Garbled Lookup Tables from Homomorphic Secret Sharing

Liqiang Liu Tianren Liu Bo Peng

Peking University
{lql,trl}@pku.edu.cn, bo.peng@stu.pku.edu.cn

Abstract

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

Recently, Heath, Kolesnikov, and Ng (Eurocrypt 2024) partially transcended 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 primitives could be constructed only 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 $C(x)$, while learning nothing else about x.

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 optimizing 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	$\tilde{O}(nmc_{\text{hash}} + n^3c_{\text{hash}})$ amortized	CCR	○
[ILL25]	1	$O(Nm\,\text{poly}(\lambda))$	Various	
[MORS25]	1	$O(Nm\,\text{poly}(\lambda))$	DCR	

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. $\text{poly}(\lambda)$ in [ILL25] and [MORS25] hides the cost of evaluating an NC^1 PRF using HSS.

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 that is compatible with free-XOR. The communication cost per such gate is $O(n\lambda_{\mathsf{DCR}} + m\lambda)$. The computation cost is $O(2^n{\lambda_{\mathsf{DCR}}}^2 + 2^n m\lambda_{\mathsf{DCR}})$, including $O(2^n n)$ exponentiations* [1]*.*

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

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

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 privately 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 restores 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, combining 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]	$n\lambda_{\mathsf{RSA}}$	$n\cdot\text{poly}(\lambda_{\mathsf{RSA}})$	$n\cdot\text{poly}(\lambda_{\mathsf{RSA}})$	$N\cdot\text{poly}(\lambda_{\mathsf{RSA}})$	iO/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 $2^{(\zeta-1)\lambda_{\mathsf{DCR}}-\lambda}$, where $\zeta > 1$ is an arbitrary constant. 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 $x \oplus c$ or $(x+c) \bmod 2^n$ instead of x. This preserves privacy, as $x \oplus c$ (resp. $(x+c) \bmod 2^n$) is uniformly distributed over $[2^n]$ independent 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 Garbler and Evaluator hold the following variant of one-hot label of $y = x \oplus c$: [2] Garbler holds keys $I_{\mathsf{G}} = (I[0], \ldots, I[2^n - 1])$, and 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

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

Since subtractive secret share is linearly homomorphic, the task is reduced to multiplying the secret share I by 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 a function with exponential-like properties: $\exp(a + b) \equiv \exp(a)\exp(b) \pmod{N^3}$ and $\exp(ab) \equiv \exp(a)^b \pmod{N^3}$. This fits particularly well with subtractive secret share, where, say, Garbler holds r and Evaluator holds $r + \mathsf{sk}$. If Garbler sends $e \leftarrow \mathsf{Enc}(N, c_i)$ to Evaluator, they can compute $w_{\mathsf{G}} = e^r$ and $w_{\mathsf{E}} = e^{r+\mathsf{sk}}$ locally, and $w_{\mathsf{E}} \cdot w_{\mathsf{G}}^{-1} \equiv \exp(c_i \cdot \mathsf{sk}) \pmod{N^3}$. Garbler and 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, Garbler and 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 $n-1$, Garbler sends $E[i] \leftarrow \mathsf{Enc}(N, c_i)$ to Evaluator, then they locally compute $E[i]^{I_{\mathsf{G}}^{(i)}}$ and $E[i]^{I_{\mathsf{E}}^{(i)}}$ (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 $f : [2^n] \to \{0,1\}$, $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ 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 $\mathsf{Evaluator}$ don't know sk, this can be solved with $\mathsf{Garbler}$ sending extra $O(\lambda)$ bits to $\mathsf{Evaluator}$.

1.2 Related Works

Arithmetic Garbled Circuits. A line of research focuses on garbling arithmetic circuits (featuring multiplication, addition, and arithmetic/boolean conversion gates), primarily aiming to optimize the garbling rate.[3] Yao's scheme, with schoolbook multiplication, offers a baseline $O(1/(\lambda\ell))$ rate arithmetic garbling from One-Way Functions (OWF). [AIK11] improved this to $O(1/\lambda_{\mathsf{LWE}})$ using LWE. While [BMR16] generalized Free-XOR for free addition, their multiplication remained exponentially expensive. Subsequently, [BLLL23] achieved the first constant rate for bounded integer computation, relying on DCR (Damgård-Jurik). [LL24] further enhanced bit decomposition gates within the random oracle model. [Hea24] then presented the first rate $O(1/\lambda)$ construction relying only on minicrypt-style assumptions (CCR hash). Recently, [MORS24a] obtained rate-1 garbling for multiplication gates from DCR; similar to our work, they also leverage techniques from Damgård-Jurik based HSS [RS21].

2-Party Homomorphic Secret Sharing. Numerous works construct 2-party Homomorphic Secret Sharing (HSS) for Branching Programs (BP), often leveraging techniques related to Distributed Discrete Logarithm (DDLog) and relying on diverse cryptographic assumptions.

Group-based HSS schemes include those built from the DDH assumption [BGI16, BGI17, BCG⁺17, DKK18], the DCR assumption [FGJS17, OSY21, RS21], and class groups [ADOS22]. Lattice-based HSS saw [BKS19] offer a direct LWE (or Ring-LWE) construction that bypassed fully/somewhat homomorphic encryption, albeit using a superpolynomial modulus. [ACK23] subsequently enhanced this to a polynomial modulus.

These existing HSS schemes generally operate within the Restricted Multiplication Straight-line (RMS) program model. While this model captures BPs, it restricts multiplication to occur only between an input value and a memory value. In contrast, our scheme supports 2PC for more complex functions.

Finally, it is important to note that directly deriving a share of a one-hot vector of x from its boolean share via HSS is non-trivial. Since neither the garbler nor the evaluator individually knows x, they cannot acquire the necessary encryptions of x's bits to perform the required multiplications.

[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. There are several other Garbled Circuit (GC) approaches for evaluating functions via lookup tables, including Classical GC, one-hot garbling [HK21a, Hea24, HKN24], stacked garbling [Kol18, HK21b, HK20], Garbled RAM (GRAM) [LO13], and Garbled Private Information Retrieval (GPIR) [HHK+22]. A detailed comparison can be found in Table 1.

Succinct Garbling. A garbling scheme is *succinct* if the bit-length of its garbled circuit is (asymptotically) smaller than the description size of the original circuit. [GKP+13, BGG+14, KLW15, CHJV15, HLL23] achieved *full succinctness*, where the garbled circuit size is independent of the size of the original circuit, but their constructions rely on the heavy assumptions such as iO or a non-black-box combination of FHE and ABE. Recently, [ILL24] posed a fully succinct garbling scheme from group-based assumptions (variants of DDH and DCR) as well as lattice assumptions. However, it only supports a limited class of functions, including bounded length branching programs and truth tables.

Programmable Distributed Point Functions. This work's partial construction is essentially a Programmable Distributed Point Function (PDPF), or equivalently, a Privately Programmable Pseudorandom Function (PP-PRF) with polynomial-size domain. In a constrained PRF, the owner of key k generates constrained keys k_f from k and a predicate f. Anyone with k_f can evaluate the PRF on inputs x where $f(x) = 0$, while PRF evaluations on other inputs remain hidden. A Privately Constrained PRF (PC-PRF) further requires k_f to hide f. [BKM17] constructed PC-PRFs for point-function constraints (Privately Puncturable PRFs) from LWE and the 1-dimensional SIS problem. Subsequent works built PC-PRFs for more complex constraints with stronger privacy, mostly from LWE [CC17, BTVW17, CVW18, DKN+20]. Boneh, Lewi, and Wu [BLW17] first proposed Privately Programmable PRF (PP-PRF): a Privately Puncturable PRF where deriving k_f allows specifying outputs where $f(x) = 1$. Their initial construction used multilinear maps, with a later improvement by [PS18] relaxing the assumption to LWE.

