punctured point. Roughly speaking, combing our technique with the tree-based PPRF construction of [GGM86] gives a PDPF where the output range is a cyclic Abelian group of size up to $2^{O(\lambda_{\mathsf{DCR}})}$. The programmed key mainly consists of a punctured key of GGM-PPRF and the garbled materials of our LUT. The key size of our PDPF is $O(n\lambda_{\mathsf{DCR}})$, while the full domain evaluation takes $O(2^n{\lambda_{\mathsf{DCR}}}^2)$ time. Generation of the master key and the programmed key takes $O(n{\lambda_{\mathsf{DCR}}}^2)$ and $O(n\lambda + \lambda_{\mathsf{DCR}})$ time respectively.

Theorem 2 (Programmable DPF)**.** *Assuming the decisional composite residuosity (DCR) assumption, there exists a programmable distributed point function for $f_{x,v} : [2^n] \to \mathbb{G}$, for any cyclic group $\mathbb{G}$ with size smaller than $2^{(\zeta-1)\lambda_{\mathsf{DCR}}-\lambda}$. Key generation runs in time $O(n{\lambda_{\mathsf{DCR}}}^2)$, programming runs in time $O(n\lambda + \lambda_{\mathsf{DCR}})$, key size is $O(n\lambda_{\mathsf{DCR}})$, and full-domain evaluation runs in time $O(2^n{\lambda_{\mathsf{DCR}}}^2)$.*

We compare our PDPF construction with previous works in Table 2.

	Key Size	**Key Gen.**	**Key Prog.**	**Full Eval.**	**Assumpt.**	**Decomp.**
Ours	$n\lambda_{\mathsf{DCR}}$	$n\lambda_{\mathsf{DCR}}c_{\text{mult}}$	$nc_{\text{hash}} + \lambda_{\mathsf{DCR}}$	$N\lambda_{\mathsf{DCR}}c_{\text{mult}}$	DCR	
Ours	$n\lambda_{\mathsf{DCR}}$	$N\lambda_{\mathsf{DCR}} + n\lambda_{\mathsf{DCR}}c_{\text{mult}}$	$nc_{\text{hash}} + \lambda_{\mathsf{DCR}}$	$N\lambda_{\mathsf{DCR}}c_{\text{mult}}$	DCR	○
[BLW17]	$n\lambda_{\mathsf{RSA}}$	$n \cdot \text{poly}(\lambda_{\mathsf{RSA}})$	$n \cdot \text{poly}(\lambda_{\mathsf{RSA}})$	$N \cdot \text{poly}(\lambda_{\mathsf{RSA}})$	MMap	○
[PS18]	$\text{poly}(\lambda_{\mathsf{LWE}})$	$\text{poly}(\lambda_{\mathsf{LWE}})$	$\text{poly}(\lambda_{\mathsf{LWE}})$	$N \cdot \text{poly}(\lambda_{\mathsf{LWE}})$	subexp-LWE	○
[BGIK22](∗)	$nm\lambda$	λ	$\frac{Nm^2}{\epsilon^2}c_{\text{hash}}$	$\frac{Nm^2}{\epsilon^2}c_{\text{hash}}$	OWF	
[BGIK22]	$\text{poly}(n)$	λ	$\text{poly}(N)$	$\text{poly}(N)$	OWF	

Table 2: Comparison of key size and computation complexities for different approaches for constructing PDPF with domain size $N = 2^n$ and m-bit output. Constant factors are ignored. [PS18] supports efficient evaluation on a single point, while all other approaches only support full-domain evaluation. c_{mult} denotes the cost of multiplying two λ_{DCR}-bit integers. c_{hash} denotes the cost of evaluating a hash function or a pseudorandom number generator. [BGIK22] gives two constructions, with the first (*) only offering ϵ-privacy. We assume $m \le \lambda_{\mathsf{DCR}} \le N$ and $\epsilon \ge 1/N$ for simplicity.

Our PDPF construction can be further modified to achieve a property we call *decomposability*, at the cost of increasing the generation time of the master key to $O(2^n\lambda_{\mathsf{DCR}} + n{\lambda_{\mathsf{DCR}}}^2)$. Basically, decomposability means that the programmed key can be decomposed into n parts, where the i-th part depends solely on the i-th bit of the programming point. The decomposability property is particularly valuable when the programmed key is generated in a distributed manner, eliminating the need for a trusted setup or a generic MPC protocol in many cases. Therefore, we believe that our PDPF construction is of independent interest.

Theorem 3 (Programmable DPF with Decomposable Key)**.** *Assuming the decisional composite residuosity (DCR) assumption, there exists a programmable distributed point function for $f_{x,v} : [2^n] \to \mathbb{G}$, for any cyclic group $\mathbb{G}$ with size smaller than $2^{(\zeta-1)\lambda_{\mathsf{DCR}}-\lambda}$. Key generation runs in time $O(2^n\lambda_{\mathsf{DCR}} + n{\lambda_{\mathsf{DCR}}}^2)$, programming runs in time $O(n\lambda + \lambda_{\mathsf{DCR}})$, key size is $O(n\lambda_{\mathsf{DCR}})$, and full-domain evaluation runs in time $O(2^n{\lambda_{\mathsf{DCR}}}^2)$. Furthermore, the programmed key is decomposable with respect to the bits of the programming position x.*

1.1 Technical Overview

Background: Shifted Boolean One-Hot Label. Consider a wire value $x \in \mathbb{Z}_{2^n}$ where $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$ is its binary representation. The garbler (denoted by $\mathsf{Garbler}$) holds keys $X_{\mathsf{G}} = (X[0], \ldots, X[n-1])$, and the evaluator (denoted by $\mathsf{Evaluator}$) holds labels $X_{\mathsf{E}} = (X[0] \oplus x_0\Delta, \ldots, X[n-1] \oplus x_{n-1}\Delta)$, where $X[0], \ldots, X[n-1], \Delta$ are random λ-length boolean vectors. We call $(X_{\mathsf{G}}, X_{\mathsf{E}})$ a Boolean share of x. The goal of $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ is to obtain a Boolean share of $f(x)$, where $f : [2^n] \to \{0,1\}^m$ is a predetermined function.

The techniques in [HK21a] and [Hea24] allow $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ to obtain a so-called *one-hot label* of x, such that $\mathsf{Garbler}$ holds keys $I_{\mathsf{G}} = (I[0], \ldots, I[2^n-1])$, and $\mathsf{Evaluator}$ holds labels $I_{\mathsf{E}} = (I[0], \ldots, I[x-1], I[x] \oplus \Delta, I[x+1], \ldots, I[2^n-1])$, where $I[0], \ldots, I[2^n-1]$ are random λ-length boolean vectors, and $\oplus$ denotes the bitwise XOR operation. Observe that for any predetermined function $f : [2^n] \to \{0,1\}$, $\mathsf{Garbler}$ can calculate $Y_{\mathsf{G}} = \bigoplus_{i=0}^{2^n-1} I_{\mathsf{G}}[i] \cdot f(i)$, and $\mathsf{Evaluator}$ can calculate $Y_{\mathsf{E}} = \bigoplus_{i=0}^{2^n-1} I_{\mathsf{E}}[i] \cdot f(i)$, such that $Y_{\mathsf{E}} = Y_{\mathsf{G}} \oplus f(x)\Delta$, i.e., $(Y_{\mathsf{G}}, Y_{\mathsf{E}})$ is a Boolean share of $f(x)$. The same process can be repeated for many functions without additional communication. There is a caveat, however: To generate a one-hot label of x, x must be leaked to $\mathsf{Evaluator}$.

[HK21a] and [Hea24] got around this issue by introducing a random offset $c \in [2^n]$, and generating a one-hot label of $(x + c) \bmod 2^n$ instead of x. This preserves privacy, as $(x + c) \bmod 2^n$ is uniformly distributed over $[2^n]$ indepedent of x. However, a Boolean share of $f(x + c)$ is useless in most setting, and applications of one-hot labels have been limited to computing multiplication, with the single exception of [HKN24].

Our Contribution: Real One-Hot Label. Our core contribution is a way to remove the random offset c from the one-hot label, without compromising privacy. Once this is

achieved, we can use one-hot labels to compute any function $f : [2^n] \to \{0,1\}^m$, with almost no additional communication.

Suppose $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ hold the following variant of one-hot label of $y = x \oplus c$: [2] $\mathsf{Garbler}$ holds keys $I_\mathsf{G} = (I[0], \ldots, I[2^n - 1])$, and $\mathsf{Evaluator}$ holds labels $I_\mathsf{E} = (I[0], \ldots, I[y-1], I[y] + w, I[y+1], \ldots, I[2^n - 1])$, where $I[0], \ldots, I[2^n - 1]$ are random integers, and w is a payload to be specified later. From now on, it will be more convenient to view $(I_\mathsf{G}, I_\mathsf{E})$ as a subtractive secret share of the one-hot vector

$$\mathcal{I}^{(n)}(y, w) := (\underbrace{0, \ldots, 0}_{y}, w, \underbrace{0, \ldots, 0}_{2^n - y - 1}).$$

Our goal is to transform $\mathcal{I}^{(n)}(y, w)$ into $\mathcal{I}^{(n)}(x, w)$. For an array $I = (I[0], \ldots, I[2^n - 1])$, let $\mathsf{shift}(I, t)$ denote the array where each index is XORed with t, that is, $\mathsf{shift}(I, t)[j] = I[j \oplus t]$. Then we want to obtain $\mathsf{shift}(\mathcal{I}^{(n)}(y, w), c)$ from $\mathcal{I}^{(n)}(y, w)$.

Let $c = \sum_{i=0}^{n-1} c_i 2^i$ be its binary representation. We decompose the task into n steps, where the i-th step is to shift the secret share by 2^i if $c_i = 1$, and do nothing otherwise. Note that for a secret-shared array I, the i-th step is equivalent to computing the linear combination

$$c_i \cdot \mathsf{shift}(I, 2^i) + (1 - c_i) \cdot I.$$

Since subtractive secret share is linearly homomorphic, the task is reduced to multiplying the secret share I by c_i and $1 - c_i$ entry-wise, without leaking c.

To this end, we borrow techniques from the Homomorphic Secret Sharing (HSS) literature. In the Damgård-Jurik cryptosystem [DJ01], a public key is an RSA modulus $N = pq$, and its corresponding secret key is $\mathsf{sk} = (p-1)(q-1)$. For any $m \in \mathbb{Z}_{N^2}$, we have

$$\mathsf{Enc}(N, m)^{\mathsf{sk}} \equiv \exp(m \cdot \mathsf{sk}) \pmod{N^3},$$

where $\exp : \mathbb{Z}_{N^2} \to 1 + N\mathbb{Z}_{N^3}$ is an exponential function in some sense. This fits particularly well with subtractive secret share, where, say, $\mathsf{Garbler}$ holds r and $\mathsf{Evaluator}$ holds $r + \mathsf{sk}$. If $\mathsf{Garbler}$ sends $e \leftarrow \mathsf{Enc}(N, c_i)$ to $\mathsf{Evaluator}$, they can compute $w_\mathsf{G} = e^r$ and $w_\mathsf{E} = e^{r+\mathsf{sk}}$ locally, and $w_\mathsf{E} \cdot w_\mathsf{G}^{-1} \equiv \exp(c_i \cdot \mathsf{sk}) \pmod{N^3}$. $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ can then obtain a subtractive secret share of $c_i \cdot \mathsf{sk}$ by computing the logarithm of w_G and w_E locally. This is the so-called *distributed discrete logarithm* technique introduced in [OSY21].

Now the path is clear: Initially, $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ hold $I_\mathsf{G}^{(0)} := I_\mathsf{G}, I_\mathsf{E}^{(0)} := I_\mathsf{E}$ respectively, which form a subtractive secret share of $\mathcal{I}^{(n)}(y, \mathsf{sk})$. For each i from 0 to $n-1$, $\mathsf{Garbler}$ sends $E[i][0] \leftarrow \mathsf{Enc}_N(1 - c_i)$ and $E[i][1] \leftarrow \mathsf{Enc}_N(c_i)$ to $\mathsf{Evaluator}$, then they locally compute $E[i][0]^{I_\mathsf{G}^{(i)}} \cdot E[i][1]^{\mathsf{shift}(I_\mathsf{G}^{(i)}, 2^i)}$ and $E[i][0]^{I_\mathsf{E}^{(i)}} \cdot E[i][1]^{\mathsf{shift}(I_\mathsf{E}^{(i)}, 2^i)}$ (the exponentiations and multiplications are done entry-wise), and use distributed discrete logarithm of the result to obtain $I_\mathsf{G}^{(i+1)}$ and $I_\mathsf{E}^{(i+1)}$. Finally, $I_\mathsf{G}^{(n)}$ and $I_\mathsf{E}^{(n)}$ form a subtractive secret share of $\mathcal{I}^{(n)}(x, \mathsf{sk})$.

[2] We choose $x \oplus c$ instead of $x + c$ for simplicity in the formal description of the protocol.

Note that a subtractive secret share cannot be converted to a Boolean share directly. By doing inner product with the truth table of a function $f : [2^n] \to \{0,1\}$, Garbler and Evaluator would obtain a subtractive secret share of $f(x) \cdot \mathsf{sk}$, which is still one step away from the desired Boolean share of $f(x)$. Fortunately, since Evaluator don't know sk, this can be solved with Garbler sending extra $O(\lambda)$ bits to Evaluator.

1.2 Related Work

Arithmetic Garbled Circuits. There is a line of works garbling circuits that are composed of arithmetic multiplication/addition gates and the conversion gates between arithmetic labels and boolean labels. They mainly focused on optimizing the rate [3]. By Yao's scheme, one can build a rate-$O(1/(\lambda\ell))$ arithmetic garbling scheme with schoolbook multiplication, relying only on OWF. [AIK11] achieved rate $O(1/\lambda_{\mathsf{LWE}})$ from LWE. [BMR16] generalized Free-XOR and supported free addition, while multiplication is exponentially expensive. [BLLL23] got the first constant rate scheme for bounded integer computation from DCR (by Damgård-Jurik). [LL24] further supported improved bit decomposition gates in the random oracle model. [Hea24] gave the first construction of rate $O(1/\lambda)$ that only relies on minicrypt-style assumption (CCR hash). Recently [MORS24a] obtained rate-1 garbling of multiplication gates from DCR. Similar to our work, they also borrowed techniques from Damgård-Jurik based HSS [RS21].

