Breaking the $1/\lambda$-Rate Barrier for Arithmetic Garbling

Geoffroy Couteau[1], Carmit Hazay[2], Aditya Hegde[3⋆], and Naman Kumar[4]

[1] Université Paris Cité, CNRS, IRIF couteau@irif.fr
[2] Bar-Ilan University carmit.hazay@biu.ac.il
[3] Johns Hopkins University ahegde@cs.jhu.edu
[4] Oregon State University kumarnam@oregonstate.edu

Abstract. Garbled circuits, introduced in the seminal work of Yao (FOCS, 1986), have received considerable attention in the boolean setting due to their efficiency and application to round-efficient secure computation. In contrast, arithmetic garbling schemes have received much less scrutiny. The main efficiency measure of garbling schemes is their rate, defined as the bit size of each gate's output divided by the size of the (amortized) garbled gate. Despite recent progress, state-of-the-art garbling schemes for arithmetic circuits suffer from important limitations: all existing schemes are either restricted to B-bounded integer arithmetic circuits (a computational model where the arithmetic is performed over $\mathbb{Z}$ and correctness is only guaranteed if no intermediate computation exceeds the bound B) and achieve constant rate only for very large bounds $B = 2^{\Omega(\lambda^3)}$, or have a rate at most $O(1/\lambda)$ otherwise, where λ denotes a security parameter. In this work, we improve this state of affairs in both settings.

- As our main contribution, we introduce the first arithmetic garbling scheme over modular rings $\mathbb{Z}_B$ with rate $O(\log \lambda/\lambda)$, breaking for the first time the $1/\lambda$-rate barrier for modular arithmetic garbling. Our construction relies on the power-DDH assumption.
- As a secondary contribution, we introduce a new arithmetic garbling scheme for B-bounded integer arithmetic that achieves a constant rate for bounds B as low as $2^{O(\lambda)}$. Our construction relies on a new non-standard KDM-security assumption on Paillier encryption with small exponents.

1 Introduction

Garbled circuits, first introduced by Yao in the 1980s [Yao86], have become a fundamental and versatile tool in modern cryptography. Since their inception, a substantial body of work has expanded on Yao's construction, offering various optimizations, abstractions, and extensions. In addition to being one of the fundamental paradigms for achieving constant-round secure computation [Yao86, BMR90, FKN94, IK00, LP09], garbled circuits have also found applications in many other cryptographic areas. Their popularity stems from the ability to encode a function f, represented as a boolean circuit, in such a way that evaluating the encoded function on some input x reveals no information other than $f(x)$. Despite all the efforts to improve these constructions, the current state of the art of garbled circuits, based solely on symmetric-key assumptions, implies a multiplicative factor of $O(\lambda)$ in the encoded circuit size. Namely, the rate between the original circuit size C and the garbled circuit size $\widehat{C}$ is $1/O(\lambda)$, where λ is the security parameter. This implies limited scalability as the size of the circuit increases.

Another restriction of boolean circuits is their limited suitability for capturing a broader class of function representations, such as arithmetic circuits, when the computation is performed over integers, real numbers, or rings. Arithmetic circuits are often used to represent computations over real-world data, such as financial transactions, machine learning, or privacy-preserving protocols for statistics and data mining. For such applications, applying naive boolean garbling techniques to the arithmetic setting proves too inefficient. In particular, while traditional garbled circuits are better suited to boolean operations such as bitwise comparison, many real-world computations are more naturally expressed using arithmetic operations performed over larger domains.

Compared to the vast literature on garbled circuits for boolean circuits, not much progress has been made in the field of arithmetic garbling. This is partly because the traditional techniques do not scale with the underlying ring. Consequently, arithmetic garbling requires new approaches that

⋆ This work was done in part while the author was at IRIF.

get beyond the bit representation of the circuit wires. The arithmetic computation model can be captured by two different representations: (1) *Bounded integer computations.* This computational model considers circuits defined over the integer ring $\mathbb{Z}$ with addition and multiplication gates and a predefined bound B, where *any* wire value falls within $[-B, B]$. (2) *Modular arithmetic.* Here the computations are performed over a finite ring $\mathcal{R} = \mathbb{Z}_p$ (where p is not necessarily a prime number).

A popular approach to arithmetic garbling in the bounded integer setting reduces the problem to a variant of Yao's original garbling scheme, but this typically requires bit decomposition [BGG+14, BMR16], meaning that arithmetic operations need to be broken down into individual bitwise operations, which increases the rate. An alternative construction was proposed by Applebaum et al. [AIK11], offering a way to garble computations over bounded integers directly without bit decomposition. This method allows for computations over integers $\mathbb{Z}$ within a bounded domain $[-B, B]$, where B can be large, possibly exponential, and the security of this scheme is based on the Learning With Errors (LWE) assumption. However, their rate is still proportional to the security parameter for the LWE assumption.

Building on the work of Applebaum et al. [AIK11], Ball et al. [BLLL23] introduced an instantiation of the key expansion gadget introduced in [AIK11] using a variant of Paillier's key and message linearly homomorphic encryption scheme [Pai99, DJN10]. This approach achieves the first constant multiplicative communication overhead relative to the bit-length representation of the computed arithmetic circuit (when given in the clear). In more detail, Ball et al. introduced two variants of arithmetic garbling schemes under the Decisional Composite Residuosity (DCR) assumption with a rate of 1/36 and under the short exponent DCR assumption with a rate of 1/24. In both cases, the bound B is of the order of $B \approx 2^{4000}$. Both papers further study the modular arithmetic setting with $O(1/\lambda)$ rate.

A recent work by Meyer et al. [MORS24] improves on the concrete constants of [BLLL23] by incorporating a different technique for computing the multiplication gates using Homomorphic Secret Sharing (HSS). This approach enables to achieve rate 1/2 assuming DCR and rate-1 garbling by additionally assuming key-dependent message (KDM) security. However, these approaches are fundamentally limited to the model of B-bounded integer computation, and furthermore, their achieved bound B is still large on the order of $N^2 \approx 2^{\Omega(\lambda^3)}$ due to incorporating the same cryptographic building blocks of Paillier. In another concurrent work [Hea24], Heath proposed a garbling scheme based only on symmetric-key assumptions that garbles a circuit C over $\mathbb{Z}_m$ for an arbitrary m with rate $O(1/\lambda)$.

Two important questions are at hand in light of this state of affairs. (1) Can we surpass the $1/\lambda$ barrier for modular arithmetic garbling? (2) To what extent can we reduce the bound B while maintaining a constant rate?

1.1 Our Contributions

A Rate-$\omega(1/\lambda)$ Arithmetic Garbling Scheme. We construct the first arithmetic garbling scheme that provides an asymptotic improvement in rate over $1/\lambda$ for modular arithmetic computation. The security of our scheme is based on a mild variant of the well-known Power-DDH Assumption [GJM03, CNs07]. Our scheme forms part of a unified approach towards arithmetic garbling, which we introduce as the 'VOLE-to-OLE' framework that interprets the (both arithmetic as well as boolean) garbling of a gate as a single-message conversion from a VOLE (Vector Oblivious Linear Evaluation) correlation to an OLE (Oblivious Linear Evaluation). Our viewpoint unifies several recent developments in garbling ([ZRE15, HKN24, Hea24, MORS24]) as an instantiation of the same general framework. While previous approaches have achieved better (even constant) rates for schemes based on B-bounded integer computation, all known schemes set the value of B to be *exponential* in the size of the security parameter. Our scheme is the first to improve the rate over $1/\lambda$ for garbling over $\mathbb{Z}_B$, for small, polynomial-size domains, from a standard cryptographic assumption.

Theorem 1 (Modular Garbling from Power-DDH, Informal). *Let λ be a security parameter, B be a polynomial-size modulus $B \in \mathsf{poly}(\lambda)$, and c be any arbitrarily slow growing function $\omega(1)$. Set $B' = B^2 + c \cdot \lambda \cdot B^3$. Then, assuming the B'-power-DDH assumption and the existence of tweakable correlation-robust hash functions (a property of hash functions that holds unconditionally in the random oracle model), there exists an arithmetic garbling scheme for P/poly that supports $\mathbb{Z}_B$-modular arithmetic computation, has statistical correctness failure probability $O(1/2^c)$, and achieves a rate $O(\log B/\lambda)$.*

A New PPRF from Power-DDH. Puncturable Pseudorandom Functions are primitives that support a master key sk and a *punctured* key sk_x such that an underlying PRF can be evaluated at all points in the domain with sk and on all but one point, x, with the punctured key sk_x. In this work, we make use of a PPRF that features multiple attractive properties: the key size is small, consisting of only a single element of $\mathbb{Z}_p^*$, and it supports *reusable setup*, i.e., the same setup, that generates a master secret key and a (possibly large) master public key, can be reused for multiple key generations, which is advantageous in the amortized setting (*i.e.*, although the master public key can be large, its size overhead can be amortized over many instances). We use this PPRF to realize our VOLE-to-OLE framework, in which it features the additional advantage that provided a VOLE correlation in the form of setup, the punctured key sk_x can be derived *at runtime without interaction*. The security of our PPRF relies on the standard Power-DDH assumption (plus the existence of a suitable correlation-robust hash function, in the reusable-setup setting), and we believe the construction is of independent interest.

Improved Constant-Rate Garbling for Small Integers. Our third contribution is a concrete improvement over the HSS-based garbling scheme of [MORS24], which provides a constant rate for an improved class of integers. In particular, our scheme provides constant-rate arithmetic garbling for B-bounded integers where B can be as small as $2^{2\lambda}$. We instantiate our scheme with Homomorphic Secret Sharing from the Paillier-ElGamal cryptosystem ([OSY21]) with *short exponent*, a scheme believed to be secure under the DCR and the SEI assumptions. We base the security of our garbling scheme under a strong variant of circular security for the underlying Paillier-ElGamal cryptosystem combined with a computational one-time encryption scheme we term *circular security of the hybrid*.

Theorem 2 (Small-integer Arithmetic Garbling from Circular Security of Paillier-ElGamal, Informal). *Let λ be a security parameter. Then, assuming (a variant of) circular security of Paillier-ElGamal with small exponent and the existence of a pseudorandom function in NC^1, for every $B < 2^{O(\lambda)}$ there exists an arithmetic garbling scheme for P/poly that supports B-bounded integer computation and achieves rate $1/2 - \epsilon$ for arbitrary constant ϵ.*

Organization. In Section 2, we provide a detailed overview of our main contribution. We introduce our new PPRF in Section 3 and our modular arithmetic garbling scheme in Section 4. Being quite different in nature, our secondary contribution is deferred to the Appendix. In Appendix A and Appendix B, we provide additional preliminaries on arithmetic garbling and homomorphic secret sharing. Appendix C covers our rate-1/2 arithmetic garbling scheme for B-bounded arithmetic with $B = 2^{O(\lambda)}$. We give a detailed technical overview in Appendix C.1 and the formal construction in Appendix C.2.

Follow-up works. In a follow-up work [CHHK25], the same authors uses the techniques introduced in the power-DDH-based arithmetic garbling scheme to construct a Boolean garbling scheme with $\lambda/\sqrt{\log \lambda}$ bits per gate, beating the $O(\lambda)$-bit per gate barrier without resorting to heavy-hammer cryptography, and making only a black-box use of cryptography.

In other subsequent works, Meyer, Orlandi, Roy, and Scholl [MORS25] and Ishai, Li, and Lin [ILL25] introduced the first sublinear garbling scheme for Boolean circuits from DCR. Their scheme essentially subsumes our second contribution, and is incomparable to our first contribution and to [CHHK25] (in particular and unlike these works, both our work and the follow-up make a black-box use of cryptography).

2 Technical Overview: Modular Arithmetic Garbling from Power-DDH

We start with an overview of our construction of a modular arithmetic garbling scheme with rate $\omega(1/\lambda)$. Our starting point is a natural view on a common template for (boolean and arithmetic) garbling as a non-interactive procedure from vector oblivious linear evaluation (VOLE) to oblivious linear evaluation (OLE). We elaborate below.

2.1 Preliminaries on Power-DDH

We let $\mathsf{GrpGen}(1^\lambda)$ denote a deterministic algorithm that, on inputs the security parameter λ in unary, outputs a tuple $(\mathbb{G}, p, g)$ where $\mathbb{G}$ is (the description of) a cyclic group of order p, p is a prime of

length 2λ bits, and g is a generator of $\mathbb{G}$. Given an integer n, we write $[\pm n]$ for the set $\{-n, \cdots, n\}$ and $[n]$ for the set $\{0, \cdots, n\}$.

Definition 3 (B-power-DDH assumption). *Let $B = B(\cdot)$ be a polynomial. The B-power-DDH assumptions holds with respect to* GrpGen *if for all large enough security parameter λ, denoting $(\mathbb{G}, p, g) := \mathsf{GrpGen}(1^\lambda)$, the following distributions are computationally indistinguishable:*

$$\mathcal{D}_0 := \left\{ (g_i)_{i\in[\pm B(\lambda)]} \ : \ \alpha \leftarrow\$ \mathbb{Z}_p^*, h \leftarrow\$ \mathbb{G}, (g_i)_{i\in[\pm B(\lambda)]} \leftarrow (h^{\alpha^i})_{i\in[\pm B(\lambda)]} \right\}$$
$$\mathcal{D}_1 := \left\{ (g_i)_{i\in[\pm B(\lambda)]} \ : \ \alpha \leftarrow\$ \mathbb{Z}_p^*, h, g_0 \leftarrow\$ \mathbb{G}, (g_i)_{i\in[\pm B(\lambda)]^*} \leftarrow (h^{\alpha^i})_{i\in[\pm B(\lambda)]} \right\}.$$

The traditional formulation of the power-DDH assumption [GJM03, CNs07] states that given $(h, h^\alpha, \cdots, h^{\alpha^{B-1}})$, one cannot distinguish h^{α^B} from random. Other common variants [AMN+18], often called power-DDH as well (or DH-inversion), ask that it is hard to distinguish $h^{1/\alpha}$ from random given $(h, h^\alpha, \cdots, h^{\alpha^{B-1}})$. In this work, we use yet another slight variant, where we ask that given $(h^{\alpha^{-B}}, \cdots, h^{\alpha^{-1}}, h^\alpha, \cdots, h^{\alpha^B})$, it should be hard to distinguish the missing "center term" h from random. All variants can be shown to be equivalent, up to a factor-2 loss in the bound B. Before we sketch the simple proof, we introduce the "all-random" power-DDH variant below:

Definition 4 (All-random B-power-DDH assumption). *Let $B = B(\cdot)$ be a polynomial. The all-random B-power-DDH assumptions holds with respect to* GrpGen *if for all large enough security parameter λ, denoting $(\mathbb{G}, p, g) := \mathsf{GrpGen}(1^\lambda)$, the following distributions are computationally indistinguishable:*

$$\mathcal{D}_0^{\mathsf{ar}} := \left\{ (g_i)_{i\in[B(\lambda)]} \ : \ \alpha \leftarrow\$ \mathbb{Z}_p^*, h \leftarrow\$ \mathbb{G}, (g_i)_{i\in[B(\lambda)]} \leftarrow (h^{\alpha^i})_{i\in[B(\lambda)]} \right\}$$
$$\mathcal{D}_1^{\mathsf{ar}} := \left\{ (g_i)_{i\in[\pm B(\lambda)]} \ : \ (g_i)_{i\in[B(\lambda)]^*} \leftarrow\$ \mathbb{G}^{B(\lambda)} \right\}.$$

Theorem 5. *If the B-power-DDH assumption holds, then the all-random B-power-DDH assumption holds.*

Note that there is a slight loss in the parameters in Theorem 5 because the B-power-DDH assumption involves $2B+1$ exponents, while the all-random B-power-DDH assumption only involves $B+1$ exponents. Before we proceed with the proof, we note that the all-random B-power-DDH assumption implies all known flavors of power-DDH, including those from [GJM03, CNs07] and [AMN+18]. Given that all variants are equivalent, in this work, we simply call our variant "power-DDH".

Proof. We proceed through a sequence of hybrid distributions, moving gradually from $\mathcal{D}_0^{\mathsf{ar}}$ to $\mathcal{D}_1^{\mathsf{ar}}$. We start with the distribution $\mathcal{D}_0^{\mathsf{ar}}$ ("Hybrid$_{-1}$").

Hybrid$_0$. We modify $\mathcal{D}_0^{\mathsf{ar}}$ by replacing $g_0 \leftarrow h^{\alpha^0}$ with $g_0 \leftarrow\$ \mathbb{G}$, leaving everything else the same. We denote $\mathcal{D}_{0.0}^{\mathsf{ar}}$ the new distribution.

Hybrid$_1$. We modify $\mathcal{D}_{0.0}^{\mathsf{ar}}$ by replacing $g_1 \leftarrow h^{\alpha^1}$ with $g_1 \leftarrow\$ \mathbb{G}$, leaving everything else the same. We denote $\mathcal{D}_{0.1}^{\mathsf{ar}}$ the new distribution.

Hybrid$_{i+1}$. We modify $\mathcal{D}_{0.i}^{\mathsf{ar}}$ by replacing $g_{i+1} \leftarrow h^{\alpha^{i+1}}$ with $g_{i+1} \leftarrow\$ \mathbb{G}$, leaving everything else the same.

Hybrid$_B$. This is the distribution $\mathcal{D}_{0.B}^{\mathsf{ar}} = \mathcal{D}_1^{\mathsf{ar}}$.

Claim. Under the B-power-DDH assumption, Hybrid$_{-1}$ and Hybrid$_0$ are indistinguishable.

Proof. The reduction receives a challenge $(g_i)_{i\in\pm[B(\lambda)-1]}$ from either $\mathcal{D}_0$ or $\mathcal{D}_1$. It sets $(g'_0, \cdots, g'_B) \leftarrow (g_0, \cdots, g_B)$ and runs the distinguisher on $(g'_i)_{i\in[B(\lambda)]}$. Observe that if the challenge comes from $\mathcal{D}_0$, $(g'_i)_{i\in[B(\lambda)]}$ is distributed as in Hybrid$_0$, while if the challenges comes from $\mathcal{D}_1$, $(g_i)_{i\in[B(\lambda)]}$ is distributed as in Hybrid$_1$. ■

Claim. Under the $(B-i-1)$-power-DDH assumption, Hybrid$_i$ and Hybrid$_{i+1}$ are indistinguishable.

Proof. The reduction receives a challenge $(g_j)_{j\in\pm[B(\lambda)-i-1]}$ from either $\mathcal{D}_0$ or $\mathcal{D}_1$. It first samples $(g'_0, \cdots, g'_i) \leftarrow\$ \mathbb{G}^{i+1}$ and sets $(g'_{i+1}, \cdots, g'_B) \leftarrow (g_0, \cdots, g_{B-i-1})$. Then, it runs the distinguisher on $(g'_j)_{j\in[B(\lambda)]}$. Observe that if the challenge comes from $\mathcal{D}_0$, $(g'_j)_{j\in[B(\lambda)]}$ is distributed as in Hybrid$_i$, while if the challenges comes from $\mathcal{D}_1$, $(g_j)_{j\in[B(\lambda)]}$ is distributed as in Hybrid$_{i+1}$. ■

Eventually, we observe that the $(B-i-1)$-power-DDH assumptions are all implied by the B-power-DDH assumption for $i = -1$ to $B-1$. This concludes the proof. ■

2.2 A Template for Arithmetic Garbling

We start by outlining a template used, implicitly or explicitly, in various recent works [ZRE15, HKN24, Hea24, MORS24]. For each wire x, the garbler generates a key K_x associated with the label $L_x = \Delta x + K_x$ where Δ is a (randomly generated) global parameter known only to the garbler. Here, (L_x, K_x) can also be interpreted as a *secret sharing* $\langle \Delta x \rangle$; similarly, the garbler can also generate (l_x, k_x) to be a sharing of x, and the parties maintain an invariant sharing $(\langle x \rangle, \langle \Delta \cdot x \rangle)$ with the garbler's share being (l_x, L_x), and the evaluator's share being (k_x, K_x) for each wire x.

With this invariant in place, the garbled circuit evaluation procedure reduces to computing successive output wire values $(\langle g(x,y) \rangle, \langle \Delta \cdot g(x,y) \rangle)$. For additive gates, this follows for free by the additively homomorphic property of secret sharing – the garbler and the evaluator can compute their shares as $(\langle x+y \rangle, \langle \Delta \cdot (x+y) \rangle) = (\langle x \rangle + \langle y \rangle, \langle \Delta \cdot x \rangle + \langle \Delta \cdot y \rangle)$. For a multiplication gate consider the identities

$$z = x \cdot y = \langle x \rangle_{\mathsf{E}} \cdot \langle y \rangle_{\mathsf{E}} + \langle x \rangle_{\mathsf{E}} \cdot \langle y \rangle_{\mathsf{G}} + \langle y \rangle_{\mathsf{E}} \cdot \langle x \rangle_{\mathsf{G}} + \langle x \rangle_{\mathsf{G}} \cdot \langle y \rangle_{\mathsf{G}} \tag{1}$$

$$\Delta \cdot z = \Delta x \cdot y = \langle \Delta x \rangle_{\mathsf{E}} \cdot \langle y \rangle_{\mathsf{E}} + \langle \Delta x \rangle_{\mathsf{E}} \cdot \langle y \rangle_{\mathsf{G}} + \langle y \rangle_{\mathsf{E}} \cdot \langle \Delta x \rangle_{\mathsf{G}} + \langle \Delta x \rangle_{\mathsf{G}} \cdot \langle y \rangle_{\mathsf{G}} \tag{2}$$

The first terms of both these identities can be computed without interaction by the evaluator, while the garbler can calculate the last terms without interaction. Note, however, that the remaining 'cross' terms are secret-dependent – computing them requires garbling material.

The case of boolean circuits. The work of [ZRE15] was the first to use this identity explicitly to design a (boolean) garbling scheme. We sketch the main intuition; we assume that x, y are bits, and we focus for now on the cross term $\langle x \rangle_{\mathsf{G}} \cdot \langle y \rangle_{\mathsf{E}}$. Recall that the parties hold shares $\langle \Delta y \rangle$. Observe that $\langle \Delta y \rangle_{\mathsf{G}} - \Delta \cdot \langle y \rangle_{\mathsf{G}}$ (which the garbler can compute locally) and $\langle \Delta y \rangle_{\mathsf{E}}$ form shares of $\Delta y - \Delta \cdot \langle y \rangle_{\mathsf{G}} = \Delta \cdot \langle y \rangle_{\mathsf{E}}$. Abstracting out, the garbler and the evaluator therefore hold shares $(\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{G}}, \langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{E}})$ of $\Delta \langle y \rangle_{\mathsf{E}}$.[5] Note that the evaluator share $\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{E}}$ satisfies $\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{E}} = -\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{G}}$ if $\langle y \rangle_{\mathsf{E}} = 0$, and $\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{E}} = \Delta - \langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{G}}$ if $\langle y \rangle_{\mathsf{E}} = 1$. Then, the parties proceed as follows:

Garbler: pick a random bit r_{G} and add $(h_0, h_1) \coloneqq (\mathsf{H}(-\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{G}}) - r_{\mathsf{G}}, \mathsf{H}(\Delta - \langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{G}}) + (\langle x \rangle_{\mathsf{G}} - r_{\mathsf{G}}))$ to the garbling of the gate. Define G's share of $\langle x \rangle_{\mathsf{G}} \cdot \langle y \rangle_{\mathsf{E}}$ to be r_{G}.
Evaluator: given the gate garbling (h_0, h_1), compute $r_{\mathsf{E}} \coloneqq h_{\langle y \rangle_{\mathsf{E}}} - \mathsf{H}(\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{E}})$ and output r_{E}.