[BGIK22] proposed a PP-PRF construction based solely on OWF. [4] However, their key generation and evaluation times are linear in domain size N, even for $1/\text{poly}$ security. Achieving negligible security error required error-correcting codes, leading to super-linear, impractical key generation and evaluation times.

In our work, we construct a PDPF with full security from the DCR assumption, with the additional property of *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. This is crucial for our application in lookup table garbling and may offer benefits in other contexts as well.

[4] [BGIK22] modeled their construction as a PDPF, where evaluations are typically performed on the entire domain. Both our work and the [BGIK22] construction cannot evaluate on a single point in sublinear time with respect to the domain size.

1.3 Concurrent Works

Succinct Garbling from HSS. Recent advancements in Boolean garbling schemes have achieved remarkable efficiency. Building on the algebraic homomorphic MAC (aHMAC) from [ILL24] and the HSS technique, [ILL25] introduced a scheme achieving 1-bit per gate. This construction is versatile, supporting assumptions like Power-DDH (in Paillier or prime-order groups) or Power-RLWE, and can generalize to garble $O(\log \lambda)$-fan-in gates with the same 1-bit efficiency. Independently, [MORS25] developed a similar framework with comparable efficiency, focusing on standard and circular DCR assumptions.

However, a key limitation in both [ILL25] and [MORS25] arises when garbling an n-fan-in gate. Their approaches require the garbler and evaluator to compute an array of length 2^n. At each entry, they perform a white-box evaluation of an NC^1 PRF using HSS. When instantiated with DCR, this results in a computational cost of $O(2^n {\lambda_{\mathsf{DCR}}}^2 \operatorname{poly}(\lambda))$, where $\operatorname{poly}(\lambda)$ depends on the specific PRF. In contrast, for an n-input, m-output gate, our method requires only $O(2^n {\lambda_{\mathsf{DCR}}}^2 + 2^n m \lambda_{\mathsf{DCR}})$ computation, making it significantly more efficient.

Privately Puncturable PRFs from DCR. Following [CMPR23]'s template, [BMO$^+$25] constructed a privately puncturable PRF from the DCR assumption, supporting an exponentially large domain and also featuring a decomposable punctured key. However, *puncturable* is a weaker property than *programmable*, as the punctured key does not allow specifying the output at the punctured point. In addition, their work also requires white-box evaluation of an NC^1 PRF using HSS, which limits their practical efficiency.

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, 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 $\mathcal{D}$ 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 denote 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).
- $\zeta \geq 1$ 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 $\zeta \geq 1$ 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

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

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].

Definition 7 (Correctness)**.** A garbling scheme is *correct* if for any circuit C and all input x,

$$\Pr\left[(\hat{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C) : \mathsf{Decode}(d, \mathsf{Evaluate}(C, \hat{C}, \mathsf{Encode}(e, x))) = C(x)\right] \geq 1 - \mathrm{negl}(\lambda).$$

Definition 8 (Obliviousness)**.** A garbling scheme is *oblivious* if there exists a simulator $\mathsf{Sim}_{\mathsf{obv}}$ such that for any circuit C and all inputs x, the pair $(\hat{C}, X_{\mathsf{E}})$ is computationally indistinguishable from $\mathsf{Sim}_{\mathsf{obv}}(1^\lambda, C)$, where $(\hat{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C)$ and $X_{\mathsf{E}} \leftarrow \mathsf{Encode}(e, x)$.

Definition 9 (Privacy)**.** A garbling scheme is *private* if there exists a simulator $\mathsf{Sim}_{\mathsf{priv}}$ such that for any circuit C and all inputs x, the tuple $(\hat{C}, X_{\mathsf{E}}, d)$ is computationally indistinguishable from $\mathsf{Sim}_{\mathsf{priv}}(1^\lambda, C, y)$, where $(\hat{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C)$, $X_{\mathsf{E}} \leftarrow \mathsf{Encode}(e, x)$ and $y \leftarrow C(x)$.

Definition 10 (Authenticity)**.** A garbling scheme is *authentic* if for all circuits C, all inputs x, and all PPT adversaries $\mathcal{A}$, the following probability is negligible:

$$\Pr\left[\mathsf{Evaluate}(C, \hat{C}, X_{\mathsf{E}}) \neq Y'_{\mathsf{E}} \wedge \mathsf{Decode}(d, Y'_{\mathsf{E}}) \neq \perp\right]$$

where $(\hat{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C)$, $X_{\mathsf{E}} \leftarrow \mathsf{Encode}(e, x)$ and $Y'_{\mathsf{E}} \leftarrow \mathcal{A}(C, \hat{C}, X_{\mathsf{E}})$.

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 11. *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. Our approach utilizes the distributed discrete logarithm technique presented in [RS21]. The construction is presented in Figure 2.

Claim 12. *Let z_{G} and z_{E} be the outputs of* Garbler *and* Evaluator *in Π_{DDLog}, Figure 2. Then $(z_{\mathsf{G}}, z_{\mathsf{E}}) \in [\![x]\!]_{N^\zeta}^{\mathsf{sub}}$. Further, Π_{DDLog} takes no communication and $O(\log N (\log\log N)^2)$ 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 $z_{\mathsf{E}} - z_{\mathsf{G}} \equiv x \pmod{N^\zeta}$.

The computation bottleneck is the $O(1)$ modular divisions, where each division takes $O(\log N (\log\log N)^2)$ time using Fast Fourier Transform (FFT) [CT65], Half-GCD [Sch71], and Barrett reduction [Bar87].

We further remark that, when Π_{DDLog} is applied to vectors of length $\Omega((\log\log N)^2)$, the computation time can be reduced to amortized $O(\log N)$ per element. This is because computing multiple modular inverses is much more efficient than computing them one by one. □

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 almost the same as 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.

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 element-wise to vectors.*

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

Claim 13. *Let* (I'_{G}, c) *and* (I'_{E}, y) *be the outputs of* Garbler *and* Evaluator *in* $\Pi^{\mathit{shift}}_{\mathit{one\text{-}hot}}$, *Figure 9. Then* $y = x \oplus c$, *and* $(I'_{\mathsf{G}}, I'_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(y, w)]\!]^{\mathit{sub}}_{N^\zeta}$. *Further,* $\Pi^{\mathit{shift}}_{\mathit{one\text{-}hot}}$ *takes* $O(n\lambda + \log N)$ *communication and* $O(2^n \log N)$ *computation.*

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

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

so the problem is reduced to multiplying I' by c_i, without leaking c_i.