2-Party Homomorphic Secret Sharing. There are amount of works on constructing 2-party HSS for Branching Programs (BP) via Distributed Discrete Logarithm (DDLog) based on various assumptions. For group based HSS, [BGI16, BGI17, BCG$^+$17, DKK18] gave constructions from DDH assumption. [FGJS17, OSY21, RS21] gave constructions from DCR assumption. [ADOS22] gave a construction form class groups. For lattice based HSS, [BKS19] bypassed fully/somewhat homomorphic encryption and gave a direct construction from LWE (or Ring-LWE) with superpolynomial modulus (and thus large ciphertexts). [ACK23] enhanced it to polynomial modulus but with a more involved reconstruction procedure. All these HSS schemes works on a relatively restricted computing model called Restricted Multiplication Straight-line (RMS) program, in which multiplication is only allowed between an input value and a memory value and addition is allowed between two memory values (this model captures BP). In comparison, our scheme allows 2PC for more complex functions at the expense of sublinear communication. One should note that the share of one-hot vector of x is not directly tractable from the boolean share of x via HSS: neither the garbler nor the evaluator knows x thus they cannot acquire the encryptions of (the bits of) x to do multiplication.

[3] The rate of a garbling scheme is roughly defined as $(|C|+n)\ell/(|\widetilde{C}|+|L|)$. $\widetilde{C}$ denotes the garbled circuit, $|L|$ is the size of all the input labels, n is the number of inputs to C, and ℓ is the bit-length of wire values.

Other Approaches to Garble Lookup Table [HKN24] mentioned several other GC approaches that can be used to evaluate functions via their lookup tables. Classical GC requires $O(|f|\cdot\lambda)$ communication, which is $\tilde{O}(2^n m\lambda)$ in the worst case. The one-hot garbling approach costs only $O(n\lambda)$ communication to get shares of $f(x)$, but x must be leaked to the Evaluator [HK21a, Hea24]. Stacked garbling [Kol18, HK21b, HK20], which can efficiently handle programs with conditional branching, can be used recursively to obtain an $O(n^2\lambda + nm\lambda)$ communication scheme for garbling f. But it requires $O((2^n)^{2.389}m\lambda)$ hash function calls [HKN24, Hea22]. Garbled RAM [LO13] requires nearly $O(nm\lambda + n^3\lambda)$ amortized communication and computation per access to an 2^n-length array of size-m elements [PLS23, HKO22]. If the cost is not amortized, $\tilde{O}(2^n m\lambda)$ communication is required for each new lookup table. [HHK+22] considered how to garble a PIR gate, where the database is publicly agreed by the garbler and evaluator, and the input index is computed inside the evaluated circuit. They achieved $\tilde{O}(\sqrt{2^n}m\lambda)$ communication and sublinear (to 2^n) evaluator's computation.

2-Round Secure 2-Party Computation. We can place our garbled LUT scheme in the general context of 2-round 2PC, where Alice holds input $x_A \in \{0,1\}^n$ and Bob holds input $x_B \in \{0,1\}^n$. Alice sends the first round message to Bob and eventually gets the output $y = f(x_A, x_B) \in \{0,1\}^m$ after getting the second round message from Bob. The core measures in this model are the size of the two messages and the computation of Alice and Bob. Note that in the insecure setting they can simply send the inputs/outputs in plaintext and let only one party evaluate f. For a circuit f, using Fully Homomorphic Encryption (FHE) [Gen09] or Laconic Function Evaluation (LFE) [QWW18], one can get a protocol that the communication is sublinear to $|f|$ and the computation of one of Alice/Bob is also sublinear to $|f|$ [4]. However, they are all based on lattice assumptions. [IP07] enabled computation of branching programs on ciphertexts based on DCR, and their cipertext size is proportional to the length of the BP.

Our garbled LUT scheme is suitable for securely computing functions with some small segments that are sophisticated to be represented as boolean/arithmetic circuits. The reason is that our communication is only linear to the input/output size of the function – no matter how high its circuit complexity is – but the computation is exponentially large (almost the worst case circuit size). For those suitable functions, one can use our LUT garbling on those 'small but sophisticated' segments to get a garbled circuit of sublinear size. Starting from this, one can further derive various 2PC with sublinear overall communication under non-lattice assumptions. For example, one can combine it with Laconic OT [CDG+17] to get a Non-Interactive Secure Computation (NISC) for circuits, with the public message sublinear to the database size and the online message sublinear to the circuit size.

[4] To be precise, with FHE, the communication and Alice's computation are proportional to the input and output size of f. While with LFE, the communication and Bob's computation are proportional to the input and ouput size of f and the circuit depth of f.

Programmable Distributed Point Functions. What we have constructed partially in this work is essentially a Privately Programmable Pseudorandom Function (PP-PRF) or Programmable Distributed Point Function (PDPF). In a constrained PRF, the owner of the PRF key k can generate constrained keys k_f from k and a predict f, such that anyone equipped with k_f can evaluate the PRF on inputs x where $f(x) = 0$, while no information is revealed about the PRF evaluation on other inputs. A Privately Constrained PRF (PC-PRF) additionally requires that the constrained key k_f hides the predicate f. [BKM17] gave a construction from LWE and the 1-dimensional SIS problem for the class of point-function constraints (which is also known as Privately Puncturable PRF). Afterwards, there are many works constructing PC-PRF for more involved constraints and with stronger privacy, mostly from LWE [CC17, BTVW17, CVW18, DKN+20]. Boneh, Lewi and Wu [BLW17] initially proposed the concept of Privately Programmable PRF (PP-PRF), which is a privately puncturable PRF along with an additional programmability property: when deriving a constrained key, one can specify the outputs the key yields at the points that $f(x) = 1$. They posed a construction from multilinear maps. [PS18] later gave a construction from LWE. [BGIK22] proposed an equivalent primitive called Programmable Distributed Point Function (PDPF) [5] and degraded the assumption to OWF. But their key generation time and the evaluation time are almost linear to the domain size N even in the 1/poly-secure setting. To obtain negligible security error and arbitrary payload set, their key size goes to $\mathsf{polylog}(N)$, the key generation time and the evaluation time goes to $\mathsf{poly}(N)$.

Early works on Distributed Point Functions (DPF) gave graceful and efficient constructions based on OWF [GI14, BGI15], but they are not programmable. Furthermore, [BDSS25] considered non-interactive DPF, where two parties can locally derive DPF keys by simply reading each other's public keys. Their construction has key size $O(N^{2/3})$, and evaluation time $O(N^{5/3})$, and can be based on various assumptions including DCR.

There are some other works on PC-PRF under non-lattice assumptions. [CMPR23] constructed PC-PRF from HSS for inner-product constraints. Their construction relies on HSS in a relatively direct way and the technique is different from ours. [AMN+18] constructed PC-PRF under DDH in prime-order cyclic groups, but only for bit-fixing constraints.

2 Preliminaries

2.1 Notations

Let $\mathbb{N}^+ = \{0, 1, \ldots\}$. For every $n \in \mathbb{N}^+$, we use $[n]$ to represent the set $\{0, \ldots, n-1\}$. We use $\mathbb{Z}_n$ to denote the ring of integers modulo n. We will use $\mathbb{Z}_n$ and $[n]$ interchangeably,

[5] In most applications of PDPF, evaluations are done on the whole domain (a.k.a full-domain evaluation), because the main task is to allocate additive shares of a point vector to two parties. We note that both ours and [BGIK22]'s construction cannot evaluate on a single point in sublinear time w.r.t. the domain size.

and assume $x \bmod n$ always falls into $[n]$. We use $\mathbb{Z}_n^*$ to denote the multiplicative group of units in $\mathbb{Z}_n$. We use $\leftarrow$ to denote assignment. We let $x \xleftarrow{\$} \mathcal{D}$ denote sampling x according to the distribution $\mathcal{D}$. If D is a set, we abuse the notation and let $x \xleftarrow{\$} \mathcal{D}$ denote uniformly sampling from the elements of $\mathcal{D}$. We denote using $\oplus$ the bitwise xor operation. Capital letters denotes vectors, with an exception that N denotes the modulus in Damgård-Jurik cryptosystem. All vectors are 0-indexed unless otherwise specified.

2.2 Damgård-Jurik Cryptosystem

The Damgård-Jurik cryptosystem [DJ01], as described in Figure 1, is a generalization of the Paillier cryptosystem [Pai99].

The Damgård-Jurik Cryptosystem

Require:

- $\mathsf{RSA.Gen}(\cdot)$ is an RSA modulus generation algorithm which, on input a security parameter λ, samples two primes p, q from range $[2^{\lambda_{\mathsf{DCR}}-1}, 2^{\lambda_{\mathsf{DCR}}}]$ (where $\lambda_{\mathsf{DCR}} = \lambda_{\mathsf{DCR}}(\lambda)$ is some polynomial chosen appropriately in order for the cryptosystem to achieve λ bits of security) and then computes $N \leftarrow p \cdot q$, and outputs (N, p, q).
- $\zeta \geq 2$ is a constant defining the plaintext size.
- Functions $\exp : \mathbb{Z}_{N^\zeta} \to 1 + N\mathbb{Z}_{N^{\zeta+1}}$ and $\log : 1 + N\mathbb{Z}_{N^{\zeta+1}} \to \mathbb{Z}_{N^\zeta}$ defined by the following expressions, as in [RS21] and [MORS24b]:

$$\exp(x) := \sum_{k=0}^{\zeta} \frac{(Nx)^k}{k!} \bmod N^{\zeta+1} \quad \text{and} \quad \log(1+Nx) := \sum_{k=1}^{\zeta} \frac{(-N)^{k-1}x^k}{k} \bmod N^\zeta.$$

$\mathsf{Gen}\left(1^\lambda\right)$:

- Sample (N, p, q) such that $N = p \cdot q$.
- Output $\mathsf{pk} = N, \mathsf{sk} = (p-1)(q-1)$.

$\mathsf{Enc}\left(\mathsf{pk} = N, x\right)$:

- Sample a random $g \xleftarrow{\$} \mathbb{Z}_{N^{\zeta+1}}^*$.
- Output $\mathrm{ct} = g^{N^\zeta} \exp(x) \bmod N^{\zeta+1}$.

$\mathsf{Dec}(\mathsf{sk}, \mathrm{ct})$:

- Output $x = \mathsf{sk}^{-1} \log(\mathrm{ct}^{\mathsf{sk}}) \bmod N^\zeta$.

Figure 1: Damgård-Jurik Cryptosystem.

Theorem 4 (Damgård-Jurik Cryptosystem [DJ01])**.** *Assuming the DCR assumption, the construction of Figure 1 is a CPA-secure encryption scheme.*

In this paper, we always let $\zeta \geq 2$ denote the positive integer constant used in Damgård-Jurik cryptosystem.

Definition 5 (Decision Composite Residuosity (DCR) Assumption, [Pai99])**.** Let $\mathsf{RSA.Gen}$ be a polynomial-time algorithm which, on input a security parameter λ, outputs (N, p, q) where p and q are λ-bit primes and $N = pq$. We say that the Decision Composite Residuosity (DCR) problem is hard relative to modulus-sampling algorithm $\mathsf{RSA.Gen}$ if

$$\left\{(N,x): \begin{array}{c}(N,p,q) \xleftarrow{\$} \mathsf{RSA.Gen} \\ x \xleftarrow{\$} \mathbb{Z}^*_{N^2}\end{array}\right\} \overset{\mathrm{c}}{\approx} \left\{\left(N, x^N \bmod N\right): \begin{array}{c}(N,p,q) \xleftarrow{\$} \mathsf{RSA.Gen} \\ x \xleftarrow{\$} \mathbb{Z}^*_{N^2}\end{array}\right\}$$

2.3 Garbled Circuit

A garbling scheme [BHR12] is a tuple of procedures specifying how to garble a class of circuits.

Definition 6 (Garbling Schemes)**.** A garbling scheme for a class of circuits $\mathcal{C}$ is a tuple of procedures $(\mathsf{Garble}, \mathsf{Encode}, \mathsf{Evaluate}, \mathsf{Decode})$, where:

- Garble maps a circuit $C \in \mathcal{C}$ to garbled circuit material $\hat{C}$, an input encoding string e, and an output decoding string d;
- Encode maps an input encoding string e and a cleartext bitstring x to an encoded input;
- $\mathsf{Evaluate}$ maps a circuit C, garbled circuit material $\hat{C}$, and an encoded input to an encoded output;
- Decode maps an output decoding string d and an encoded output to a cleartext output string (or outputs $\perp$ if the encoded output is invalid).

A garbling scheme must be **correct**, and it may satisfy any combination of **obliviousness**, **privacy**, and **authenticity** [BHR12]. The most significant of these properties is obliviousness, which informally states that the garbled material and encoded inputs reveal nothing about the computation to the evaluator:

Definition 7 (Oblivious Garbling Scheme)**.** A garbling scheme is oblivious if there exists a simulator Sim such that for any circuit $C \in \mathcal{C}$ and for all inputs x, the pair $(\hat{C}, \mathsf{Encode}(e, x))$ is computationally indistinguishable from $\mathsf{Sim}(1^\lambda, C)$, where $(\hat{C}, e, \cdot) \leftarrow \mathsf{Garble}(1^\lambda, C)$.

For most garbled circuit techniques (including ours), authenticity and privacy naturally follow from obliviousness.

3 Encoding and Secret Share

Boolean Encoding. For an n-bit integer x whose binary representation is $x = \sum_{i=0}^{n-1} x_i 2^i$, and a λ-length boolean vector Δ, let $\mathfrak{B}(x, \Delta)$ denote $(x_0 \cdot \Delta, \ldots, x_{n-1} \cdot \Delta)$.

One-Hot Encoding. For an n-bit integer x and an integer w, let $\mathcal{I}^{(n)}(x, w)$ denote the length-2^n vector where the x-th entry is w and all other entries are 0.

Secret Share. For any value v (which may be a scalar or a vector), a secret share of v consists of two values $v_{\mathsf{G}}, v_{\mathsf{E}}$ held by garbler and evaluator respectively. We will use three types of secret share in this paper: XOR secret share, subtractive secret share, and divisional secret share.

- For an length-n bit string v, we use $[\![v]\!]^{\mathsf{xor}} := \{(v_{\mathsf{G}}, v_{\mathsf{E}}) : v_{\mathsf{G}}, v_{\mathsf{E}} \in \{0,1\}^n, v_{\mathsf{G}} \oplus v_{\mathsf{E}} = v\}$ to denote the set of all possible XOR secret shares of v.
- For an integer v and a modulus N, we use $[\![v]\!]^{\mathsf{sub}}_N := \{(v_{\mathsf{G}}, v_{\mathsf{E}}) : v_{\mathsf{G}}, v_{\mathsf{E}} \in [N], v_{\mathsf{E}} - v_{\mathsf{G}} \equiv v \pmod N\}$ to denote the set of all possible subtractive secret shares of v.
- For an integer v, a Damgård-Jurik public key N and a fixed constant ζ, we use

$$[\![v]\!]^{\mathsf{div}}_N := \left\{(v_{\mathsf{G}}, v_{\mathsf{E}}) : v_{\mathsf{G}}, v_{\mathsf{E}} \in \mathbb{Z}^*_{N^{\zeta+1}}, v_{\mathsf{E}} \cdot v_{\mathsf{G}}^{-1} \equiv \exp(v) \pmod{N^{\zeta+1}}\right\}$$

  to denote the set of all possible divisional secret shares of v, where $\exp(x) := \sum_{k=0}^{\zeta} \frac{(Nx)^k}{k!} \bmod N^{\zeta+1}$ is defined as in Figure 1.