Observe that $h_{\langle y \rangle_{\mathsf{E}}} = \mathsf{H}(\Delta \langle y \rangle_{\mathsf{E}} - \langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{G}}) + (\langle y \rangle_{\mathsf{E}} \langle x \rangle_{\mathsf{G}} - r_{\mathsf{G}}) = \mathsf{H}(\langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{E}}) + (\langle y \rangle_{\mathsf{E}} \langle x \rangle_{\mathsf{G}} - r_{\mathsf{G}})$ by construction, hence r_{G} and r_{E} form additive shares of $\langle y \rangle_{\mathsf{E}} \langle x \rangle_{\mathsf{G}}$: this takes care of the first cross term. A similar strategy is used to handle the cross terms $\langle y \rangle_{\mathsf{G}} \langle x \rangle_{\mathsf{E}}$ and $\langle y \rangle_{\mathsf{E}} \langle \Delta x \rangle_{\mathsf{G}}$ (where each time we use the fact that a bit known to E "selects" between two possible strings known to G). Eventually, the remaining cross term $\langle \Delta x \rangle_{\mathsf{E}} \langle y \rangle_{\mathsf{G}}$ is slightly more tedious, but we can rewrite it as

$$\begin{aligned} \langle \Delta x \rangle_{\mathsf{E}} \langle y \rangle_{\mathsf{G}} &= (\Delta x - \langle \Delta x \rangle_{\mathsf{G}}) \cdot \langle y \rangle_{\mathsf{G}} = (\Delta(\langle x \rangle_{\mathsf{G}} + \langle x \rangle_{\mathsf{E}}) - \langle \Delta x \rangle_{\mathsf{G}}) \cdot \langle y \rangle_{\mathsf{G}} \\ &= (\Delta \langle x \rangle_{\mathsf{G}} - \langle \Delta x \rangle_{\mathsf{G}}) \cdot \langle y \rangle_{\mathsf{G}} + \langle x \rangle_{\mathsf{E}} \cdot (\Delta \langle y \rangle_{\mathsf{G}}), \end{aligned}$$

where in the last term, the leftmost part can be computed locally by G, and the rightmost part is again the product of a bit known to E and a value known to G, hence can be handled via the same approach as before.

Security of the garbling scheme hinges upon the fact that $\mathsf{H}(\Delta - \langle \Delta \langle y \rangle_{\mathsf{E}} \rangle_{\mathsf{E}})$ looks random to E, which holds assuming a suitable notion of circular correlation robustness for H. We note that in [ZRE15], r_{G} was chosen so that $h_0 = 0$ to avoid having to send it, reducing the size of the garbled gate to $2\lambda + 2$ bits (where λ is such that $\Delta \in \mathbb{F}_{2^\lambda}$), but we omit this row-reduction optimization here for simplicity (our focus is on asymptotic security and we do not optimize the constants).

The VOLE-to-OLE viewpoint. From a bird's-eye view, the approach of [ZRE15] consists of applying the celebrated IKNP protocol [IKNP03] to convert many pairs of "Δ-correlated oblivious transfers" (where the sender's inputs are pairs of strings whose difference is always Δ) into truly random oblivious transfers (OTs) via a correlation-robust hash function, and then "derandomizing" these OTs (by adding the derandomization messages to the garbled gate) to obtain the target shares of the cross terms.

[5] Over $\mathbb{F}_2$, addition and subtraction coincide, but we keep the distinction to facilitate the discussion when we generalize later to other rings.

This view suggests an immediate generalization to more general arithmetic structures (rings or fields). Fix a ring $\mathcal{R}$. The garbler and the evaluator will want to compute shares of terms of the form $u \cdot v$, where $u \in \mathcal{R}$ is known to the evaluator, and v is known to the garbler. The parties initially hold shares of $\Delta \cdot u$ (where Δ is a global constant known to the garbler), and the computation should proceed via a single message from G to E (this message will be appended to the garbling of the gate). In secure computation, computing shares of $u \cdot v$ is often called executing an OLE (for Oblivious Linear Evaluation: the evaluator should obliviously obtain $f(u) = u \cdot v - r_{\mathsf{G}}$ where r_{G} is the garbler's share and f is an affine function), while shares of Δu for many u's but a single global Δ (known to G) are generally called a VOLE (for vector-OLE: denoting $\mathbf{u}$ the vector of all the u's, and $\mathbf{r}_{\mathsf{E}}$ the vector of all the evaluator's shares of the Δu's, the garbler holds $g(\Delta) = \Delta \cdot \mathbf{u} - \mathbf{r}_{\mathsf{E}}$, which is an OLE between two vectors held by E). Using this terminology, we want to achieve the following: the garbler and the evaluator hold a single large VOLE (where the garbler plays the role of the receiver) and want to convert it into a large number of OLEs (where the roles are swapped) using a single G-to-E message.

2.3 Instantiating the Non-Interactive VOLE-to-OLE

The literature provides a few options for converting a VOLE into many OLEs using a single message. One option is to use homomorphic secret sharing (HSS) [BGI16, BGI17, BCG+17]; this was the route taken in the recent work of [MORS24]. For sufficiently large rings, this approach yields a minimal communication overhead: it achieves rate-1 garbling, where the (amortized) size of a garbling gate amounts to a single element of $\mathcal{R}$. However, this optimal rate is only achieved over extremely large rings, and due to limitations inherent to HSS-based OLEs, their construction is limited to *B-bounded arithmetic*: that is, arithmetic circuits over the integers (without modular reduction) where all intermediate values of the computation (for all possible "admissible" inputs) are guaranteed to be bounded by B.

Another one-message VOLE-to-OLE transform was recently described by Roy in [Roy22]. The approach of Roy is a direct generalization of IKNP [IKNP03] to larger fields. Under the hood, the method of Roy is the one used in the recent works of [HKN24, Hea24] to garble general lookup tables (in [HKN24]) and arithmetic circuits (in [Hea24]). A convenient way to describe the approach at a high level is to rely on *puncturable pseudorandom functions* (PPRFs). A PPRF is a pseudorandom function equipped with a *puncturing* algorithm which, given a PRF key K and a point z, output a *punctured key* K_z. Given a punctured key K_z, one can evaluate the PPRF $\mathsf{F}_{K_z}(x) = \mathsf{F}_K(x)$ at all points *except* $x = z$. Furthermore, given K_z, the value $\mathsf{F}_K(z)$ should remain indistinguishable from random. Now, assume for now that the garbler holds a PPRF key K and that the evaluator somehow managed to obtain the corresponding key K_u punctured at the point u. Further, assume that the ring $\mathcal{R}$ has polynomial size. The garbler computes the following values: $a \coloneqq \sum_{z \in \mathcal{R}} \mathsf{F}_K(z), b \coloneqq \sum_{z \in \mathcal{R}} z \cdot \mathsf{F}_K(z)$. Observe that the evaluator can compute $c \coloneqq \sum_{z \in \mathcal{R}} (u - z) \cdot \mathsf{F}_K(z) = u \cdot a - b$, because K_u allows computing all terms $\mathsf{F}_K(z)$ except for $\mathsf{F}_K(u)$, and this missing term is multiplied by $u - z = 0$. Therefore, b and c form additive shares of $u \cdot a$. Furthermore, the garbler can send $v - a$, allowing the evaluator to compute $c + u \cdot (v - a) = u \cdot v - b$, obtaining shares of $u \cdot v$. Because $a = \mathsf{F}_K(u) + \sum_{z \neq u} \mathsf{F}_K(z)$ is pseudorandom from the viewpoint of E (under the security of the PPRF), $v - a$ computationally hides v.

It remains at this stage to explain how the evaluator can obtain the punctured key K_u. In [HKN24, Hea24], this is done via the following steps:

- First, the PPRF is instantiated via the GGM PPRF [GGM86, KPTZ13, BW13, BGI14].
- To securely distribute the punctured key K_u (where G holds K and E holds u), [HKN24, Hea24] use under the hood the Doerner-shelat protocol [Ds17] that distributes a punctured PPRF key for the GGM PPRF in two rounds via $\log |\mathcal{R}|$ parallel calls to a 2-round oblivious transfer.
- To obtain the necessary OTs (with a single G-to-E message), the same approach as [ZRE15] is used: using shares of $\Delta \cdot u_i$ for all the bits u_i of u, the parties derive pseudorandom oblivious transfers using a circular correlation robust hash function, and the garbler derandomizes the OTs using a single message (added to the garbled gate).

We note that the above description is (voluntarily) oversimplified and uses a language very different from the terminology used in [HKN24, Hea24] (we believe that the authors of [HKN24, Hea24] are

aware of this "MPC-style" view of their construction, but found more convenient to describe it with a different terminology). Nevertheless, it captures the intuition of their constructions.

The main downside of the approach taken in [HKN24, Hea24] is that to execute the Doerner-shelat protocol, the parties must hold a "bitwise" VOLE for u – i.e., they must hold shares of $\Delta \cdot u_i$ for all bits u_i of u. Concretely, this means that the size of the labels must scale as $\Omega(\lambda \cdot \log |\mathcal{R}|)$. In turn, because each of the $\log |\mathcal{R}|$ OTs must be derandomized by adding material to the garbled gate, the size of each garbled gate also scales as $\Omega(\lambda \cdot \log |\mathcal{R}|)$. Therefore, the rate of the scheme is $O(1/\lambda)$.

2.4 A Garbling-Friendly PPRF from Power-DDH

We overcome the limitations of the previous approaches by introducing a new construction of puncturable pseudorandom function. Our PPRF enjoys the following appealing feature: given additive shares of $\Delta \cdot u$, the parties can compute a pair (K, K_u) (where K is a PPRF key and K_u is the key punctured at u) using *zero communication*. Our PPRF enjoys very short key size (a single element of $\mathbb{Z}_p$, where p is the order of a discrete-log-hard group) in an amortized setting (where many pair (K, K_u) are generated) and can be computed with low complexity (evaluating the PPRF boils down to computing a single exponentiation, which can be done in the complexity class NC^1). We believe that our new PPRF could enjoy other applications.

The security of our PPRF reduces to the following assumption (B is some polynomial bound): fix a group $\mathbb{G}$ of order p with generator g. Sample $\alpha \leftarrow\!\!\$\, \mathbb{Z}_p$ and $h \leftarrow\!\!\$\, \mathbb{G}$. Then, given

$$(h^{\alpha^{-B}}, h^{\alpha^{-B+1}}, \cdots, h^{\alpha^{-2}}, h^{\alpha^{-1}}, h^{\alpha^{1}}, h^{\alpha^{2}}, \cdots, h^{\alpha^{B-1}}, h^{\alpha^{B}}),$$

(that is, all terms h^{α^i} for $i = -B$ to B except for $i = 0$), it should be infeasible to distinguish $h = h^{\alpha^0}$ from random. This assumption is a static and falsifiable variant of the standard power-DDH assumption [GJM03, CNs07]. It follows from existing meta-theorems that it holds in the generic group model. For simplicity, we call it B-power-DDH in this work. We describe our PPRF construction with domain $[B] = \{0, \cdots, B\}$ below:

Setup. Output the description of a group $\mathbb{G}$ of order p with generator g, as well as a "master public key" $(g_i)_{i \in [\pm B]^*} := (h^{\alpha^i})_{i \in [\pm B]^*}$, where $[\pm B]$ stands for $\{-B, \cdots, B\} \setminus \{0\}$. Set $g_0 := h$. Set α to be the corresponding master secret key.
KeyGen. Sample a secret key $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$.
Eval. On input $x \in [B]$, output $g_x^{\mathsf{sk}} = h^{\alpha^x \mathsf{sk}}$.
Puncture. Given a point $z \in [B]$, output $\mathsf{psk} := \alpha^z \cdot \mathsf{sk}$.
Punctured Eval. On input $x \in [B]$, if $x \neq z$, output g_{x-z}^{psk}.

Correctness can be checked easily, as $g_{x-z}^{\mathsf{psk}} = h^{\alpha^{x-z} \cdot \mathsf{psk}} = h^{\alpha^{x-z} \cdot \alpha^z \mathsf{sk}} = h^{\alpha^x \mathsf{sk}} = g_x^{\mathsf{sk}}$. Security follows from the fact that when $x = z$, then $g_{x-z} = h$ is indistinguishable from random under the B-power-DDH assumption. In our scheme, the output of the PPRF will additionally be hashed into an element of $\mathbb{Z}_{p-1}$, using a hash function satisfying a suitable correlation-robustness property (the reader can think of this hash as a random oracle for simplicity). In our construction, the hash will also take some salt as input; we omit the details in this simplified overview.

From VOLE to punctured keys with zero communication. Recall that our goal is to let G and E agree on PPRF keys (K, K_u) where K_u is a key punctured at a value u known to E. Assume for now that we want to perform arithmetic computations over $\mathbb{Z}_B$, and therefore have $u \in \mathbb{Z}_B$. Furthermore, assume that G and E hold shares of $\Delta \cdot u$ over $\mathbb{Z}_{p-1}$.

The core observation is the following: let G denote a generator of $\mathbb{Z}_p^*$, and let $\alpha := G^{\Delta} \bmod p$. Assume that the master public key $(g_i)_{i \in [\pm B]^*} := (h^{\alpha^i})_{i \in [\pm B]^*}$ was generated and added to the garbling by G (as a "header" of the garbled circuit). Then, if we denote $\mathsf{sk} := G^{\langle \Delta u \rangle_{\mathsf{G}}} \bmod p$, it holds that

$$G^{\langle \Delta u \rangle_{\mathsf{E}}} = G^{\Delta u - \langle \Delta u \rangle_{\mathsf{G}}} = \alpha^u \cdot G^{\langle \Delta u \rangle_{\mathsf{G}}} = \alpha^u \cdot \mathsf{sk} \bmod p,$$

hence the evaluator can set $\mathsf{psk} := G^{\langle \Delta u \rangle_{\mathsf{E}}}$ and obtain the punctured key at u with respect to the secret key sk.

2.5 Rate-$\Omega(1/\lambda)$ Modular Arithmetic Garbling

Fix a security parameter λ and a polynomial modulus $B = B(\lambda)$. The previous scheme suggests the following garbling scheme over $\mathbb{Z}_B$ (below, the sharing is done modulo $p-1$):

- The garbler initially generates $\Delta \leftarrow\$ \mathbb{Z}_{p-1}$, sets $\alpha := G^{\Delta} \bmod p$, and samples the master public key $(g_i)_{i\in[\pm B]^*}$ using exponent α.
- For each input wire, the garbler samples her shares $(\langle x\rangle_{\mathsf{G}}, \langle \Delta x\rangle_{\mathsf{G}})$ of the (yet-undefined) input x uniformly at random. Once x is defined, the input labels will be constructed as the corresponding evaluator shares $(\langle x\rangle_{\mathsf{E}}, \langle \Delta x\rangle_{\mathsf{E}})$ of x. We will maintain the invariant that, before "computing" each gate with inputs x_0, x_1 during the evaluation of the circuit on x, the garbler and the evaluator will hold shares $(\langle x_b\rangle, \langle \Delta \cdot x_b\rangle)$ for $b = 0, 1$.
- For each multiplication gate and for each of the 4 cross-products $u \cdot v$ of the gate, where $u \in \mathbb{Z}_B$ will be known to E and v is known to G, the garbler computes a PPRF key $\mathsf{sk} := G^{\langle \Delta u\rangle_{\mathsf{G}}}$. Then, she computes $a := \sum_{z\in[\pm B]} \mathsf{F}_{\mathsf{sk}}(z)$ and $b := \sum_{z\in[\pm B]} z \cdot \mathsf{F}_{\mathsf{sk}}(z)$. She sets her share of $u \cdot v$ to b and adds $v - a$ to the garbled gate.

Given the garbled circuit and the input labels, the evaluator will iteratively compute the labels of each outgoing wire of a gate from the labels of the incoming wire. For each cross-product $u \cdot v$ in a multiplication gate $\gamma : x, y \mapsto z = x \cdot y$, the evaluator sets $\mathsf{psk} := G^{\langle \Delta u\rangle_{\mathsf{E}}}$ and uses it to compute $u \cdot (v - a) + \sum_{z\in\mathcal{R}} (u - z) \cdot \mathsf{F}_K(z) = u \cdot v - b$ (over $\mathbb{Z}_{p-1}$). Using the template for arithmetic garbling described in Section 2.2, this allows him to obtain shares $\langle xy\rangle_{\mathsf{E}}, \langle \Delta xy\rangle_{\mathsf{E}}$, i.e., the label of the outgoing wire z.

Overall, assuming that $\log p = O(\lambda)$ and that the group elements have $O(\lambda)$-bit representations, the size of a garbling of a $\mathbb{Z}_B$-arithmetic circuit is bounded by $O(\lambda \cdot B + |C| \cdot \lambda)$. Asymptotically, when $|C|$ becomes larger than B, this translates to an amortized cost of $O(\lambda)$ bits per gate over $\mathbb{Z}_B$. However, the method described above does not work – in fact, it is not even correct!

Modulus mismatch, and how to fix it. The issue stems from a modulus mismatch between the $\mathbb{Z}_B$-arithmetic computation, and the computation modulo $\mathbb{Z}_{p-1}$. Concretely, we would ideally like to have the following: a label for a wire w carrying a value x is a pair $(\langle x\rangle_{\mathsf{E}}, \langle \Delta x\rangle_{\mathsf{E}})$, and the corresponding key (computed independently of x) is $(\langle x\rangle_{\mathsf{G}}, \langle \Delta x\rangle_{\mathsf{G}})$, where

- $\langle x\rangle_{\mathsf{E}}, \langle x\rangle_{\mathsf{G}}$ form additive shares of x *over* $\mathbb{Z}_B$, and
- $\langle \Delta \cdot x\rangle_{\mathsf{E}}, \langle \Delta \cdot x\rangle_{\mathsf{G}}$ form additive shares of $\Delta \cdot x$ *over* $\mathbb{Z}_{p-1}$.

However, this cannot work. First, adding shares of Δx and Δy yields shares of $\Delta(x + y)$ over $\mathbb{Z}_{p-1}$, but $(x + y)$ is not reduced modulo B. Second, going back to the template described in Section 2.2, recall that to compute shares of the cross term $\langle x\rangle_{\mathsf{G}} \cdot \langle y\rangle_{\mathsf{E}}$ (when computing the keys and labels for a product gate with inputs x and y), the first step was for the parties to compute shares of $\Delta\langle y\rangle_{\mathsf{E}}$ of the form $(\langle \Delta y\rangle_{\mathsf{G}} - \Delta \cdot \langle y\rangle_{\mathsf{G}}, \langle \Delta y\rangle_{\mathsf{E}})$. However, this is incorrect, because $y - \langle y\rangle_{\mathsf{G}}$ is not equal to $\langle y\rangle_{\mathsf{E}}$ when the computation is done modulo $p - 1$.

We could attempt to fix this by letting $\langle x\rangle_{\mathsf{G}}, \langle x\rangle_{\mathsf{E}}$ be shares of a value x over $\mathbb{Z}_{p-1}$. However, we would not anymore be computing a circuit over $\mathbb{Z}_B$ (but rather a bounded integer arithmetic circuit), missing our target goal. But even more fundamentally, recall that our PPRF-based method is restricted to puncturing values from a polynomial-sized set. In our context, to compute shares of the cross term $\langle x\rangle_{\mathsf{G}} \cdot \langle y\rangle_{\mathsf{E}}$, the evaluator wants to obtain a key punctured at $\langle y\rangle_{\mathsf{E}}$. That is, we inherently require the *shares themselves* to be small. Alas, even fixing the input labels to be small does not work: first, we cannot mask over the integers with polynomial-size values (and we cannot use rejection-sampling-based methods as the garbler shares must be picked before the inputs to be masked are even defined), and second, the size of the shares will increase significantly after each gate (by a factor at least B^2).

First idea: PPRF-based modular reduction. The above issues are non-trivial to solve, and our solution combines several ideas to address each of the shortcomings of the naive approach. We start by changing the invariant that we wish to maintain. For each wire carrying a value x, the parties will hold:

- values $(k_x, \ell_x) := (\langle x\rangle_{\mathsf{E}}, \langle x\rangle_{\mathsf{G}}) \in \mathbb{Z}_B$ forming additive shares of x over $\mathbb{Z}_B$, and
- values $(K_x, L_x) := (\langle \Delta \cdot \langle x\rangle_{\mathsf{E}}\rangle_{\mathsf{E}}, \langle \Delta \cdot \langle x\rangle_{\mathsf{E}}\rangle_{\mathsf{G}})$ forming additive shares of $\Delta \cdot \langle x\rangle_{\mathsf{E}}$ over $\mathbb{Z}_{p-1}$, where $\langle x\rangle_{\mathsf{E}}$ is viewed as an element of $\{0, \cdots, B-1\}$ embedded into $\mathbb{Z}_{p-1}$.

We must now explain how we maintain the invariant, which is non-trivial even for addition gates. We start with addition gates, as they are simpler and already use one of our core ideas. Given an addition gates with incoming wires carrying values x and y, the garbler sets $\tilde{k}_z \coloneqq k_x + k_y$ and $\tilde{K}_z \coloneqq K_x + K_y$, while the evaluator sets $\tilde{\ell}_z \coloneqq \ell_x + \ell_y$ and $\tilde{L}_z \coloneqq L_x + L_y$. Observe that $(\tilde{K}_z, \tilde{L}_z)$ form additive shares of $\Delta \cdot (\ell_x + \ell_y) = \Delta \cdot \tilde{\ell}_z \bmod p-1$. However, while $(\tilde{k}_z, \tilde{\ell}_z)$ sum to $x+y$ when reduced mod B, they are *not* elements of $\mathbb{Z}_B$ as the sum is computed over the integers.

To control the growth, we build a gadget allowing the parties, given shares of $\Delta \cdot \tilde{\ell}_z$, to compute shares of $\Delta \cdot [\tilde{\ell}_z \bmod B]$ (where $[\tilde{\ell}_z \bmod B]$ denotes the remainder of the euclidean division of $\tilde{\ell}_z$ by B, which belongs to $\mathbb{Z}_B$). This is done by relying on the fact that in a PPRF-based OLE, the parties can actually compute an *arbitrary function of the punctured point* for free. Indeed, fix an arbitrary function f and let $\mathsf{sk} \coloneqq G^{\tilde{K}_z}$ and $\mathsf{psk} \coloneqq G^{\tilde{L}_z} = \alpha^{\tilde{\ell}_z} \cdot \mathsf{sk}^{-1} \bmod p$. Using sk, let the garbler compute

$$a \coloneqq \sum_{i \in [2B]} \mathsf{F}_{\mathsf{sk}}(i) \bmod p-1, \qquad K_z \coloneqq \sum_{i \in [2B]} f(i) \cdot \mathsf{F}_{\mathsf{sk}}(i) \bmod p-1.$$

Above, the PPRF is assumed to have domain $[2B]$ because $\tilde{\ell}_z = \ell_x + \ell_y$ can be as large as $2B$. Then, the evaluator can compute

$$c \coloneqq \sum_{i \in [2B], i \neq \tilde{\ell}_z} (f(\tilde{\ell}_z) - f(i)) \cdot \mathsf{F}_{\mathsf{psk}}(i) = f(\tilde{\ell}_z) \cdot a - K_z \bmod p-1.$$

Hence, G and E hold shares of $f(\tilde{\ell}_z) \cdot a$ over $\mathbb{Z}_p - 1$. Eventually, the garbler sends $a - \Delta$ to let the evaluator shift his share to a share L_z of $\Delta \cdot f(\tilde{\ell}_z)$. Setting $f : x \mapsto [x \bmod B]$, the parties ultimately obtain shares (K_z, L_z) of $\Delta \cdot \ell_z$ with $\ell_z \coloneqq [\tilde{\ell}_z \bmod B]$, and set $(k_z, \ell_z) \coloneqq ([\tilde{k}_z \bmod B], [\tilde{\ell}_z \bmod B])$, maintaining the invariant.