To this end, Garbler encrypts c_i using the Damgård-Jurik encryption scheme, and sends the ciphertext $E[i]$ to Evaluator. By Lemma 11, the ciphertext can be used to multiply I' by 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 14. *Let* I_{G} *and* I_{E} *be the outputs of* Garbler *and* Evaluator *in* $\Pi^{\mathit{real}}_{\mathit{one\text{-}hot}}$, *Figure 3. Assume* $\zeta \geq 2$. *Then* $(I_{\mathsf{G}}, I_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(x, \mathit{sk})]\!]^{\mathit{sub}}_{N^\zeta}$, *except with negligible probability. Further,* $\Pi^{\mathit{real}}_{\mathit{one\text{-}hot}}$ *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_i to $I^{(i)}_{\mathsf{G}}$ and $I^{(i)}_{\mathsf{E}}$, the entries containing $[\![\mathsf{sk}]\!]^{\mathsf{sub}}_{N^\zeta}$ are rerandomized, while other entries remain equal. Since $\mathsf{sk} \ll N^\zeta$, we have $\left((I^{(i)}_{\mathsf{E}} + r_i) \bmod N^\zeta\right) - \left((I^{(i)}_{\mathsf{G}} + r_i) \bmod N^\zeta\right) = \mathcal{I}^{(n)}(y^{(i)}, \mathsf{sk})$ with overwhelming probability, even when they are viewed as vectors in $\mathbb{Z}^{2^n}$ [5]. Thus, by Lemma 11, $(\widetilde{I}^{(i)}_{\mathsf{G}}, \widetilde{I}^{(i)}_{\mathsf{E}}) \in$

[5] We assumed $\zeta \geq 2$ here. However, when $\zeta = 1$, we can use the secret share of $N - \mathsf{sk}$ instead of sk, which has almost the same effect but satisfies $N - \mathsf{sk} \ll N$. The protocol can be modified accordingly to maintain correctness and achieve better concrete efficiency.

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

Input.

- Public parameter: A positive integer n and a Damgård-Jurik public key $\mathsf{pk} = N$.
- From $\mathsf{Garbler}$: $X_\mathsf{G} = (X_\mathsf{G}[0], \ldots, X_\mathsf{G}[n-1]) \in (\{0,1\}^\lambda)^n$, $\Delta \in 1\{0,1\}^{\lambda-1}$, and the Damgård-Jurik secret key sk corresponding to N.
- 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 $k_r \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$ and sends k_r to $\mathsf{Evaluator}$. $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ expands k_r to $r_0, \ldots, r_{n-1} \in [N^\zeta]$ using a PRG.
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 $E[i] \leftarrow \mathsf{Enc}(N, c_i)$, and sends $E[i]$ to $\mathsf{Evaluator}$.
   (c) $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ compute $\widetilde{I}^{(i)}_\mathsf{G} = E[i]^{(I^{(i)}_\mathsf{G} + r_i) \bmod N^\zeta} \bmod N^{\zeta+1}$ and $\widetilde{I}^{(i)}_\mathsf{E} = E[i]^{(I^{(i)}_\mathsf{E} + r_i) \bmod N^\zeta} \bmod N^{\zeta+1}$, where r_i is added entry-wise.
   (d) $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ call $(\hat{I}^{(i)}_\mathsf{G}, \hat{I}^{(i)}_\mathsf{E}) \leftarrow \Pi_{\mathsf{DDLog}}(N; \widetilde{I}^{(i)}_\mathsf{G}; \widetilde{I}^{(i)}_\mathsf{E})$.
   (e) $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ compute $I^{(i+1)}_\mathsf{G} = (I^{(i)}_\mathsf{G} - \hat{I}^{(i)}_\mathsf{G} + \mathsf{shift}(\hat{I}^{(i)}_\mathsf{G}, 2^i)) \bmod N^\zeta$ and $I^{(i+1)}_\mathsf{E} = (I^{(i)}_\mathsf{E} - \hat{I}^{(i)}_\mathsf{E} + \mathsf{shift}(\hat{I}^{(i)}_\mathsf{E}, 2^i)) \bmod N^\zeta$, where $\mathsf{shift}(I, 2^i)[j] = I[j \oplus 2^i]$.
4. $\mathsf{Garbler}$ outputs $I^{(n)}_\mathsf{G}$, and $\mathsf{Evaluator}$ outputs $I^{(n)}_\mathsf{E}$.

Figure 3: Real One-Hot Gate

$[\![\mathcal{I}^{(n)}(y^{(i)}, c_i \cdot \mathsf{sk})]\!]_N^{\mathsf{div}}$, and by correctness of Π_{DDLog}, $(\hat{I}_{\mathsf{G}}^{(i)}, \hat{I}_{\mathsf{E}}^{(i)}) \in [\![\mathcal{I}^{(n)}(y^{(i)}, c_i \cdot \mathsf{sk})]\!]_{N^\zeta}^{\mathsf{sub}}$. Finally, $I_{\mathsf{G}}^{(i+1)}$ and $I_{\mathsf{E}}^{(i+1)}$ are obtained by a linear combination of $I_{\mathsf{G}}^{(i)}, I_{\mathsf{E}}^{(i)}, \hat{I}_{\mathsf{G}}^{(i)}, \hat{I}_{\mathsf{E}}^{(i)}$.

Calling $\Pi_{\text{one-hot}}^{\text{shift}}$ takes $O(n\lambda + \log N)$ communication and $O(2^n \log N)$ computation. Sending $E[i]$ takes $O(n \log N)$ communication in total, and computing $\widetilde{I}_{\mathsf{G}}^{(i)}, \widetilde{I}_{\mathsf{E}}^{(i)}$ takes $O(2^n \log^2 N)$ computation in total [6]. Calling Π_{DDLog} takes $O(2^n n \log N) = o(2^n \log^2 N)$ 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 15. *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 $O(n\lambda_{\mathsf{DCR}} + m\lambda)$ communication and $O(2^n {\lambda_{\mathsf{DCR}}}^2 + 2^n m \lambda_{\mathsf{DCR}})$ computation.*

Proof. By correctness of $\Pi_{\text{one-hot}}^{\text{real}}$, $(I_{\mathsf{G}}, I_{\mathsf{E}}) \in [\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]_{N^\zeta}^{\mathsf{sub}}$. It follows that $W_{\mathsf{E}}[i] = W_{\mathsf{G}}[i] + f_i(x)\mathsf{sk} \pmod{N^\zeta}$ and $H_3(i, W_{\mathsf{E}}[i]) = H_3(i, (W_{\mathsf{G}}[i] + f_i(x)\mathsf{sk}) \bmod N^\zeta)$. 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 $\Pi_{\text{one-hot}}^{\text{real}}$ 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 $O(m\lambda)$ communication, and computing $W_{\mathsf{G}}[i], W_{\mathsf{E}}[i]$ for all $i \in [m]$ takes $O(2^n m \lambda_{\mathsf{DCR}})$ computation. Therefore, the total communication is $O(n\lambda_{\mathsf{DCR}} + m\lambda)$, and the total computation is $O(2^n {\lambda_{\mathsf{DCR}}}^2 + 2^n m \lambda_{\mathsf{DCR}})$. □

4.5 Garbling Scheme with Lookup Gate

We now formalize our garbling scheme (Definition 6). The garbling scheme assembles XOR gates, AND gates and lookup gates in the same way as [HKN24].

Construction 16. We consider circuits with three gate types:

- Standard two-input, one-output XOR gates and AND gates.
- Lookup gates. A lookup gate is parameterized over functions $f_0, \ldots, f_{m-1} : \{0,1\}^n \to \{0,1\}$. It takes an n-bit input x and outputs m bits $f_0(x), \ldots, f_{m-1}(x)$.