For a vector v, the notations $[\![v]\!]^{\mathsf{xor}}, [\![v]\!]^{\mathsf{sub}}_N, [\![v]\!]^{\mathsf{div}}_N$ are extended element-wise.

We note that all three types of secret shares are linearly homomorphic in a certain sense. We formulate the linear homomorphism property of the divisional secret share in the following lemma.

Lemma 8. *Fix a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk})$ and a constant ζ. Let $(a_{\mathsf{G}}, a_{\mathsf{E}}) \in [\![a]\!]^{\mathsf{div}}_N, (b_{\mathsf{G}}, b_{\mathsf{E}}) \in [\![b]\!]^{\mathsf{div}}_N$ and $c_{\mathsf{E}}, c_{\mathsf{G}} \in \mathbb{Z}$ such that $c_{\mathsf{E}} - c_{\mathsf{G}} = c \cdot \mathsf{sk}$, for some integers a, b, c. Let $e \leftarrow \mathsf{Enc}(N, t)$ for some integer t. Then we have:*

- *Addition:* $(a_{\mathsf{G}} \cdot b_{\mathsf{G}} \bmod N^{\zeta+1}, a_{\mathsf{E}} \cdot b_{\mathsf{E}} \bmod N^{\zeta+1}) \in [\![a+b]\!]^{\mathsf{div}}_N$.
- *Multiplication:* $(a_{\mathsf{G}}^k \bmod N^{\zeta+1}, a_{\mathsf{E}}^k \bmod N^{\zeta+1}) \in [\![k \cdot a]\!]^{\mathsf{div}}_N$, *for any constant k.*
- *Exponentiation:* $(e^{c_{\mathsf{G}}} \bmod N^{\zeta+1}, e^{c_{\mathsf{E}}} \bmod N^{\zeta+1}) \in [\![t \cdot c \cdot \mathsf{sk}]\!]^{\mathsf{div}}_N$.

Proof. We note that the function $\exp(x) := \sum_{k=0}^{\zeta} \frac{(Nx)^k}{k!} \bmod N^{\zeta+1}$ indeed behaves like an exponential function, in the sense that $\exp(x+y) \equiv \exp(x) \cdot \exp(y) \pmod{N^{\zeta+1}}$, which can be proved by direct calculation.

Thus, the first two items follow directly from the definition of the divisional secret share. For the third item, we note that $e^{c_{\mathsf{E}}} \cdot (e^{c_{\mathsf{G}}})^{-1} \equiv e^{c \cdot \mathsf{sk}} \equiv \exp(t \cdot c \cdot \mathsf{sk}) \pmod{N^{\zeta+1}}$ by correctness of the Damgård-Jurik cryptosystem in Figure 1. □

4 Full Construction

For ease of presentation, we will split the full construction into several parts, where each part is a subprotocol that realizes a specific functionality, with its own correctness and efficiency guarantee. In this section, we will present the construction of each part, and prove their correctness and efficiency. The security of the full construction will be analyzed in the next section.

Notations. We usually use lower-case letters like x, y to denote integers, and upper-case letters like X, Y, I to denote bit strings or vectors, or vectors of bit strings. For an n-bit integer x, we use subscript x_i to denote its i-th bit. For a bit string (vector) X, we use $X[i]$ to denote its i-th bit (entry). We use $x^{(0)}, x^{(1)}, \cdots$ and $X^{(0)}, X^{(1)}, \cdots$ to denote relevant (but different) values.

4.1 DDLog Gate

The DDLog gate converts a divisional secret share $[\![x]\!]_N^{\mathsf{div}}$ to a subtractive secret share $[\![x]\!]_{N^\zeta}^{\mathsf{sub}}$, without any communication. It is inspired by the distributed discrete logarithm technique in HSS literature [OSY21, RS21]. The construction is presented in Figure 2.

Claim 9. *Let z_{G} and z_{E} be the outputs of* Garbler *and* Evaluator *in Π_{DDLog}, Figure 2. Then $(z_{\mathsf{G}}, z_{\mathsf{E}}) \in [\![x]\!]_{N^\zeta}^{sub}$. Further, Π_{DDLog} takes no communication and $O(\log N)$ computation.*

Proof. Since $(x_{\mathsf{G}}, x_{\mathsf{E}}) \in [\![x]\!]_N^{\mathsf{div}}$, i.e., $x_{\mathsf{G}} \equiv x_{\mathsf{E}} \pmod N$ and $\log(x_{\mathsf{G}}^{-1} \cdot x_{\mathsf{E}}) = x$, we have $y_{\mathsf{G}} \equiv y_{\mathsf{E}} \equiv 1 \pmod N$ and $(y_{\mathsf{G}}, y_{\mathsf{E}}) \in [\![x]\!]_N^{\mathsf{div}}$. Therefore $z_{\mathsf{E}} - z_{\mathsf{G}} \equiv x \pmod{N^\zeta}$.

Computation bottleneck is the $O(1)$ modular divisions and modular multiplications, where each division and multiplication takes $O(\log N)$ time using algorithms like Fast Fourier Transform (FFT) and Barrett reduction. □

4.2 Shifted One-Hot Gate

The shifted one-hot gate converts an XOR secret share $[\![\mathfrak{B}(x, \Delta)]\!]^{\mathsf{xor}}$ and a integer w into $[\![\mathcal{I}^{(n)}(x \oplus c, w)]\!]_{N^\zeta}^{\mathsf{sub}}$, i.e., a subtractive secret share of the one-hot encoding of $x \oplus c$, where c is randomly chosen from $[2^n]$. The construction is conceptually similar to a bin-to-hot gate followed by a scale gate, as defined in [Hea24]. Therefore, we defer the construction to Figure 9, and the correctness proof to Appendix A.

Claim 10. *Let (I'_{G}, c) and (I'_{E}, y) be the outputs of* Garbler *and* Evaluator *in $\Pi_{one\text{-}hot}^{shift}$, Figure 9. Then $y = x \oplus c$, and $(I'_{\mathsf{G}}, I'_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(y, w)]\!]_{N^\zeta}^{sub}$. Further, $\Pi_{one\text{-}hot}^{shift}$ takes $O(n\lambda + \log N)$ communication and $O(2^n \log N)$ computation.*

Π_{DDLog}: DDLog Gate

Input.

- Public parameter: A Damgård-Jurik public key N.
- From $\mathsf{Garbler}$: $x_{\mathsf{G}} \in \mathbb{Z}^*_{N^{\zeta+1}}$.
- From $\mathsf{Evaluator}$: $x_{\mathsf{E}} \in \mathbb{Z}^*_{N^{\zeta+1}}$.
- Required: $(x_{\mathsf{G}}, x_{\mathsf{E}}) \in [\![x]\!]^{\mathsf{div}}_N$, where $x \in [N^\zeta]$.

Output.

- $\mathsf{Garbler}$: $z_{\mathsf{G}} \in [N^\zeta]$.
- $\mathsf{Evaluator}$: $z_{\mathsf{E}} \in [N^\zeta]$.
- Expected: $(z_{\mathsf{G}}, z_{\mathsf{E}}) \in [\![x]\!]^{\mathsf{sub}}_{N^\zeta}$.

Protocol. *Can be applied to vectors element-wise.*

1. $\mathsf{Garbler}$ computes $y_{\mathsf{G}} = x_{\mathsf{G}} \cdot (x_{\mathsf{G}}^{-1} \bmod N) \bmod N^{\zeta+1}$ and $z_{\mathsf{G}} = \log(y_{\mathsf{G}})$, where $\log(1+Ny) := \sum_{k=1}^{\zeta} \frac{(-N)^{k-1} y^k}{k} \bmod N^\zeta$.
2. $\mathsf{Evaluator}$ computes $y_{\mathsf{E}} = x_{\mathsf{E}} \cdot (x_{\mathsf{E}}^{-1} \bmod N) \bmod N^{\zeta+1}$ and $z_{\mathsf{E}} = \log(y_{\mathsf{E}})$, where $\log(1+Ny) := \sum_{k=1}^{\zeta} \frac{(-N)^{k-1} y^k}{k} \bmod N^\zeta$.
3. $\mathsf{Garbler}$ outputs z_{G}, and $\mathsf{Evaluator}$ outputs z_{E}.

Figure 2: DDLog Gate

4.3 Real One-Hot Gate

The real one-hot gate is our main building block for the lookup gate. It converts an XOR secret share $[\![\mathfrak{B}(x,\Delta)]\!]^{\mathsf{xor}}$ into $[\![\mathcal{I}^{(n)}(x,\mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$, i.e., a subtractive secret share of the one-hot encoding of x, where sk is a random Damgård-Jurik secret key. The construction is presented in Figure 3.

We briefly explain the intuition behind our construction. We can obtain a subtractive secret share of $\mathcal{I}^{(n)}(x \oplus c, \mathsf{sk})$ from the shifted one-hot gate, where c is known to Garbler and $x \oplus c$ is known to Evaluator. Now we want to transform it into a subtractive secret share of $\mathcal{I}^{(n)}(x, \mathsf{sk})$.

Let $I' = \mathcal{I}^{(n)}(x \oplus c, \mathsf{sk})$ be the secret-shared array. We do the transformation step by step, where in the i-th step we shift I' by 2^i if $c_i = 1$ and keep it unchanged if $c_i = 0$, where c_i is the i-th bit of c. We note that this equivalent to computing

$$c_i \cdot \mathsf{shift}(I', 2^i) + (1 - c_i) \cdot I',$$

so the problem is reduced to multiplying I' by c_i and $1 - c_i$, without leaking c_i.

To this end, Garbler encrypts $1 - c_i$ and c_i using the Damgård-Jurik encryption scheme, and sends the ciphertexts $E[i][0], E[i][1]$ to Evaluator. By Lemma 8, the ciphertexts can be used to multiply I' by $1 - c_i$ and c_i, and the result is a divisional secret share. Finally, Garbler and Evaluator use the DDLog gate to reduce it back to a subtractive secret share.

Claim 11. *Let (I_G, N, sk) and (I_E, N) be the outputs of* Garbler *and* Evaluator *in $\Pi^{real}_{one\text{-}hot}$, Figure 3. Then $(I_\mathsf{G}, I_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]^{sub}_{N^\zeta}$, except with negligible probability. Further, $\Pi^{real}_{one\text{-}hot}$ takes $O(n\lambda_{\mathsf{DCR}})$ communication and $O(2^n {\lambda_{\mathsf{DCR}}}^2)$ computation.*

Proof. We will prove the loop invariant by induction on i. The claim directly follows from the loop invariant, since $y^{(n)} = y \oplus c = x$.

For $i = 0$, we have $y^{(i)} = y$, and by correctness of $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$, $(I^{(0)}_\mathsf{G}, I^{(0)}_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(y, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$.

Suppose the loop invariant holds for i. By adding r to all entries in $I^{(i)}_\mathsf{G}$ and $I^{(i)}_\mathsf{E}$, it remains that $(I^{(i)}_\mathsf{G}, I^{(i)}_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(y^{(i)}, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$. Further, since $\mathsf{sk} \ll N^\zeta$, we now have $I^{(i)}_\mathsf{E} - I^{(i)}_\mathsf{G} = \mathcal{I}^{(n)}(y^{(i)}, \mathsf{sk})$ with overwhelming probability, even when they are viewed as vectors in $\mathbb{Z}^{2^n}$. Thus, by Lemma 8, $(E[i][0]^{I^{(i)}_\mathsf{G}}, E[i][0]^{I^{(i)}_\mathsf{E}})$ is a divisional share of $\mathcal{I}^{(n)}(y^{(i)}, (1 - c_i)\mathsf{sk})$, and $(E[i][1]^{\mathsf{shift}(I^{(i)}_\mathsf{G}, 2^i)}, E[i][1]^{\mathsf{shift}(I^{(i)}_\mathsf{E}, 2^i)})$ is a divisional share of $\mathcal{I}^{(n)}(y^{(i)} \oplus 2^i, c_i\mathsf{sk})$. Putting them together, $(\widetilde{I}^{(i)}_\mathsf{G}, \widetilde{I}^{(i)}_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(y^{(i)} \oplus c_i 2^i, \mathsf{sk})]\!]^{\mathsf{div}}_N$, and by correctness of Π_{DDLog}, $(I^{(i+1)}_\mathsf{G}, I^{(i+1)}_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(y^{(i)} \oplus c_i 2^i, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$. Note that $y^{(i)} \oplus c_i 2^i = y^{(i+1)}$.

Calling $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$ takes $O(n\lambda + \log N)$ communication and $O(2^n \log N)$ computation. Sending $E[i][0], E[i][1]$ takes $O(n \log N)$ communication in total, and computing $\widetilde{I}^{(i)}_\mathsf{G}, \widetilde{I}^{(i)}_\mathsf{E}$ takes $O(2^n \log^2 N)$ computation in total [6]. Calling Π_{DDLog} takes $O(2^n n \log N) = O(2^n {\lambda_{\mathsf{DCR}}}^2)$

[6] Naively, computing each $\widetilde{I}^{(i)}_\mathsf{G}$ and $\widetilde{I}^{(i)}_\mathsf{E}$ requires $O(2^n \log^2 N)$ time, leading to a total computation time

$\Pi^{\mathsf{real}}_{\mathsf{one\text{-}hot}}$: Real One-Hot Gate

Input.

- Public parameter: A positive integer n.
- From $\mathsf{Garbler}$: $X_{\mathsf{G}} = (X_{\mathsf{G}}[0], \ldots, X_{\mathsf{G}}[n-1]) \in (\{0,1\}^\lambda)^n$, and $\Delta \in 1\{0,1\}^{\lambda-1}$.
- From $\mathsf{Evaluator}$: $X_{\mathsf{E}} = (X_{\mathsf{E}}[0], \ldots, X_{\mathsf{E}}[n-1]) \in (\{0,1\}^\lambda)^n$.
- Required: $(X_{\mathsf{G}}, X_{\mathsf{E}}) \in [\![\mathfrak{B}(x, \Delta)]\!]^{\mathsf{xor}}$, where $x \in [2^n]$.