Second idea: leakage-resilient computation on small shares. The above idea suffices to maintain the invariant for addition gates. However, for multiplication gates, we still have the issue that after executing a PPRF-based OLE, the shares obtained are large. Concretely, consider a multiplication gate with inputs x and y, where the garbler holds the keys (k_x, K_x) and (k_y, K_y), and the evaluator holds the labels (ℓ_x, L_x) and (ℓ_y, L_y). The parties start by computing

- $\tilde{k}_z \coloneqq k_x k_y + \langle k_x \ell_y \rangle_{\mathsf{G}} + \langle k_y \ell_x \rangle_{\mathsf{G}}$ (over the integers)
- $\tilde{\ell}_z \coloneqq \ell_x \ell_y + \langle k_x \ell_y \rangle_{\mathsf{E}} + \langle k_y \ell_x \rangle_{\mathsf{E}}$ (over the integers),

where we crucially require the shares of the cross terms $k_x \ell_y$ and $k_y \ell_x$ to be computed *over the integers*. It now remains for the parties to compute shares of $\Delta \cdot \tilde{\ell}_z$; after that, both parties will apply the same modular reduction step as for addition gates to obtain shares of $\Delta \cdot [\tilde{\ell}_z \bmod B] \bmod p-1$. The issue is that, unlike for addition gates, the size of $\tilde{\ell}_z$ is influenced by the size of the *shares* $\langle k_x \ell_y \rangle_{\mathsf{E}}, \langle k_y \ell_x \rangle_{\mathsf{E}}$. Hence, to let the evaluator obtain a key punctured at $\tilde{\ell}_z$, we must guarantee that $\tilde{\ell}_z$ remains polynomially bounded.

To do that, we let the PPRF (used to compute the shares of the cross terms) output *polynomially-bounded values*. Concretely, let us focus on the cross term $k_x \ell_y$. Let $\mathsf{sk} \coloneqq G^{\langle \Delta \ell_y \rangle_{\mathsf{G}}} \bmod p$ and $\mathsf{psk} \coloneqq G^{\langle \Delta \ell_y \rangle_{\mathsf{E}}} = \alpha^{\ell_y} \cdot \mathsf{sk} \bmod p$. Let $\tilde{B}$ be a (polynomial) bound on $\mathsf{F}_{\mathsf{sk}}(i)$ (for any $i \in [B]$). Recall that $\langle k_x \ell_y \rangle_{\mathsf{E}}$ is computed as $\ell_x \cdot (k_x - a) + \sum_{i \neq \ell_y} (\ell_y - i) \cdot \mathsf{F}_{\mathsf{psk}}(i)$, where $a = \sum_i \mathsf{F}_{\mathsf{sk}}(i)$. Then, the size of $\langle k_x \ell_y \rangle_{\mathsf{E}}$ is approximately bounded by $B^2 \cdot \tilde{B}$. This translates to a bound on the size of $\tilde{\ell}_z$ of about $2B^2\tilde{B}$. Hence, to execute the PPRF-based modular reduction procedure described above, it suffices at this stage to rely on another PPRF with domain $[2B^2\tilde{B}]$ and outputs over $\mathbb{Z}_{p-1}$.

The main issue of this approach is that setting $\tilde{B}$ to a polynomial introduces *leakage* on k_x. Concretely, the evaluator is given $k_x - a$, where $a = \sum_i \mathsf{F}_{\mathsf{sk}}(i)$. Substracting the terms that E can compute, we are left with $k_x - \mathsf{F}_{\mathsf{sk}}(\ell_x)$: the (secret) wire key k_x is masked with $\mathsf{F}_{\mathsf{sk}}(\ell_x)$ *over the integers*, but $\mathsf{F}_{\mathsf{sk}}(\ell_x)$ is bounded by $\tilde{B}$! Fortunately, we can measure precisely the leakage that this value induces: concretely, if $k_x - \mathsf{F}_{\mathsf{sk}}(\ell_x)$ belongs to $\{B - \tilde{B}, \cdots, 0\}$, no leakage occurs; else, E learns information of the form "$k_x \leq i$" or "$k_x \geq i$" (for example, if $k_x - \mathsf{F}_{\mathsf{sk}}(\ell_x) = 2$, E learns that $k_x \geq 2$).

To handle this leakage, we adopt the following strategy: instead of using k_x directly in the computation, the garbler shares k_x into c shares $(k_x^1, \cdots, k_x^c)$ over $\mathbb{Z}_B$, where c is a value to be determined later. The intermediate values $\tilde{k}_z, \tilde{\ell}_z$ are computed via the modified equations below (over the integers):

- $\tilde{k}_z := \sum_{i,j \le c} k_x^i k_y^j + \sum_{i=1}^{c} (\langle k_x^i \ell_y \rangle_{\mathsf{G}} + \langle k_y^i \ell_x \rangle_{\mathsf{G}})$
- $\tilde{\ell}_z := \ell_x \ell_y + \sum_{i=1}^{c} (\langle k_x^i \ell_y \rangle_{\mathsf{E}} + \langle k_y^i \ell_x \rangle_{\mathsf{E}})$.

Above, each of the shares $\langle k_x^i \ell_y \rangle_{\mathsf{E}}, \langle k_y^i \ell_x \rangle_{\mathsf{E}}$ are computed via the PPRF-based procedure, using a polynomial bound $\tilde{B} := \lambda \cdot B$. In addition, we rely on a "salted" variant of the PPRF to ensure independent executions: F_{sk} now takes as input a pair (i, salt) where salt is any string, and the PPRF security game is extended to a multi-instance security notion where multiple values $(\mathsf{F}_{\mathsf{sk}}(i, \mathsf{salt}_j))_j$ are simultaneously indistinguishable from random even given the punctured key psk punctured at i, provided that the salt_j are pairwise distinct.

Observe that the probability that a leakage occurs, that is, e.g., $k_x^i - \mathsf{F}_{\mathsf{sk}}(\ell_x, \mathsf{salt}_i)$ does not belong to $\{B - \lambda B, \cdots, 0\}$, happens with probability $1/\lambda$. For k_x to leak, it must happen that *all* the k_x^i leak simultaneously, which happens with probability $1/\lambda^c$. Setting c to be any superconstant function suffices to make this quantity negligible. An important observation[6], that yields a factor-c efficiency improvement in the garbled circuit size, is that to compute the sum $\sum_{i=1}^{c} (\langle k_x^i \ell_y \rangle_{\mathsf{E}} + \langle k_y^i \ell_x \rangle_{\mathsf{E}})$, it is not necessary to transmit all masked values $\mathsf{shift}_x^i := k_x^i - a_x^i$ and $\mathsf{shift}_y^i := k_y^i - a_y^i$ (even though it would be secure to do so): the evaluator computes this sum as $\sum_{i=1}^{c} \ell_x \cdot (k_x^i - a_x^i) + \sum_{i=1}^{c} \ell_y \cdot (k_y^i - a_y^i)$ (where the a_x^i, a_y^i denote the values $\sum_j \mathsf{F}_{\mathsf{sk}}(j, \mathsf{salt}_i^x), \sum_j \mathsf{F}_{\mathsf{sk}}(j, \mathsf{salt}_i^y)$ for appropriate distinct salts). Therefore, as ℓ_x and ℓ_y factor out, it suffices to let the evaluator transmit only $\sum_{i=1}^{c} (k_x^i - a_x^i)$ and $\sum_{i=1}^{c} (k_y^i - a_y^i)$. As a consequence, we can set $c := \Omega(\lambda)$ to achieve a leakage probability of $O(1/2^\lambda)$ without harming the rate of our garbling scheme.

Using the above ideas, the parties have all the required ingredients to compute shares $(\tilde{K}_z, \tilde{L}_z)$ of $\Delta \cdot \tilde{\ell}_z \bmod p-1$ (using this time PPRF computations with domain $\mathbb{Z}_{p-1}$). Concretely, the parties compute

- $\tilde{K}_z := \langle K_x \ell_y \rangle_{\mathsf{G}} - \Delta \cdot (\sum_{i=1}^{c} (\langle k_x^i \ell_y \rangle_{\mathsf{G}} + \langle k_y^i \ell_x \rangle_{\mathsf{G}}) + \sum_{i=1}^{c} (\langle \Delta k_x^i \ell_y \rangle_{\mathsf{G}} + \langle \Delta k_y^i \ell_x \rangle_{\mathsf{G}}) \bmod p-1$, and
- $\tilde{L}_z := \langle K_x \ell_y \rangle_{\mathsf{E}} + L_x \ell_y + \sum_{i=1}^{c} (\langle \Delta k_x^i \ell_y \rangle_{\mathsf{E}} + \langle \Delta k_y^i \ell_x \rangle_{\mathsf{E}}) \bmod p-1$.

For the same reason as before, the term $\sum_{i=1}^{c} (\langle \Delta k_x^i \ell_y \rangle_{\mathsf{E}} + \langle \Delta k_y^i \ell_x \rangle_{\mathsf{E}})$ can be computed by E using only two $O(\lambda)$-bit "sums of shifts" from G, instead of $2c$. A simple (but slightly tedious) calculation shows that $\tilde{K}_z + \tilde{L}_z = \Delta \cdot \tilde{\ell}_z \bmod p-1$. Furthermore, the value of $\tilde{\ell}_z$ is bounded by (approximately) $2c \cdot \lambda B^3$, which is polynomial; hence, the parties can use the previous PPRF-based modular reduction to compute shares (K_z, L_z) of $\Delta \cdot [\tilde{\ell}_z \bmod B]$ over $\mathbb{Z}_{p-1}$ obtaining the desired invariant.

Dealing with circular security. An issue that we have overlooked so far is that the construction uses PPRF outputs to hide terms that depend on Δ, but Δ is (the discrete log of) the master secret key of the PPRF itself. While the construction sketched above is plausibly secure, proving security requires assuming a non-standard circularly secure flavor of the power-DDH assumption. To get a reduction to power-DDH, we take instead a leveled approach, using D different keys $\Delta_1, \cdots, \Delta_D$ for each of the $D = \mathsf{depth}(C)$ levels of the circuit C, and make sure that every time a Δ_i is masked with a term $\mathsf{F}_{\mathsf{sk}_j}(x)$, it holds that $j < i$. This introduces additional overhead, as the garbled circuit must now contain D master secret keys, and additional complications, as some of the equations we used here break down when we have multiple Δ's (specifically, the previous equations for computing $\tilde{L}_z, \tilde{K}_z$ don't work anymore). However, correct equations for the leveled setting can be recovered with slightly more work.

Wrapping up. Putting everything together, the size of the garbled circuit scales as $|\hat{C}| = O(\lambda \cdot |C| + \lambda B' \cdot \mathsf{depth}(C))$, where $B' = O(c\lambda B^3)$ and c is $\omega(1)$ (we can take $c = \Omega(\lambda)$ to get an exponentially small failure probability). Asymptotically, and for not too "tall-and-skinny" circuits, this yields a circuit with an amortized cost of $O(\lambda)$ bits per gate, a $\log B = O(\log \lambda)$ improvement over the state-of-the-art garbling scheme for modular arithmetic from [Hea24] (which uses $O(\lambda \cdot \log B$ bits per gate), and the first garbling scheme ever to beat the $1/\lambda$-rate barrier for general modular arithmetic circuits.

Eventually, while the construction is tailored to small integer rings $\mathbb{Z}_B$, it immediately extends to rings $\mathbb{Z}_N$ of arbitrary size, provided that N is smooth (i.e., has only polynomial-size coprime factors) by using CRT decomposition and running the modular garbling scheme for each of the coprime factors. It also extends to extension rings of $\mathbb{Z}_B$, isomorphic to $\mathbb{Z}_B^t$, by using "schoolbook" multiplication to write a product over $\mathbb{Z}_B^t$ as t^2 cross products over $\mathbb{Z}_B$; in this setting, the rate improves asymptotically over the state of the art as long as $t^2 \ll \log B$, which still suffices to handle superpolynomial-size extension rings.

[6] Due to an anonymous Eurocrypt'25 reviewer

3 A Puncturable PRF with Short Keys from Power-DDH

3.1 Correlation-Robust Hashing

A hash family H is *correlation-robust* for a class of correlations C if samples of the form $\mathsf{H}(C(x,s))$ for public inputs x's and secret s are indistinguishable from random. Common classes of correlations in the literature include additive correlations [IKNP03] ($C(x,s) = x + s$), affine correlations [Ser24] ($C((x_0,x_1),s) = x_0 \cdot s + x_1$), group-induced correlations [AMN+18] ($C(x,s) = x \cdot s$ where x, s belong to some group $(\mathbb{G},\cdot)$) and exponential correlations [BCM+24] ($C(x,s) = s^x$ where $s \in \mathbb{G}$ and $x \in \mathbb{Z}_{\mathsf{ord}(\mathbb{G})}$). In this work, we rely on a correlation-robust hash function for the latter correlation. We further need the hash family to be *tweakable*: A hash function H is *tweakable correlation-robust* for a class of correlations C if samples of the form $\mathsf{H}(C(x,s),y)$ are indistinguishable from random given public inputs x's and public *tweaks* y's (where all pairs (x,y) are distinct).

Tweakable correlation-robust hashing for exponential correlations. Given a security parameter λ, fix group parameters $(\mathbb{G},p,g,G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$ (recall that g generates $\mathbb{G}$, and G generates $\mathbb{Z}_p^*$). We introduce below the notion of *tweakable correlation-robustness for exponential correlations*. The definition is reproduced verbatim from [CHHK25]:[7]

Definition 6 (Tweakable correlation-robust hashing for exponential correlations over $\mathbb{G}$). *Given a security parameter λ, let $(\mathbb{G},p,g,G) \coloneqq \mathsf{GrpGen}^*(1^\lambda)$. Let $\mathsf{H} = \{\mathsf{H}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of hash functions $\mathsf{H}_\lambda : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}$. Given $h \in \mathbb{G}$, let $\mathcal{O}_{\mathsf{H},h}$ denote the oracle that, on input $(x,y) \in \mathbb{Z}_{p-1} \times \{0,1\}^*$, returns $\mathsf{H}(h^x,y)$. We call y the* tweak.

We say that the hash family $\mathsf{H} = \{\mathsf{H}_\lambda\}_{\lambda\in\mathbb{N}}$ is a TCR hash for exponential correlations over $\mathbb{G}$ *if for every probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$\left|\Pr[\mathcal{A}^{\mathsf{H},\mathcal{O}_{\mathsf{H},h}}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\mathsf{H},\mathsf{RO}}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random choice of $h \leftarrow\!\!\$\ \mathbb{G}$ and of a random oracle $\mathsf{RO} : \mathbb{Z}_{p-1} \times \{0,1\}^* \to \mathbb{Z}_{p-1}$. When H and $\mathcal{O}_{\mathsf{H},h}$ do not take as input a tweak y, we say that H is correlation-robust for exponential correlations over $\mathbb{G}$.

In particular, we note that random oracles are tweakable correlation-robust for exponential correlations, hence security under the existence of a TCR hash implies security in the random oracle model. TCR hashing for exponential correlations is a relatively simple security notion that can be plausibly conjectured for any standard (unstructed) hash function, such as SHA2 ror SHA3.

3.2 Puncturable Pseudorandom Functions

Pseudorandom functions (PRF), introduced in [GGM86], are keyed functions which are indistinguishable from truly random functions. A *puncturable pseudorandom function* (PPRF) is a PRF F such that given an input z, and a PRF key sk, one can generate a *punctured* key, denoted psk, which allows evaluating F at every point except for z, and does not reveal any information about the value $\mathsf{F.Eval}(\mathsf{sk},z)$. PPRFs have been introduced in [KPTZ13, BW13, BGI14].

In this work, we introduce and build upon a puncturable pseudorandom function which features a reusable master public key and master secret key: the same pair $(\mathsf{mpk},\mathsf{msk})$ can be reused for an arbitrary number of generations of PPRF secret keys sk (and corresponding punctured secret keys psk). The main advantage of our construction is its key size: the keys sk and psk contain a single element of $\mathbb{Z}_p^*$ (where p is the order of a dlog-hard group). This is particularly useful in an amortized setting, where the cost of storing mpk is amortized across many PPRF key generations. To capture this setting, we define below a variant of selectively-secure puncturable pseudorandom functions *with setup*.

Definition 7 (Puncturable Pseudorandom Function). *A puncturable pseudorandom function (PPRF) with domain $\mathsf{Y} = \mathsf{Y}_{B,\lambda}$ is a 5-tuple of probabilistic polynomial-time algorithms* $(\mathsf{F.Setup}, \mathsf{F.KeyGen}, \mathsf{F.Eval}, \mathsf{F.Punct}, \mathsf{F.PEval})$ *such that*

[7] Though [CHHK25] is a follow-up to the current work, our work was updated *a posteriori* to include the formal definition of this flavor of correlation-robustness, that was first formally defined in [CHHK25].

Experiment $\mathsf{ExpPPRF}(1^\lambda, B, b)$

Setup Phase. Upon receiving a setup query from $\mathcal{A}$, the challenger samples $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{F.Setup}(1^\lambda, B)$ and sends mpk to $\mathcal{A}$. Ignore all further setup queries from $\mathcal{A}$.

Challenge Phase. Upon receiving a query $(\mathsf{challenge}, z)$ from $\mathcal{A}$ with $z \in [B]$, the challenger picks $\mathsf{sk} \leftarrow\!\!\$\ \mathsf{F.KeyGen}(\mathsf{mpk})$. The challenger sends $\mathsf{psk} \leftarrow\!\!\$\ \mathsf{F.Punct}(\mathsf{msk}, \mathsf{sk}, z)$ to $\mathcal{A}$. If $b = 0$, the challenger additionally sends $\mathsf{F.Eval}(\mathsf{msk}, \mathsf{sk}, z)$ to $\mathcal{A}$; otherwise, if $b = 1$, the challenger picks $y \leftarrow\!\!\$\ \mathsf{Y}$ and sends it to $\mathcal{A}$.

Fig. 1: Selective security game with reusable setup for puncturable pseudorandom functions. The adversary can request an arbitrary number of challenges. At the end of the experiment, $\mathcal{A}$ sends a guess b' and wins if $b' = b$.

- $\mathsf{F.Setup}(1^\lambda, B)$ *outputs a master public key and master secret key* $(\mathsf{mpk}, \mathsf{msk})$. *The integer* B *specifies the input domain* $[B]$ *of the PRF, and* (λ, B) *specifies the range* $\mathsf{Y}_{\lambda,B}$ *(we assume that* (λ, B) *are included in* mpk*).*
- $\mathsf{F.KeyGen}(\mathsf{mpk})$ *outputs a PRF key* sk.
- $\mathsf{F.Eval}(\mathsf{msk}, \mathsf{sk}, x)$*, where* $x \in [B]$*, outputs a PPRF evaluation* $y \in \{0,1\}^\lambda$.
- $\mathsf{F.Punct}(\mathsf{msk}, \mathsf{sk}, z)$*, outputs a punctured key* psk.
- $\mathsf{F.PEval}(\mathsf{mpk}, \mathsf{psk}, z, x)$*, on input a punctured key* psk*, a punctured point* z*, and a point* x*, outputs* $y \in \{0,1\}^\lambda$ *if* $x \neq z$*, and* $\perp$ *otherwise.*

A PPRF F *is* selectively secure with reusable setup *no probabilistic polynomial-time adversary wins the experiment* $\mathsf{ExpPPRF}$ *represented on Figure 1 with non-negligible advantage over the random guess.*

Remark 8. Our definition of PPRFs differs from the standard presentation on two aspects. First, we consider PPRFs where the same setup can be reused for many key generations; we recover the standard selective security game for PPRFs by restricting $\mathcal{A}$ to make a single challenge query. Second, the standard definition of PPRFs typically requires that $\mathsf{F.Setup}, \mathsf{F.KeyGen}, \mathsf{F.Eval}$ is itself a pseudorandom function (note that this is not implied by the selective security game, which only ensures pseudorandomness at the punctured point given the punctured key). This stronger security notion is not needed in our work, but we note that it can be nevertheless shown to hold as well for our construction using a slightly stronger assumption.

3.3 An Efficient PPRF from Power-DDH

We represent on Protocol 1 a construction of punctured pseudorandom function over a polynomial-size domain $[B]$ from the power-DDH assumption.

Algorithm Puncturable PRF From Power-DDH

$\mathsf{F.Setup}(1^\lambda, B)$

1 : $(\mathbb{G}, p, g) \coloneqq \mathsf{GrpGen}(1^\lambda)$
2 : $(\alpha, h) \leftarrow\!\!\$\ \mathbb{Z}_p^* \times \mathbb{G}$
3 : **for** $i \in [\pm B], g_i \coloneqq h^{\alpha^i}$
4 : $\mathsf{mpk} \coloneqq \left(\mathbb{G}, p, (g_i)_{i \in [\pm B]^*}\right)$
5 : $\mathsf{msk} \coloneqq (\mathsf{mpk}, \alpha, g_0)$
6 : **return** $(\mathsf{mpk}, \mathsf{msk})$

$\mathsf{F.KeyGen}(\mathsf{mpk})$

1 : **parse** p from mpk
2 : **return** $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p^*$

$\mathsf{F.Punct}(\mathsf{msk}, \mathsf{sk}, z)$

1 : **parse** (p, α) from msk
2 : **return** $\mathsf{psk} = \alpha^z \mathsf{sk} \bmod p$

$\mathsf{F.PEval}^{\mathsf{H}}(\mathsf{mpk}, \mathsf{psk}, z, x)$

1 : **parse** $(g_i)_{i \neq 0}$ from mpk
2 : **return** $\mathsf{H}\left(g_{x-z}^{\mathsf{psk}}\right)$

$\mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x)$

1 : **parse** $(g_i)_{i \in [\pm B]}$ from msk
2 : **return** $\mathsf{H}\left(g_x^{\mathsf{sk}}\right)$

Protocol 1: Puncturable PRF from the power-DDH assumption.

The construction involves a large master public key mpk, containing $\Omega(B)$ elements of a group $\mathbb{G}$ of order p. However, in an amortized setting where the same master public key is reused across $n \gg B$ generations of PRF keys, it features extremely compact keys: both the secret key sk and the punctured secret key psk contain a single element of $\mathbb{Z}_p^*$. The procedures $\mathsf{F.Eval}^{\mathsf{H}}$ and $\mathsf{F.PEval}^{\mathsf{H}}$ rely on a hash function $\mathsf{H} : \mathbb{G} \to \mathsf{Y}$, where $\mathsf{Y} = \mathsf{Y}_{\lambda,B}$ is the range of the PPRF. Formally, H is a family of functions $\mathsf{H}_{\lambda,B}$ parametrized by (λ, B) (note that $\mathbb{G}$ is fully determined by λ as GrpGen is a deterministic procedure), but we drop the transcript for readability. We require H to be a TCR hash for exponential correlations (see Definition 6).

Theorem 9. *Assume that the B-power-DDH assumption holds (Definition 3) and that H is a tweakable correlation-robust hash for exponential correlations over $\mathbb{G}$. Then the construction $(\mathsf{F.Setup}_B, \mathsf{F.KeyGen}, \mathsf{F.Eval}, \mathsf{F.Punct}, \mathsf{F.PEval})$ from Protocol 1 is a* selectively-secure puncturable pseudorandom function with reusable setup *with polynomial-size input domain $[B]$.*

We note that this basic version of the PPRF does not use tweaks: standard (non-tweakable) correlation-robustness actually suffices (but the tweak will be necessary in the version used in our construction, that we discuss next).

Proof of Theorem 9. First, we argue correctness. Fix any $\mathsf{mpk} = \left(\mathbb{G}, p, (g_i)_{i\in[\pm B]^*}\right)$ and $\mathsf{msk} = (\mathsf{mpk}, \alpha, g_0)$ in the support of $\mathsf{F.Setup}_B$. Fix any $\mathsf{sk} \in \mathbb{Z}_p^*$, and $x, z \in [\pm B]$ with $x \neq z$. Then,

$$\mathsf{F.Eval}(\mathsf{msk}, \mathsf{sk}, x) = \mathsf{H}(g_x^{\mathsf{sk}}) = \mathsf{H}(g^{\alpha^x \cdot \mathsf{sk}}) = \mathsf{H}(g^{\alpha^{x-z} \cdot (\alpha^z \mathsf{sk})}) = \mathsf{H}(g_{x-z}^{\mathsf{psk}}) = \mathsf{F.PEval}(\mathsf{mpk}, \mathsf{psk}, z, x).$$

We now argue security through a sequence of straightforward game hops:

Game $0.b$**.** This is the experiment $\mathsf{ExpPPRF}(1^\lambda, B, b)$ from Figure 1:

Setup Phase. The challenger picks $(\mathbb{G}, p, g) \coloneqq \mathsf{GrpGen}(1^\lambda)$ and samples $\alpha, h \leftarrow\!\!\$\ \mathbb{Z}_p^* \times \mathbb{G}$. It sets $(g_i)_{i\in[\pm B]} \coloneqq (h^{\alpha^i})_{i\in[\pm B]}$ and sends $\left(\mathbb{G}, p, (g_i)_{i\in[\pm B]^*}\right)$ to $\mathcal{A}$. Upon receiving $z \in [\pm B]$, it samples $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p^*$.