[6] Naively, computing each $\widetilde{I}_{\mathsf{G}}^{(i)}$ and $\widetilde{I}_{\mathsf{E}}^{(i)}$ requires $O(2^n \log^2 N)$ time, leading to a total computation time of $O(n 2^n \log^2 N)$. 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: Two positive integers n, m, a Damgård-Jurik public key $\mathsf{pk} = N$, m functions $f_0, \ldots, f_{m-1} : \{0,1\}^n \to \{0,1\}$, and two random oracles $H_3, H_4 : \mathbb{Z} \times \mathbb{Z} \to \{0,1\}^\lambda$.
- From Garbler: $X_\mathsf{G} = (X_\mathsf{G}[0], \ldots, X_\mathsf{G}[n-1]) \in (\{0,1\}^\lambda)^n$, $\Delta_O \in \{0,1\}^\lambda$, $\Delta \in 1\{0,1\}^{\lambda-1}$, and the Damgård-Jurik secret key sk corresponding to N.
- 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 $\Pi^{\mathsf{real}}_{\text{one-hot}}(n, N; X_\mathsf{G}, \Delta, \mathsf{sk}; X_\mathsf{E})$, and obtain I_G and I_E respectively. // $(I_\mathsf{G}, I_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$
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, for $t \in \{0,1\}$,
   $$\mathrm{ct}_t = \Big(H_3\big(i, (W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta\big), H_4\big(i, (W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta\big) \oplus Y_\mathsf{G}[i] \oplus t\Delta_O\Big),$$
   and 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 $u_t = H_3(i, W_\mathsf{E}[i])$, and let $Y_\mathsf{E}[i] = v_t \oplus H_4(i, W_\mathsf{E}[i])$.
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

The garbling procedures are defined as follows:

- $\mathsf{Garble}(1^\lambda, C)$ proceeds in several steps:
  - Uniformly sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$ and $(N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Set $\Delta_O = \Delta$.
  - For each input wire $x[i]$, sample $\mathsf{Garbler}$'s share (same as the 'label' of 0 in standard GC literature) $X_{\mathsf{G}}[i] \xleftarrow{\$} \{0,1\}^\lambda$.
  - The input encoding string e is defined as a vector where the i-th element is $(X_{\mathsf{G}}[i], X_{\mathsf{G}}[i] \oplus \Delta)$.
  - Go through the circuit gate by gate. For each gate, a distinct and pseudorandom nonce is employed for invocations of random oracles. For each XOR gate, $\mathsf{Garbler}$'s share of the output wire is defined as the XOR of $\mathsf{Garbler}$'s share of the two input wires. For each AND gate, run the AND gate garbling procedure formalized by [ZRE15]. For each lookup gate, run Π_{lookup} as $\mathsf{Garbler}$, and treat the output as $\mathsf{Garbler}$'s share of the output wires.
  - Let the garbled material $\hat{C}$ be the concatenation of the materials for XOR and AND gates, and messages sent by $\mathsf{Garbler}$ in the lookup gates.
  - For each output wire $y[i]$, let $Y_{\mathsf{G}}[i]$ be $\mathsf{Garbler}$'s share of the output wire. The output decoding string d is defined as a vector where the i-th element is $(H_1(v, Y_{\mathsf{G}}[i]) \| Y_{\mathsf{G}}[i][0], H_1(v, Y_{\mathsf{G}}[i] \oplus \Delta) \| (Y_{\mathsf{G}}[i][0] \oplus 1))$, where $H_1 : \mathbb{Z} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a random oracle (also used in $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$), and $v \xleftarrow{\$} [2^\lambda]$ is a random nonce. v is appended to d as the last element.
  - Output $(\hat{C}, e, d)$.
- $\mathsf{Encode}(e, x)$: For each input bit $x[i]$, output $X_{\mathsf{E}}[i] = e[i][x[i]]$.
- $\mathsf{Evaluate}(C, \hat{C}, X_{\mathsf{E}})$: Step through the circuit gate by gate, and use the garbling material $\hat{C}$ to map $\mathsf{Evaluator}$'s share of the input wires to $\mathsf{Evaluator}$'s share of the output wires. More specifically, the procedure is as follows: For each XOR gate, the output share is the XOR of the two input shares. For each AND gate, run the AND gate evaluation procedure formalized by [ZRE15]. For each lookup gate, run Π_{lookup} as $\mathsf{Evaluator}$, and treat the output as $\mathsf{Evaluator}$'s output share. Finally, collect the shares Y_{E} of the output wires and output them as the encoded output.
- $\mathsf{Decode}(d, Y_{\mathsf{E}})$: For each encoded output bit $Y_{\mathsf{E}}[i]$, compute:

$$y[i] = \begin{cases} 0, & \text{if } d[i][0] = H_1(v, Y_{\mathsf{E}}[i]) \| Y_{\mathsf{E}}[i][0]; \\ 1, & \text{if } d[i][1] = H_1(v, Y_{\mathsf{E}}[i]) \| Y_{\mathsf{E}}[i][0]; \\ \bot, & \text{otherwise.} \end{cases}$$

  If any $y[i] = \bot$, output $\bot$. Otherwise, output y as the decoded output.

Theorem 17. *The garbling scheme defined in Construction 16 is correct (Definition 7).*

Proof. By the correctness of individual gates. □

We defer the proofs of obliviousness, privacy, and authenticity to Appendix B.

5 Privacy

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

Lemma 18. *Under the random oracle model, 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.*

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

We defer the proof of Lemma 18 to Appendix A. Next, we define and prove the privacy of the lookup gate.

Lemma 19. *Under the random oracle model and the DCR assumption, 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.*

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

Note that Lemma 19 only proves the privacy of a single lookup gate. We defer the proof of the privacy of the full garbling scheme to Appendix B.

5.1 Proof of Lemma 19

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

Experiment Hyb_0**.**

1. Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output N.
2. 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.
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 $c = \sum_{i=0}^{n-1} c_i \cdot 2^i$ be its binary representation. For $i \in [n]$, let $E[i] \leftarrow \mathsf{Enc}(N, c_i)$. Output E.
5. Sample and output $k_r \xleftarrow{\$} \{0,1\}^\lambda$. Compute $(I_\mathsf{G}, I_\mathsf{E}) \in [\![\mathcal{I}^{(n)}(x, \mathsf{sk})]\!]^{\mathsf{sub}}_{N^\zeta}$ with $I'_\mathsf{G}, c, I'_\mathsf{E}, y, k_r, E$ [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 $\mathrm{ct}_t = \big(H_3(i, (W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta), H_4(i, (W_\mathsf{G}[i] + t\mathsf{sk}) \bmod N^\zeta) \oplus Y_\mathsf{G}[i] \oplus t\Delta_O\big)$ 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 statistically indistinguishable from Hyb_0. The changes are marked in blue.

Experiment Hyb_1**.**

1. Sample a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Output N.
2. 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.
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 $E[i] \leftarrow \mathsf{Enc}(N, x_i \oplus y_i)$. Output E.
5. Sample and output $k_r \xleftarrow{\$} \{0,1\}^\lambda$. Compute $I_\mathsf{E} \in \mathbb{Z}_{N^\zeta}^{2^n}$ with I'_E, y, k_r, E.
6. For $i \in [m]$, sample $Y_\mathsf{E}[i] \xleftarrow{\$} \{0,1\}^\lambda$, let $W_\mathsf{E}[i] = \sum_{j=0}^{2^n-1} f_i(j) I_\mathsf{E}[j] \pmod{N^\zeta}$,

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

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

[7] Concretely, I_E is computed by acting as Evaluator in Step 3 of Figure 3, and I_G is computed by acting as Garbler in Step 3 of Figure 3.