Output.

- $\mathsf{Garbler}$: $I_{\mathsf{G}}, N, \mathsf{sk}$.
- $\mathsf{Evaluator}$: I_{E}, N.
- Expected: $(I_{\mathsf{G}}, I_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$.

Protocol.

1. $\mathsf{Garbler}$ samples a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and sends N to $\mathsf{Evaluator}$. $\mathsf{Garbler}$ samples $r \xleftarrow{\$} [N^\zeta]$ and sends r to $\mathsf{Evaluator}$.
2. $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ call $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}(n, N; X_{\mathsf{G}}, \mathsf{sk}, \Delta; X_{\mathsf{E}})$, and obtain $(I^{(0)}_{\mathsf{G}}, c)$ and $(I^{(0)}_{\mathsf{E}}, y)$, respectively. Let $c = \sum_{i=0}^{n-1} c_i 2^i$ be its binary representation.
3. For i from 0 to $n-1$:
   (a) **Invariant**: $(I^{(i)}_{\mathsf{G}}, I^{(i)}_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(y^{(i)}, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$, where $y^{(i)} = y \oplus \sum_{j=0}^{i-1} c_j 2^j$.
   (b) $\mathsf{Garbler}$ computes $E[i][0] \leftarrow \mathsf{Enc}(N, 1 - c_i), E[i][1] \leftarrow \mathsf{Enc}(N, c_i)$, and sends $E[i][0], E[i][1]$ to $\mathsf{Evaluator}$.
   (c) $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ add r to all entries in $I^{(i)}_{\mathsf{G}}$ and $I^{(i)}_{\mathsf{E}}$, modulo N^ζ.
   (d) $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ compute $\widetilde{I}^{(i)}_{\mathsf{G}} = E[i][0]^{I^{(i)}_{\mathsf{G}}} \cdot E[i][1]^{\mathsf{shift}(I^{(i)}_{\mathsf{G}}, 2^i)} \bmod N^{\zeta+1}$ and $\widetilde{I}^{(i)}_{\mathsf{E}} = E[i][0]^{I^{(i)}_{\mathsf{E}}} \cdot E[i][1]^{\mathsf{shift}(I^{(i)}_{\mathsf{E}}, 2^i)} \bmod N^{\zeta+1}$, where $\mathsf{shift}(I, 2^i)[j] = I[j \oplus 2^i]$.
   (e) $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ call $\Pi_{\mathsf{DDLog}}(N; \widetilde{I}^{(i)}_{\mathsf{G}}; \widetilde{I}^{(i)}_{\mathsf{E}})$, and obtain $I^{(i+1)}_{\mathsf{G}}$ and $I^{(i+1)}_{\mathsf{E}}$.
4. $\mathsf{Garbler}$ outputs $(I^{(n)}_{\mathsf{G}}, N, \mathsf{sk})$, and $\mathsf{Evaluator}$ outputs $(I^{(n)}_{\mathsf{E}}, N)$.

Figure 3: Real One-Hot Gate

computation in total. Therefore, the total communication is $O(n\lambda_{\mathsf{DCR}})$, and the total computation is $O(2^n {\lambda_{\mathsf{DCR}}}^2)$. □

4.4 Lookup Gate

The lookup gate is a simple application of the real one-hot gate. It takes m functions $f_0, \ldots, f_{m-1} : \{0,1\}^n \to \{0,1\}$, and converts an XOR secret share $[\![\mathfrak{B}(x, \Delta)]\!]^{\mathsf{xor}}$ into $[\![\mathfrak{B}(f_i(x), \Delta_O)]\!]^{\mathsf{xor}}$ for $i \in [m]$, where Δ_O is another λ-length boolean vector used to encode the output. The construction is presented in Figure 4.

Claim 12. *Let* $(Y_{\mathsf{G}}[0], \ldots, Y_{\mathsf{G}}[m-1])$ *and* $(Y_{\mathsf{E}}[0], \ldots, Y_{\mathsf{E}}[m-1])$ *be the outputs of* Garbler *and* Evaluator *in* Π_{lookup}*, Figure 4. Then* $(Y_{\mathsf{G}}[i], Y_{\mathsf{E}}[i]) \in [\![\mathfrak{B}(f_i(x), \Delta_O)]\!]^{\mathsf{xor}}$ *for* $i \in [m]$*, except with negligible probability. Further,* Π_{lookup} *takes* $O(n\lambda_{\mathsf{DCR}} + \lambda m)$ *communication and* $O(2^n {\lambda_{\mathsf{DCR}}}^2 + 2^n \lambda_{\mathsf{DCR}} m)$ *computation.*

Proof. By correctness of $\Pi^{\mathsf{real}}_{\text{one-hot}}$, $(I_{\mathsf{G}}, I_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$. It follows that $W_{\mathsf{E}}[i] = W_{\mathsf{G}}[i] + f_i(x)\mathsf{sk} \pmod{N^\zeta}$ and $H_3(W_{\mathsf{E}}[i]) = H_3((W_{\mathsf{G}}[i] + f_i(x)\mathsf{sk}) \bmod N^\zeta)$. Thus $Y_{\mathsf{E}}[i] = Y_{\mathsf{G}}[i] \oplus f_i(x)\Delta_O$, i.e., $(Y_{\mathsf{G}}[i], Y_{\mathsf{E}}[i]) \in [\![\mathfrak{B}(f_i(x), \Delta_O)]\!]^{\mathsf{xor}}$.

Calling $\Pi^{\mathsf{real}}_{\text{one-hot}}$ takes $O(n\lambda_{\mathsf{DCR}})$ communication and $O(2^n {\lambda_{\mathsf{DCR}}}^2)$ computation. Sending $\mathrm{ct}_0, \mathrm{ct}_1$ for all $i \in [m]$ takes $O(\lambda m)$ communication, and computing $W_{\mathsf{G}}[i], W_{\mathsf{E}}[i]$ for all $i \in [m]$ takes $O(2^n \lambda_{\mathsf{DCR}} m)$ computation. Therefore, the total communication is $O(n\lambda_{\mathsf{DCR}} + \lambda m)$, and the total computation is $O(2^n {\lambda_{\mathsf{DCR}}}^2 + 2^n \lambda_{\mathsf{DCR}} m)$. □

5 Privacy

We first present a privacy lemma for the shifted one-hot gate.

Lemma 13. *There exists a PPT simulator* Sim *such that for any positive integer* n*, integer* $x \in [2^n]$*, Damgård-Jurik public key* $N < 2^{\lambda_{\mathsf{DCR}}}$*, integer* $w \in [N^\zeta]$*, the following experiments are computationally indistinguishable.*

- RealShiftOneHotPriv*: Uniformly sample* $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ *and* $(X_{\mathsf{G}}, X_{\mathsf{E}}) \xleftarrow{\$} [\![\mathfrak{B}(x, \Delta)]\!]^{\mathsf{xor}}$*. Run the protocol* $\Pi^{\mathsf{shift}}_{\text{one-hot}}$ *with public parameters* n, N *and inputs* $X_{\mathsf{G}}, w, \Delta, X_{\mathsf{E}}$*, and output the view of* Evaluator*.*
- IdealShiftOneHotPriv*: Output* $\mathsf{Sim}(n, N)$*.*

We defer the proof of Lemma 13 to Appendix A. Next, we define and prove the privacy of our full construction.

of $O(n2^n \log^2 N)$. However, since the base is the same for every 2^n exponentiations, we can optimize this using the Method of Four Russians, reducing the overall computation to $O(2^n \log^2 N)$.

Π_{lookup}: Lookup Gate

Input.

- Public parameter: A positive integer n, a positive integer m, m functions $f_0, \ldots, f_{m-1} : \{0,1\}^n \to \{0,1\}$, and two random oracles $H_3, H_4 : \mathbb{Z} \to \{0,1\}^\lambda$.
- From Garbler: $X_\mathsf{G} = (X_\mathsf{G}[0], \ldots, X_\mathsf{G}[n-1]) \in (\{0,1\}^\lambda)^n$, and $\Delta_O \in \{0,1\}^\lambda$, $\Delta \in 1\{0,1\}^{\lambda-1}$.
- From Evaluator: $X_\mathsf{E} = (X_\mathsf{E}[0], \ldots, X_\mathsf{E}[n-1]) \in (\{0,1\}^\lambda)^n$.
- Required: $(X_\mathsf{G}, X_\mathsf{E}) \in [\![\mathfrak{B}(x, \Delta)]\!]^{\mathsf{xor}}$, where $x \in [2^n]$.

Output.

- Garbler: $(Y_\mathsf{G}[0], \ldots, Y_\mathsf{G}[m-1])$.
- Evaluator: $(Y_\mathsf{E}[0], \ldots, Y_\mathsf{E}[m-1])$.
- Expected: $(Y_\mathsf{G}[i], Y_\mathsf{E}[i]) \in [\![\mathfrak{B}(f_i(x), \Delta_O)]\!]^{\mathsf{xor}}$ for $i \in [m]$.

Protocol.

1. Garbler and Evaluator call $\Pi^{\mathsf{real}}_{\mathsf{one\text{-}hot}}(n; X_\mathsf{G}, \Delta; X_\mathsf{E})$, and obtain $(I_\mathsf{G}, N, \mathsf{sk})$ and (I_E, N) respectively. // $(I_\mathsf{G}, I_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(x, \mathit{sk})]\!]^{\mathit{sub}}_{N^\zeta}$
2. For i from 0 to $m-1$:
   (a) Garbler samples $Y_\mathsf{G}[i] \xleftarrow{\$} \{0,1\}^\lambda$.
   (b) Garbler computes $W_\mathsf{G}[i] \equiv \sum_{j=0}^{2^n-1} f_i(j) I_\mathsf{G}[j] \pmod{N^\zeta}$, and Evaluator computes $W_\mathsf{E}[i] \equiv \sum_{j=0}^{2^n-1} f_i(j) I_\mathsf{E}[j] \pmod{N^\zeta}$.
   (c) Garbler computes $\mathrm{ct}_t = \Big(H_3\big((W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta\big), H_4\big((W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta\big) \oplus Y_\mathsf{G}[i] \oplus t\Delta_O\Big)$ for $t \in \{0,1\}$, randomly permutes $\mathrm{ct}_0, \mathrm{ct}_1$, and sends them to Evaluator.
   (d) Evaluator receives $\mathrm{ct}'_0 = (u_0, v_0), \mathrm{ct}'_1 = (u_1, v_1)$. Let $t \in \{0,1\}$ be the index such that $u_t = H_3(W_\mathsf{E}[i])$, and let $Y_\mathsf{E}[i] = v_t \oplus H_4(W_\mathsf{E}[i])$.
3. Garbler outputs $(Y_\mathsf{G}[0], \ldots, Y_\mathsf{G}[m-1])$, and Evaluator outputs $(Y_\mathsf{E}[0], \ldots, Y_\mathsf{E}[m-1])$.

Figure 4: Lookup Gate

Theorem 14. *There exists a PPT simulator* Sim *such that for any positive integers* n, m*, integer* $x \in [2^n]$*, bit string* $\Delta_O \in \{0,1\}^\lambda$*, and functions* $f_0, f_1, \ldots, f_{m-1} : \{0,1\}^n \to \{0,1\}$*, the following experiments are computationally indistinguishable.*

- $\mathsf{RealLookupPriv}$*: Uniformly sample* $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ *and* $(X_\mathsf{G}, X_\mathsf{E}) \xleftarrow{\$} [\![\mathfrak{B}(x,\Delta)]\!]^{\mathsf{xor}}$*. Run the protocol* Π_{lookup} *with public parameters* $n, m, f_0, \ldots, f_{m-1}$ *and inputs* $X_\mathsf{G}, \Delta_O, \Delta, X_\mathsf{E}$*, and output the view of* $\mathsf{Evaluator}$*.*
- $\mathsf{IdealLookupPriv}$*: Output* $\mathsf{Sim}(n, m, f_0, \ldots, f_{m-1})$*.*

5.1 Proof of Theorem 14

We first expand all subroutine calls in the experiment $\mathsf{RealLookupPriv}$ (except $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$).

Experiment Hyb_0**.**

1. Uniformly sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ and $(X_\mathsf{G}, X_\mathsf{E}) \xleftarrow{\$} [\![\mathfrak{B}(x,\Delta)]\!]^{\mathsf{xor}}$. Output X_E.
2. Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output N.
3. Run $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$ with public parameters n, N, $\mathsf{Garbler}$ input $(X_\mathsf{G}, \mathsf{sk}, \Delta)$, $\mathsf{Evaluator}$ input X_E. Let $(I'_\mathsf{G}, c), (I'_\mathsf{E}, y)$ denote the output of $\mathsf{Garbler}$ and $\mathsf{Evaluator}$, respectively. Output the view of $\mathsf{Evaluator}$.
4. Let $c = \sum_{i=0}^{n-1} c_i \cdot 2^i$ be its binary representation. For $i \in [n]$, let $E[i][0] \leftarrow \mathsf{Enc}(N, 1 - c_i), E[i][1] \leftarrow \mathsf{Enc}(N, c_i)$. Output E.
5. Sample and output $r \xleftarrow{\$} [N^\zeta]$. Compute $(I_\mathsf{G}, I_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$ with I'_E, y and r, E [7].
6. For $i \in [m]$, sample $Y_\mathsf{G}[i] \xleftarrow{\$} \{0,1\}^\lambda$, let $W_\mathsf{G}[i] = \sum_{j=0}^{2^n-1} f_i(j) I_\mathsf{G}[j] \pmod{N^\zeta}$, let $\mathrm{ct}_t = \big(H_3((W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta), H_4((W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta) \oplus Y_\mathsf{G}[i] \oplus t\Delta_O\big)$ for $t \in \{0,1\}$, randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Identity Substitution. We start by replacing c with $x \oplus y$, and I_G with $I_\mathsf{E} - \mathcal{I}^{(n)}(x, \mathsf{sk})$. The new experiment Hyb_1 is statistical indistinguishable from Hyb_0. The changes are marked in blue.

Experiment Hyb_1**.**

1. Uniformly sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ and $(X_\mathsf{G}, X_\mathsf{E}) \xleftarrow{\$} [\![\mathfrak{B}(x,\Delta)]\!]^{\mathsf{xor}}$. Output X_E.
2. Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output N.

[7] Concretely, act as $\mathsf{Evaluator}$ in Step 3 in Figure 3 to compute $I^{(n)}_\mathsf{E}$, let $I_\mathsf{E} := I^{(n)}_\mathsf{E}$ and let $I_\mathsf{G} := I^{(n)}_\mathsf{E} - \mathcal{I}^{(n)}(x, \mathsf{sk})$. We will omit the details in the following hybrid worlds.