Challenge Phase. The challenger sets $\mathsf{psk} \coloneqq \alpha^z \mathsf{sk}$. If $b = 0$, the challenger sets $y \coloneqq \mathsf{H}(g_0^{\mathsf{psk}})$; else, the challenger sets $y \leftarrow\!\!\$\ \mathsf{Y}$. The challenger sends (psk, y) to $\mathcal{A}$.

Game $1.b$**.** In this game, the challenger does not construct psk as $\alpha^z\mathsf{sk}$ in the challenge phase; Instead, it samples $\mathsf{psk} \leftarrow\!\!\$\ \mathbb{Z}_p^*$. Because sk is uniformly distributed over $\mathbb{Z}_p^*$, so is $\alpha^z\mathsf{sk}$, hence Game $1.b$ is perfectly indistinguishable from Game $0.b$.

Game $2.b$**.** In this game, the challenger sets $(g_i)_{i\in[\pm B]^*} \coloneqq (h^{\alpha^i})_{i\in[\pm B]^*}$, but behaves differently in the challenge phase:

- If $b = 0$, it behaves as in Game 1.0, sets $g_0 \coloneqq h$ and sends $y \coloneqq \mathsf{H}(g_0^{\mathsf{psk}})$.
- If $b = 1$, it samples instead a fresh uniform $g_0 \leftarrow\!\!\$\ \mathbb{G}$, and sends $y \coloneqq \mathsf{H}(g_0^{\mathsf{psk}})$.

The only difference with Game $1.b$ happens when $b = 1$, where y is computed as $\mathsf{H}(g_0^{\mathsf{psk}})$ for a fresh uniformly random group element g_0. Observe that with overwhelming probability, all punctured secret keys psk are distinct. Therefore, by the tweakable correlation-robustness of H for exponential correlations over $\mathbb{G}$ (Definition 6), the distribution of every y is indistinguishable from the uniform distribution over $\mathbb{G}$, hence Game $1.b$ and Game $2.b$ are indistinguishable.

Claim. Under the B-power-DDH assumption, the advantage of any adversary in distinguishing Game 2.0 and Game 2.1 is negligible.

Proof. We show how to turn an adversary against Game $2.b$ into an adversary with the same advantage against the B-power-DDH assumption. The reduction receives a sample $((h^{\alpha^i})_{i\in[\pm B]^*}, g_0)$ from the distribution $\mathcal{D}_b$ from Definition 3. The reduction sends $(h^{\alpha^i})_{i\in[\pm B]^*}$ to $\mathcal{A}$ in the setup phase. In the

challenge phase, it ignores the value z received from $\mathcal{A}$, samples $\mathsf{psk} \leftarrow\!\!\$\ \mathbb{Z}_p^*$, and computes $y := \mathsf{H}(g_0^{\mathsf{psk}})$. It sends (psk, y) to $\mathcal{A}$. Observe that the view of $\mathcal{A}$ is perfectly identical to Game 2.b. Upon receiving a guess b' from $\mathcal{A}$, the reduction outputs the guess $b = b'$; the advantage of the reduction against B-power-DDH is exactly that of $\mathcal{A}$ in distinguishing Game 2.0 and Game 2.1. This concludes the proof. ■

■

Remark 10 (Pseudorandomness). As noted in Remark 8, the standard definition of PPRFs typically requires that $\mathsf{F.Setup}, \mathsf{F.KeyGen}, \mathsf{F.Eval})$ is itself a pseudorandom function, and this property is not implied by the selective security game. We note that our candidate can straightforwardly be shown to be a PRF assuming the all-random power-DDH assumption (Definition 4), which is equivalent to the B-power-DDH assumption (see Theorem 5).

3.4 A Modified PPRF Procedure

In the previous section, we described the PPRF $(\mathsf{F.Setup}, \mathsf{F.KeyGen}, \mathsf{F.Eval}, \mathsf{F.Punct}, \mathsf{F.PEval})$ in a self-contained way. In this section, we introduce several modifications of the PPRF that facilitate its use within our modular garbling scheme. At a high level, we introduce two types of modifications:

- We change the procedures $(\mathsf{F.Setup}, \mathsf{F.KeyGen}, \mathsf{F.Punct})$ to generate pairs $(\mathsf{sk}, \mathsf{psk})$ (where psk is a key punctured at $z \in [B]$) from substractive shares of Δz where $\Delta \in \mathbb{Z}_{p-1}$ is a value such that $\alpha = G^{\Delta} \bmod p$ (for some generator G of $\mathbb{Z}_p$).
- We let H take an auxiliary salt salt as input. Looking ahead, we will use this salt to ensure independence across various events in our analysis. For simplicity, we keep the notation $\mathsf{F.Eval}^{\mathsf{H}}$ and $\mathsf{F.PEval}^{\mathsf{H}}$ and treat salt as an additional *optional* input (when salt is not passed as input, we assume that it is replaced with the dummy string 0^λ).

Let G be a generator of $\mathbb{Z}_p^*$ and let $\alpha = G^{\Delta} \bmod p$ for some $\Delta \in \mathbb{Z}_{p-1}$. Consider a secret key $\mathsf{sk} \leftarrow\!\!\$\ \mathsf{F.KeyGen}(\mathsf{mpk})$ and a punctured key $\mathsf{psk} := \mathsf{F.Punct}(\mathsf{msk}, \mathsf{sk}, z)$. Write $\mathsf{sk} = G^{\Gamma} \bmod p$. Then, by construction, we have $\mathsf{psk} = G^{\Delta z + \Gamma} \bmod p$. That is, the pair $(\mathsf{psk}, \mathsf{sk})$ can be viewed as a pair of substractive shares $(\Gamma_{\mathsf{G}}, \Gamma_{\mathsf{E}}) = (\Gamma, \Delta z + \Gamma)$ of Δz *lifted to the exponent of* G. We call Γ_{E} the *evaluator share* and Γ_{G} the *garbler share*. We make use of this observation by defining alternative PPRF algorithms $\mathsf{F.Setup}^*, \mathsf{F.KeyGen}^*, \mathsf{F.Punct}^*$ that generate secret keys sk and punctured secret keys psk respectively from shares of Δz. In the procedures below, $\mathsf{GrpGen}^*(1^\lambda)$ refers to an algorithm that behaves identically to $\mathsf{GrpGen}(1^\lambda)$ and additionally outputs a generator G of $\mathbb{Z}_p^*$.
In the procedure below, $(\Gamma_{\mathsf{G}}, \Gamma_{\mathsf{E}})$ are assumed to form subtractive shares of Δz over $\mathbb{Z}_{p-1}$ (where $z \in [B]$ is the point to be punctured), that is, $\Gamma_{\mathsf{E}} + \Gamma_{\mathsf{G}} = \Delta z \bmod p - 1$. When the garbler share Γ_{G} is uniformly distributed, these alternative procedures yield keys $(\mathsf{sk}, \mathsf{psk})$ distributed identically as with the procedures $\mathsf{F.KeyGen}$ and $\mathsf{F.Punct}$. The main advantage of these alternative procedures is that if the two parties, the garbler and the evaluator, happen to have shares $(\Gamma_{\mathsf{G}}, \Gamma_{\mathsf{E}})$ of Δz, then they can compute PPRF keys sk and psk respectively *without any interaction* simply by lifting their shares to the exponent of G.

$\mathsf{F.Setup}^*(1^\lambda, B)$

1 : $(\mathbb{G}, p, g, G) := \mathsf{GrpGen}^*(1^\lambda)$
2 : $(\Delta, h) \leftarrow\!\!\$\ \mathbb{Z}_{p-1} \times \mathbb{G}, \alpha := G^{\Delta} \bmod p$
3 : **for** $i \in [\pm B], g_i := h^{\alpha^i}$
4 : $\mathsf{mpk} := \left(\mathbb{G}, p, G, (g_i)_{i \in [\pm B]^*}\right)$
5 : $\mathsf{msk} := (\mathsf{mpk}, \Delta, g_0)$
6 : **return** $(\mathsf{mpk}, \mathsf{msk})$

$\mathsf{F.KeyGen}^*(\mathsf{mpk}, \Gamma_{\mathsf{G}})$

1 : **parse** p, G from mpk
2 : **return** $\mathsf{sk} = G^{\Gamma_{\mathsf{G}}} \bmod p$

$\mathsf{F.Punct}^*(\mathsf{mpk}, \Gamma_{\mathsf{E}})$

1 : **parse** p, G from mpk
2 : **return** $\mathsf{psk} = G^{\Gamma_{\mathsf{E}}} \bmod p$

$\mathsf{F.PEval}^{\mathsf{H}}(\mathsf{mpk}, \mathsf{psk}, z, x, \mathsf{salt})$

1 : **parse** $(g_i)_{i \neq 0}$ from mpk
2 : **return** $\mathsf{H}\left(g_{x-z}^{\mathsf{psk}}, \mathsf{salt}\right)$

$\mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt})$

1 : **parse** $(g_i)_{i \in [\pm B]}$ from msk
2 : **return** $\mathsf{H}\left(g_x^{\mathsf{sk}}, \mathsf{salt}\right)$

3.5 Security of the Modified PPRF

We show that the modified PPRF remains selectively secure (with reusable inputs) as long as Γ_{E} is statistically independent of Δ. Formally, we consider the following modified selective security experiment $\mathsf{ExpPPRF}^*(1^\lambda, B, b, \mathsf{salt})$ (for an arbitrary value of salt):

Setup Phase. Upon receiving a setup query from $\mathcal{A}$, the challenger samples, the challenger picks $(\mathbb{G}, p, g, G) := \mathsf{GrpGen}^*(1^\lambda)$ and samples $\Delta, h \leftarrow\$ \mathbb{Z}_{p-1} \times \mathbb{G}$. It sets $\alpha := G^\Delta$, $(g_i)_{i\in[\pm B]} := (h^{\alpha^i})_{i\in[\pm B]}$, and sends $\left(\mathbb{G}, p, (g_i)_{i\in[\pm B]^*}\right)$ to $\mathcal{A}$. Ignore all further setup queries from $\mathcal{A}$.

Challenge Phase. Upon receiving a query $(\mathsf{challenge}, z, \Gamma_{\mathsf{E}})$ with $z \in [B]$ and $\Gamma_{\mathsf{E}} \in \mathbb{Z}_{p-1}$ from $\mathcal{A}$, if $b = 0$, the challenger sets $y := \mathsf{H}(g_0^{G^{\Gamma_{\mathsf{E}}}}, \mathsf{salt})$; else, the challenger sets $y \leftarrow\$ \mathsf{Y}$. The challenger sends y to $\mathcal{A}$.

Definition 11 (selective security). *Fix $B = B(\lambda) \in \mathsf{poly}(\lambda)$. We say that the modified PPRF $(\mathsf{F.Setup}^*, \mathsf{F.KeyGen}^*, \mathsf{F.Eval}, \mathsf{F.Punct}^*, \mathsf{F.PEval})$ is* selectively secure with reusable setup *if for all large enough λ and every $\mathsf{salt} \in \mathsf{S}$, the advantage of any PPT adversary $\mathcal{A}$ against $\mathsf{ExpPPRF}^*(1^\lambda, B, b, \mathsf{salt})$ is negligible.*

Note that by defining $\mathsf{sk} := G^{\Gamma_{\mathsf{E}} - \Delta z}$, it holds that $\mathsf{H}(g_0^{G^{\Gamma_{\mathsf{E}}}}, \mathsf{salt}) = \mathsf{H}(g_z^{G^{\Gamma_{\mathsf{E}} - \Delta z}}, \mathsf{salt}) = \mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, z, \mathsf{salt})$, hence the case $b = 0$ corresponds to the correct evaluation of the PPRF at the punctured point z. Security follows via a sequence of game hops similar to the proof of Theorem 9 , setting g_0 to be either h if $b = 0$ or random if $b = 1$, embedding a B-power-DDH instance, and finally invoking the tweakable correlation-robustness of H for exponential correlations over $\mathbb{G}$ (where salt plays the role of the tweak; note that in this experiment, the tweak is the same for all queries). We omit the straightforward details.

We also consider a t-fold variant of the above experiment where the adversary received t PPRF evaluations y_i using different salts. Concretely, we define the t-instance selective security of the modified PPRF with the experiment $t\text{-}\mathsf{ExpPPRF}^*(1^\lambda, B, b, (\mathsf{salt}_i)_{i\le t})$ below:

Setup Phase. Same as before: upon receiving a setup query from $\mathcal{A}$, the challenger picks $(\mathbb{G}, p, g, G) := \mathsf{GrpGen}^*(1^\lambda)$ and samples $\Delta, h \leftarrow\$ \mathbb{Z}_{p-1} \times \mathbb{G}$. It sets $\alpha := G^\Delta$, $(g_i)_{i\in[\pm B]} := (h^{\alpha^i})_{i\in[\pm B]}$, and sends $\left(\mathbb{G}, p, (g_i)_{i\in[\pm B]^*}\right)$ to $\mathcal{A}$. Ignore all further setup queries from $\mathcal{A}$.

Challenge Phase. Upon receiving a query $(\mathsf{challenge}, z, \Gamma_{\mathsf{E}})$ with $z \in [B]$ and $\Gamma_{\mathsf{E}} \in \mathbb{Z}_{p-1}$ from $\mathcal{A}$, if $b = 0$, for $i = 1$ to t, the challenger sets $y_i := \mathsf{H}(g_0^{G^{\Gamma_{\mathsf{E}}}}, \mathsf{salt}_i)$; else, the challenger sets $y_i \leftarrow\$ \mathsf{Y}$. The challenger sends $(y_1, \cdots, y_t)$ to $\mathcal{A}$.

Definition 12 (t-instance selective security). *Fix $t = t(\lambda)$ and $B = B(\lambda) \in \mathsf{poly}(\lambda)$. We say that the modified PPRF $(\mathsf{F.Setup}^*, \mathsf{F.KeyGen}^*, \mathsf{F.Eval}, \mathsf{F.Punct}^*, \mathsf{F.PEval})$ is* t-instance selectively secure with reusable setup *if for all large enough λ and every t-tuple of pairwise distinct $(\mathsf{salt}_1, \cdots, \mathsf{salt}_t) \in \mathsf{S}^t$, the advantage of any PPT adversary $\mathcal{A}$ against $t\text{-}\mathsf{ExpPPRF}^*(1^\lambda, B, b, (\mathsf{salt}_i)_{i\le t})$ is negligible.*

The security analysis is again essentially identical to the analysis in the single-instance setting, up to using the salt as (distinct) tweaks for the TCR hash function.

4 Modular Garbling from Power-DDH

In this section, we introduce our modular arithmetic garbling scheme from the power-DDH assumption. Our construction builds upon multiple variants of a PPRF-based VOLE-to-OLE conversion protocol, which we introduce below.

4.1 VOLE to OLE Conversion

Fix a security parameter 1^λ and a modulus $B = B(\lambda) \in \mathsf{poly}(\lambda)$. Let $\mathsf{H} : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}$ be a tweakable correlation-robust hash function over $\mathbb{G}$, where $\mathbb{G}, p$ are parsed from $(\mathbb{G}, p, g, G) := \mathsf{GrpGen}^*$ (note that this is a deterministic algorithm).

Let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\$ \mathsf{F.Setup}^*(1^\lambda, B)$. Let $v_{\mathsf{G}} \in \mathbb{Z}_{p-1}$ denote an element of $\mathbb{Z}_{p-1}$ known to the garbler G $v_{\mathsf{E}} \in [B]$ denote a value known to the evaluator E. In this section, we introduce a single-message

deterministic conversion protocol, called VtO, that converts additive shares $(\langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ of Δv_E over $\mathbb{Z}_{p-1}$ (where $\Delta \in \mathbb{Z}_{p-1}$ is the second component of $\mathsf{msk} := (\mathsf{mpk}, \Delta, g_0)$) to $\mathbb{Z}_{p-1}$-shares $(\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E}) \in \mathbb{Z}_{p-1}^2$ of $v_\mathsf{G} v_\mathsf{E}$, using the PPRF with reusable inputs of the previous section.

$\mathsf{VtO}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt})$

$\mathsf{sk} := \mathsf{F.KeyGen}^*(\mathsf{mpk}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G})$
$(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}) := (-v_\mathsf{G}, 0)$
for $x \in [B]$
  $y_x := \mathsf{F.Eval}^\mathsf{H}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt})$
  $\mathsf{shift} := \mathsf{shift} + y_x \bmod p-1$
  $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G} := \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G} + x \cdot y_x \bmod p-1$
return $(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G})$

$\mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift})$

$\mathsf{psk} := \mathsf{F.Punct}^*(\mathsf{mpk}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$
$\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G} := -v_\mathsf{E} \cdot \mathsf{shift} \bmod p-1$
for $x \in [B] \setminus \{v_\mathsf{E}\}$
  $y_x := \mathsf{F.PEval}^\mathsf{H}(\mathsf{mpk}, \mathsf{psk}, v_\mathsf{E}, x, \mathsf{salt})$
  $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} + (v_\mathsf{E} - x) \cdot y_x \bmod p-1$
return $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E}$

Correctness and security. We require the above procedures to satisfy the following correctness and security properties:

Definition 13 (Correctness and security of VtO^H).

- Correctness. *For every* $(\mathsf{mpk}, \mathsf{msk})$ *in the support of* $\mathsf{F.Setup}^*(1^\lambda, B)$*, denoting* $\mathsf{msk} := (\mathsf{mpk}, \Delta, g_0)$*, for any* $v_\mathsf{G} \in \mathbb{Z}_{p-1}$*,* $v_\mathsf{E} \in [B]$*, any* $\langle \Delta v_\mathsf{E} \rangle_\mathsf{G} \in \mathbb{Z}_{p-1}$*, and any salt* salt*, denoting* $\langle \Delta v_\mathsf{E} \rangle_\mathsf{E} := \Delta v_\mathsf{E} - \langle \Delta v_\mathsf{E} \rangle_\mathsf{G} \bmod p-1$ *and*

$$(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}) := \mathsf{VtO}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt})$$
$$\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}, \mathsf{shift}),$$

  it holds that $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G} + \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} = v_\mathsf{G} v_\mathsf{E} \bmod p-1$.
- t-instance security. *For any* t*-tuple of pairwise distincts salts* $\mathbf{salt} = (\mathsf{salt}_1, \cdots, \mathsf{salt}_t)$*, the advantage of any stateful polynomial-time adversary* $\mathcal{A}$ *against the experiment* $\mathsf{ExpVtO}^\mathsf{H}$ *defined below is negligible.*

In the experiments below, we assume that $\mathcal{A}$ is deterministic; by a standard averaging argument, this is without loss of generality. At the end of the experiment, $\mathcal{A}$ outputs a guess b.

$\mathsf{ExpVtO}^\mathsf{H}_0(1^\lambda, B, \mathbf{salt})$

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$
$i := 1$
Upon receiving $(v_\mathsf{G}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk})$:
  $i := i + 1$; **if** $i > t$ **return** $\bot$
  $\langle \Delta v_\mathsf{E} \rangle_\mathsf{G} := \langle \Delta v_\mathsf{E} \rangle_\mathsf{E} - \Delta v_\mathsf{E} \bmod p-1$
  $(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}) := \mathsf{VtO}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}_i)$
  $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift})$
  return $(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E})$

$\mathsf{ExpVtO}^\mathsf{H}_1(1^\lambda, B, \mathbf{salt})$

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$
Upon receiving $(v_\mathsf{G}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk})$:
  $i := i + 1$; **if** $i > t$ **return** $\bot$
  $\mathsf{shift} \leftarrow\!\$\ \mathbb{Z}_{p-1}$
  $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift})$
  return $(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E})$

Correctness and Security of VtO^H.

Correctness. Let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$ with $\mathsf{msk} := (\mathsf{mpk}, \Delta, g_0)$, let $v_\mathsf{G}, v_\mathsf{E} \in [\pm B]$ and let

$(\langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ be two elements of $\mathbb{Z}_{p-1}$ such that $\langle \Delta v_\mathsf{E} \rangle_\mathsf{G} + \langle \Delta v_\mathsf{E} \rangle_\mathsf{E} = \Delta v_\mathsf{E} \bmod p-1$. Consider the outputs $(\mathsf{msg}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G})$ of $\mathsf{VtO}^\mathsf{H}_\mathsf{G}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G})$ and $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E}$ of $\mathsf{VtO}^\mathsf{H}_\mathsf{G}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{msg})$, and parse $\mathsf{msg} := (\mathsf{shift}, \mathsf{salt})$. We have:

$$\begin{aligned}
\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} &= -v_{\mathsf{E}} \cdot \mathsf{shift} + \sum_{x \in [\pm B] \setminus \{v_{\mathsf{E}}\}} (v_{\mathsf{E}} - x) \cdot \mathsf{F.PEval}^{\mathsf{H}}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, x, \mathsf{salt}) \bmod p - 1 \\
&= -v_{\mathsf{E}} \cdot \mathsf{shift} + \sum_{x \in [\pm B] \setminus \{v_{\mathsf{E}}\}} (v_{\mathsf{E}} - x) \cdot \mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt}) \bmod p - 1 \text{ by PPRF security} \\
&= -v_{\mathsf{E}} \cdot \mathsf{shift} + \sum_{x \in [\pm B]} (v_{\mathsf{E}} - x) \cdot \mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt}) \bmod p - 1 \text{ as } v_{\mathsf{E}} - v_{\mathsf{E}} = 0 \\
&= -v_{\mathsf{E}} \cdot \mathsf{shift} + v_{\mathsf{E}} \cdot \sum_{x \in [\pm B]} \mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt}) - \sum_{x \in [\pm B]} x \cdot \mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt}) \bmod p - 1 \\
&= -v_{\mathsf{E}} \cdot \mathsf{shift} + v_{\mathsf{E}} \cdot (\mathsf{shift} + v_{\mathsf{G}}) - \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} \bmod p - 1 \\
&= v_{\mathsf{G}} v_{\mathsf{E}} - \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} \bmod p - 1.
\end{aligned}$$

Security. Fix t pairwise distinct salts salt. We argue security through a sequence of game hops:

Game 0. This is the experiment $\mathsf{ExpVtO}_0^{\mathsf{H}}(1^\lambda, B, \mathbf{salt})$.

Game 1. In this game, we modify the algorithm $\mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt}_i)$:
- On line 1 we additionally compute $\mathsf{psk} \coloneqq \mathsf{F.Punct}^*(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$.
- On line 4 we instead compute y_x as $y_x \coloneqq \mathsf{F.PEval}^{\mathsf{H}}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, x, \mathsf{salt}_i)$ for every $x \neq v_{\mathsf{E}}$. For $x = v_{\mathsf{E}}$, y_x is still computed as $\mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt}_i)$.

By (perfect) correctness of the PPRF, this game is perfectly indistinguishable from the previous one.

Game 2. In this game, we modify again $\mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt}_i)$ for $i = 1$ to t: on line 4, we set instead $y_{v_{\mathsf{E}}} \leftarrow\!\!\$\ \mathbb{Z}_{p-1}$. Under the t-instance selective security of the PPRF (Definition 12), this game is indistinguishable from the previous one. Note that the value shift output by the modified $\mathsf{VtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt}_i)$ satisfies:

$$\mathsf{shift} = y_{v_{\mathsf{E}}} + \sum_{x \neq v_{\mathsf{E}}} y_x - v_{\mathsf{G}} \bmod p - 1.$$

Since all $y_{v_{\mathsf{E}}}$ for $i = 1$ to t are sampled uniformly at random (independently of v_{G} and $(y_x)_{x \neq 0}$), it follows that the marginal distribution of all t shifts shift is the uniform distribution over $(\mathbb{Z}_{p-1})^t$.