Claim 20. *The experiments* Hyb_0 *and* Hyb_1 *are statistically indistinguishable.*

Proof. In Step 4, we replace c_i with $x_i \oplus y_i$, using the fact that $c_i = x_i \oplus y_i$ with overwhelming probability, as guaranteed by the correctness of $\Pi^{\mathsf{shift}}_{\mathsf{one\text{-}hot}}$ in Claim 13.

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 $Y_{\mathsf{G}}[i] = Y_{\mathsf{E}}[i] \oplus f_i(x)\Delta_O$ and $W_{\mathsf{G}}[i] = W_{\mathsf{E}}[i] - f_i(x)\mathsf{sk}$, which holds with overwhelming probability as guaranteed by the correctness of $\Pi^{\mathsf{real}}_{\mathsf{one\text{-}hot}}$ in Claim 14. The order of $\mathrm{ct}_0, \mathrm{ct}_1$ may be changed, but they will be randomly permuted anyway. $\square$

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 18.

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 $E[i] \leftarrow \mathsf{Enc}(N, x_i \oplus y_i)$. Output E.
4. Sample and output $k_r \xleftarrow{\$} \{0,1\}^\lambda$. Compute $I_{\mathsf{E}} \in \mathbb{Z}^{2n}_{N^\zeta}$ with $I'_{\mathsf{E}}, y, k_r, E$.
5. For $i \in [m]$, sample $Y_{\mathsf{E}}[i] \xleftarrow{\$} \{0,1\}^\lambda$, let $W_{\mathsf{E}}[i] = \sum_{j=0}^{2^n-1} f_i(j) I_{\mathsf{E}}[j] \pmod{N^\zeta}$,
$$\mathrm{ct}_t = \Big(H_3(i, (W_{\mathsf{E}}[i] + (-1)^{f_i(x)} t\mathsf{sk}) \bmod N^\zeta), \\ H_4(i, (W_{\mathsf{E}}[i] + (-1)^{f_i(x)} t\mathsf{sk}) \bmod N^\zeta) \oplus Y_{\mathsf{E}}[i] \oplus t\Delta_O\Big),$$
randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Claim 21. *The experiments* Hyb_1 *and* Hyb_2 *are computationally indistinguishable.*

Proof. Follows from Lemma 18. $\square$

Random Oracle. Next, since $\mathsf{Evaluator}$ cannot compute the secret key sk, it's safe to use sk as an 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 $E[i] \leftarrow \mathsf{Enc}(N, x_i \oplus y_i)$. Output E.

4. Sample and output $k_r \xleftarrow{\$} \{0,1\}^\lambda$. Compute $I_{\mathsf{E}} \in \mathbb{Z}_{N^\zeta}^{2^n}$ with $I'_{\mathsf{E}}, y, k_r, E$.
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 $\mathrm{ct}_0 = (H_3(i, W_{\mathsf{E}}[i]), H_4(i, W_{\mathsf{E}}[i]) \oplus Y_{\mathsf{E}}[i])$, and let $\mathrm{ct}_1 \xleftarrow{\$} \{0,1\}^{2\lambda}$. Randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Claim 22. *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 the following event must happen with non-negligible probability, when $\mathcal{A}$ is run on the output of Hyb_3: $\mathcal{A}$ queries H_3 or H_4 on $(i, W_{\mathsf{E}}[i] \pm \mathsf{sk})$ 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, and gives the output to $\mathcal{A}$. Now, whenever $\mathcal{A}$ queries the random oracles H_3 or H_4 at position (i, p), $\mathcal{A}'$ checks if $p = W_{\mathsf{E}}[i] \pm \mathsf{sk} \pmod{N^\zeta}$ (Note that $\mathcal{A}'$ can efficiently check guesses for sk with only knowledge of N). 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 $E[i] \leftarrow \mathsf{Enc}(N, 0)$. Output E.
4. Sample and output $k_r \xleftarrow{\$} \{0,1\}^\lambda$. Compute $I_{\mathsf{E}} \in \mathbb{Z}_{N^\zeta}^{2^n}$ with $I'_{\mathsf{E}}, y, k_r, E$.
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 $\mathrm{ct}_0 = (H_3(i, W_{\mathsf{E}}[i]), H_4(i, W_{\mathsf{E}}[i]) \oplus Y_{\mathsf{E}}[i])$, and let $\mathrm{ct}_1 \xleftarrow{\$} \{0,1\}^{2\lambda}$. Randomly permute $\mathrm{ct}_0, \mathrm{ct}_1$ and output them.

Claim 23. *The experiments* Hyb_3 *and* Hyb_4 *are computationally indistinguishable.*

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

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

Remark 24. *The proof also works if H_3, H_4 are modeled as Circular Correlation Robust Hash functions (CCRH), under an appropriate definition that allows replacing $H_3(i, W_{\mathsf{E}}[i] \pm \mathsf{sk})$ and $H_4(i, W_{\mathsf{E}}[i] \pm \mathsf{sk}) \oplus \Delta_O$ with random values. Combining this proof with another version of Lemma 18, we can prove the privacy of lookup gates 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.

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.

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

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

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:

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

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. Gen_0 generates the garbled materials without the programmed point; Gen_1, knowing the programmed point, sends part of the garbled materials to $\mathsf{Evaluator}$. Eval_0 and Eval_1 act as $\mathsf{Garbler}$ and $\mathsf{Evaluator}$ respectively, using the garbled materials to compute the one-hot share $I_{\mathsf{G}}, I_{\mathsf{E}}$. 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.

We follow the idea of [RS21] to remove sk from the result. Let (N', sk') be another Damgård-Jurik key pair, and $\zeta' \geq 1$ be a constant. Let $d' = \mathsf{sk}' \cdot (\mathsf{sk}'^{-1} \bmod N'^{\zeta'})$, which satisfies $\mathsf{Enc}(N', c)^{d'} = \exp(c)$ for any $c \in [N'^{\zeta'}]$ – note how d' disappears after the exponentiation. The naive solution, which we employ in this paper, is to share $v \cdot \mathsf{sk} \cdot d'$ under N^ζ, and then use $\mathsf{Enc}(N', \mathsf{sk}^{-1} \bmod N'^{\zeta'})$ to remove sk and d' in one go. ζ is set to be $\zeta' + 2 + \lceil (\lambda + \log v)/\lambda_{\mathsf{DCR}} \rceil$ such that $v \cdot \mathsf{sk} \cdot d' \ll N^\zeta$.

[RS21] introduced a technique to eliminate the need for ζ to depend on ζ', at the cost of doubling the number of ciphertexts. However, since $\zeta' = \lceil (\lambda + \log v)/\lambda_{\mathsf{DCR}} \rceil = 1$ in most applications (we usually don't need v to be more than 4000 bits long), we will not use this technique in this paper.