3. Run $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$ with public parameters n, N, Garbler input $(X_{\mathsf{G}}, \mathsf{sk}, \Delta)$, Evaluator input X_{E}. Let $(I'_{\mathsf{G}}, c), (I'_{\mathsf{E}}, y)$ denote the output of Garbler and Evaluator, respectively. Output the view of Evaluator.
4. Let $x = \sum_{i=0}^{n-1} x_i \cdot 2^i, y = \sum_{i=0}^{n-1} y_i \cdot 2^i$ be their binary representation. For $i \in [n]$, let $E[i][0] \leftarrow \mathsf{Enc}(N, 1 - x_i \oplus y_i), E[i][1] \leftarrow \mathsf{Enc}(N, x_i \oplus y_i)$. Output E.
5. Sample and output $r \xleftarrow{\$} [N^\zeta]$. Compute $I_{\mathsf{E}} \in \mathbb{Z}^{2^n}_{N^\zeta}$ with I'_{E}, y and r, E.
6. For $i \in [m]$, sample $Y_{\mathsf{E}}[i] \xleftarrow{\$} \{0,1\}^\lambda$, let $W_{\mathsf{E}}[i] = \sum_{j=0}^{2^n-1} f_i(j) I_{\mathsf{E}}[j] \pmod{N^\zeta}$,

$$ct_t = \Big(H_3((W_{\mathsf{E}}[i] + (-1)^{f_i(x)} t\mathsf{sk}) \bmod N^\zeta), \\ H_4((W_{\mathsf{E}}[i] + (-1)^{f_i(x)} t\mathsf{sk}) \bmod N^\zeta) \oplus Y_{\mathsf{E}}[i] \oplus t\Delta_O\Big),$$

randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Claim 15. *The experiments* Hyb_0 *and* Hyb_1 *are statistical indistinguishable.*

Proof. In Step 4, we replace c_i with $x_i \oplus y_i$, using the fact that $c_i = x_i \oplus y_i$ as guaranteed by the correctness of $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$ in Claim 10.

In Step 6, we sample $Y_{\mathsf{E}}[i] \xleftarrow{\$} \{0,1\}^\lambda$ and compute $W_{\mathsf{E}}[i]$ using I_{E}, while implicitly setting $Y_{\mathsf{G}}[i] = Y_{\mathsf{E}}[i] \oplus \Delta_O$ and $W_{\mathsf{G}}[i] = W_{\mathsf{E}}[i] - f_i(x)\mathsf{sk}$. The order of $\mathrm{ct}_0, \mathrm{ct}_1$ may be changed, but they will be randomly permuted anyway. □

Remove $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$. Next, we replace the invocation of protocol $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$ with a suitable simulator Sim_0, as guaranteed by Lemma 13.

Experiment Hyb_2.

1. Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output N.
2. Run $\mathsf{Sim}_0(n, N)$ and output the result. Use the result to compute I'_{E} and y.
3. Let $x = \sum_{i=0}^{n-1} x_i \cdot 2^i, y = \sum_{i=0}^{n-1} y_i \cdot 2^i$ be their binary representation. For $i \in [n]$, let $E[i][0] \leftarrow \mathsf{Enc}(N, 1 - x_i \oplus y_i), E[i][1] \leftarrow \mathsf{Enc}(N, x_i \oplus y_i)$. Output E.
4. Sample and output $r \xleftarrow{\$} [N^\zeta]$. Compute $I_{\mathsf{E}} \in \mathbb{Z}^{2^n}_{N^\zeta}$ with I'_{E}, y and r, E.
5. For $i \in [m]$, sample $Y_{\mathsf{E}}[i] \xleftarrow{\$} \{0,1\}^\lambda$, let $W_{\mathsf{E}}[i] = \sum_{j=0}^{2^n-1} f_i(j) I_{\mathsf{E}}[j] \pmod{N^\zeta}$,

$$\mathrm{ct}_t = \Big(H_3((W_{\mathsf{E}}[i] + (-1)^{f_i(x)} t\mathsf{sk}) \bmod N^\zeta), \\ H_4((W_{\mathsf{E}}[i] + (-1)^{f_i(x)} t\mathsf{sk}) \bmod N^\zeta) \oplus Y_{\mathsf{E}}[i] \oplus t\Delta_O\Big),$$

randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Claim 16. *The experiments* Hyb_1 *and* Hyb_2 *are computationally indistinguishable.*

Proof. Follows from Lemma 13. □

Random Oracle. Next, since $\mathsf{Evaluator}$ cannot compute the secret key sk, it's safe to use sk as encryption key for the ciphertexts.

Experiment Hyb_3.

1. Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output N.
2. Run $\mathsf{Sim}_0(n, N)$ and output the result. Use the result to compute I'_{E} and y.
3. Let $x = \sum_{i=0}^{n-1} x_i \cdot 2^i, y = \sum_{i=0}^{n-1} y_i \cdot 2^i$ be their binary representation. For $i \in [n]$, let $E[i][0] \leftarrow \mathsf{Enc}(N, 1 - x_i \oplus y_i), E[i][1] \leftarrow \mathsf{Enc}(N, x_i \oplus y_i)$. Output E.
4. Sample and output $r \xleftarrow{\$} [N^\zeta]$. Compute $I_{\mathsf{E}} \in \mathbb{Z}_{N^\zeta}^{2^n}$ with I'_{E}, y and r, E.
5. For $i \in [m]$, sample $Y_{\mathsf{E}}[i] \xleftarrow{\$} \{0,1\}^\lambda$, let $W_{\mathsf{E}}[i] = \sum_{j=0}^{2^n-1} f_i(j) I_{\mathsf{E}}[j] \pmod{N^\zeta}$. Let $\mathrm{ct}_0 = (H_3(W_{\mathsf{E}}[i]), H_4(W_{\mathsf{E}}[i]) \oplus Y_{\mathsf{E}}[i])$, and let $\mathrm{ct}_1 \xleftarrow{\$} \{0,1\}^{2\lambda}$. Randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Claim 17. *The experiments Hyb_2 and Hyb_3 are computationally indistinguishable.*

Proof. Consider any PPT adversary $\mathcal{A}$ that can distinguish Hyb_3 from Hyb_2. It's clear that one of the following events must happen with non-negligible probability, when $\mathcal{A}$ is run on the output of Hyb_3: 1) $\mathsf{sk} = W_{\mathsf{E}}[i] - W_{\mathsf{E}}^{(j)} \pmod{N}^\zeta$ for some $i \neq j$, or 2) $\mathcal{A}$ queries H_3 or H_4 on $W_{\mathsf{E}}[i] \pm \mathsf{sk}$ for some i.

We will show that from such an adversary $\mathcal{A}$, we can construct a PPT adversary $\mathcal{A}'$ that breaks the security of the Damgård-Jurik encryption scheme with non-negligible probability.

The adversary $\mathcal{A}'$ works as follows. Given a Damgård-Jurik public key $\mathsf{pk} = N$ and a ciphertext, it simulates Hyb_3 starting from Step 2, checks if $\mathsf{sk} = W_{\mathsf{E}}[i] - W_{\mathsf{E}}^{(j)} \pmod{N}^\zeta$ for some $i \neq j$, and if not, give the output to $\mathcal{A}$. Now, whenever $\mathcal{A}$ queries the random oracles H_3 or H_4 at position p, $\mathcal{A}'$ checks if $p = W_{\mathsf{E}}[i] \pm \mathsf{sk} \pmod{N^\zeta}$ for some i. If any of the above checks succeeds, $\mathcal{A}'$ recovers sk, so it can decrypt the ciphertext.

This contradicts the security of the Damgård-Jurik encryption scheme, so such $\mathcal{A}$ cannot exist. □

Damgård-Jurik Encryption. Finally, we replace the Damgård-Jurik ciphertexts with encryptions of zero.

Experiment Hyb_4.

1. Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output N.
2. Run $\mathsf{Sim}_0(n, N)$ and output the result. Use the result to compute I'_{E} and y.
3. For $i \in [n]$, let $E[i][0] \leftarrow \mathsf{Enc}(N, 0), E[i][1] \leftarrow \mathsf{Enc}(N, 0)$. Output E.
4. Sample and output $r \xleftarrow{\$} [N^\zeta]$. Compute $I_{\mathsf{E}} \in \mathbb{Z}_{N^\zeta}^{2^n}$ with I'_{E}, y and r, E.

5. For $i \in [m]$, sample $Y_{\mathsf{E}}[i] \xleftarrow{\$} \{0,1\}^{\lambda}$, let $W_{\mathsf{E}}[i] = \sum_{j=0}^{2^n-1} f_i(j) I_{\mathsf{E}}[j] \pmod{N^{\zeta}}$. Let $\mathrm{ct}_0 = (H_3(W_{\mathsf{E}}[i]), H_4(W_{\mathsf{E}}[i]) \oplus Y_{\mathsf{E}}[i])$, and let $\mathrm{ct}_1 \xleftarrow{\$} \{0,1\}^{2\lambda}$. Randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Claim 18. *The experiments* Hyb_3 *and* Hyb_4 *are computationally indistinguishable.*

Proof. Follows from the CPA security of the Damgård-Jurik encryption scheme. □

Note that Hyb_4 only requires knowledge of the public parameters $n, m, f_0, \ldots, f_{m-1}$, so it's simulatable by a PPT simulator Sim. This concludes the proof of the theorem.

Remark 19. *The proof also works if H_3, H_4 are modeled as Circular Correlation Robust Hash functions (CCRH), under appropriate definition that allows replacing $H_3(W_{\mathsf{E}}[i] \pm \mathsf{sk})$ and $H_4(W_{\mathsf{E}}[i] \pm \mathsf{sk}) \oplus \Delta_O$ with random values. Combining this proof with another version of Lemma 13, we can prove the privacy of the full construction under the CCRH assumption in the plain model. See Appendix B for details.*

6 Programmable Distributed Point Functions

In this section, we demonstrate how to construct small-domain programmable distributed point functions (PDPFs) using the techniques developed in previous sections. We present two constructions: the first offers highly efficient key generation and programming times (poly-logarithmic in the domain size), while the second introduces a property we call *decomposability*. Decomposability means that the programmed key can be decomposed into n parts, where the i-th part depends solely on the i-th bit of the programming point.

The decomposability property is particularly valuable when the programmed key is generated in a distributed manner. Consider a scenario where one party (the sender) knows the programming value v, and another party (the receiver) knows the programming point x. The sender generates the master key and, for each $i \in [n]$ and $b \in \{0,1\}$, computes the i-th part of the programmed key corresponding to the i-th bit of x being b. The sender and receiver then execute n parallel instances of oblivious transfer (OT), such that the receiver obtains the correct parts. This results in a highly efficient, two-round protocol for distributed key generation.

This protocol can be extended to the case where the two parties hold x_0, v_0 and x_1, v_1, respectively, such that $x_0 \oplus x_1 = x$ and $v_0 + v_1 = v$. In this case, the parties run two parallel instances of the previous protocol, with each party acting as the sender in one instance and the receiver in the other. When the party holding x_t, v_t acts as the sender, it simply uses v_t as the payload and permutes the i-th part of the programmed key according to the i-th bit of x_t. This gives the two parties shares of f_{x,v_0} and f_{x,v_1} with payloads at different sides. Finally they can locally subtract the two shares they take and get shares of $f_{x,v}$. The resulting protocol remains two-round, with each round involving simultaneous messages from both parties.

$\underline{\mathsf{RealProgPriv}^{\mathcal{A}}(1^\lambda, M, \mathbb{G})}$:	$\underline{\mathsf{IdealProgPriv}^{\mathcal{A},\mathsf{Sim}}(1^\lambda, M, \mathbb{G})}$:
$x, v \leftarrow \mathcal{A}(1^\lambda, M, \mathbb{G})$	$x, v \leftarrow \mathcal{A}(1^\lambda, M, \mathbb{G})$
$k_0 \leftarrow \mathsf{Gen}_0(1^\lambda, M, \mathbb{G})$	$k_1 \leftarrow \mathsf{Sim}(1^\lambda, M, \mathbb{G})$
$k_1 \leftarrow \mathsf{Gen}_1(k_0, (M, \mathbb{G}, x, v))$	Output $\mathcal{A}(k_1)$
Output $\mathcal{A}(k_1)$	

Figure 5: Security experiments for Programmable DPF, where the adversary $\mathcal{A}$ is stateful.

6.1 Definition

We follow the definition of PDPF in [BGIK22].

Notations. We use $\mathbb{G}$ to denote an Abelian group. Given a domain size M and an Abelian group $\mathbb{G}$, a *point function* $f_{x,v} : [M] \to \mathbb{G}$ evaluates to v on input x and to 0 on all other inputs.

Syntax. A programmable DPF is a tuple $(\mathsf{Gen}_0, \mathsf{Gen}_1, \mathsf{Eval}_0, \mathsf{Eval}_1)$ of possibly randomized algorithms with the following syntax:

- $\mathsf{Gen}_0(1^\lambda, M, \mathbb{G})$: given the security parameter λ, the input domain M and group description $\mathbb{G}$, output a key k_0.
- $\mathsf{Gen}_1(k_0, \hat{f})$: given the key k_0 and the description of a point function $\hat{f} = (M, \mathbb{G}, x, v)$, output a key k_1.
- $\mathsf{Eval}_i(k_i, x)$: given a key k_i and an input $x \in [M]$, output the evaluation outcome $v \in \mathbb{G}$.

Correctness. For any polynomially bounded function $M(\cdot)$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that for all λ, for all point function descriptions $\hat{f} = (M, \mathbb{G}, x, v)$ where $x \in [M]$ and $v \in \mathbb{G}$, we have the following:

$$\Pr\left[\begin{array}{cc} k_0 \leftarrow \mathsf{Gen}_0(1^\lambda, M, \mathbb{G}), & \mathsf{Eval}_1(k_1, x) = \mathsf{Eval}_0(k_0, x) + v \ \text{ and} \\ k_1 \leftarrow \mathsf{Gen}_1(k_0, \hat{f}) \quad : & \forall x' \neq x, \mathsf{PEval}(k_1, x') = \mathsf{Eval}(k_0, x') \end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Security. We require there exists a PPT algorithm Sim such that for any polynomially bounded function $M(\cdot)$, the experiments $\mathsf{RealProgPriv}$ and $\mathsf{IdealProgPriv}$ given in Figure 5 are computationally indistinguishable.

6.2 Construction

In this section, we assume the domain size M is a power of 2, and set $n = \log_2 M$.