Game 3. This is the experiment $\mathsf{ExpVtO}_1^{\mathsf{H}}(1^\lambda, B, \mathbf{salt})$. The difference with the previous game is that shift is sampled uniformly from $\mathbb{Z}_{p-1}$ for $i = 1$ to t. As the marginal distribution of all shifts in Game 2 is already uniform over $(\mathbb{Z}_{p-1})^t$, this is a purely syntactic change and this game is perfectly indistinguishable from Game 2. This concludes the proof.

■

4.2 Modular reduction of authenticated shares

We observe that the correctness and security analysis above remains identical if the parties replace y_x with $f(y_x)$, for an arbitrary function f, in the $\mathsf{VtO}^{\mathsf{H}}$ procedures. Concretely, consider the following "functional" variants where $f : \mathbb{Z}_{p-1} \to \mathbb{Z}_{p-1}$ is an arbitrary efficient function:

$\mathsf{fVtO}_{\mathsf{G}}^{\mathsf{H}}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt}, f)$

1 : $\mathsf{sk} \coloneqq \mathsf{F.KeyGen}^*(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}})$
2 : $(\mathsf{shift}, \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}}) \coloneqq (-v_{\mathsf{G}}, 0)$
3 : **for** $x \in [B]$
4 : $\quad y_x \coloneqq \mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt})$
5 : $\quad \mathsf{shift} \coloneqq \mathsf{shift} + y_x \bmod p - 1$
6 : $\quad \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} \coloneqq \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} + f(x) \cdot y_x \bmod p - 1$
7 : **return** $(\mathsf{shift}, \langle v_{\mathsf{G}} \cdot f(v_{\mathsf{E}}) \rangle_{\mathsf{G}})$

$\mathsf{fVtO}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}, \mathsf{salt}, \mathsf{shift}, f)$

1 : $\mathsf{psk} \coloneqq \mathsf{F.Punct}^*(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$
2 : $\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} \coloneqq -f(v_{\mathsf{E}}) \cdot \mathsf{shift} \bmod p - 1$
3 : **for** $x \in [B] \setminus \{v_{\mathsf{E}}\}$
4 : $\quad y_x \coloneqq \mathsf{F.PEval}^{\mathsf{H}}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, x, \mathsf{salt})$
5 : $\quad \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} \coloneqq \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} + (f(v_{\mathsf{E}}) - f(x)) \cdot y_x \bmod p - 1$
6 : **return** $\langle v_{\mathsf{G}} \cdot f(v_{\mathsf{E}}) \rangle_{\mathsf{E}}$

The same correctness analysis as before shows that with probability 1, it holds that $\langle v_{\mathsf{G}} \cdot f(v_{\mathsf{E}}) \rangle_{\mathsf{G}} + \langle v_{\mathsf{G}} \cdot f(v_{\mathsf{E}}) \rangle_{\mathsf{E}} = v_{\mathsf{G}} \cdot f(v_{\mathsf{E}}) \bmod p - 1$. We use this observation below to define a procedure that lets two parties compute modular reductions inside authenticated shares.

The MR procedure. Fix a security parameter 1^λ and two moduli $B = B(\lambda) \in \mathsf{poly}(\lambda)$ and $B' = B'(\lambda) \in \mathsf{poly}(\lambda)$ with $B' > B$. Let $\mathsf{H} : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_{p-1}$ be a TCR hash function for exponential correlations over $\mathbb{G}$. Let $(\mathsf{mpk}', \mathsf{msk}') \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B')$ and parse $(\mathsf{mpk}', \Delta', g_0) := \mathsf{msk}'$. Let $\tilde{v}_\mathsf{E} \in [B']$ denote a value known to the evaluator E. In this section, we introduce one more procedure to convert, using our PPRF and via a single-message deterministic protocol called MR (for Modular Reduction), $\mathbb{Z}_{p-1}$-shares $(\langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{G}, \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E})$ of $\Delta' \tilde{v}_\mathsf{E}$ to $\mathbb{Z}_{p-1}$-shares $(\langle \Delta'' v_\mathsf{E} \rangle_\mathsf{G}, \langle \Delta'' v_\mathsf{E} \rangle_\mathsf{E})$ of the value $\Delta'' v_\mathsf{E} = \Delta'' \cdot [\tilde{v}_\mathsf{E} \bmod B]$. That is, $\langle \Delta'' v_\mathsf{E} \rangle_\mathsf{G} + \langle \Delta'' v_\mathsf{E} \rangle_\mathsf{E} = \Delta'' \cdot [\tilde{v}_\mathsf{E} \bmod B] \bmod p - 1$. Looking ahead, Δ'' will be set in our protocol to be the "Δ component" of another PPRF secret key msk''.

$\mathsf{MR}^\mathsf{H}_\mathsf{G}(\mathsf{msk}', \Delta'', \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}, B)$

1 : **parse** $(\mathsf{mpk}', \Delta', g_0) := \mathsf{msk}'$
2 : **define** $f : x \mapsto [x \bmod B]$
3 : $(\mathsf{shift}, \langle \Delta'' \cdot v_\mathsf{E} \rangle_\mathsf{G}) := \mathsf{fVtO}^\mathsf{H}_\mathsf{G}(\mathsf{msk}', \Delta', \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}, f)$
4 : **return** $(\mathsf{shift}, \langle v_\mathsf{G} \cdot v_\mathsf{E} \rangle_\mathsf{G})$

$\mathsf{MR}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}', \tilde{v}_\mathsf{E}, \langle \Delta' v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}, \mathsf{shift}, B)$

1 : **define** $f : x \mapsto [x \bmod B]$
2 : $\langle v_\mathsf{G} \cdot v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{fVtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}', \tilde{v}_\mathsf{E}, \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}, f)$
3 : **return** $\langle \Delta'' \cdot v_\mathsf{E} \rangle_\mathsf{E}$

The above procedure lets two parties compute, given $(\langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{G}, \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E})$, shares of $\Delta'' \cdot [\tilde{v}_\mathsf{E} \bmod B] \bmod p - 1$ using the functional VOLE-to-OLE procedure, where G sets its OLE input v_G to Δ'' and the parties use the modular reduction function $f : x \mapsto [x \bmod B]$. Note that we run the setup $(\mathsf{mpk}', \mathsf{msk}') \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B')$ using a larger modulus $B' > B$. Looking ahead, we use this procedure to shrink back labels in our garbling scheme to values in $[B]$ after their size has been increased (but remains bounded by B') through computations over the integers.

Security. We define security using a slight modification of the ExpVtO to account for the function f and for the fact that we fix v_G to Δ (hence we don't let the adversary choose it).

$\mathsf{ExpMR}^\mathsf{H}_0(1^\lambda, B', B, \mathbf{salt})$

1 : $(\mathsf{mpk}', \mathsf{msk}') \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B')$
2 : $i := 1$
3 : Upon receiving $(\Delta'', \tilde{v}_\mathsf{E}, \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk}')$:
4 : $\quad i := i + 1$; **if** $i > t$ **return** $\bot$
5 : $\quad \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{G} := \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E} - \Delta' \tilde{v}_\mathsf{E} \bmod p - 1$
6 : $\quad (\mathsf{shift}, \langle \Delta'' v_\mathsf{E} \rangle_\mathsf{G}) := \mathsf{MR}^\mathsf{H}_\mathsf{G}(\mathsf{msk}', \Delta'', \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}_i, B)$
7 : $\quad \langle \Delta'' v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{MR}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}', \tilde{v}_\mathsf{E}, \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E}, \mathsf{shift})$
8 : $\quad$ **return** $(\mathsf{shift}, \langle \Delta'' v_\mathsf{E} \rangle_\mathsf{E})$

$\mathsf{ExpMR}^\mathsf{H}_1(1^\lambda, B', B, \mathbf{salt})$

1 : $(\mathsf{mpk}', \mathsf{msk}') \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B')$
2 : $i := 1$
3 : Upon receiving $(\Delta'', \tilde{v}_\mathsf{E}, \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk}')$:
4 : $\quad i := i + 1$; **if** $i > t$ **return** $\bot$
5 : $\quad \mathsf{shift} \leftarrow\!\$\ \mathbb{Z}_{p-1}$
6 : $\quad \langle \Delta'' v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{MR}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}', \tilde{v}_\mathsf{E}, \langle \Delta' \tilde{v}_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift}, B)$
7 : $\quad$ **return** $(\mathsf{shift}, \langle \Delta'' v_\mathsf{E} \rangle_\mathsf{E})$

Definition 14 (Security of MR). *We say that* MR *is* t-instance secure *if for any $B < B'$ and any t-tuple of pairwise distinct salts* $\mathbf{salt}$*, the advantage of any stateful polytime adversary $\mathcal{A}$ against the experiment* $\mathsf{ExpMR}^\mathsf{H}$ *defined above is negligible.*

The security analysis is an immediate adaptation of the security analysis of ExpVtO and we omit it.

4.3 Leaky VOLE to OLE Conversion

We introduce another variant of the VtO procedure, called LVtO, with the following characteristics:

- LVtO is a single-message conversion protocol where the garbler holds an input $v_\mathsf{G} \in [B]$. LVtO converts additive shares of Δv_E (over $\mathbb{Z}_{p-1}$) into additive shares of $v_\mathsf{G} v_\mathsf{E}$ *over the integers.*
- The LVtO procedure produces *small integer shares*: the output shares $(\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E})$ are guaranteed to belong to $[\lambda B^3]$.
- This combination of sharing over the integers and achieving a small share size comes at a cost in security: with probability $1/\lambda$, the procedure LVtO leaks information about v_G to the evaluator (in a sense that will be made precise).

Fix a security parameter 1^λ and a modulus $B = B(\lambda) \in \mathsf{poly}(\lambda)$. Let $\mathsf{H}' : \mathbb{G} \times \{0,1\}^\lambda \to [\lambda \cdot B]$ be a tweakable correlation-robust hash function for exponential correlations over $\mathbb{G}$. Let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$. Let $v_\mathsf{G}, v_\mathsf{E} \in [B]$ denote two values known to the garbler G and the evaluator E respectively.

$\mathsf{LVtO}_{\mathsf{G}}^{\mathsf{H}'}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt})$

1 : $\mathsf{sk} \coloneqq \mathsf{F.KeyGen}^*(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}})$
2 : $(\mathsf{shift}, \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}}) \coloneqq (-v_{\mathsf{G}}, 0)$
3 : **for** $x \in [B]$
4 : $\quad y_x \coloneqq \mathsf{F.Eval}^{\mathsf{H}'}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt})$
5 : $\quad \mathsf{shift} \coloneqq \mathsf{shift} + y_x$
6 : $\quad \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} \coloneqq \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} + x \cdot y_x$
7 : **return** $(\mathsf{shift}, \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}})$

$\mathsf{LVtO}_{\mathsf{E}}^{\mathsf{H}'}(\mathsf{mpk}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}, \mathsf{salt}, \mathsf{shift})$

1 : $\mathsf{psk} \coloneqq \mathsf{F.Punct}^*(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$
2 : $\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} \coloneqq -v_{\mathsf{E}} \cdot \mathsf{shift}$
3 : **for** $x \in [B] \setminus \{v_{\mathsf{E}}\}$
4 : $\quad y_x \coloneqq \mathsf{F.PEval}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, x, \mathsf{salt})$
5 : $\quad \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} \coloneqq \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} + (v_{\mathsf{E}} - x) \cdot y_x$
6 : **return** $\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}}$

Correctness and efficiency. We define *correctness* as for VtO. The same analysis as in Section 4.1 shows that $\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} + \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} = v_{\mathsf{G}} v_{\mathsf{E}}$ (over the integers) with probability 1. In the procedures above, it holds that $y_x \in [\lambda \cdot B]$ for every $x \in [B]$. Since $\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} = \sum_{x \in [B]} x \cdot y_x$ and $\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} = \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}} - v_{\mathsf{G}} v_{\mathsf{E}}$, we have $|\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}}| \leq \lambda \cdot B^2(B-1)/2 \leq \lambda \cdot B^3/2$, and $|\langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}}| \leq \lambda B^2(B-1)/2 + B^2 \leq \lambda \cdot B^3/2$.

Security. To define *multi-instance security*, we introduce below a modified experiment $\mathsf{ExpLVtO}$. Compared to ExpVtO_1, the simulated experiment $\mathsf{ExpLVtO}_1$ tosses a biased coin $\sigma \leftarrow\!\!\$\ \mathsf{Ber}(1/\lambda)$. With probability $1 - 1/\lambda$, it samples shift independently of v_{G} (no leakage occurred). With probability $1/\lambda$, it samples shift from a distribution that depends on v_{G} (a leakage occurred).
The distribution of the simulated shifts in $\mathsf{ExpLVtO}_1$ are a bit more involved compared to ExpVtO_1 (where shift is simply simulated as $\mathsf{shift} \leftarrow\!\!\$\ \mathbb{Z}_{p-1}$). For convenience, we define two algorithms $\mathsf{SimShift}$ and $\mathsf{SimLShift}$ that simulate shift in the case where no leakage occurs ($\sigma = 0$) and where leakage occurs ($\sigma = 1$) respectively:

$\mathsf{SimShift}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, \mathsf{salt})$

1 : $\mathsf{shift} \leftarrow\!\!\$\ [(\lambda - 1) \cdot B]$
2 : **for** $x \in [B] \setminus \{v_{\mathsf{E}}\}$
3 : $\quad \mathsf{shift} \coloneqq \mathsf{shift} + \mathsf{F.PEval}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, x, \mathsf{salt})$
4 : **return** shift

$\mathsf{SimLShift}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{G}}, v_{\mathsf{E}}, \mathsf{salt})$

1 : $\mathsf{shift} \leftarrow\!\!\$\ [-v_{\mathsf{G}}, \lambda B - v_{\mathsf{G}}] \setminus [(\lambda - 1) \cdot B]$
2 : **for** $x \in [B] \setminus \{v_{\mathsf{E}}\}$
3 : $\quad \mathsf{shift} \coloneqq \mathsf{shift} + \mathsf{F.PEval}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, x, \mathsf{salt})$
4 : **return** shift

Definition 15 (t-instance security of $\mathsf{LVtO}^{\mathsf{H}'}$). *For all large enough λ, all $v_{\mathsf{E}} \in [B]$ and pairwise-distinct t-tuple of salts $\mathbf{salt} = (\mathsf{salt}_1, \cdots, \mathsf{salt}_t)$, the advantage of any polynomial-time stateful adversary $\mathcal{A}$ against $\mathsf{ExpLVtO}^{\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$ is negligible.*

$\mathsf{ExpLVtO}_0^{\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$

1 : $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$
2 : $i \coloneqq 1$
3 : Upon receiving $(v_{\mathsf{G}}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$ from $\mathcal{A}(\mathsf{mpk})$:
4 : $\quad i \coloneqq i + 1$; **if** $i > t$ **return** $\bot$
5 : $\quad \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}} \coloneqq \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}} - \Delta v_{\mathsf{E}} \bmod p - 1$
6 : $\quad (\mathsf{shift}, \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{G}}) \coloneqq \mathsf{LVtO}_{\mathsf{G}}^{\mathsf{H}'}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt}_i)$
7 : $\quad \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}} \coloneqq \mathsf{LVtO}_{\mathsf{E}}^{\mathsf{H}'}(\mathsf{mpk}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}, \mathsf{salt}_i, \mathsf{shift})$
8 : $\quad$ **return** $(\mathsf{shift}, \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}})$

$\mathsf{ExpLVtO}_1^{\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$

1 : $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$
2 : $i \coloneqq 1$
3 : Upon receiving $(v_{\mathsf{G}}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$ from $\mathcal{A}(\mathsf{mpk})$:
4 : $\quad i \coloneqq i + 1$; **if** $i > t$ **return** $\bot$
5 : $\quad \sigma_i \leftarrow\!\!\$\ \mathsf{Ber}(1/\lambda)^t$
6 : $\quad \mathsf{shift}_0 \leftarrow\!\!\$\ \mathsf{SimShift}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{E}}, \mathsf{salt}_i)$
7 : $\quad \mathsf{shift}_1 \leftarrow\!\!\$\ \mathsf{SimLShift}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_{\mathsf{G}}, v_{\mathsf{E}}, \mathsf{salt}_i)$
8 : $\quad \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}}^i \coloneqq \mathsf{LVtO}_{\mathsf{E}}^{\mathsf{H}'}(\mathsf{mpk}, v_{\mathsf{E}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}}, \mathsf{salt}_i, \mathsf{shift})$
9 : $\quad$ **return** $(\mathsf{shift}_{\sigma_i}, \langle v_{\mathsf{G}} v_{\mathsf{E}} \rangle_{\mathsf{E}})$

The proof of correctness is identical to the previous section; we argue security below:

Proof of Security of $\mathsf{LVtO}^{\mathsf{H}'}$, Section 4.3. Fix any t-tuple of salts $\mathbf{salt}$.

Game 0. This is the experiment $\mathsf{ExpLVtO}_0^{\mathsf{H}}(1^\lambda, B, \mathbf{salt})$.
Game 1. In this game, we modify the algorithm $\mathsf{LVtO}_{\mathsf{G}}^{\mathsf{H}'}(\mathsf{msk}, v_{\mathsf{G}}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{G}}, \mathsf{salt}_i)$:
– On line 1 we additionally compute $\mathsf{psk} \coloneqq \mathsf{F.Punct}^*(\mathsf{mpk}, \langle \Delta v_{\mathsf{E}} \rangle_{\mathsf{E}})$.

– On line 4, for $i = 1$ to t, we instead compute y_x as $y_x := \mathsf{F.PEval}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_\mathsf{E}, x, \mathsf{salt}_i)$ for every $x \neq v_\mathsf{E}$. For $x = v_\mathsf{E}$, y_x is still computed as $\mathsf{F.Eval}^{\mathsf{H}}(\mathsf{msk}, \mathsf{sk}, x, \mathsf{salt}_i)$.

By (perfect) correctness of the PPRF, this game is perfectly indistinguishable from the previous one.

Game 2. In this game, we further modify $\mathsf{LVtO}_\mathsf{G}^{\mathsf{H}'}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}_i)$ for $i = 1$ to t: on line 4, we set instead $y_{v_\mathsf{E}} \leftarrow\!\$\ [\lambda \cdot B]$. Under the t-instance selective security of the PPRF (Definition 12), this game is indistinguishable from the previous one. Note that the value shift output by the modified $\mathsf{LVtO}_\mathsf{G}^{\mathsf{H}'}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}_i)$ satisfies:

$$\mathsf{shift} = y_{v\mathsf{E}} - v_\mathsf{G} + \sum_{x \neq v_\mathsf{E}} y_x.$$

In this equation, y_{v_E} is sampled uniformly at random from $[\lambda \cdot B]$ (independently for all iterations $i = 1$ to t). Equivalently, $\mathsf{shift} = z + \sum_{x \neq v_\mathsf{E}} y_x$ where $z := y_{v_\mathsf{E}} - v_\mathsf{G}$ is sampled uniformly at random from $[-v_\mathsf{G}, \lambda \cdot B - v_\mathsf{G}]$. Before we proceed, we prove a simple claim:

Claim. $\Pr[z \notin [(\lambda - 1) \cdot B]] = 1/\lambda$.

Proof. $\Pr[z \notin [(\lambda-1)\cdot B]] = \Pr[z \in [-v_\mathsf{G}, 0)] + \Pr[z \in ((\lambda-1)B, \lambda B - v_\mathsf{G}]] = \frac{v_\mathsf{G}}{\lambda B} + \frac{B - v_\mathsf{G}}{\lambda B} = \frac{1}{\lambda}$. ■

Game 3. In this experiment, upon receiving each query from $\mathcal{A}$ for $i = 1$ to t, we sample z differently: we first pick $\sigma_i \leftarrow\!\$\ \mathsf{Ber}(1/\lambda)$. If $\sigma_i = 0$ (which happens with probability $1 - 1/\lambda$), we sample $z \leftarrow\!\$\ [(\lambda-1)\cdot B]]$. Else, ($\sigma_i = 1$, probability $1/\lambda$) we sample $z \leftarrow\!\$\ [-v_\mathsf{G}, \lambda B - v_\mathsf{G}] \setminus [(\lambda-1)\cdot B]]$. Note that by the claim above, this induces exactly the same distribution as sampling $z \leftarrow\!\$\ [-v_\mathsf{G}, \lambda B - v_\mathsf{G}]$ (where a leakage occurs if $z \notin [(\lambda - 1) \cdot B]]$), hence this game is perfectly indistinguishable from Game 2. This game corresponds exactly to $\mathsf{ExpLVtO}_0^{\mathsf{H}}(1^\lambda, B, v_\mathsf{E}, \mathbf{salt})$; this concludes the proof.

■

Remark 16. Multi-instance security allows running multiple LVtO instances in parallel using the same pair $(\mathsf{mpk}, \mathsf{msk})$. We note that the same analysis of multi-instance security immediately extends to show that security is preserved when running concurrently both VtO and LVtO instances using the same pair $(\mathsf{mpk}, \mathsf{msk})$. We state below the corresponding experiment (denoted $\mathsf{ExpBothVtO}$) for completeness and introduce the corresponding security notion in Definition 17, but omit the proof, which remains essentially identical.

Definition 17 (t-instance joint security of $\mathsf{VtO}^{\mathsf{H}}, \mathsf{LVtO}^{\mathsf{H}'}$). *For all large enough λ, all $v_\mathsf{E} \in [B]$ and pairwise-distinct t-tuple of salts $\mathbf{salt} = (\mathsf{salt}_1, \cdots, \mathsf{salt}_t)$, the advantage of any polynomial-time stateful adversary $\mathcal{A}$ against $\mathsf{ExpBothVtO}^{\mathsf{H},\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$ is negligible.*

$\mathsf{ExpBothVtO}_0^{\mathsf{H},\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$

1 : $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$
2 : $i := 1$
3 : Upon receiving $(\mathsf{VtO}, v_\mathsf{G}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk})$:
4 : $\quad i := i + 1$; **if** $i > t$ **return** $\perp$
5 : $\quad \langle \Delta v_\mathsf{E} \rangle_\mathsf{G} := \langle \Delta v_\mathsf{E} \rangle_\mathsf{E} - \Delta v_\mathsf{E} \bmod p - 1$
6 : $\quad (\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}) := \mathsf{VtO}_\mathsf{G}^{\mathsf{H}}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}_i)$
7 : $\quad \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{VtO}_\mathsf{E}^{\mathsf{H}}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift})$
8 : $\quad$ **return** $(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E})$
9 : Upon receiving $(\mathsf{LVtO}, v_\mathsf{G}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk})$:
10 : $\quad i := i + 1$; **if** $i > t$ **return** $\perp$
11 : $\quad \langle \Delta v_\mathsf{E} \rangle_\mathsf{G} := \langle \Delta v_\mathsf{E} \rangle_\mathsf{E} - \Delta v_\mathsf{E} \bmod p - 1$
12 : $\quad (\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}) := \mathsf{LVtO}_\mathsf{G}^{\mathsf{H}'}(\mathsf{msk}, v_\mathsf{G}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{G}, \mathsf{salt}_i)$
13 : $\quad \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{LVtO}_\mathsf{E}^{\mathsf{H}'}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift})$
14 : $\quad$ **return** $(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E})$

$\mathsf{ExpBothVtO}_1^{\mathsf{H},\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$

1 : $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\$\ \mathsf{F.Setup}^*(1^\lambda, B)$
2 : $i := 1$
3 : Upon receiving $(\mathsf{VtO}, v_\mathsf{G}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk})$:
4 : $\quad i := i + 1$; **if** $i > t$ **return** $\perp$
5 : $\quad \mathsf{shift} \leftarrow\!\$\ \mathbb{Z}_{p-1}$
6 : $\quad \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{VtO}_\mathsf{E}^{\mathsf{H}}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift})$
7 : $\quad$ **return** $(\mathsf{shift}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E})$
8 : Upon receiving $(\mathsf{LVtO}, v_\mathsf{G}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E})$ from $\mathcal{A}(\mathsf{mpk})$:
9 : $\quad i := i + 1$; **if** $i > t$ **return** $\perp$
10 : $\quad \sigma_i \leftarrow\!\$\ \mathsf{Ber}(1/\lambda)^t$
11 : $\quad \mathsf{shift}_0 \leftarrow\!\$\ \mathsf{SimShift}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_\mathsf{E}, \mathsf{salt}_i)$
12 : $\quad \mathsf{shift}_1 \leftarrow\!\$\ \mathsf{SimLShift}^{\mathsf{H}'}(\mathsf{mpk}, \mathsf{psk}, v_\mathsf{G}, v_\mathsf{E}, \mathsf{salt}_i)$
13 : $\quad \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E}^i := \mathsf{LVtO}_\mathsf{E}^{\mathsf{H}'}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift})$
14 : $\quad$ **return** $(\mathsf{shift}_{\sigma_i}, \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E})$

4.4 The Garbling Scheme

Equipped with the procedures introduced in the previous sections, we are now ready to present our garbling scheme for modular arithmetic circuit. Let λ be a security parameter and $B = B(\lambda) \in \mathsf{poly}(\lambda)$ be a polynomial-size modulus. We consider a polynomial-size arithmetic circuit C over $\mathbb{Z}_B$ with n inputs, m outputs, s gates (additions, substractions, and multiplications), of depth D. We let $I(C)$ denote the set of input wires, $W(C)$ denote the set of all wires, $O(C)$ denote the set of output wires, and $\Gamma(C)$ denote the set of gates. We write $|C|$ to denote the size of C (the number of gates in C), and $\mathsf{depth}(C)$ to denote its depth (the length of the longest path from an input to an output).