Formal Construction. For the 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 $\mathrm{ddlog}_N(x) := \log_N(x \cdot (x^{-1} \bmod N) \bmod N^{\zeta+1})$. For the Damgård-Jurik public key N', ζ is replaced with ζ' in $\exp_{N'}$, $\log_{N'}$, $\mathrm{ddlog}_{N'}$, and in the Damgård-Jurik encryption scheme $\mathsf{Enc}(N', \cdot)$. Further, log and ddlog 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 25. *Let $(N, \mathsf{sk}), (N', \mathsf{sk}')$ be two Damgård-Jurik key pairs where $N, N' \in [2^{\lambda_{\mathsf{DCR}}-1}, 2^{\lambda_{\mathsf{DCR}}}]$, and let $d' = \mathsf{sk}' \cdot (\mathsf{sk}'^{-1} \bmod N'^{\zeta'})$. Let $0 \leq v \leq 2^{\min(\zeta', \zeta-\zeta'-2)\lambda_{\mathsf{DCR}}-\lambda}$ be an integer. Let $I'_{\mathsf{G}}, I'_{\mathsf{E}} \in \mathbb{Z}^{2^n}$, such that $I'_{\mathsf{E}} - I'_{\mathsf{G}} = \mathcal{I}^{(n)}(y, v \cdot \mathsf{sk} \cdot d')$. Let $c \in [2^n]$ be an integer with binary representation $c = \sum_{i=0}^{n-1} c_i 2^i$. Let $E[i]$ be an encryption of c_i under N^ζ and F be an encryption of $\mathsf{sk}^{-1} \bmod N'^{\zeta'}$ under $N'^{\zeta'}$. Let $k_r \in \{0,1\}^\lambda$ be a random bit string. Then $\mathsf{OblivShift}(N, E, F, k_r, I'_{\mathsf{E}}) - \mathsf{OblivShift}(N, E, F, k_r, I'_{\mathsf{G}}) = \mathcal{I}^{(n)}(y \oplus c, v)$ except with negligible probability in λ.*

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

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

- $I^{(i)}_{\mathsf{E}} - I^{(i)}_{\mathsf{G}} = \mathcal{I}^{(n)}(y^{(i)}, v \cdot \mathsf{sk} \cdot d')$.

Procedure $\mathsf{OblivShift}$

Input.

- Two Damgård-Jurik public keys N, N'.
- n Damgård-Jurik ciphertexts $E[0], \dots, E[n-1]$ encrypted under N^{ζ}.
- Another Damgård-Jurik ciphertext F encrypted under $N'^{\zeta'}$.
- A random bit string $k_r \in \{0,1\}^{\lambda}$.
- A vector $I' \in \mathbb{Z}^{2^n}$.

Procedure.

1. Expand k_r to $r_0, \dots, r_n \in [2^{\zeta \lambda_{\mathsf{DCR}}}]$ using a PRG.
2. Let $I^{(0)} = I'$.
3. For i from 0 to $n-1$:
   (a) Let $\widetilde{I}^{(i)} = E[i]^{I^{(i)}} \bmod N^{\zeta+1}$, and $\hat{I}^{(i)} = \mathrm{ddlog}_N(\widetilde{I}^{(i)})$.
   (b) Let $I^{(i+1)} = (I^{(i)} - \hat{I}^{(i)} + \mathsf{shift}(\hat{I}^{(i)}, 2^i) + r_i) \bmod N^{\zeta}$, where r_i is added entry-wise, and $\mathsf{shift}(I, 2^i)[j] = I[j \oplus 2^i]$. $I^{(i+1)}$ is now viewed as integers.
4. Let $\widetilde{I}^{(n)} = F^{I^{(n)}} \bmod N'^{\zeta'}$, and $I^{(n+1)} = (\mathrm{ddlog}_{N'}(\widetilde{I}^{(n)}) + r_n) \bmod N'^{\zeta'}$. Output $I^{(n+1)}$ as integers.

Figure 6: Oblivious Shift

- $(\widetilde{I}_{\mathsf{G}}^{(i)}, \widetilde{I}_{\mathsf{E}}^{(i)}) \in [\![\mathcal{I}^{(n)}(y^{(i+1)}, c_i \cdot v \cdot \mathsf{sk} \cdot d')]\!]_N^{\mathsf{div}}$. This is because $\mathsf{Enc}(N, c_i)^{v \cdot \mathsf{sk} \cdot d'} \equiv \exp_N(c_i \cdot v \cdot \mathsf{sk} \cdot d') \pmod{N^{\zeta+1}}$.
- $(\hat{I}_{\mathsf{G}}^{(i)}, \hat{I}_{\mathsf{E}}^{(i)}) \in [\![\mathcal{I}^{(n)}(y^{(i+1)}, c_i \cdot v \cdot \mathsf{sk} \cdot d')]\!]_{N^\zeta}^{\mathsf{sub}}$.
- $I_{\mathsf{G}}^{(i+1)} - I_{\mathsf{E}}^{(i+1)} = \mathcal{I}^{(n)}(y^{(i+1)}, v \cdot \mathsf{sk} \cdot d')$. Note the implicit conversion from $\mathbb{Z}_{N^\zeta}^{2^n}$ to $\mathbb{Z}^{2^n}$, which incurs $|v \cdot \mathsf{sk} \cdot d'|_\infty / N^\zeta = \mathrm{negl}(\lambda)$ failure probability.

Now we have $I_{\mathsf{G}}^{(n)} - I_{\mathsf{E}}^{(n)} = \mathcal{I}^{(n)}(y \oplus c, v \cdot \mathsf{sk} \cdot d')$ with overwhelming probability. Since $\mathsf{Enc}(N', \mathsf{sk}^{-1} \bmod N'^{\zeta'})^{v \cdot \mathsf{sk} \cdot d'} \equiv \exp_{N'}(v) \pmod{N'^{\zeta'+1}}$, we have $I_{\mathsf{G}}^{(n+1)} - I_{\mathsf{E}}^{(n+1)} = \mathcal{I}^{(n)}(y \oplus c, v)$ with overwhelming probability. □

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 26. *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 $2^{\min(\zeta', \zeta - \zeta' - 2)\lambda_{\mathsf{DCR}} - \lambda}$. Gen_0 runs in time $O(n{\lambda_{\mathsf{DCR}}}^2)$, 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 $O(2^n {\lambda_{\mathsf{DCR}}}^2)$.*

Correctness. It's clear from definition that $I'_{\mathsf{E}} - I'_{\mathsf{G}} = \mathcal{I}^{(n)}(y, v \cdot \mathsf{sk} \cdot d')$, 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 $[2^{\zeta\lambda_{\mathsf{DCR}}}]$, which is much larger than $v \cdot \mathsf{sk} \cdot d'$, so w is also computationally indistinguishable from a random integer in range $[2^{\zeta\lambda_{\mathsf{DCR}}}]$. Now the dependency on d' is removed, so F can be replaced by an encryption of zero. Finally, the dependency on sk is removed, so E can be replaced by encryption of zeros.

Efficiency. Gen_0 runs in time $O(n{\lambda_{\mathsf{DCR}}}^2)$ for generating $O(n)$ ciphertexts. Gen_1 only needs to expand the GGM tree through a single path, and do several calculation in $[2^{\zeta\lambda_{\mathsf{DCR}}}]$, 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], our construction has the same efficiency for evaluating at a single point and for evaluating at all points.

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

Programmable Distributed Point Function

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

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

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 two Damgård-Jurik key pairs $(\mathsf{pk} = N, \mathsf{sk}), (\mathsf{pk}' = N', \mathsf{sk}') \leftarrow \mathsf{Gen}(1^\lambda)$ with $N \geq N'$.
4. For $i \in [n]$, let $E[i] \leftarrow \mathsf{Enc}(N, c_i)$. Let $F \leftarrow \mathsf{Enc}(N', \mathsf{sk}^{-1} \bmod N'^{\zeta'})$.
5. Sample $k_r \xleftarrow{\$} \{0,1\}^\lambda$.
6. Output $k_0 = (L^{(0)}, c, N, E, F, k_r)$.