Overview. The construction is in the same spirit as our real one-hot gate. Basically, Gen_0 generates the Boolean share X_G and the one-hot share I_G held by $\mathsf{Garbler}$, and Gen_1 generates the garbled materials and the Boolean share X_E held by $\mathsf{Evaluator}$. Eval_1 acts as $\mathsf{Evaluator}$, using X_E and the garbled materials to compute the one-hot share I_E. Since our real one-hot gate is secure, the resulting PDPF does not leak any information about the programmed point.

However, the result of our real one-hot gate is always an instance of $[\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$, while the PDPF requires replacing sk with a specific value v. We can send $v \cdot \mathsf{sk}$ instead of sk when calling $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$, such that the result would be $[\![\mathcal{I}^{(n)}(x, v \cdot \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$. We can then naively use $\mathsf{Enc}(N, \mathsf{sk}^{-1} \bmod N^\zeta)$ to remove sk from the result. It works, but requires the key-dependent message (KDM) security of the Damgård-Jurik encryption scheme, which is not ideal.

A natural idea is to use $d = \mathsf{sk} \cdot (\mathsf{sk}^{-1} \bmod N^\zeta)$ instead of sk, which satisfies $\mathsf{Enc}(N, c)^d = \exp(c)$ for any $c \in [N^\zeta]$, so the extra sk is removed from the result. This has a fatal flaw, though: d can be as large as $N^{\zeta+1}$, too large to be stored in a ciphertext or a subtractive secret share, where the values are modulo N^ζ.

Following the idea of [RS21, MORS24b], we use $\nu = N^{-\zeta} \bmod \mathsf{sk}$ instead of d in the secret share. By Chinese Remainder Theorem, it must be that $d \equiv 1 - N^\zeta \cdot \nu \pmod{\mathsf{sk} \cdot N^\zeta}$, so we can obtain a subtractive secret share of $v \cdot d$ by combining the subtractive secret shares of v and $v \cdot \nu$.

Now d disappears after exponentiation, but we still need secret shares of $v \cdot \nu$ in the next round. We can send $\mathsf{Enc}(N, (1 - c_i)\nu)$ and $\mathsf{Enc}(N, c_i\nu)$ alongside $\mathsf{Enc}(N, 1 - c_i)$ and $\mathsf{Enc}(N, c_i)$, but this defeats the purpose of not relying on KDM security. Instead, we use the key switching technique, where we use different Damgård-Jurik keys for different rounds, and use the current key to encrypt the ν of the next round.

Formal Construction. For a Damgård-Jurik public key N, let $\exp_N(x) := \sum_{k=0}^{\zeta} \frac{(Nx)^k}{k!} \bmod N^{\zeta+1}$, $\log_N(1 + Nx) := \sum_{k=1}^{\zeta} \frac{(-N)^{k-1}x^k}{k} \bmod N^\zeta$, and $\mathrm{ddlog}_N(x) := \log_N(x \cdot (x^{-1} \bmod N) \bmod N^{\zeta+1})$. Further, these functions are extended element-wise to vectors.

We first define the procedure $\mathsf{OblivShift}$ in Figure 6, which corresponds to the transform from a shifted one-hot share to a real one-hot share. The procedure satisfies the following correctness property:

Claim 20. *Let $(N_0, \mathsf{sk}_0), \ldots, (N_{n-1}, \mathsf{sk}_{n-1})$ be Damgård-Jurik key pairs, let $d_i = \mathsf{sk}_i \cdot (\mathsf{sk}_i^{-1} \bmod N_i^\zeta)$, and let $\nu_i = N_i^{-\zeta} \bmod \mathsf{sk}_i$. Let $0 \le v \le 2^{(\zeta-1)\lambda_{\mathsf{DCR}}-\lambda}$ be an integer. Let $I'_\mathsf{G}, I'_\mathsf{E} \in \mathbb{Z}^{2^n}$, such that $I'_\mathsf{E} - I'_\mathsf{G} = \mathcal{I}^{(n)}(y, v \cdot d_0)$. Let $c \in [2^n]$ be an integer with binary representation $c = \sum_{i=0}^{n-1} c_i 2^i$. Let $E[i][b]$ be an encryption of $c_i \oplus b \oplus 1$ and $F[i][b]$ be an encryption of $(c_i \oplus b \oplus 1) \cdot \nu_{i+1}$, for $i \in [n]$ and $b \in \{0, 1\}$. In particular, $F[n-1][b]$ can be anything. Let r be a random integer in $[2^{2\zeta\lambda_{\mathsf{DCR}}}]$. Then $\mathsf{OblivShift}(N, E, F, r, I'_\mathsf{E}) - \mathsf{OblivShift}(N, E, F, r, I'_\mathsf{G}) = \mathcal{I}^{(n)}(y \oplus c, v)$ expect with negligible probability in λ.*

Procedure OblivShift

Input.

- n Damgård-Jurik public keys $N_0, \ldots, N_{n-1}$.
- $2n$ Damgård-Jurik ciphertexts $E[0][0], E[0][1], \ldots, E[n-1][0], E[n-1][1]$, where $E[i][0], E[i][1]$ are encrypted under N_i.
- $2n$ Damgård-Jurik ciphertexts $F[0][0], F[0][1], \ldots, F[n-1][0], F[n-1][1]$, where $F[i][0], F[i][1]$ are encrypted under N_i.
- A random integer $r \in [2^{2\zeta\lambda_{\mathsf{DCR}}}]$.
- A vector $I' \in \mathbb{Z}^{2^n}$.

Procedure.

1. Let $I^{(0)} = I'$.
2. For i from 0 to $n-1$:
   (a) Let $\widetilde{I}^{(i)}[0] = E[i][0]^{I^{(i)}} \cdot E[i][1]^{\mathsf{shift}(I^{(i)}, 2^i)}$, $\widetilde{I}^{(i)}[1] = F[i][0]^{I^{(i)}} \cdot F[i][1]^{\mathsf{shift}(I^{(i)}, 2^i)}$, where $\mathsf{shift}(I, 2^i)[j] = I[j \oplus 2^i]$.
   (b) Let $\widehat{I}^{(i)}[0] = \mathrm{ddlog}_{N_i}(\widetilde{I}^{(i)}[0])$ and $\widehat{I}^{(i)}[1] = \mathrm{ddlog}_{N_i}(\widetilde{I}^{(i)}[1])$. Then add r to all entries in $\widehat{I}^{(i)}$ modulo N_i^{ζ}.
   (c) If $i \neq n-1$, let $I^{(i+1)} = \widehat{I}^{(i)}[0] - N_{i+1}^{\zeta} \cdot \widehat{I}^{(i)}[1]$, viewed as integers.
3. Output $\widehat{I}^{(n)}[0]$ as integers.

Figure 6: Oblivious Shift

Proof. Define $I_{\mathsf{G}}^{(i)}$ to be the intermediate value $I^{(i)}$ when running $\mathsf{OblivShift}(I'_{\mathsf{G}})$, and define $I_{\mathsf{E}}^{(i)}, \widetilde{I}_{\mathsf{G}}^{(i)}, \widetilde{I}_{\mathsf{E}}^{(i)}, \widehat{I}_{\mathsf{G}}^{(i)}, \widehat{I}_{\mathsf{E}}^{(i)}$ in a similar way.

Let $y^{(i)} = y \oplus \sum_{j=0}^{i-1} c_j 2^j$. By using induction on i, we can prove that except with negligible probability in λ (which comes from converting vectors from $\mathbb{Z}_{N_i^\zeta}^{2^n}$ to $\mathbb{Z}^{2^n}$),

- $I_{\mathsf{E}}^{(i)} - I_{\mathsf{G}}^{(i)} \equiv \mathcal{I}^{(n)}(y^{(i)}, v \cdot d_i) \pmod{\mathsf{sk}_i \cdot N_i^\zeta}$.
- $(\widetilde{I}_{\mathsf{G}}^{(i)}[b], \widetilde{I}_{\mathsf{E}}^{(i)}[b]) \in [\![\mathcal{I}^{(n)}(y^{(i+1)}, v \cdot \nu_{i+1}^b)]\!]_{N_i}^{\mathsf{div}}$.
- $\widehat{I}_{\mathsf{E}}^{(i)}[b] - \widehat{I}_{\mathsf{G}}^{(i)}[b] = \mathcal{I}^{(n)}(y^{(i+1)}, v \cdot \nu_{i+1}^b)$ (viewed as integers, after adding r).
- $I_{\mathsf{G}}^{(i+1)} - I_{\mathsf{E}}^{(i+1)} \equiv \mathcal{I}^{(n)}(y^{(i+1)}, v \cdot d_{i+1}) \pmod{\mathsf{sk}_{i+1} \cdot N_{i+1}^\zeta}$.

where we used the facts that $d \equiv 1 - N^\zeta \cdot \nu \pmod{\mathsf{sk} \cdot N^\zeta}$, $\mathsf{Enc}(N, w)^d \equiv \exp_N(w) \pmod{N^{\zeta+1}}$ and $\mathsf{Enc}(N, w)^{\mathsf{sk} \cdot N^\zeta} \equiv 1 \pmod{N^{\zeta+1}}$ for any $w \in [N^\zeta]$. □

Now all we need to do is generate the initial I'_{G} and I'_{E}. Since we are allowed to reveal the punctured point $x \oplus c$, this can be achieved using the classical puncturable PRF construction based on the GGM tree. The full construction is given in Figure 7.

Theorem 21. *Assuming the DCR assumption, the construction in Figure 7 is a programmable distributed point function for $f_{x,v} : [2^n] \to \mathbb{G}$, for any cyclic group $\mathbb{G}$ with size smaller than $2^{(\zeta-1)\lambda_{\mathsf{DCR}}-\lambda}$. Gen_0 runs in time $O(n\lambda{\lambda_{\mathsf{DCR}}}^2)$, Gen_1 runs in time $O(n\lambda+\lambda_{\mathsf{DCR}})$, key size is $O(n\lambda_{\mathsf{DCR}})$, and full-domain evaluation runs in time $O(2^n{\lambda_{\mathsf{DCR}}}^2)$.*

Correctness. It's clear from definition that $I'_{\mathsf{E}} - I'_{\mathsf{G}} = \mathcal{I}^{(n)}(y, v \cdot d_0)$, and the rest follows from the correctness of $\mathsf{OblivShift}$.

Security. Note that L is generated similar to a GGM tree, and each $P[i]$ represents a sibling to the path from the root to $L^{(n)}[y]$, so P and $L^{(n)}[y]$ are pseudorandom. Then $G_2(L^{(n)}[y])$ is pseudorandom in range $[2^{2\zeta\lambda_{\mathsf{DCR}}}]$, which is much larger than $v \cdot d_0$, so w is also computationally indistinguishable from a random integer in range $[2^{2\zeta\lambda_{\mathsf{DCR}}}]$. Now we removed all dependency on sk_0, so $E[0][0], E[0][1], F[0][0], F[0][1]$ can all be replaced by encryptions of zero. We can then continue to replace $E[1][0], E[1][1], F[1][0], F[1][1]$, and so on.

Efficiency. Gen_0 runs in time $O(n\lambda{\lambda_{\mathsf{DCR}}}^2)$ for generating $O(n)$ ciphertexts. Gen_1 only needs to expand the GGM tree through a single path, and do several calculation in $[2^{2\zeta\lambda_{\mathsf{DCR}}}]$, so it runs in time $O(n\lambda+\lambda_{\mathsf{DCR}})$.[8] Both Eval_0 and Eval_1 have the same bottleneck, which occurs during the execution of $\mathsf{OblivShift}$, running in $O(2^n{\lambda_{\mathsf{DCR}}}^2)$ time. Similar to [BGIK22],

[8] We disregarded the time required to output N, E, F, as it merely involves data transfer without any actual computation.

Programmable Distributed Point Function

Notation. We assume two pseudorandom number generators (PRG) $G_1 : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}, G_2 : \{0,1\}^\lambda \to [2^{2\zeta\lambda_{\mathsf{DCR}}}]$.

$\mathsf{Gen}_0(1^\lambda, M, \mathbb{G})$:

1. Sample $L^{(0)} \xleftarrow{\$} (\{0,1\}^\lambda)^1$, i.e. a vector with a single element.
2. Sample $c \xleftarrow{\$} [2^n]$, and let $c = \sum_{i=0}^{n-1} c_i 2^i$ be its binary representation.
3. Sample n Damgård-Jurik key pairs $(N_0, \mathsf{sk}_0), \ldots, (N_{n-1}, \mathsf{sk}_{n-1}) \leftarrow \mathsf{Gen}(1^\lambda)$. Let $\nu_i = N_i^{-\zeta} \bmod \mathsf{sk}_i$ for $i \in [n]$.
4. For $i \in [n]$, let $E[i][0] \leftarrow \mathsf{Enc}(N_i, 1 - c_i), E[i][1] \leftarrow \mathsf{Enc}(N_i, c_i)$.
5. For $i \in [n]$, let $F[i][0] \leftarrow \mathsf{Enc}(N_i, (1 - c_i)\nu_{i+1}), F[i][1] \leftarrow \mathsf{Enc}(N_i, c_i\nu_{i+1})$.
6. Sample $r \xleftarrow{\$} [2^{2\zeta\lambda_{\mathsf{DCR}}}]$.
7. Output $k_0 = (L^{(0)}, c, N, E, F, r)$.

$\mathsf{Gen}_1(k_0 = (L^{(0)}, c, N, E, F, r), \hat{f} = (M, \mathbb{G}, x, v))$:

1. For $i \in [n]$, define $L^{(i+1)}$ to be a vector of length 2^{i+1}, where $L^{(i+1)}[j] \| L^{(i+1)}[j + 2^i] = G_1(L^{(i)}[j])$ for $j \in [2^i]$.
2. Let $y = x \oplus c$, and let $y = \sum_{i=0}^{n-1} y_i 2^i$ be its binary representation.
3. For $i \in [n-1]$, let $P[i] = L^{(i+1)}\left[(1 - y_i)2^i + \sum_{j=0}^{i-1} y_j 2^j\right]$.
4. Let $w = v \cdot d_0 + G_2(L^{(n)}[y])$, where $d_0 = \mathsf{sk}_0 \cdot (\mathsf{sk}_0^{-1} \bmod N_0^\zeta)$.
5. Output $k_1 = (N, E, F, r, y, P, w)$.

$\mathsf{Eval}_0(k_0 = (L^{(0)}, c, N, E, F, r), x)$:

1. Compute $L^{(n)}$ as defined in Gen_1.
2. Let $I'_{\mathsf{G}}[j] = G_2(L^{(n)}[j])$ for $j \in [2^n]$.
3. Output $\mathsf{OblivShift}(N, E, F, r, I'_{\mathsf{G}})[x]$.