Theorem 18. *Let $B = B(\lambda) \in \mathsf{poly}(\lambda)$ be a polynomial-size modulus. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ denote the class of all polynomial-size arithmetic circuits over the ring $\mathbb{Z}_B$. Let $c = c(\lambda)$ denote a function such that $c = \omega(1)$, and set $B' := B^2 + c \cdot \lambda \cdot B^3$. Assume that the B'-power-DDH assumption holds over a group $(\mathbb{G}, p, g) := \mathsf{GrpGen}(1^\lambda)$ of order $p = 2^{O(\lambda)}$ and that H is a tweakable correlation-robust hash function for exponential correlations over $\mathbb{G}$. Then there exists an arithmetic garbling scheme $\mathsf{AGC} = (\mathsf{AGC.Garble}, \mathsf{AGC.Eval})$ for $\mathcal{C}$ where, on input $(1^\lambda, C)$ with $C \in \mathcal{C}_\lambda$, $\mathsf{AGC.Garble}$ outputs a garbled circuit $\hat{C}$ of size*

$$|\hat{C}| = O(\lambda \cdot |C| + \lambda B' \cdot \mathsf{depth}(C)).$$

In particular, the rate of the scheme approaches $O(\log B/\lambda)$ when $\mathsf{depth}(C) \ll |C|$.

More concretely, the garbled circuit $\hat{C}$ contains a "header" of size $(B + B') \cdot \log p \cdot \mathsf{depth}(C)$ and garbled gates $\hat{C}_\gamma$, where $\hat{C}_\gamma = 3 \log p$ for addition/substraction gates, and $\hat{C}_\gamma \le 3 \log p + 2 \cdot (\log \lambda + 3 \log B + 2 \log p + 2 \log c)$ for multiplication gates. The result of Theorem 18 represents a logarithmic improvement in rate compared with the state of the art: our scheme achieves asymptotic rate $\Omega(\log \lambda)/\lambda$, while the best known arithmetic garbling scheme for modular arithmetic is the scheme of [Hea24], which achieves asymptotic rate $O(1/\lambda)$.

Remark 19. The dependency in $\mathsf{depth}(C)$ in Theorem 18 stems from the fact that our construction is *leveled*: $\mathsf{AGC.Garble}$ samples fresh PPRF master keys mpk_d at each level $d \le \mathsf{depth}(C)$. This leveled construction allows us to avoid circularity, by ensuring that PPRF evaluations at a given level d are always used to mask values that depend on $\mathsf{msk}_{d'}$ for $d' > d$. The dependency can be removed at the cost of assuming a non-standard circular variant of tweakable correlation-robustness. Concretely, fix a hash H. Let $(g_i)_{i\in[\pm B']} \leftarrow\!\$\ \mathcal{D}_0$, where $\mathcal{D}_0$ is the power-DDH distribution of Definition 3 with bound B', and let $(\alpha, h) \in \mathbb{Z}_p^* \times \mathbb{G}$ be such that $g_i = h^{\alpha^i}$ for all $i \in [\pm B']$. Then the "circular B'-power-DDH assumption" states that no PPT adversary $\mathcal{A}$ (who receives the g_i for $i \neq 0$ as input) can, given at most t samples of the form $\mathsf{H}(g_0^u, \mathsf{salt}) + \alpha v$, for queries (u, v, salt) of his choice with distinct salts, distinguish the samples from random. In a follow-up work [CHHK25], we showed that this assumption holds unconditionally in the generic group model when H is additionally modeled as a random oracle. Under this assumption, the $2\mathsf{depth}(C)$ different Δ's in our construction can be replaced with a single global Δ.

Garbling algorithm We represent on Algorithm 1 the garbling algorithm $\mathsf{AGC.Garble}(1^\lambda, C)$. The garbling algorithm maintains the following invariant: for every wire w at depth d of C, the garbler constructs a wire key $(k_w, K_w) \in \mathbb{Z}_B \times \mathbb{Z}_{p-1}$. $\mathsf{AGC.Garble}$ outputs a global keys Δ_1, a set of wire keys $\hat{K} = (k_w, K_w)_{w\in W(C)}$, and a garbled circuit $\hat{C}$. Given a wire value $x_w \in \mathbb{Z}_B$ at depth d, the corresponding wire label is defined as $(\ell_w, L_w) := (x_w - k_w \bmod B, \Delta_d \cdot \ell_w - K_w \bmod p - 1)$.

Definition 20. *We let $\mathsf{InputLabels}$ denote the function which, on input $(1^\lambda, C, \Delta_1, \hat{K}, x)$, outputs $(\ell_i, L_i) := (x_i - k_i \bmod B, \Delta_1 \cdot x_i - K_i \bmod p - 1)_{i\in I(C)}$.*

Algorithm Modular Garbling from Power-DDH: $\mathsf{AGC.Garble}(1^\lambda, C)$

Parameters. Let $(\mathbb{G}, p, g, G) := \mathsf{GrpGen}^*(1^\lambda)$. Let $c = c(\lambda)$ denote a functions $c = \omega(1)$, and set $B' := B^2 + c \cdot \lambda \cdot B^3$. Let $\mathsf{H} : \mathbb{G} \to \mathbb{Z}_{p-1}$ and $\mathsf{H}' : \mathbb{G} \times \{0,1\}^* \to [\lambda \cdot B]$ denote TCR hash functions for exponential correlations over $\mathbb{G}$. Let $\mathsf{F} = (\mathsf{F.Setup}^*, \mathsf{F.KeyGen}^*, \mathsf{F.Eval}, \mathsf{F.Punct}^*, \mathsf{F.PEval})$ denote the modified PPRF of Section 3.4.

Initialization. For $d = 1$ to D,
- sample $(\mathsf{mpk}_d, \mathsf{msk}_d) \leftarrow\!\$\ \mathsf{F.Setup}(1^\lambda, B)$. Parse $\mathsf{msk}_d := (\mathsf{mpk}_d, \Delta_d, g_{0,d})$.
- Sample $(\mathsf{mpk}'_d, \mathsf{msk}'_d) \leftarrow\!\$\ \mathsf{F.Setup}(1^\lambda, B')$. Parse $\mathsf{msk}'_d := (\mathsf{mpk}'_d, \Delta'_d, g'_{0,d})$.

Input keys. For each input wire i, sample $(k_i, K_i) \leftarrow\!\$\ \mathbb{Z}_B \times \mathbb{Z}_{p-1}$.

Addition/Subtraction Gate $\gamma = (u, v, w)$**.** Given the incoming wires u and v with keys $(k_u, K_u), (k_v, K_v)$ and depth d_u, d_v respectively, set $d := \max(d_u, d_v) + 1$. Set

For wire u**.** $(\mathsf{shift}_u, \langle \Delta'_d \ell_u \rangle_{\mathsf{G}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_u}, \Delta'_d, K_u, \gamma||u)$

For wire v**.** $(\mathsf{shift}_v, \langle \Delta'_d \ell_v \rangle_{\mathsf{G}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_v}, \Delta'_d, K_v, \gamma||v)$

For wire w**.**
- $k_w := k_u \pm k_v \bmod B$
- $\tilde{K}_w := \langle \Delta'_d \ell_u \rangle_{\mathsf{G}} \pm \langle \Delta'_d \ell_v \rangle_{\mathsf{G}} \bmod p - 1$
- $(\mathsf{shift}_w, K_w) := \mathsf{MR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}'_d, \Delta_d, \tilde{K}_w, \gamma, B)$
- Set the keys of the outgoing wire w at depth d to (k_w, K_w) and the garbled gate to

$$\hat{C}_\gamma := (\mathsf{shift}_u, \mathsf{shift}_v, \mathsf{shift}_w).$$

Multiplication Gate $\gamma = (u, v, w)$**.** Given the incoming wires u and v with keys $(k_u, K_u), (k_v, K_v)$ and depth d_u, d_v respectively, set $d := \max(d_u, d_v) + 1$ and sample c uniformly random shares $(k^i_u, k^i_v)_{i \le c}$ of (k_u, k_v) over $\mathbb{Z}_B$.

For wire u**.** For $i = 1$ to c,
- $(\mathsf{leakyshift}^i_u, \langle k^i_u \ell_v \rangle_{\mathsf{G}}) := \mathsf{LVtO}^{\mathsf{H}'}_{\mathsf{G}}(\mathsf{msk}_{d_v}, k^i_u, K_v, \gamma||u||i)$
- $(\mathsf{shift}^i_u, \langle \Delta'_d k^i_u \ell_v \rangle_{\mathsf{G}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_v}, \Delta'_d k^i_u, K_v, \gamma||u||i)$

- $(\mathsf{shift}_u, K'_u) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_u}, \Delta'_d, K_u, \gamma||u)$
- $(\mathsf{shift}'_u, \langle K'_u \ell_v \rangle_{\mathsf{G}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_v}, K'_u, K_v, \gamma)$

For wire v**.** For $i = 1$ to c,
- $(\mathsf{leakyshift}^i_v, \langle k^i_v \ell_u \rangle_{\mathsf{G}}) := \mathsf{LVtO}^{\mathsf{H}'}_{\mathsf{G}}(\mathsf{msk}_{d_u}, k^i_v, K_u, \gamma||v||i)$
- $(\mathsf{shift}^i_v, \langle \Delta'_d k^i_v \ell_u \rangle_{\mathsf{G}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_u}, \Delta'_d k^i_v, K_u, \gamma||v||i)$

For wire w**.** For $i = 1$ to c,
- $\tilde{k}^i_w := \langle k^i_u \ell_v \rangle_{\mathsf{G}} + \langle k^i_v \ell_u \rangle_{\mathsf{G}}$
- $\tilde{K}^i_w := \langle \Delta'_d k^i_u \ell_v \rangle_{\mathsf{G}} + \langle \Delta'_d k^i_v \ell_u \rangle_{\mathsf{G}} \bmod p - 1$

- $\tilde{k}_w := \left(\sum_{i=1}^c k^i_u\right) \cdot \left(\sum_{i=1}^c k^i_v\right) + \sum_{i=1}^t \tilde{k}^i_w$
- $\tilde{K}_w := \langle K'_u \ell_v \rangle_{\mathsf{G}} - \Delta'_d \cdot \left(\sum_{i=1}^c \tilde{k}^i_w\right) + \sum_{i=1}^c \tilde{K}^i_w \bmod p - 1$
- $k_w := [\tilde{k}_w \bmod B]$
- $(\mathsf{shift}_w, K_w) := \mathsf{MR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}'_d, \Delta_d, \tilde{K}_w, \gamma, B)$
- Set the keys of the outgoing wire w at depth d to (k_w, K_w) and the garbled gate to

$$\hat{C}_\gamma := \left(\sum_{i=1}^{c} (\mathsf{leakyshift}^i_u, \mathsf{leakyshift}^i_v, \mathsf{shift}^i_u, \mathsf{shift}^i_v), \mathsf{shift}_u, \mathsf{shift}'_u, \mathsf{shift}_w\right).$$

Output. The garbler outputs the global keys Δ_1, the wire keys $\hat{K}$ and the garbled circuit $\hat{C}$, where

$$\hat{K} = (k_w, K_w)_{w \in W(C)}, \qquad \hat{C} = \left((\mathsf{mpk}_d, \mathsf{mpk}'_d)_{d \le D}, (\hat{C}_\gamma)_{\gamma \in \Gamma(C)}, (k_o)_{o \in O(C)}\right).$$

Algorithm 1: Garbling Algorithm of the Modular Arithmetic Garbling Scheme over $\mathbb{Z}_B$ with Rate $\Theta(\log B/\lambda)$ from Power DDH.

Evaluation algorithm We represent on Algorithm 2 the evaluation algorithm $\mathsf{AGC.Eval}((\ell_i, L_i)_{i \in I(C)}, \hat{C})$. The evaluation algorithm takes as input the input labels (ℓ_i, L_i) for every $i \in I(C)$, where we recall that labels are defined for every wire w with value x_w as $(\ell_w, L_w) := (x_w - k_w \bmod B, \Delta_d \cdot \ell_w - K_w \bmod p - 1)$. It also receives the garbled circuit $\hat{C} = \left((\mathsf{mpk}_d, \mathsf{mpk}'_d)_{d \le D}, (\hat{C}_\gamma)_{\gamma \in \Gamma(C)}\right)$.

Algorithm Modular Garbling from Power-DDH: $\mathsf{AGC.Eval}((\ell_i, L_i)_{i\in I(C)}, \hat{C})$

Parameters. Let $(\mathbb{G}, p, g, G) := \mathsf{GrpGen}^*(1^\lambda)$. Let $c = c(\lambda)$ denote an arbitrarily slowly growing functions $c = \omega(1)$, and set $B' := B^2 + c \cdot \lambda \cdot B^3$. Let $\mathsf{H} : \mathbb{G} \to \mathbb{Z}_{p-1}$ and $\mathsf{H}' : \mathbb{G} \times \{0,1\}^* \to [\lambda \cdot B]$ denote TCR hash functions for exponential correlations over $\mathbb{G}$. Let $\mathsf{F} = (\mathsf{F.Setup}^*, \mathsf{F.KeyGen}^*, \mathsf{F.Eval}, \mathsf{F.Punct}^*, \mathsf{F.PEval})$ denote the modified PPRF of Section 3.4. Parse $\hat{C} = \Big((\mathsf{mpk}_d, \mathsf{mpk}'_d)_{d\le D}, (\hat{C}_\gamma)_{\gamma\in\Gamma(C)}, (k_o)_{o\in O(C)}\Big)$. The evaluator computes the labels of all outgoing wires of a gate using the labels of the gate's incoming wires, starting with the input wire labels and computing each gate's outgoing wire's label as soon as its incoming wire labels have been computed.

Addition/Subtraction Gate $\gamma = (u, v, w)$**.** Given the incoming wires u and v with labels $(\ell_u, L_u), (\ell_v, L_v)$ and depth d_u, d_v respectively, set $d := \max(d_u, d_v) + 1$. Parse $\hat{C}_\gamma := (\mathsf{shift}_u, \mathsf{shift}_v, \mathsf{shift}_w)$. Set

For wire u**.** $\langle \Delta'_d \ell_u \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma||u, \mathsf{shift}_u)$

For wire v**.** $\langle \Delta'_d \ell_v \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma||v, \mathsf{shift}_v)$

For wire w**.**

- $\tilde{\ell}_w := \ell_u \pm \ell_v$
- $\ell_w := [\tilde{\ell}_w \bmod B]$
- $\tilde{L}_w := \langle \Delta'_d \ell_u \rangle_\mathsf{E} \pm \langle \Delta'_d \ell_v \rangle_\mathsf{E} \bmod p-1$
- $L_w := \mathsf{MR}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}'_d, \tilde{\ell}_w, \tilde{L}_w, \gamma, \mathsf{shift}_w, B)$
- Set the outgoing wire label to (ℓ_w, L_w).

Multiplication Gate $\gamma = (u, v, w)$**.** Given the incoming wires u and v with label $(\ell_u, L_u), (\ell_v, L_v)$ and depth d_u, d_v respectively, set $d := \max(d_u, d_v) + 1$ and parse $\hat{C}_\gamma := (\mathsf{leakyshift}_u, \mathsf{leakyshift}_v, \mathsf{shift}^0_u, \mathsf{shift}^0_v, \mathsf{shift}_u, \mathsf{shift}'_u, \mathsf{shift}_w)$.

For wire u**.**

- $\langle \sum_{i=1}^c k_u^i \ell_v \rangle_\mathsf{E} := \mathsf{LVtO}^{\mathsf{H}'}_\mathsf{E}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma||u, \mathsf{leakyshift}_u)$
- $\langle \Delta'_d \sum_{i=1}^c k_u^i \ell_v \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma||u, \mathsf{shift}^0_u)$
- $L'_u := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma||u, \mathsf{shift}_u)$
- $\langle K'_u \ell_v \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma, \mathsf{shift}'_u)$

For wire v**.**

- $\langle \sum_{i=1}^c k_v^i \ell_u \rangle_\mathsf{E} := \mathsf{LVtO}^{\mathsf{H}'}_\mathsf{E}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma||v, \mathsf{leakyshift}_v)$
- $\langle \Delta'_d \sum_{i=1}^c k_v^i \ell_u \rangle_\mathsf{E} := \mathsf{VtO}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma||v, \mathsf{shift}^0_v)$

For wire w**.**

- $\tilde{\ell}_w := \ell_u\ell_v + \langle \sum_{i=1}^c k_u^i \ell_v \rangle_\mathsf{E} + \langle \sum_{i=1}^c k_v^i \ell_u \rangle_\mathsf{E}$
- $\tilde{L}_w := \langle K'_u \ell_v \rangle_\mathsf{E} + L'_u \ell_v + \langle \Delta'_d \sum_{i=1}^c k_u^i \ell_v \rangle_\mathsf{E} + \langle \Delta'_d \sum_{i=1}^c k_v^i \ell_u \rangle_\mathsf{E} \bmod p-1$
- $\ell_w := [\tilde{\ell}_w \bmod B]$
- $L_w := \mathsf{MR}^\mathsf{H}_\mathsf{E}(\mathsf{mpk}'_d, \tilde{\ell}_w, \tilde{L}_w, \gamma, \mathsf{shift}_w, B)$
- Set the outgoing wire label to (ℓ_w, L_w).

Output. For all output wire $o \in O(C)$, set $z_o := \ell_o + k_o \bmod B$. Output $z := (z_o)_{o\in O(C)}$.

Algorithm 2: Evaluation Algorithm of the Modular Arithmetic Garbling Scheme over $\mathbb{Z}_B$ with Rate $\Theta(\log B/\lambda)$ from Power DDH.

4.5 Proof of Theorem 18

We provide a complete proof of correctness and security of our modular garbling scheme below.

Correctness Fix a circuit C and an input x. For every wire w of C, let x_w denote the value carried by this wire during the computation of $C(x)$. We prove that the following invariant is maintained for every wire w:

$$(\ell_w, L_w) := (x_w - k_w \bmod B, \Delta_d \cdot \ell_w - K_w \bmod p-1),$$

where $\ell_w \leq B$, and d denote the depth $\mathsf{depth}(w)$ of w. Note that in particular, this implies that $z_o = \ell_o + k_o = x_o \mod B$ for every output wire $o \in O(C)$, hence correctness is satisfied if the invariant is maintained. Fix the input labels $(\ell_i, L_i)_{i \in I(C)} \coloneqq \mathsf{InputLabels}(1^\lambda, C, \Delta_1, \hat{K}, x)$, which satisfy the invariant by definition.

Addition/Subtraction Gate $\gamma = (u, v, w)$ with incoming wires at depth d_u, d_v with keys $(k_u, K_u), (k_v, K_v)$ and labels $(\ell_u, L_u), (\ell_v, L_v)$. Assume that the invariant is maintained for u, v and set $d \coloneqq \max(d_u, d_v) + 1$.

For wire u. $\langle \Delta'_d \ell_u \rangle_{\mathsf{G}} + \langle \Delta'_d \ell_u \rangle_{\mathsf{E}} = \Delta'_d \ell_u \mod p - 1$ ▶ by correctness of $\mathsf{VtO}^{\mathsf{H}}$

For wire v. $\langle \Delta'_d \ell_v \rangle_{\mathsf{G}} + \langle \Delta'_d \ell_v \rangle_{\mathsf{E}} = \Delta'_d \ell_v \mod p - 1$ ▶ by correctness of $\mathsf{VtO}^{\mathsf{H}}$

For wire w.

- $\ell_w = \ell_u \pm \ell_v \mod B$
- $\tilde{K}_w + \tilde{L}_w = (\langle \Delta'_d \ell_u \rangle_{\mathsf{G}} + \langle \Delta'_d \ell_u \rangle_{\mathsf{E}}) \pm (\langle \Delta'_d \ell_v \rangle_{\mathsf{G}} + \langle \Delta'_d \ell_v \rangle_{\mathsf{E}}) = \Delta'_d \cdot (\ell_u + \ell_v) \mod p - 1$
- $K_w + L_w = \Delta'_d \cdot [\ell_u + \ell_v \mod B] = \Delta'_d \ell_w \mod p - 1$ ▶ by correctness of MR^{H}

Multiplication Gate $\gamma = (u, v, w)$ with incoming wires at depth d_u, d_v with keys $(k_u, K_u), (k_v, K_v)$ and labels $(\ell_u, L_u), (\ell_v, L_v)$. Assume that the invariant is maintained for u, v and set $d \coloneqq \max(d_u, d_v) + 1$.

For wire u.

- $\sum_{i=1}^{c} \langle k_u^i \ell_v \rangle_{\mathsf{G}} + \langle \sum_{i=1}^{c} k_u^i \ell_v \rangle_{\mathsf{E}} = \sum_{i=1}^{c} k_u^i \ell_v$ (over $\mathbb{Z}$) ▶ by correctness of $\mathsf{LVtO}^{\mathsf{H}'}$
- $\sum_{i=1}^{c} \langle \Delta'_d k_u^i \ell_v \rangle_{\mathsf{G}} + \langle \Delta'_d \sum_{i=1}^{c} k_u^i \ell_v \rangle_{\mathsf{E}} = \Delta'_d \sum_{i=1}^{c} k_u^i \ell_v \mod p - 1$ ▶ by correctness of $\mathsf{VtO}^{\mathsf{H}}$

- $K'_u + L'_u = \Delta'_d \ell_u \mod p - 1$ ▶ by correctness of $\mathsf{VtO}^{\mathsf{H}}$
- $\langle K'_u \ell_v \rangle_{\mathsf{G}} + \langle K'_u \ell_v \rangle_{\mathsf{E}} = K'_u \ell_v \mod p - 1$ ▶ by correctness of $\mathsf{VtO}^{\mathsf{H}}$

For wire v.