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

1. 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 $j \in [2^i]$. This is only a definition without any computation.
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 $P[i] = L^{(i+1)} \left[(1 - y_i)2^i + \sum_{j=0}^{i-1} y_j 2^j\right]$. The entry of $L^{(i+1)}$ can be computed efficiently without expanding the entire tree.
4. Let $w = v \cdot \mathsf{sk} \cdot d' + G_2(L^{(n)}[y])$, where $d' = \mathsf{sk}' \cdot (\mathsf{sk}'^{-1} \bmod N'^{\zeta'})$.
5. Output $k_1 = (N, E, F, k_r, y, P, w)$.

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

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 $\mathsf{OblivShift}(N, E, F, k_r, I'_{\mathsf{G}})[x]$.

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

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 $\mathsf{OblivShift}(N, E, F, k_r, I'_{\mathsf{E}})[x]$.

Figure 7: Programmable Distributed Point Function.

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 27. *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 $2^{\min(\zeta',\zeta-\zeta'-2)\lambda_{\mathsf{DCR}}-\lambda}$. 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) $G_1 : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}, G_2 : \{0,1\}^\lambda \to [2^{\zeta\lambda_{\mathsf{DCR}}}]$.

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

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 two Damgård-Jurik key pairs $(\mathsf{pk} = N, \mathsf{sk}), (\mathsf{pk}' = N', \mathsf{sk}') \leftarrow \mathsf{Gen}(1^\lambda)$ with $N \geq N'$.
4. For $i \in [n]$, let $E[i] \leftarrow \mathsf{Enc}(N, c_i)$. Let $F \leftarrow \mathsf{Enc}(N', \mathsf{sk}^{-1} \bmod N'^{\zeta'})$.
5. Sample $k_r \xleftarrow{\$} \{0,1\}^\lambda$.
6. 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 $j \in [2^i]$.
7. For $i \in [n-1]$, let $Q[i][b] = \bigoplus_{j=b2^i}^{(b+1)2^i-1} L^{(i+1)}[j]$. Let $s = \sum_{j=0}^{2^n-1} G_2(L^{(n)}[j])$.
8. Output $k_0 = (L^{(0)}, c, N, E, F, k_r, Q, s)$.

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

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 $w = v \cdot \mathsf{sk} \cdot d' + s$, where $d' = \mathsf{sk}' \cdot (\mathsf{sk}'^{-1} \bmod N'^{\zeta'})$.
4. Output $k_1 = (N, E, F, k_r, y, P, w)$.

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

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 $\mathsf{OblivShift}(N, E, F, k_r, I'_{\mathsf{G}})[x]$.

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

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 $\mathsf{OblivShift}(N, E, F, k_r, I'_{\mathsf{E}})[x]$.

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.

[Bar87] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 311–323. Springer, Berlin, Heidelberg, August 1987.

[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.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Nguyen and Oswald [NO14], pages 533–556.

[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.

[BMO+25] Amik Raj Behera, Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Privately constrained PRFs from DCR: Puncturing and bounded waring rank. Cryptology ePrint Archive, Paper 2025/230, 2025.

[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.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In Servedio and Rubinfeld [SR15], pages 429–437.

[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.

[CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[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.

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

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

[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.

[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.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th FOCS*, pages 415–434. IEEE Computer Society Press, November 2023.

[ILL24] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct homomorphic MACs from groups and applications. Cryptology ePrint Archive, Paper 2024/2073, 2024.

[ILL25] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. Cryptology ePrint Archive, Paper 2025/442, 2025.

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

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In Servedio and Rubinfeld [SR15], pages 419–428.

[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.

[MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. Cryptology ePrint Archive, Paper 2025/245, 2025.

[NO14] Phong Q. Nguyen and Elisabeth Oswald, editors. *EUROCRYPT 2014*, volume 8441 of *LNCS*. Springer, Berlin, Heidelberg, May 2014.

[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.

[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.

[Sch71] A. Schönhage. Schnelle berechnung von kettenbruchentwicklungen. *Acta Inf.*, 1(2):139–144, June 1971.

[SR15] Rocco A. Servedio and Ronitt Rubinfeld, editors. *47th ACM STOC*. ACM Press, June 2015.

[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 13. 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. □

A.2 Proof of Lemma 18

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 $P[i] = c_i\Delta \oplus X_{\mathsf{G}}[i] \oplus \bigoplus_{j=0}^{2^i-1} \widehat{L}_{\mathsf{G}}^{(i)}[j]$. 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 $L_{\mathsf{E}}^{(0)}$.

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

Input.

- Public parameter: A positive integer n, a Damgård-Jurik public key $N \le 2^{\lambda_{\mathsf{DCR}}}$, 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 $P[i] = c_i\Delta \oplus X_{\mathsf{G}}[i] \oplus \bigoplus_{j=0}^{2^i-1} \widehat{L}^{(i)}_{\mathsf{G}}[j]$ 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 $I'_{\mathsf{E}}[y] = w' - \sum_{i \ne y} I'_{\mathsf{E}}[i] \pmod{N^\zeta}$. // $I'_{\mathsf{E}}[y] = I'_{\mathsf{G}}[y] + w$
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

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 $P[i] = X_{\mathsf{E}}[i] \oplus y_i \Delta \oplus H_1(L_{\mathsf{E}}^{(i)}[y^{(i)}] \oplus \Delta) \oplus \bigoplus_{j \in [2^i] \setminus y^{(i)}} \widehat{L}_{\mathsf{E}}^{(i)}[j]$, and let $\widehat{L}_{\mathsf{E}}^{(i)}[y^{(i)}] = X_{\mathsf{E}}[i] \oplus P[i] \oplus \bigoplus_{j \in [2^i] \setminus y^{(i)}} \widehat{L}_{\mathsf{E}}^{(i)}[j]$. 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 28. *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 $\widehat{L}_{\mathsf{E}}^{(i)}[y^{(i)}] = X_{\mathsf{E}}[i] \oplus P[i] \oplus \bigoplus_{j \in [2^i] \setminus y^{(i)}} \widehat{L}_{\mathsf{E}}^{(i)}[j]$. 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 29. *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 30. *The proof also works when H_1, H_2 are modeled as Circular Correlation Robust Hash functions (CCRH), under appropriate definition that allows replacing $y_i \Delta \oplus H_1(L_{\mathsf{E}}^{(i)}[y^{(i)}] \oplus \Delta)$ and $w + H_2(L_{\mathsf{E}}^{(n)}[y] \oplus \Delta)$ with random values. In our construction, w is always a Damgård-Jurik secret key* sk.

B Security of the Garbling Scheme

In this section, we prove the obliviousness, privacy, and authenticity of the garbling scheme defined in Construction 16. As a bonus, we will prove them under the circular correlation robust hash (CCRH) assumption in the plain model, though the definition of CCRH is tailored for our construction.