$\mathsf{Eval}_1(k_1 = (N, E, F, r, y, P, w), x)$:

1. Use P to compute $L^{(n)}$ except $L^{(n)}[y]$. We omit the details since this is a standard construction of puncturable PRF based on GGM tree.
2. Let $I'_{\mathsf{E}}[j] = G_2(L^{(n)}[j])$ for $j \in [2^n] \setminus \{y\}$, and set $I'_{\mathsf{E}}[y] = w$.
3. Output $\mathsf{OblivShift}(N, E, F, r, I'_{\mathsf{E}})[x]$.

Figure 7: Programmable Distributed Point Function.

our construction has the same efficiency for evaluating at a single point and for evaluating at all points.

6.3 Recovering Decomposability

While the construction in Figure 7 is quite efficient in terms of key generation, it lost an important property of the real one-hot gate – independency between bits of the programmed point (i.e. decomposability).

We explain this property in more details. The input to the real one-hot gate is an XOR secret share of the Boolean label of x, where each bit of x is shared independently. The garbled materials does not depend on x. Thus, the information held by $\mathsf{Evaluator}$ can be split into n independent parts, each corresponding to a bit of x. Ideally, we would like to have the same decomposable property in the PDPF, i.e., the programmed key k_1 should be split into n independent parts, each corresponding to a bit of x.

We give a construction in Figure 8 that recovers this property, using techniques in the shifted one-hot gate. However, this comes at the cost of slower key generation. We mark the changes in blue compared to the original construction.

Theorem 22. *Assuming the DCR assumption, the construction in Figure 8 is a programmable distributed point function for $f_{x,v} : [2^n] \to \mathbb{G}$, for any cyclic group $\mathbb{G}$ with size smaller than $2^{(\zeta-1)\lambda_{\mathsf{DCR}}-\lambda}$. Gen_0 runs in time $O(2^n\lambda_{\mathsf{DCR}} + n{\lambda_{\mathsf{DCR}}}^2)$, Gen_1 runs in time $O(n\lambda+\lambda_{\mathsf{DCR}})$, key size is $O(n\lambda_{\mathsf{DCR}})$, and full-domain evaluation runs in time $O(2^n{\lambda_{\mathsf{DCR}}}^2)$.*

Proof. Correctness is satisfied by the same argument as before.

Compared to the original construction, the P is XORed with some extra terms, and w is added with some extra terms. However, the extra terms are meant to be known to the adversary anyway, so they do not affect security. □

Remark. While Gen_0 runs in time linear in the domain size, Gen_1 remains efficient. We argue that Gen_0 is generally run as the offline phase (before the point function is known) in applications of PDPF, allowing for more computational time. In contrast, an efficient Gen_1 is critical for ensuring an efficient online phase, which is often more important in practice.

References

[ACK23] Thomas Attema, Pedro Capitão, and Lisa Kohl. On homomorphic secret sharing from polynomial-modulus LWE. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part II*, volume 13941 of *LNCS*, pages 3–32. Springer, Cham, May 2023.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Dodis and Shrimpton [DS22], pages 421–452.

Programmable Distributed Point Function with decomposable key

Notation. We assume two pseudorandom number generators (PRG) $G_1 : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}, G_2 : \{0,1\}^\lambda \to [2^{2\zeta\lambda_{\mathsf{DCR}}}]$.

$\mathsf{Gen}_0(1^\lambda, M, \mathbb{G})$:

1. Sample $L^{(0)} \xleftarrow{\$} (\{0,1\}^\lambda)^1$, i.e. a vector with a single element.
2. Sample $c \xleftarrow{\$} [2^n]$, and let $c = \sum_{i=0}^{n-1} c_i 2^i$ be its binary representation.
3. Sample n Damgård-Jurik key pairs $(N_0, \mathsf{sk}_0), \ldots, (N_{n-1}, \mathsf{sk}_{n-1}) \leftarrow \mathsf{Gen}(1^\lambda)$. Let $\nu_i = N_i^{-\zeta} \bmod \mathsf{sk}_i$ for $i \in [n]$.
4. For $i \in [n]$, let $E[i][0] \leftarrow \mathsf{Enc}(N_i, 1 - c_i), E[i][1] \leftarrow \mathsf{Enc}(N_i, c_i)$.
5. For $i \in [n]$, let $F[i][0] \leftarrow \mathsf{Enc}(N_i, (1 - c_i)\nu_{i+1}), F[i][1] \leftarrow \mathsf{Enc}(N_i, c_i\nu_{i+1})$.
6. Sample $r \xleftarrow{\$} [2^{2\zeta\lambda_{\mathsf{DCR}}}]$.
7. For $i \in [n]$, let $L^{(i+1)}$ be a vector of length 2^{i+1}, where $L^{(i+1)}[j] \| L^{(i+1)}[j + 2^i] = G_1(L^{(i)}[j])$ for $j \in [2^i]$.
8. For $i \in [n-1]$, let $Q[i][b] = \bigoplus_{j=b2^i}^{(b+1)2^i - 1} L^{(i+1)}[j]$. Let $s = \sum_{j=0}^{2^n-1} G_2(L^{(n)}[j])$.
9. Output $k_0 = (L^{(0)}, c, N, E, F, r, Q, s)$.

$\mathsf{Gen}_1(k_0 = (L^{(0)}, c, N, E, F, r, Q, s), \hat{f} = (M, \mathbb{G}, x, v))$:

1. Let $y = x \oplus c$, and let $y = \sum_{i=0}^{n-1} y_i 2^i$ be its binary representation.
2. For $i \in [n-1]$, let $P[i] = Q[i][1 - y_i]$.
3. Let $w = v \cdot d_0 + s$, where $d_0 = \mathsf{sk}_0 \cdot (\mathsf{sk}_0^{-1} \bmod N_0^\zeta)$.
4. Output $k_1 = (N, E, F, r, y, P, w)$.

$\mathsf{Eval}_0(k_0 = (L^{(0)}, c, N, E, F, r, Q, s), x)$:

1. Compute $L^{(n)}$ as defined in Gen_0.
2. Let $I'_{\mathsf{G}}[j] = G_2(L^{(n)}[j])$ for $j \in [2^n]$.
3. Output $\mathsf{OblivShift}(N, E, F, r, I'_{\mathsf{G}})[x]$.

$\mathsf{Eval}_1(k_1 = (N, E, F, r, y, P, w), x)$:

1. Use P to compute $L^{(n)}$ except $L^{(n)}[y]$.
2. Let $I'_{\mathsf{E}}[j] = G_2(L^{(n)}[j])$ for $j \in [2^n] \setminus \{y\}$, and set $I'_{\mathsf{E}}[y] = w - \sum_{j \neq y} I'_{\mathsf{E}}[j]$.
3. Output $\mathsf{OblivShift}(N, E, F, r, I'_{\mathsf{E}})[x]$.

Figure 8: Programmable Distributed Point Function with decomposable key.

[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.

[AMN+18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Shacham and Boldyreva [SB18], pages 543–574.

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

[BDSS25] Elette Boyle, Lalita Devadas, and Sacha Servan-Schreiber. Non-interactive distributed point functions. Cryptology ePrint Archive, Paper 2025/095, 2025.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Oswald and Fischlin [OF15], pages 337–367.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Berlin, Heidelberg, August 2016.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193. Springer, Cham, April / May 2017.

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Dodis and Shrimpton [DS22], pages 121–151.

[BHR12] 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.

[BKM17] Dan Boneh, Sam Kim, and Hart Montgomery. Private puncturable prfs from standard lattice assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 415–445, Cham, 2017. Springer International Publishing.

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 3–33. Springer, Cham, May 2019.

[BLLL23] Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 3–34. Springer, Cham, April 2023.

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 494–524. Springer, Berlin, Heidelberg, March 2017.

[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.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 565–577. ACM Press, October 2016.

[BPP00] Joan Boyar, René Peralta, and Denis Pochuev. On the multiplicative complexity of boolean functions over the basis (cap, +, 1). *Theor. Comput. Sci.*, 235(1):43–57, 2000.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 264–302. Springer, Cham, November 2017.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 446–476. Springer, Cham, April / May 2017.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Cham, August 2017.

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 194–224. Springer, Cham, April 2023.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Shacham and Boldyreva [SB18], pages 577–607.

[DD22] Orr Dunkelman and Stefan Dziembowski, editors. *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*. Springer, Cham, May / June 2022.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Berlin, Heidelberg, February 2001.

[DKK18] Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 213–242. Springer, Cham, August 2018.

[DKN+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 559–589. Springer, Cham, August 2020.

[DS22] Yevgeniy Dodis and Thomas Shrimpton, editors. *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*. Springer, Cham, August 2022.

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *Provable Security*, pages 381–399, Cham, 2017. Springer International Publishing.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4), August 1986.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 640–658. Springer, Berlin, Heidelberg, May 2014.

[GLNP18] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. *Journal of Cryptology*, 31(3):798–844, July 2018.

[Hea22] David Heath. *New Directions in Garbled Circuits.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2022.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In Joye and Leander [JL24], pages 3–31.

[HHK+22] Abida Haque, David Heath, Vladimir Kolesnikov, Steve Lu, Rafail Ostrovsky, and Akash Shah. Garbled circuits with sublinear evaluator. In Dunkelman and Dziembowski [DD22], pages 37–64.

[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, Cham, August 2020.

[HK21a] 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.

[HK21b] David Heath and Vladimir Kolesnikov. LogStack: Stacked garbling with $O(b \log b)$ computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 3–32. Springer, Cham, October 2021.

[HKN24] David Heath, Vladimir Kolesnikov, and Lucien K. L. Ng. Garbled circuit lookup tables with logarithmic number of ciphertexts. In Joye and Leander [JL24], pages 185–215.

[HKO22] David Heath, Vladimir Kolesnikov, and Rafail Ostrovsky. EpiGRAM: Practical garbled RAM. In Dunkelman and Dziembowski [DD22], pages 3–33.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 575–594. Springer, Berlin, Heidelberg, February 2007.

[JL24] Marc Joye and Gregor Leander, editors. *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*. Springer, Cham, May 2024.

[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, Berlin, Heidelberg, August 2014.

[Kol18] Vladimir Kolesnikov. Free IF: How to omit inactive branches and implement S-universal garbled circuit (almost) for free. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 34–58. Springer, Cham, December 2018.

[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, Berlin, Heidelberg, July 2008.

[LL24] Hanjun Li and Tianren Liu. How to garble mixed circuits that combine boolean and arithmetic computations. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 331–360. Springer, Cham, May 2024.

[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, Berlin, Heidelberg, May 2013.

[MORS24a] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret-sharing. Cryptology ePrint Archive, Report 2024/820, 2024.

[MORS24b] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 71–97. Springer, Cham, December 2024.

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

[OF15] Elisabeth Oswald and Marc Fischlin, editors. *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*. Springer, Berlin, Heidelberg, April 2015.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Cham, October 2021.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[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, Cham, April 2023.

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 675–701. Springer, Cham, March 2018.

[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, Berlin, Heidelberg, December 2009.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.

[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, Cham.

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Cham.

[SB18] Hovav Shacham and Alexandra Boldyreva, editors. *CRYPTO 2018, Part II*, volume 10992 of *LNCS*. Springer, Cham, August 2018.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Oswald and Fischlin [OF15], pages 220–250.

A Shifted One-Hot Gate

A.1 Construction

We present our version of the shifted one-hot gate in Figure 9.

Proof of Claim 10. Correctness follows by verifying the loop invariant using induction.

Sending P incurs $O(n\lambda)$ communication, and sending w' incurs $O(\log N)$ communication. Computation bottleneck is the $O(2^n)$ random oracle queries, where each query takes $O(\log N)$ time. □

$\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$: Shifted One-Hot Gate

Input.

- Public parameter: A positive integer n, a Damgård-Jurik public key $N \leq 2^{\lambda_{\mathsf{DCR}}}$, two random oracles $H_1 : \{0,1\}^\lambda \to \{0,1\}^\lambda, H_2 : \{0,1\}^\lambda \to [2^{2\zeta\lambda_{\mathsf{DCR}}}]$.
- From Garbler: $X_{\mathsf{G}} = (X_{\mathsf{G}}[0], \ldots, X_{\mathsf{G}}[n-1]) \in (\{0,1\}^\lambda)^n$, an integer $w \in [N^\zeta]$, and a λ-length bit string $\Delta \in 1\{0,1\}^{\lambda-1}$.
- From Evaluator: $X_{\mathsf{E}} = (X_{\mathsf{E}}[0], \ldots, X_{\mathsf{E}}[n-1]) \in (\{0,1\}^\lambda)^n$.
- Required: $(X_{\mathsf{G}}, X_{\mathsf{E}}) \in [\![\mathfrak{B}(x, \Delta)]\!]^{\mathsf{xor}}$, where $x \in [2^n]$.

Output. Garbler outputs (I'_{G}, c), and Evaluator outputs (I'_{E}, y), where $y = x \oplus c$ and $(I'_{\mathsf{G}}, I'_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(y, w)]\!]^{\mathsf{sub}}_{N^\zeta}$.

Protocol.