- $\sum_{i=1}^{c} \langle k_v^i \ell_u \rangle_{\mathsf{G}} + \langle \sum_{i=1}^{c} k_v^i \ell_u \rangle_{\mathsf{E}} = \sum_{i=1}^{c} k_v^i \ell_u$ (over $\mathbb{Z}$) ▶ by correctness of $\mathsf{LVtO}^{\mathsf{H}'}$
- $\sum_{i=1}^{c} \langle \Delta'_d k_v^i \ell_u \rangle_{\mathsf{G}} + \langle \Delta'_d \sum_{i=1}^{c} k_v^i \ell_u \rangle_{\mathsf{E}} = \Delta'_d \sum_{i=1}^{c} k_v^i \ell_u \mod p - 1$ ▶ by correctness of $\mathsf{VtO}^{\mathsf{H}}$

For wire w.

$$\begin{aligned}
\tilde{k}_w + \tilde{\ell}_w &= \left(\sum_{i=1}^{c} k_u^i\right) \cdot \left(\sum_{i=1}^{c} k_v^i\right) + \sum_{i=1}^{t} \tilde{k}_w^i + \langle \sum_{i=1}^{c} k_u^i \ell_v \rangle_{\mathsf{E}} + \langle \sum_{i=1}^{c} k_v^i \ell_u \rangle_{\mathsf{E}} + \ell_u \ell_v \\
&= \left(\sum_{i=1}^{c} k_u^i\right) \cdot \left(\sum_{i=1}^{c} k_v^i\right) + \sum_{i=1}^{t} (k_u^i \ell_v + k_v^i \ell_u) + \ell_u \ell_v \\
&= \left(\sum_{i=1}^{c} k_u^i + \ell_u\right) \cdot \left(\sum_{i=1}^{c} k_v^i + \ell_v\right) \\
\implies k_w + \ell_w &= \left(\sum_{i=1}^{c} k_u^i + \ell_u\right) \cdot \left(\sum_{i=1}^{c} k_v^i + \ell_v\right) \mod B \\
&= (k_u + \ell_u) \cdot (k_v + \ell_v) \mod B \blacktriangleright \text{ the } k_u^i, k_v^i \text{ are shares of } k_u, k_v \text{ over } \mathbb{Z}_B \\
&= x_u \cdot x_v \mod B \blacktriangleright \text{ by the invariants for } u, v
\end{aligned}$$

$$\begin{aligned}
\tilde{K}_w + \tilde{L}_w &= (\langle K'_u \ell_v \rangle_{\mathsf{G}} + \langle K'_u \ell_v \rangle_{\mathsf{E}}) - \Delta'_d \cdot \sum_{i=1}^{c} \tilde{k}_w^i + L_u \ell_v + \sum_{i=1}^{c} \tilde{K}_w^i \\
&\quad + \langle \Delta'_d \sum_{i=1}^{c} k_u^i \ell_v \rangle_{\mathsf{E}} + \langle \Delta'_d \sum_{i=1}^{c} k_v^i \ell_u \rangle_{\mathsf{E}} \mod p - 1 \\
&= K'_u \ell_v - \Delta'_d \cdot \sum_{i=1}^{c} \tilde{k}_w^i + L'_u \ell_v + \Delta'_d \cdot \left(\sum_{i=1}^{c} \tilde{k}_w^i + \langle \sum_{i=1}^{c} k_u^i \ell_v \rangle_{\mathsf{E}} + \langle \sum_{i=1}^{c} k_v^i \ell_u \rangle_{\mathsf{E}} \right) \mod p - 1 \\
&= (K'_u + L'_u) \ell_v + \Delta'_d \cdot \left(\langle \sum_{i=1}^{c} k_u^i \ell_v \rangle_{\mathsf{E}} + \langle \sum_{i=1}^{c} k_v^i \ell_u \rangle_{\mathsf{E}} \right) \mod p - 1 \\
&= \Delta'_d \cdot \ell_u \ell_v + \Delta'_d \cdot \left(\langle \sum_{i=1}^{c} k_u^i \ell_v \rangle_{\mathsf{E}} + \langle \sum_{i=1}^{c} k_v^i \ell_u \rangle_{\mathsf{E}} \right) = \Delta'_d \cdot \tilde{\ell}_w \mod p - 1 \\
\implies K_w + L_w &= \Delta'_d \cdot [\tilde{\ell}_w \mod B] \blacktriangleright \text{ by correctness of } \mathsf{MR}^{\mathsf{H}}
\end{aligned}$$

To conclude the proof of correctness, it remains to verify that, when using the MR^{H} procedure on shares of $\Delta'_d \cdot (\tilde{\ell}_w)$ over $\mathbb{Z}_{p-1}$, the value $\tilde{\ell}_w$ is bounded by $B' = B^2 + \lambda \cdot c \cdot B^3$ (in absolute value):

- For addition/substraction gates, $\tilde{\ell}_w = \ell_u + \ell_v \leq 2B < B'$
- For multiplication gates, $\tilde{\ell}_w = \ell_u \ell_v + \sum_{i=1}^{c} \tilde{\ell}^i_w$, where $\tilde{\ell}^i_w = \langle k^i_u \ell_v \rangle_{\mathsf{E}} + \langle k^i_v \ell_u \rangle_{\mathsf{E}}$.

As shown in Section 4.3, the outputs of $\mathsf{LVtO}^{\mathsf{H}'}_{\mathsf{E}}$ have absolute value at most $\lambda \cdot B^3/2$, hence $|\langle k^i_u \ell_v \rangle_{\mathsf{E}} + \langle k^i_v \ell_u \rangle_{\mathsf{E}}| \leq \lambda \cdot B^3$, and therefore $|\tilde{\ell}_w| \leq B^2 + c \cdot \lambda \cdot B^3 = B'$. Eventually, the bound $\ell_w \leq B$ is trivial as $\ell_w := [\tilde{\ell}_w \bmod B]$. This concludes the proof of correctness.

Security We represent on Algorithm 3 a simulator Sim that, on input $(1^\lambda, C, z)$, outputs a simulated garbled circuit $\hat{C}$ together with simulated input labels $\hat{L} = (\ell_i, L_i)_{i \in I(C)}$.

Algorithm Modular Garbling from Power-DDH: $\mathsf{Sim}(1^\lambda, C, z)$

Parameters. Let $(\mathbb{G}, p, g, G) := \mathsf{GrpGen}^*(1^\lambda)$. Let $c = c(\lambda)$ denote an arbitrarily slowly growing functions $c = \omega(1)$, and set $B' := B^2 + c \cdot \lambda \cdot B^3$. Set $t \leftarrow (2c+1) \cdot s$. Let $\mathsf{H} : \mathbb{G} \to \mathbb{Z}_p$ and $\mathsf{H}' : \mathbb{G} \to [\lambda \cdot B]$ denote TCR hash functions for exponential correlations over $\mathbb{G}$. Let $\mathsf{F} = (\mathsf{F.Setup}^*, \mathsf{F.KeyGen}^*, \mathsf{F.Eval}, \mathsf{F.Punct}^*, \mathsf{F.PEval})$ denote the modified PPRF of Section 3.4.

Initialization. For $d = 1$ to D,
- sample $(\mathsf{mpk}_d, \mathsf{msk}_d) \leftarrow\!\$\ \mathsf{F.Setup}(1^\lambda, B)$. Parse $\mathsf{msk}_d := (\mathsf{mpk}_d, \Delta_d, g_{0,d})$.
- Sample $(\mathsf{mpk}'_d, \mathsf{msk}'_d) \leftarrow\!\$\ \mathsf{F.Setup}(1^\lambda, B')$. Parse $\mathsf{msk}'_d := (\mathsf{mpk}'_d, \Delta'_d, g'_{0,d})$.

Simulated Input labels. For each input wire i, sample $(\ell_i, L_i) \leftarrow\!\$\ \mathbb{Z}_B \times \mathbb{Z}_{p-1}$.

Addition/Subtraction Gate $\gamma = (u, v, w)$**.** Given the incoming wires u and v with simulated labels $(\ell_u, L_u), (\ell_v, L_v)$ and depth d_u, d_v respectively, set $d := \max(d_u, d_v) + 1$.

For wire u**.** Set $\mathsf{shift}_u \leftarrow\!\$\ \mathbb{Z}_{p-1}$ and $\langle \Delta'_d \ell_u \rangle_{\mathsf{E}} := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma || u, \mathsf{shift}_u)$.

For wire v**.** Set $\mathsf{shift}_v \leftarrow\!\$\ \mathbb{Z}_{p-1}$ and $\langle \Delta'_d \ell_v \rangle_{\mathsf{G}} := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma || v, \mathsf{shift}_v)$.

For wire w**.**
- $\tilde{\ell}_w := \ell_u \pm \ell_v$
- $\ell_w := [\tilde{\ell}_w \bmod B]$
- $\tilde{L}_w := \langle \Delta'_d \ell_u \rangle_{\mathsf{E}} \pm \langle \Delta'_d \ell_v \rangle_{\mathsf{E}} \bmod p - 1$
- Sample $\mathsf{shift}_w \leftarrow\!\$\ \mathbb{Z}_{p-1}$ and set $L_w := \mathsf{MR}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}'_d, \tilde{\ell}_w, \tilde{L}_w, \gamma, \mathsf{shift}_w, B)$
- Set the simulated outgoing wire label to (ℓ_w, L_w) and the simulated garbled gate to

$$\hat{C}_\gamma := (\mathsf{shift}_u, \mathsf{shift}_v, \mathsf{shift}_w).$$

Multiplication Gate $\gamma = (u, v, w)$**.** Given the incoming wires u and v with simulated label $(\ell_u, L_u), (\ell_v, L_v)$ and depth d_u, d_v respectively, set $d := \max(d_u, d_v) + 1$.

For wire u**.** For $i = 1$ to c,
- $\mathsf{psk}^i_u := \mathsf{F.Punct}^*(\mathsf{mpk}_{d_v}, L_v)$
- Sample $\sigma^u_i \leftarrow\!\$\ \mathsf{Ber}(1/\lambda)$.
- If $\sigma^u_i = 1$, $k^i_u \leftarrow\!\$\ \mathbb{Z}_B$, $\mathsf{leakyshift}^i_u \leftarrow\!\$\ \mathsf{SimLShift}^{\mathsf{H}'}(\mathsf{mpk}_{d_v}, \mathsf{psk}^i_u, k^i_u, \ell_v, \gamma || u || i)$.
- Else, $\mathsf{leakyshift}^i_u \leftarrow\!\$\ \mathsf{SimShift}^{\mathsf{H}'}(\mathsf{mpk}_{d_v}, \mathsf{psk}^i_u, \ell_v, \gamma || u || i)$
- $\langle k^i_u \ell_v \rangle_{\mathsf{E}} := \mathsf{LVtO}^{\mathsf{H}'}_{\mathsf{E}}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma || u || i, \mathsf{leakyshift}^i_u)$
- Sample $\mathsf{shift}^i_u \leftarrow\!\$\ \mathbb{Z}_{p-1}$
- $\langle \Delta'_d k^i_u \ell_v \rangle_{\mathsf{E}} := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma || u || i, \mathsf{shift}^i_u)$

- Sample $(\mathsf{shift}_u, \mathsf{shift}'_u) \leftarrow\!\$\ \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$
- $L'_u := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma || u, \mathsf{shift}_u)$.
- $\langle K'_u \ell_v \rangle_{\mathsf{E}} := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma, \mathsf{shift}'_u)$

For wire v**.** For $i = 1$ to c,
- $\mathsf{psk}^i_v := \mathsf{F.Punct}^*(\mathsf{mpk}_{d_u}, L_u)$
- Sample $\sigma^v_i \leftarrow\!\$\ \mathsf{Ber}(1/\lambda)$.
- If $\sigma^v_i = 1$, set $k^i_v \leftarrow\!\$\ \mathbb{Z}_B$ and $\mathsf{leakyshift}^i_v \leftarrow\!\$\ \mathsf{SimLShift}^{\mathsf{H}'}(\mathsf{mpk}_{d_u}, \mathsf{psk}^i_v, k^i_v, \ell_u, \gamma || v || i)$.
- Else, $\mathsf{leakyshift}^i_v \leftarrow\!\$\ \mathsf{SimShift}^{\mathsf{H}'}(\mathsf{mpk}_{d_u}, \mathsf{psk}^i_v, \ell_u, \gamma || v || i)$
- $\langle k^i_v \ell_u \rangle_{\mathsf{E}} := \mathsf{LVtO}^{\mathsf{H}'}_{\mathsf{E}}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma || v || i, \mathsf{leakyshift}^i_v)$

– Sample $\mathsf{shift}_v^i \leftarrow\!\$\ \mathbb{Z}_{p-1}$
– $\langle \Delta'_d k_v^i \ell_u \rangle_{\mathsf{E}} := \mathsf{VtO}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma || v || i, \mathsf{shift}_v^i)$

For wire w**.** For $i = 1$ to c,
- $\tilde{\ell}_w^i := \langle k_u^i \ell_v \rangle_{\mathsf{E}} + \langle k_v^i \ell_u \rangle_{\mathsf{E}}$
- $\tilde{L}_w^i := \langle \Delta'_d k_u^i \ell_v \rangle_{\mathsf{E}} + \langle \Delta'_d k_v^i \ell_u \rangle_{\mathsf{E}} \bmod p-1$

– $\tilde{\ell}_w := \ell_u \ell_v + \sum_{i=1}^c \tilde{\ell}_w^i$
– Sample $\mathsf{shift}_w \leftarrow\!\$\ \mathbb{Z}_{p-1}$
– $\tilde{L}_w := \langle K'_u \ell_v \rangle_{\mathsf{E}} + L'_u \ell_v + \sum_{i=1}^c \tilde{L}_w^i \bmod p-1$
– $\ell_w := [\tilde{\ell}_w \bmod B]$
– $L_w := \mathsf{MR}_{\mathsf{E}}^{\mathsf{H}}(\mathsf{mpk}'_d, \tilde{\ell}_w, \tilde{L}_w, \gamma, \mathsf{shift}_w, B)$
– Set the simulated outgoing wire label to (ℓ_w, L_w) and the simulated garbled gate to

$$\hat{C}_\gamma := (\sum_{i=1}^{c} (\mathsf{leakyshift}_u^i, \mathsf{leakyshift}_v^i, \mathsf{shift}_u^i, \mathsf{shift}_v^i), \mathsf{shift}_u, \mathsf{shift}'_u, \mathsf{shift}_w).$$

Output. Given the output string $z = (z_o)_{o \in O(C)}$, set $k_o := z_o - \ell_o \bmod B$ and output

$$\hat{L} = (\ell_i, L_i)_{i \in I(C)} \qquad \hat{C} = \left((\mathsf{mpk}_d, \mathsf{mpk}'_d)_{d \le D}, (\hat{C}_\gamma)_{\gamma \in \Gamma(C)}, (k_o)_{o \in O(C)} \right).$$

Algorithm 3: Simulator of the Modular Arithmetic Garbling Scheme over $\mathbb{Z}_B$ with Rate $\omega(\log B/\lambda)$ from Power DDH.

Lemma 21. *For all sequences of polynomial-size $\mathbb{Z}_B$-arithmetic circuits $\{C_\lambda\}_{\lambda \in \mathbb{N}}$, and every sequence of inputs $(x_\lambda)_{\lambda \in \mathbb{N}}$ with $|x_\lambda| = |I(C_\lambda)|$, the following holds:*

$$\left\{ \mathsf{Sim}(1^\lambda, C_\lambda, z) : z \leftarrow C_\lambda(x) \right\} \approx_c \left\{ ((\ell_i, L_i)_{i \in I(C_\lambda)}, \widehat{C}) : \begin{array}{l} (\Delta_1, \hat{K}, \widehat{C}) \leftarrow\!\$\ \mathsf{Garble}(1^\lambda, C) \\ (\ell_i, L_i)_{i \in I(C_\lambda)} := \mathsf{InputLabels}(1^\lambda, C_\lambda, \Delta_1, \hat{K}, x) \end{array} \right\}$$

Proof. The proof proceed through a sequence of game hops. Fix a security parameter 1^λ, a circuit $C = C_\lambda$, and an input $x = x_\lambda$. We let $\mathbf{Game}_0$ denote the real game:

$\mathbf{Game}_0$: the simulator Sim receives as input $(1^\lambda, C, x)$, computes $(\Delta_1, \hat{K}, \hat{C}) \leftarrow\!\$\ \mathsf{Garble}(1^\lambda, C)$ and $(\ell_i, L_i)_{i \in I(C)} := \mathsf{InputLabels}(1^\lambda, C, \Delta_1, \hat{K}, x)$. Sim outputs $((\ell_i, L_i)_{i \in I(C)}, \hat{C})$.

In the following sequence of games, the simulator initially knows the wire values $(x_w)_{w \in W(C)}$ for all incoming wires of all gates and the corresponding incoming wire keys $(k_w, K_w)_{w \in W(C)}$. The proof proceeds by letting the simulator forget, in a level-by-level fashion, the incoming wire values x_w and corresponding wire keys (k_w, K_w) for gates at a given level, and construct instead simulated labels (ℓ_w, L_w).

$\mathbf{Game}_d$: at the start of this game, Sim knows the wire values x_w for all wires w at depth $\mathsf{depth}(w) \ge d$. It performs the following steps:

Initialization. For $d = 1$ to D,
– sample $(\mathsf{mpk}_d, \mathsf{msk}_d) \leftarrow\!\$\ \mathsf{F.Setup}(1^\lambda, B)$. Parse $\mathsf{msk}_d := (\mathsf{mpk}_d, \Delta_d, g_{0,d})$.
– Sample $(\mathsf{mpk}'_d, \mathsf{msk}'_d) \leftarrow\!\$\ \mathsf{F.Setup}(1^\lambda, B')$. Parse $\mathsf{msk}'_d := (\mathsf{mpk}'_d, \Delta'_d, g'_{0,d})$.
Simulated Input labels. For each input wire i, sample $(\ell_i, L_i) \leftarrow\!\$\ \mathbb{Z}_B \times \mathbb{Z}_{p-1}$.

During the simulation of $\hat{C}$, Sim constructs simulated labels (ℓ_w, L_w) for every wire w at depth $\mathsf{depth}(w) < d$, and simulated wire keys (k_w, K_w) for every wire w at depth $\mathsf{depth}(w) \ge d$. We cover each of the two cases below.

Case 1: For every gate γ with outgoing wire w at depth $\mathsf{depth}(w) \le d$. Given the incoming wires u and v with simulated label $(\ell_u, L_u), (\ell_v, L_v)$ and depth $d_u, d_v < d$ respectively, run the gate garbling algorithm of Algorithm 3 to obtain the simulated outgoing wire label (ℓ_w, L_w) and the simulated garbled gate $\hat{C}_\gamma$. If $\mathsf{depth}(w) = d$, set $(k_w, K_w) := (x_w - \ell_w \bmod B, \Delta_d \ell_w - L_w \bmod p-1)$.

Case 2: For every gate γ with outgoing wire w at depth $\mathsf{depth}(z) > d$.

Case 2.1: $d_u, d_v \geq d$. Run the gate garbling algorithm of Algorithm 1 with incoming wire keys (k_u, K_u) and (k_v, K_v) to obtain the outgoing wire key (k_w, K_w) and the garbled gate $\hat{C}$.

Case 2.2: $d_u < d$, $d_v = d_w - 1 \geq d$. Given the simulated labels (ℓ_u, L_u) for the incoming wire u and the wire key (k_v, K_v) for the incoming wire v, set $(\ell_v, L_v) := (x_v - k_v \bmod B, \Delta_{d_v} x_v - K_v \bmod p - 1)$.

We provide below the detailed procedure for case 2.2. We use red and blue to denote variants of the procedure that will be used in later games hops. The procedure of case 2.2 corresponds to variant 1, where the parts in blue are ignored.

Addition/Subtraction Gate $\gamma = (u, v, w)$.

For wire u. *Simulated procedure for incoming wires*

- Set $\mathsf{shift}_u \leftarrow\!\$\ \mathbb{Z}_{p-1}$ and $\langle \Delta'_d \ell_u \rangle_{\mathsf{E}} := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma||u, \mathsf{shift}_u)$.

For wire v. *Garbler procedure for incoming wires*

- $(\mathsf{shift}_v, \langle \Delta'_{d_w} \ell_v \rangle_{\mathsf{G}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_v}, \Delta'_{d_w}, K_v, \gamma||v)$
- $\langle \Delta'_{d_w} \ell_v \rangle_{\mathsf{E}} := \Delta'_{d_w} \ell_v - \langle \Delta'_{d_w} \ell_v \rangle_{\mathsf{G}} \bmod p - 1$

For wire w. *Hybrid procedure for outgoing wires*

- $\tilde{\ell}_w := \ell_u \pm \ell_v$
- $\ell_w := [\tilde{\ell}_w \bmod B]$
- $\tilde{L}_w := \langle \Delta'_d \ell_u \rangle_{\mathsf{E}} \pm \langle \Delta'_d \ell_v \rangle_{\mathsf{E}} \bmod p - 1$
- $\tilde{K}_w := \Delta'_{d_w} \cdot (\ell_u \pm \ell_v) - \tilde{L}_w \bmod p - 1$
- $(\mathsf{shift}_w, K_w) := \mathsf{MR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}'_{d_w}, \Delta_{d_w}, \tilde{K}_w, \gamma, B)$
- Set the outgoing wire key to $(x_w - \ell_w \bmod B, K_w)$ and the simulated garbled gate to

$$\hat{C}_\gamma := (\mathsf{shift}_u, \mathsf{shift}_v, \mathsf{shift}_w).$$

Multiplication Gate $\gamma = (u, v, w)$.

For wire u. *Simulated procedure for incoming wires* (version 1, version 2)

- Sample c uniformly random shares $(k^i_u)_{i \leq c}$ of k_u over $\mathbb{Z}_B$.
- For $i = 1$ to c,
  - $\mathsf{psk}^i_u := \mathsf{F.Punct}^*(\mathsf{mpk}_{d_v}, L_v)$
  - Sample $\sigma^u_i \leftarrow\!\$\ \mathsf{Ber}(1/\lambda)$.
  - If $\sigma^u_i = 1$, set $k^i_u \leftarrow\!\$\ \mathbb{Z}_B$ and $\mathsf{leakyshift}^i_u \leftarrow\!\$\ \mathsf{SimLShift}^{\mathsf{H}'}(\mathsf{mpk}_{d_v}, \mathsf{psk}^i_u, k^i_u, \ell_v, \gamma||u||i)$.
  - Else, $\mathsf{leakyshift}^i_u \leftarrow\!\$\ \mathsf{SimShift}^{\mathsf{H}'}(\mathsf{mpk}_{d_v}, \mathsf{psk}^i_u, \ell_v, \gamma||u||i)$
  - $\langle k^i_u \ell_v \rangle_{\mathsf{E}} := \mathsf{LVtO}^{\mathsf{H}'}_{\mathsf{E}}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma||u||i, \mathsf{leakyshift}^i_u)$
  - Sample $\mathsf{shift}^i_u \leftarrow\!\$\ \mathbb{Z}_{p-1}$
  - $\langle \Delta'_{d_w} k^i_u \ell_v \rangle_{\mathsf{E}} := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma||u||i, \mathsf{shift}^i_u)$
  - Sample $(\mathsf{shift}_u, \mathsf{shift}'_u) \leftarrow\!\$\ \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$
  - $L'_u := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_u}, \ell_u, L_u, \gamma||u, \mathsf{shift}_u)$
  - $\langle K'_u \ell_v \rangle_{\mathsf{E}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{E}}(\mathsf{mpk}_{d_v}, \ell_v, L_v, \gamma, \mathsf{shift}'_u)$

For wire v. *Garbler procedure for incoming wires*

For $i = 1$ to c,

- $(\mathsf{leakyshift}^i_v, \langle k^i_v \ell_u \rangle_{\mathsf{G}}) := \mathsf{LVtO}^{\mathsf{H}'}_{\mathsf{G}}(\mathsf{msk}_{d_u}, k^i_v, K_u, \gamma||v||i)$
- $\langle k^i_v \ell_u \rangle_{\mathsf{E}} := k^i_v \ell_u - \langle k^i_v \ell_u \rangle_{\mathsf{G}}$
- $(\mathsf{shift}^i_v, \langle \Delta'_{d_w} k^i_v \ell_u \rangle_{\mathsf{G}}) := \mathsf{VtO}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}_{d_u}, \Delta'_d k^i_v, K_u, \gamma||v||i)$
- $\langle \Delta'_{d_w} k^i_v \ell_u \rangle_{\mathsf{E}} := \Delta'_{d_w} k^i_v \ell_u - \langle \Delta'_{d_w} k^i_v \ell_u \rangle_{\mathsf{G}} \bmod p - 1$

For wire w. *Hybrid procedure for outgoing wires*

For $i = 1$ to c,

- $\tilde{\ell}^i_w := \langle k^i_u \ell_v \rangle_{\mathsf{E}} + \langle k^i_v \ell_u \rangle_{\mathsf{E}}$
- $\tilde{L}^i_w := \langle \Delta'_{d_w} k^i_u \ell_v \rangle_{\mathsf{E}} + \langle \Delta'_{d_w} k^i_v \ell_u \rangle_{\mathsf{E}} \bmod p - 1$

- $\tilde{\ell}_w := \ell_u \ell_v + \sum_{i=1}^c \tilde{\ell}^i_w$
- $\ell_w := [\tilde{\ell}_w \bmod B]$
- $\langle K_u \ell_v \rangle_{\mathsf{E}} := K_u \ell_v - \langle K_u \ell_v \rangle_{\mathsf{G}} \bmod p - 1$
- $\tilde{L}_w := \langle K'_u \ell_v \rangle_{\mathsf{E}} + L'_u \ell_v + \sum_{i=1}^c \tilde{L}^i_w \bmod p - 1$
- $\tilde{K}_w := \Delta'_{d_w} \cdot \tilde{\ell}_w - \tilde{L}_w \bmod p - 1$
- $(\mathsf{shift}_w, K_w) := \mathsf{MR}^{\mathsf{H}}_{\mathsf{G}}(\mathsf{msk}'_{d_w}, \Delta'_{d_w}, \tilde{K}_w, \gamma, B)$

– Set the outgoing wire key to $(x_w - \ell_w \bmod B, K_w)$ and the simulated garbled gate to

$$\hat{C}_\gamma := (\sum_{i=1}^{c}(\mathsf{leakyshift}_u^i, \mathsf{leakyshift}_v^i, \mathsf{shift}_u^i, \mathsf{shift}_v^i), \mathsf{shift}_u, \mathsf{shift}'_u, \mathsf{shift}_w).$$

Game$_{d.1}$. In this game, we modify the simulation of the gate garbling procedure. The aim of the sequence of games starting at **Game**$_{d.1}$ is to allow Sim to *forget* all values x_w on wires at depth $\mathsf{depth}(w) = d$.