B.1 Circular Correlation Robust Hash

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

Definition 31 (Circular Correlation Robustness)**.** Let $H_1 : \mathbb{Z} \times \{0,1\}^\lambda \to \{0,1\}^\lambda, H_2 : \mathbb{Z} \times \{0,1\}^\lambda \to [2^{2\zeta\lambda_{\mathsf{DCR}}}], H_3 : \mathbb{Z} \times \mathbb{Z} \to \{0,1\}^\lambda, H_4 : \mathbb{Z} \times \mathbb{Z} \to \{0,1\}^\lambda$ be four functions. For any Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk})$, we define four oracles:

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

A sequence of oracle queries is legal if and only if $\mathcal{O}_1$ is never queried with the same i, X and different b, and $\mathcal{O}_4$ is never queried with the same i, X, b' and different b. The tuple (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 $\mathcal{A}^{\mathcal{O}_1^\Delta, \mathcal{O}_2^{\Delta,\mathsf{sk}}, \mathcal{O}_3^{\mathsf{sk}}, \mathcal{O}_4^{\Delta,\mathsf{sk}}}(N)$, 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 Obliviousness

Theorem 32. *Let H_1, H_2, H_3, H_4 be circular correlation robust hash functions, as per Definition 31. Then, the garbling scheme defined in Construction 16 is oblivious (Definition 8).*

Proof. We construct a simulator $\mathsf{Sim}_{\mathsf{obv}}$ that simulates the garbling scheme. $\mathsf{Sim}_{\mathsf{obv}}$ proceeds as follows.

First, it samples a Damgård-Jurik key pair $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and simulates each $X_{\mathsf{E}}[i]$ with a random string in $\{0,1\}^\lambda$. This is indistinguishable because the real shares are also random strings.

$\mathsf{Sim}_{\mathsf{obv}}$ then goes through the circuit C gate by gate. At each gate, it uses the input shares to simulate the garbled materials and the output shares. More specifically,

- No garbled material is needed for XOR gates. The output share is the XOR of the two input shares.
- The simulator for AND gates is explicitly given in [ZRE15], page 13.
- For lookup gates, the simulator is given as Hyb_4 in the proof of Lemma 19.

Finally, it outputs the garbled materials and the share of output wires.

Now we need to show that the simulator is indistinguishable from the real garbling. As most of the work is already done in the proof of Lemma 19 and Lemma 18, the rest is a fairly standard hybrid argument. We only outline the proof below.

Experiment Hyb_0. Hyb_0 is the same as real garbling. In Hyb_0, we sample $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}$, $(\mathsf{pk} = N, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, and sample $\mathsf{Garbler}$'s share of all input wires X_{G}. We then go through the circuit gate by gate, garble each AND gate using H_1 (i.e., use H_1 as the CCRH in [ZRE15]), and garble each lookup gate using H_1, H_2, H_3, H_4. Finally, the garbled materials are collected and output, along with the encoded input shares X_{E}.

Experiment Hyb_1. In Hyb_1, we directly sample X_{E} from random, and set $X_{\mathsf{G}}[i] = X_{\mathsf{E}}[i] \oplus x[i]\Delta$ for each input wire $x[i]$. We then prepare the garbled materials as before. This corresponds to Hyb_1 in the proof of Lemma 18 and Lemma 19.

Claim 33. *The experiment Hyb_1 can be split into two parts: The first part samples $\Delta \xleftarrow{\$} 1\{0,1\}^{\lambda-1}, (pk = N, sk) \leftarrow \mathsf{Gen}(1^\lambda)$, and provides oracle access $\mathcal{O}_1^{\Delta}, \mathcal{O}_2^{\Delta,sk}, \mathcal{O}_3^{sk}, \mathcal{O}_4^{\Delta,sk}$ to the second part, where the oracles are defined as in Definition 31. The second part cannot access Δ and sk, but it uses N and the oracles to sample the shares and prepare the garbled materials. The resulting experiment is identical to Hyb_1.*

Proof. The split is done independently for each gate.

For the AND gates, the split is possible by inspecting the proof of [ZRE15]. For the lookup gates, the split is possible by inspecting Hyb_1 in the proof of Lemma 18 and Hyb_1 in the proof of Lemma 19. □

Experiment Hyb_2. Hyb_2 is defined as the split version of Hyb_1 in Claim 33.

Experiment Hyb_3. In Hyb_3, we remove the first part of Hyb_2, and replace the oracles $\mathcal{O}_1^{\Delta}, \mathcal{O}_2^{\Delta,\mathsf{sk}}, \mathcal{O}_3^{\mathsf{sk}}, \mathcal{O}_4^{\Delta,\mathsf{sk}}$ with random oracles $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$. This is computationally indistinguishable from Hyb_2, by the circular correlation robustness of H_1, H_2, H_3, H_4. Now the garbled materials of AND gates are already random, and we arrive at Hyb_3 in the proof of Lemma 19.

Experiment Hyb_4. In Hyb_4, we replace 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.

Hyb_4 is identical to $\mathsf{Sim}_{\mathsf{obv}}(1^\lambda, C)$ defined above. □

B.3 Privacy

Theorem 34. *Let H_1, H_2, H_3, H_4 be circular correlation robust hash functions, as per Definition 31. Then, the garbling scheme defined in Construction 16 is private (Definition 9).*

Proof. We construct a simulator $\mathsf{Sim}_{\mathsf{priv}}$ that simulates the garbling scheme. $\mathsf{Sim}_{\mathsf{priv}}$ first invokes $\mathsf{Sim}_{\mathsf{obv}}$ to generate $(\hat{C}, X_{\mathsf{E}})$, then evaluates the fake circuit $\hat{C}$ on the fake encoded input X_{E} to get Y_{E}. Finally, it simulates the output decoding string as follows:

- Sample $v \xleftarrow{\$} \{0,1\}^\lambda$.
- For each i, if $y[i] = 0$, set $d[i] = (H_1(v, Y_{\mathsf{E}}[i]) \| Y_{\mathsf{E}}[i][0], R \| (Y_{\mathsf{E}}[i][0] \oplus 1))$, where $R \xleftarrow{\$} \{0,1\}^\lambda$; if $y[i] = 1$, swap the order of the two terms in the pair.
- Append v to the end of d.

$\mathsf{Sim}_{\mathsf{priv}}$ outputs the tuple $(\hat{C}, X_{\mathsf{E}}, d)$.

The proof of indistinguishability is similar to that of Theorem 32. In real garbling, for each i, the two terms in the real $d[i]$ will be $H_1(v, Y_{\mathsf{E}}[i]) \| Y_{\mathsf{E}}[i][0]$ and $H_1(v, Y_{\mathsf{E}}[i] \oplus \Delta) \| (Y_{\mathsf{E}}[i][0] \oplus 1)$, where the order depends on $y[i]$. $Y_{\mathsf{E}}[i]$ can be simulated by $\mathsf{Sim}_{\mathsf{obv}}$ in the proof of Theorem 32, while $H_1(v, Y_{\mathsf{E}}[i] \oplus \Delta)$ is indistinguishable from a random string, by the circular correlation robustness of H_1. Therefore, $\mathsf{Sim}_{\mathsf{priv}}$ is indistinguishable from the real garbling. □

B.4 Authenticity

Theorem 35. *Let H_1, H_2, H_3, H_4 be circular correlation robust hash functions, as per Definition 31. Then, the garbling scheme defined in Construction 16 is authentic (Definition 10).*

Proof. Authenticity follows from obliviousness. If an adversary breaks authenticity, it can forge $Y_{\mathsf{E}}[i] \oplus \Delta$ for at least one output wire i with non-negligible probability. However, the adversary can also obtain $Y_{\mathsf{E}}[i]$ simply by running $\mathsf{Evaluate}(C, \hat{C}, X_{\mathsf{E}})$. Therefore the adversary obtains Δ with non-negligible probability, which breaks obliviousness. □