1. Garbler let $c_i = X_{\mathsf{G}}[i][0]$ for $i \in [n]$, and let $c = \sum_{i=0}^{n-1} c_i 2^i$.
2. Evaluator let $y_i = X_{\mathsf{E}}[i][0]$ for $i \in [n]$, and let $y = \sum_{i=0}^{n-1} y_i 2^i$. // $y = x \oplus c$
3. Garbler samples $L^{(0)}_{\mathsf{G}} \xleftarrow{\$} (\{0,1\}^\lambda)^1$, i.e. a vector where the only element is a random λ-bit string. Garbler sends $L^{(0)}_{\mathsf{G}} \oplus \Delta$ to Evaluator, who sets it to be $L^{(0)}_{\mathsf{E}}$.
4. For i from 0 to $n-1$:
   (a) **Invariant**: $(L^{(i)}_{\mathsf{G}}, L^{(i)}_{\mathsf{E}}) \in [\![\mathcal{I}^{(i)}(y^{(i)}, \Delta)]\!]^{\mathsf{xor}}$, where $y^{(i)} := \sum_{i'=0}^{i-1} y_{i'} 2^{i'}$. [a]
   (b) Garbler computes $\widehat{L}^{(i)}_{\mathsf{G}}[j] = H_1(L^{(i)}_{\mathsf{G}}[j])$ for $j \in [2^i]$, and sends $P[i] = X_{\mathsf{G}}[i] \oplus \bigoplus_{j=0}^{2^i-1} \widehat{L}^{(i)}_{\mathsf{G}}[j]$ to Evaluator.
   (c) Evaluator computes $\widehat{L}^{(i)}_{\mathsf{E}}[j] = H_1(L^{(i)}_{\mathsf{E}}[j])$ for $j \in [2^i] \setminus \{y^{(i)}\}$, and sets $\widehat{L}^{(i)}_{\mathsf{E}}[y^{(i)}] = X_{\mathsf{E}}[i] \oplus P[i] \oplus \bigoplus_{j \in [2^i] \setminus \{y^{(i)}\}} \widehat{L}^{(i)}_{\mathsf{E}}[j]$. // $(\widehat{L}^{(i)}_{\mathsf{G}}, \widehat{L}^{(i)}_{\mathsf{E}}) \in [\![\mathcal{I}^{(i)}(y^{(i)}, y_i\Delta)]\!]^{\mathsf{xor}}$
   (d) Garbler and Evaluator set $L^{(i+1)}_{\mathsf{G}} = (\widehat{L}^{(i)}_{\mathsf{G}} \oplus L^{(i)}_{\mathsf{G}}) \| \widehat{L}^{(i)}_{\mathsf{G}}$ and $L^{(i+1)}_{\mathsf{E}} = (\widehat{L}^{(i)}_{\mathsf{E}} \oplus L^{(i)}_{\mathsf{E}}) \| \widehat{L}^{(i)}_{\mathsf{E}}$.
5. Garbler computes $I'_{\mathsf{G}}[i] = H_2(L^{(n)}_{\mathsf{G}}[i])$ for $i \in [2^n]$, and Evaluator computes $I'_{\mathsf{E}}[i] = H_2(L^{(n)}_{\mathsf{E}}[i])$ for $i \in [2^n] \setminus \{y\}$.
6. Garbler computes $w' = w + \sum_{i=0}^{2^n-1} I'_{\mathsf{G}}[i] \pmod{N^\zeta}$, sends w' to Evaluator, and Evaluator sets $I'_{\mathsf{E}}[y] = w' - \sum_{i \neq y} I'_{\mathsf{E}}[i] \pmod{N^\zeta}$. // $I'_{\mathsf{E}}[y] = I'_{\mathsf{G}}[y] + w$
7. Garbler outputs (I'_{G}, c), and Evaluator outputs (I'_{E}, y).

[a] We slightly abused notation here by putting Δ in the one-hot encoding. It should be viewed as an integer in $[2^\lambda]$.

Figure 9: Shifted One-Hot Gate

A.2 Proof of Lemma 13

We first rewrite the experiment $\mathsf{RealShiftOneHotPriv}$, highlighting the outputs.

Experiment Hyb_0.

1. Uniformly sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ and $(X_{\mathsf{G}}, X_{\mathsf{E}}) \xleftarrow{\$} [\![\mathfrak{B}(x,\Delta)]\!]^{\mathsf{xor}}$. Output X_{E}.
2. Sample $L_{\mathsf{G}}^{(0)} \xleftarrow{\$} (\{0,1\}^{\lambda})^1$, and output $L_{\mathsf{G}}^{(0)} \oplus \Delta$.
3. For $i \in [n]$, for $j \in [2^i]$, let $\widehat{L}_{\mathsf{G}}^{(i)}[j] = H_1(L_{\mathsf{G}}^{(i)}[j])$, and output $P[i] = X_{\mathsf{G}}[i] \oplus \bigoplus_{j=0}^{2^i-1} \widehat{L}_{\mathsf{G}}^{(i)}[j]$. Let $L_{\mathsf{G}}^{(i+1)} = (\widehat{L}_{\mathsf{G}}^{(i)} \oplus L_{\mathsf{G}}^{(i)}) \| \widehat{L}_{\mathsf{G}}^{(i)}$.
4. Let $w' \equiv w + \sum_{i=0}^{2^n-1} H_2(L_{\mathsf{G}}^{(n)}[i]) \pmod{N^{\zeta}}$. Output w'.

Identity Substitution. Next, we replace $X_{\mathsf{G}}[i]$ with $X_{\mathsf{E}}[i] \oplus x_i\Delta$, $L_{\mathsf{G}}^{(i)}$ with $L_{\mathsf{E}}^{(i)} \oplus \mathcal{I}^{(i)}(y^{(i)}, \Delta)$, and $\widehat{L}_{\mathsf{G}}^{(i)}$ with $\widehat{L}_{\mathsf{E}}^{(i)} \oplus \mathcal{I}^{(i)}(y^{(i)}, y_i\Delta)$.

Experiment Hyb_1.

1. Uniformly sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ and $X_{\mathsf{E}} \xleftarrow{\$} \{0,1\}^{\lambda}$. Output X_{E}.
2. Sample $L_{\mathsf{E}}^{(0)} \xleftarrow{\$} (\{0,1\}^{\lambda})^1$, and output $L_{\mathsf{E}}^{(0)}$.
3. For $i \in [n]$, let $y_i = X_{\mathsf{E}}[i][0]$, and $y^{(i)} = \sum_{i'=0}^{i-1} y_{i'} 2^{i'}$. Let $y = \sum_{i=0}^{n-1} y_i 2^i$.
4. For $i \in [n]$, for $j \in [2^i] \setminus y^{(i)}$, let $\widehat{L}_{\mathsf{E}}^{(i)}[j] = H_1(L_{\mathsf{E}}^{(i)}[j])$, output $P[i] = X_{\mathsf{E}}[i] \oplus x_i\Delta \oplus H_1(L_{\mathsf{E}}^{(i)}[y^{(i)}] \oplus \Delta) \oplus \bigoplus_{j \in [2^i] \setminus y^{(i)}} \widehat{L}_{\mathsf{E}}^{(i)}[j]$, and let $\widehat{L}_{\mathsf{E}}^{(i)}[y^{(i)}] = X_{\mathsf{E}}[i] \oplus P[i] \oplus \bigoplus_{j \in [2^i] \setminus y^{(i)} \oplus \{y^{(i)}\}} \widehat{L}_{\mathsf{E}}^{(i)}[j]$. Let $L_{\mathsf{E}}^{(i+1)} = (\widehat{L}_{\mathsf{E}}^{(i)} \oplus L_{\mathsf{E}}^{(i)}) \| \widehat{L}_{\mathsf{E}}^{(i)}$.
5. Let $w' \equiv w + H_2(L_{\mathsf{E}}^{(n)}[y] \oplus \Delta) + \sum_{i \in [2^n] \setminus \{y\}} H_2(L_{\mathsf{E}}^{(n)}[i]) \pmod{N^{\zeta}}$. Output w'.

Claim 23. *The experiments Hyb_0 and Hyb_1 are identical.*

Proof. We are substituting equal values in Hyb_0 and Hyb_1. □

Random Oracles. Next, we note that Δ is only used in computing $x_i\Delta \oplus H_1(L_{\mathsf{E}}^{(i)}[y^{(i)}] \oplus \Delta)$ and $w + H_2(L_{\mathsf{E}}^{(n)}[y] \oplus \Delta)$. Since Δ is uniformly random from $2^{\lambda-1}$ possibilities, we can replace them with random values.

Experiment Hyb_2.

1. Uniformly sample $X_{\mathsf{E}} \xleftarrow{\$} \{0,1\}^{\lambda}$. Output X_{E}.
2. Sample $L_{\mathsf{E}}^{(0)} \xleftarrow{\$} (\{0,1\}^{\lambda})^1$, and output $L_{\mathsf{E}}^{(0)}$.
3. For $i \in [n]$, let $y_i = X_{\mathsf{E}}[i][0]$, and $y^{(i)} = \sum_{i'=0}^{i-1} y_{i'} 2^{i'}$. Let $y = \sum_{i=0}^{n-1} y_i 2^i$.

4. For $i \in [n]$, for $j \in [2^i] \setminus y^{(i)}$, let $\widehat{L}_{\mathsf{E}}^{(i)}[j] = H_1(L_{\mathsf{E}}^{(i)}[j])$, output $P[i] \xleftarrow{\$} \{0,1\}^\lambda$, and let $\widehat{L}_{\mathsf{E}}^{(i)}[y^{(i)}] = X_{\mathsf{E}}[i] \oplus P[i] \oplus \bigoplus_{j \in [2^i] \setminus y^{(i)} \oplus \{y^{(i)}\}} \widehat{L}_{\mathsf{E}}^{(i)}[j]$. Let $L_{\mathsf{E}}^{(i+1)} = (\widehat{L}_{\mathsf{E}}^{(i)} \oplus L_{\mathsf{E}}^{(i)}) \| \widehat{L}_{\mathsf{E}}^{(i)}$.
5. Let $w' \xleftarrow{\$} [N^\zeta]$. Output w'.

Claim 24. *The experiments* Hyb_1 *and* Hyb_2 *are computationally indistinguishable.*

Proof. Follows immediately from the definition of H_1, H_2. □

Note that Hyb_2 only requires knowledge of the public parameters n, N, so it's simulatable by a PPT simulator Sim. This concludes the proof of the theorem.

Remark 25. *The proof also works when* H_1, H_2 *are modeled as Circular Correlation Robust Hash functions (CCRH), under appropriate definition that allows replacing* $x_i\Delta \oplus H_1(L_{\mathsf{E}}^{(i)}[y^{(i)}] \oplus \Delta)$ *and* $w + H_2(L_{\mathsf{E}}^{(n)}[y] \oplus \Delta)$ *with random values.*

B Compatibility with Free XOR

In this section, we show that the lookup gate is compatible with the free XOR technique under the circular correlation robust hash (CCRH) assumption, in the plain model. The proof can also be extended to show compatibility with other techniques that use the CCRH assumption, e.g., arithmetic garbled circuits [Hea24].

B.1 Circular Correlation Robust Hash

We define a new notion of circular correlation robustness that is tailored for our purpose.

Definition 26 (Circular Correlation Robustness)**.** Let $H_1 : \{0,1\}^\lambda \to \{0,1\}^\lambda, H_2 : \{0,1\}^\lambda \to [2^{2\zeta\lambda_{\mathsf{DCR}}}], H_3 : \mathbb{Z} \to \{0,1\}^\lambda, H_4 : \mathbb{Z} \to \{0,1\}^\lambda$ be four functions. For any Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk})$, we define four oracles:

- $\mathcal{O}_1^\Delta(X, b)$: On input $X \in \{0,1\}^\lambda, b \in \{0,1\}$, output $H_1(X \oplus \Delta) \oplus b\Delta$.
- $\mathcal{O}_2^{\Delta,\mathsf{sk}}(X)$: On input $X \in \{0,1\}^\lambda$, output $H_2(X \oplus \Delta) + \mathsf{sk}$.
- $\mathcal{O}_3^{\Delta,\mathsf{sk}}(X, b)$: On input $X \in \mathbb{Z}_{N^\zeta}, b \in \{-1, 1\}$, output $H_3((X + b\mathsf{sk}) \bmod N^\zeta)$.
- $\mathcal{O}_4^{\Delta,\mathsf{sk}}(X, b)$: On input $X \in \mathbb{Z}_{N^\zeta}, b \in \{-1, 1\}$, output $H_4((X + b\mathsf{sk}) \bmod N^\zeta)$.

A sequence of oracle queries is legal if and only if $\mathcal{O}_1$ is never queried with the same X and different b. H_1, H_2, H_3, H_4 is circular correlation robust if the following two experiments are computationally indistinguishable:

- $\mathsf{RealCCRH}$: Sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ and a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Run $\mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\mathcal{O}_3,\mathcal{O}_4}(N)$, but only allowing legal queries.

- $\mathsf{IdealCCRH}$: Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Run $\mathcal{A}^{\mathcal{R}_1,\mathcal{R}_2,\mathcal{R}_3,\mathcal{R}_4}(N)$, but only allowing legal queries, where $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ are random oracles with the same domain and range as $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4$.

B.2 Lookup Gate with Free XOR

We show that the lookup gate is compatible with the free XOR technique. Concretely, we set Δ to be the free XOR key, which is also used in Π_{lookup}, and set $\Delta_O = \Delta$ such that the output of the lookup gate can directly be used afterwards. We use H_1 as the hash function for garbling the AND gates.

Theorem 27. *Let H_1, H_2, H_3, H_4 be circular correlation robust hash functions, as per Definition 26. Let Π_{lookup} be the lookup gate protocol with $\Delta_O = \Delta$. Then, a garbled circuit with Free XOR and lookup gates is secure.*

Proof. We only outline the proof, as most of the work is already done in the proof of Lemma 13 and Theorem 14.

Experiment Hyb_0. In Hyb_0, we sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}, (\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and sample the labels of all wire values in the circuit (including the input labels X_G and output labels Y_G of the lookup gates). We then prepare the garbled materials for each AND gate using H_1, and the garbled materials for each lookup gate using H_1, H_2, H_3, H_4.

Experiment Hyb_1. In Hyb_1, we view the labels as secret shares held by $\mathsf{Garbler}$. Instead of sampling the shares of $\mathsf{Garbler}$ from random, we equivalently sample the shares of $\mathsf{Evaluator}$ from random. We then prepare the garbled materials as before. This corresponds to Hyb_1 in the proof of Lemma 13 and Theorem 14.

Experiment Hyb_2. In Hyb_2, we split the experiment into two parts, where the first part samples $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}, (\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and the second part samples the shares and prepares the garbled materials, but only has oracle access to $\mathcal{O}_1^\Delta, \mathcal{O}_2^{\Delta,\mathsf{sk}}, \mathcal{O}_3^{\Delta,\mathsf{sk}}, \mathcal{O}_4^{\Delta,\mathsf{sk}}$, where the oracles are defined as in Definition 26. As seen from Hyb_1 in the proof of Lemma 13 and Hyb_2 in the proof of Theorem 14, this split is possible.

Experiment Hyb_3. In Hyb_3, we replace $\mathcal{O}_1^\Delta, \mathcal{O}_2^{\Delta,\mathsf{sk}}, \mathcal{O}_3^{\Delta,\mathsf{sk}}, \mathcal{O}_4^{\Delta,\mathsf{sk}}$ with $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$, i.e., random oracles. This is computationally indistinguishable from Hyb_2, by the circular correlation robustness of H_1, H_2, H_3, H_4. After this is done, we arrive at Hyb_2 in the proof of Lemma 13 and Hyb_3 in the proof of Theorem 14.

Experiment Hyb_4. In Hyb_4, we conclude by replacing the Damgård-Jurik ciphertexts generated in Π_{lookup} with encryptions of zero. This is computationally indistinguishable from Hyb_3 by the security of the Damgård-Jurik encryption scheme. Now we arrive at Hyb_4 in the proof of Theorem 14, and the experiment is independent of all private information held by Garbler. □