For every gate γ with outgoing wire w at depth $\mathsf{depth}(w) \leq d$. Sim behaves as in **Game**$_d$, except that if $\mathsf{depth}(w) = d$, it does not reconstruct (k_w, K_w) anymore (and instead simply stores w's wire label (ℓ_w, L_w).

For every gate γ with outgoing wire w at depth $\mathsf{depth}(w) > d$.

Case $d_u \neq d$ and $d_v \neq d$. Sim behaves exactly as in **Game**$_d$.
Case $d_u = d$ or $d_v = d$.
- For $a \in \{u, v\}$, if $d_a \leq d$, denoting b the other incoming wire, run the version 2 of the *simulated procedure for incoming wires* for wire b and gate γ. Else, run the *garbler procedure for incoming wires* for wire b and gate γ. Note that this differs from **Game**$_2$ only when $d_a = d$.
- As in **Game**$_d$, run the *hybrid procedure for outgoing wires* for the gate γ.

Claim. Under the t-instance joint security of $\mathsf{VtO}^{\mathsf{H}}, \mathsf{LVtO}^{\mathsf{H}'}$ (Definition 17), **Game**$_d$ and **Game**$_{d.1}$ are indistinguishable.

Proof. The only difference between **Game**$_d$ and **Game**$_{d.1}$ is that for all wires w at depth $\mathsf{depth}(w) = d$ (incoming to a gate γ), **Game**$_{d.1}$ uses the simulated procedure for incoming wires to γ, while **Game**$_d$ uses the garbled procedure for incoming wires to γ. The reduction to the t-instance joint security of $\mathsf{VtO}^{\mathsf{H}}, \mathsf{LVtO}^{\mathsf{H}'}$ is done by replacing the procedure for a depth-d incoming wire v to a gate γ (with other incoming wire u and outgoing wire w) with

- One call $(\mathsf{VtO}, \Delta'_{d_w}, \ell_u, L_u)$ to the experiment $\mathsf{ExpBothVtO}^{\mathsf{H},\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$ (where the salts $\mathbf{salt} := (\gamma, \gamma||a, \gamma||a||i)_{\gamma \in \Gamma(C), a \in W(C), i \leq c}$ are all pairwise-distinct by construction) if γ is an addition/substraction gate;
- c calls $(\mathsf{LVtO}, k_u^i, \ell_v, L_v)$, c calls $(\mathsf{VtO}, \Delta'_d k_u^i, \ell_v, L_v)$, and one call $(\mathsf{VtO}, K_u, \ell_v, L_v)$ to $\mathsf{ExpBothVtO}^{\mathsf{H},\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$ if γ is a multiplication gate.

The total number of calls is at most $(2c + 1)$ for each wire, and there can be at most s wire at any given level, hence the total number of calls is at most $(2c + 1)s = t$. Answering the queries as in $\mathsf{ExpBothVtO}_1^{\mathsf{H},\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$ yields exactly **Game**$_{d.1}$. Answering the queries as in $\mathsf{ExpBothVtO}_0^{\mathsf{H},\mathsf{H}'}(1^\lambda, B, \mathbf{salt})$ yields **Game**$_d$, up to one minor difference: the answers $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E}$ of $\mathsf{ExpBothVtO}_0$ are computed as $\langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{E} := \mathsf{LVtO}_\mathsf{E}^{\mathsf{H}'}(\mathsf{mpk}, v_\mathsf{E}, \langle \Delta v_\mathsf{E} \rangle_\mathsf{E}, \mathsf{salt}_i, \mathsf{shift})$, while they are computed as $v_\mathsf{G} v_\mathsf{E} - \langle v_\mathsf{G} v_\mathsf{E} \rangle_\mathsf{G}$ ($\bmod B$ if it is a VtO query, over $\mathbb{Z}$ otherwise) in **Game**$_d$. However, by perfect correctness of VtO and LVtO, this is perfectly equivalent to $\mathsf{ExpBothVtO}_0$. This concludes the proof. ■

Game$_{d.2}$. This game is identical to **Game**$_{d.1}$, except that we use version 1 in the simulated procedure for an incoming wire to a multiplication gate.

Claim. The advantage of any adversary in distinguishing **Game**$_{d.1}$ from **Game**$_{d.2}$ is at most $s/\lambda^c = \mathsf{negl}(\lambda)$.

Proof. The only difference between **Game**$_{d.1}$ and **Game**$_{d.2}$ lies in how the shares k_u^i are sampled: in version 2, the k_u^i are random $(c-1)$-out-of-c shares of k_u, while in version 1, the k_u^i are sampled independently. Observe that whenever $\sigma_i^u = 0$, the share k_u^i is not use by the procedure, and therefore, the two versions become perfectly indistinguishable: the view of a distinguisher between both games differs if and only if, for some incoming wire u, *all* the bits $(\sigma_i^u)_{i \leq c}$ are equal to 1. As the σ_i^u are independent Bernoulli samples equal to 1 with probability $1/\lambda$, this happens with probability $1/\lambda^c$, and we conclude with a union bound over all gates. ■

$\mathbf{Game}_{d.3}$. In this game, we modify the procedure for all gates where both incoming wires use the simulated procedure, but the outgoing wire uses the hybrid procedure. Instead, we let all such outgoing wires w use the simulated procedure for outgoing wires of Algorithm 3.

Claim. Under the t-instance security of MR^{H} (Definition 14), $\mathbf{Game}_{d.2}$ and $\mathbf{Game}_{d.3}$ are indistinguishable.

Proof. The claim is proven via a direct reduction to the $\mathsf{ExpMR}^{\mathsf{H}}(1^\lambda, B', B, \mathbf{salt})$ security game, with $\mathbf{salt} = (\gamma)_{\gamma \in \Gamma(C)}$: the procedure for an outgoing wire w is replaced by a call $(\Delta'_{d_w}, \tilde{\ell}_w, \tilde{L}_w)$ to $\mathsf{ExpMR}^{\mathsf{H}}(1^\lambda, B', B, \mathbf{salt})$. Answering the queries as in $\mathsf{ExpMR}_1^{\mathsf{H},\mathsf{H}'}(1^\lambda, B', B, \mathbf{salt})$ yields exactly $\mathbf{Game}_{d.3}$, and answering as in $\mathsf{ExpMR}_0^{\mathsf{H},\mathsf{H}'}(1^\lambda, B', B, \mathbf{salt})$ yields exactly $\mathbf{Game}_{d.2}$ using the perfect correctness of MR. ■

We now finish the proof of Lemma 21. First, observe that $\mathbf{Game}_{d.3}$ is exactly $\mathbf{Game}_{d+1}$. Second, $\mathbf{Game}_D$ is exactly the simulator of Algorithm 3 (and the wire values x_w for all wires w at depth $\mathsf{depth}(w) \geq D$ are the output wire values $z = (z_o)_{O(C)}$, which are passed as input to Sim). This concludes the proof. ■

Acknowledgements. We thank the anonymous reviewers for helpful comments and suggestions that improved the presentation and results in the paper, in particular on an optimization that improved the rate of the scheme. Geoffroy Couteau, Aditya Hegde, and Naman Kumar acknowledge the support of the French Agence Nationale de la Recherche (ANR) under grant ANR-20-CE39-0001 (project SCENE) and under the France 2030 ANR Project ANR-22-PECY-003 SecureCompute. This work is supported by ERC grant OBELiSC (101115790). Carmit Hazay acknowledges the support of the United States-Israel Binational Science Foundation (BSF) through Grant No. 2020277. Aditya Hegde was supported by NSF CNS-1814919, NSF CAREER 1942789, and the Johns Hopkins University Catalyst Award.

References

AIK11. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

AMN+18. Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 543–574, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

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: 24th Conference on Computer and Communications Security*, pages 2105–2122, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

BCM+24. Dung Bui, Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Fast public-key silent OT and more from constrained Naor-Reingold. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 88–118, Zurich, Switzerland, May 26–30, 2024. Springer, Heidelberg, Germany.

BCP03. Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 37–54, Taipei, Taiwan, November 30 – December 4, 2003. Springer, Heidelberg, Germany.

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 Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

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, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

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, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 163–193, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

BLLL23. Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 3–34, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

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: 23rd Conference on Computer and Communications Security*, pages 565–577, Vienna, Austria, October 24–28, 2016. ACM Press.

BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany.

CHHK25. Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. omega(1/lambda)-rate boolean garbling scheme from generic groups. In *Advances in Cryptology - CRYPTO 2025*, 2025.

CM21. Geoffroy Couteau and Pierre Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 842–870, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

CMPR23. Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 194–224, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

CNs07. Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 573–590, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

CS02. Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

DJN10. Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9:371–385, 2010.

Ds17. Jack Doerner and abhi shelat. Scaling ORAM for secure computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 523–535, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

FKN94. Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th Annual ACM Symposium on Theory of Computing*, pages 554–563, Montréal, Québec, Canada, May 23–25, 1994. ACM Press.

GGM86. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

GJM03. Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In Matt Blaze, editor, *FC 2002: 6th International Conference on Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 120–135, Southampton, Bermuda, March 11–14, 2003. Springer, Heidelberg, Germany.

Hea24. David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 3–31, Zurich, Switzerland, May 26–30, 2024. Springer, Heidelberg, Germany.

HKN24. David Heath, Vladimir Kolesnikov, and Lucien K. L. Ng. Garbled circuit lookup tables with logarithmic number of ciphertexts. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 185–215, Zurich, Switzerland, May 26–30, 2024. Springer, Heidelberg, Germany.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

IKNP03. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

ILL25. Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling from homomorphic secret sharing. In *Advances in Cryptology - CRYPTO 2025*, 2025.

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

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

MORS24. Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret-sharing. In *Theory of Cryptography Conference – TCC 2024*, 2024. https://eprint.iacr.org/2024/820.

MORS25. Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. In *Advances in Cryptology - CRYPTO 2025*, 2025.

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, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

Roy22. Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 657–687, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

RS21. Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

Ser24. Sacha Servan-Schreiber. Constrained pseudorandom functions for inner-product predicates from weaker assumptions. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part II*, volume 15485 of *Lecture Notes in Computer Science*, pages 232–265. Springer, 2024.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

A Continued Preliminaries

Notation. Throughout this paper, we let G denote the garbler, and E denote the evaluator. We use the notation $\langle x \rangle$ for additive (or subtractive) shares of x. Since this sharing is frequently between a garbler and an evaluator, we will use $\langle x \rangle_{\mathsf{G}}$ to denote the garbler's share of x and $\langle x \rangle_{\mathsf{E}}$ to denote the evaluator's share of x. Given an integer B, we denote by $[B]$ the set $\{0, \cdots, B\}$, by $[\pm B]$ the set

$\{-B, -B+1, \cdots, 0, \cdots, B-1, B\}$, and by $[B]^*, [\pm B]^*$ the sets sets $[B], [\pm B]$ without 0. Given an integer x, we write $[x \bmod B]$ to denote the integer remainder of the euclidean division of x by B. Given a distribution $\mathcal{D}$ (resp. a set S), we write $x \leftarrow\!\!\$\ \mathcal{D}$ (resp. $x \leftarrow\!\!\$\ S$) to denote that x is sampled from $\mathcal{D}$ (resp. that x is sampled uniformly over S). Given a parameter $p \in (0,1)$, we let $\mathsf{Ber}(p)$ denote the Bernoulli distribution that outputs 0 with probability $1-p$ and 1 with probability p.

A.1 Arithmetic Garbled Circuits

We consider a model of arithmetic circuits consisting of addition, subtraction and multiplication gates with fan-in two. A circuit consists of a series of connected gates that computes some function over its input domain $\mathcal{D}$. We refer to the *size* of a circuit, denoted by $|C|$, as the number of gates in the circuit.
Our model and definitions are heavily borrowed from [BLLL23] and [MORS24]. In this work we will be considering two models of circuit computation, defined over different domains. We refer to a valid input as *admissible*; the output value of the circuit is defined only over admissible inputs.

Modular Arithmetic Computation. In this model, the domain of the circuit is defined to be the ring $\mathbb{Z}_N$. All input and intermediate computations are carried out over $\mathbb{Z}_N$. If C has n input wires, every input vector $\mathbf{x} \in \mathbb{Z}_N^n$ is considered admissible.

Bounded Integer Computation. In this model, the domain of the circuit is defined to be $\mathbb{Z}$ and there exists a bound $B = B(\lambda) \in \mathbb{N}$ that bounds the magnitude of circuit wire values. If C has n input wires, then an input vector $\mathbf{x} \in \mathbb{Z}^n$ is considered admissible with respect to bound B if the inputs, outputs, and every intermediate wire value obtained during the evaluation of C are in the range $[-B, B]$.

Arithmetic Garbling

Definition 22 (Arithmetic Garbled Circuit, adapted from [MORS24]). *Let $\mathbb{Z}_N$-modular arithmetic computation (resp. B-bounded integer computation) be the model of computation. A garbling scheme for a family of circuit classes $\mathcal{C} = \{C_\lambda\}_{\lambda \in \mathbb{N}^*}$ for modular arithmetic computation (resp. B-bounded integer computation) with label space $\mathcal{L} = \mathcal{L}(\lambda)$ is a pair of PPT algorithms $\mathsf{AGC} = (\mathsf{AGC.Garble}, \mathsf{AGC.Eval})$ with the following syntax and properties:*

- $\mathsf{Garble}(1^\lambda, C)$*: On input a security parameter 1^λ and a circuit $C \in \mathcal{C}_\lambda$ with n inputs,* Garble *outputs n key pairs $(k_0^i, k_1^i)_{i\in[n]} \in \mathcal{L}$ and a garbled circuit $\widehat{C}$.*
- $\mathsf{Eval}((L_i)_{i\in[n]}, \widehat{C})$*: On input n input labels $(L_i)_{i\in[n]} \in \mathcal{L}^n$ and a garbled circuit $\widehat{C}$,* Eval *outputs a value $y \in \mathbb{Z}_N$ (resp. $y \in [B]$).*
- ***Correctness.*** *A garbling scheme is correct if there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $\lambda \in \mathbb{N}$, every circuit $C \in \mathcal{C}_\lambda$ with n inputs, and every $x_1, \ldots, x_n \in \mathcal{X}^n$ where $\mathcal{X}$ is the set of admissible inputs of C, the following holds:*

$$\Pr\left[\mathsf{Eval}((L_i)_{i\in[n]}, \widehat{C}) =_{\mathbb{Z}} C(x_1, \ldots, x_n) : \begin{array}{l}((k_0^i, k_1^i)_{i\in[n]}, \widehat{C}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C) \\ L_i \leftarrow k_0^i \cdot x_i + k_1^i\end{array}\right] \leq \mathsf{negl}(\lambda)$$

- ***Privacy.*** *A garbling scheme is secure if there exists a PPT simulator* Sim *such that for all sequences of circuits $\{C_\lambda\}_{\lambda\in\mathbb{N}}$, where $C_\lambda \in \mathcal{C}_\lambda$ with $n = n(\lambda)$ inputs, and every admissible sequence of inputs $\{(x_{1,\lambda}, \ldots, x_{n,\lambda}\}_{\lambda\in\mathbb{N}}$, the following holds:*

$$\left\{\mathsf{Sim}(1^\lambda, C_\lambda, y) : y \leftarrow C_\lambda(x)\right\} \approx_c \left\{((L_i)_{i\in[n]}, \widehat{C}) : \begin{array}{l}((k_0^i, k_1^i)_{i\in[n]}, \widehat{C}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C) \\ L_i \leftarrow k_0^i \cdot x_i + k_1^i\end{array}\right\}$$

Definition 23 (Rate of Arithmetic Garbled Circuit, adapted from [MORS24]). *Let $\mathcal{C}$ be a class of arithmetic circuits, let* AGC *be an arithmetic garbling scheme for $\mathcal{C}$ with $\mathbb{Z}_N$-modular arithmetic computation (resp. B-bounded computation), and let $\ell = \log(N)$ (resp. $\ell = \log(2B+1)$). The rate of* AGC *for $\mathcal{C}$ is the quantity*

$$\mathsf{rate} = \liminf_{C\in\mathcal{C}} \min_x \frac{(|C| + n)\ell}{|\widehat{C}| + \sum_i \mathsf{size}(k_0^i \cdot x_i + k_1^i)}$$

where the minimum is taken over admissible inputs to the circuit $\mathcal{C}$, and the limit infimum is taken over all circuits in $\mathcal{C}$.

B Homomorphic Secret Sharing

We start by recalling the standard definition of homomorphic secret sharing, as well as of Restricted Multiplication Straight-line (RMS) programs which is the common model of computation in the context of HSS. Our definitions have been taken from [CMPR23].

Definition 24 (Homomorphic Secret Sharing). Let λ be a security parameter. A Homomorphic Secret Sharing (HSS) scheme for a class of programs $\mathcal{P}$ which is defined over a ring $\mathcal{R}$ and has input space $\mathcal{I} \subseteq \mathcal{R}$ consists of three PPT algorithms $(\mathsf{Setup}, \mathsf{Input}, \mathsf{Eval})$ such that:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1))$: On input the security parameter λ, the setup algorithm outputs a public key pk and a pair of evaluation keys $(\mathsf{ek}_0, \mathsf{ek}_1)$.
- $\mathsf{Input}(\mathsf{pk}, x) \to (\mathsf{I}_0, \mathsf{I}_1)$: On input the public key pk and an input $x \in \mathcal{I}$, the input algorithm outputs a pair of input information $(\mathsf{I}_0, \mathsf{I}_1)$.
- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma = (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$: On input a party index $\sigma \in \{0,1\}$, an evaluation key ek_σ, a vector of ρ input values $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$, and a program $P \in \mathcal{P}$, the evaluation algorithm outputs the party σ's corresponding share of the output y_σ.

We require scheme to satisfy the following two properties:

- **Correctness.** For any security parameter $\lambda \in \mathbb{N}$, and any program $P \in \mathcal{P}$ with input space $\mathcal{I} \subseteq \mathcal{R}$, we have:
$$\Pr\left[y_0 - y_1 = P(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda) \ ,$$
where the probability is taken over $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{I}_0^{(i)}, \mathsf{I}_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$ for $i \in [\rho]$, and $y_\sigma \leftarrow \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P)$, for $\sigma \in \{0,1\}$.
- **Security.** For any PPT adversaries $\mathcal{A}, \mathcal{A}'$, and any bit $\sigma \in \{0,1\}$ the following value should be negligible in λ:
$$\left| \Pr\left[b' = b : \begin{array}{l} (x_0, x_1, \mathsf{state}) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda) \\ b \xleftarrow{\$} \{0,1\} \\ (\mathsf{I}_0, \mathsf{I}_1) \leftarrow \mathsf{Input}(x_b) \\ b' \leftarrow \mathcal{A}'(\mathsf{state}, \mathsf{pk}, \mathsf{ek}_\sigma, \mathsf{I}_\sigma) \end{array} \right] - \frac{1}{2} \right| \ .$$

We now remind the definition of Restricted Multiplication Straight-line (RMS) programs. RMS programs form a class of programs which encompasses polynomial-size branching programs and therefore NC^1 circuits. In an RMS program, the multiplication is restricted to happen between an input value and an intermediate value of the computation (so-called "memory" value).

Definition 25 (RMS Programs). *An RMS program with magnitude bound B is defined as a sequence of the instructions as follows:*

- $\mathsf{ConvertInput}(\mathsf{I}^x) \to \mathsf{M}^x$*: Loads an input x into memory.*
- $\mathsf{Add}(\mathsf{M}^x, \mathsf{M}^y) \to \mathsf{M}^{x+y}$*: Adds two memory values.*
- $\mathsf{Mul}(\mathsf{I}^x, \mathsf{M}^y) \to \mathsf{M}^{x \cdot y}$*: Multiplies an input value and a memory value to produce a memory value of their product.*
- $\mathsf{Output}(\mathsf{M}^x, n) \to x \bmod n$*: Outputs a memory value with respect to a modulus $n < B$.*

B.1 Template for RMS Programs

We now outline a template construction for HSS in the context of RMS programs, adapted from Section 4 of [CMPR23].

Definition 26 (HSS Following the RMS Template). A homomorphic secret sharing scheme $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ following the RMS template is scheme as defined in Definition 24 with an additional algorithm MemGen which serves to produce memory values as follows:

- $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, x) \to \mathsf{M}_\sigma$: On input a party index $\sigma \in \{0,1\}$, an evaluation key ek_σ, and an input $x \in \mathcal{I}$, the memory generator algorithm outputs a memory value M_σ.

Moreover, the Eval algorithm proceeds with sub-routines following the RMS operations $\mathsf{ConvertInput}$, Add, Mul, Output as follows:

- $\mathsf{Eval}(\sigma, \mathsf{ek}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$: On input a party index $\sigma \in \{0,1\}$, an evaluation key ek_σ, a vector of ρ input values $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$, and an RMS program P, this algorithm follows the instructions of P and processes them as follows:
  - $\mathsf{ConvertInput}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma^x) \to \mathsf{M}_\sigma^x$: This algorithm simply uses the MemGen and Mult algorithms as follows:
    - Run $\mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, 1) \to \mathsf{M}_\sigma^1$.
    - Run $\mathsf{Mult}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}_\sigma^x, \mathsf{M}_\sigma^1) \to \mathsf{M}_\sigma^x$.
  - $\mathsf{Add}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}^x, \mathsf{M}^y) \to \mathsf{M}^{x+y}$: This algorithm directly adds the given memory values of x and y. Namely, $\mathsf{M}_\sigma^{x+y} = \mathsf{M}_\sigma^x + \mathsf{M}_\sigma^y$.
  - $\mathsf{Mul}(\sigma, \mathsf{ek}_\sigma, \mathsf{I}^x, \mathsf{M}^y) \to \mathsf{M}^{x \cdot y}$: It multiplies an input value I_x and a memory value M_y and outputs a memory value of $x \cdot y$. The template does not impose any non-black box requirement on this algorithm.
  - $\mathsf{Output}(\sigma, \mathsf{M}^x, n) \to x \bmod n$: It uses M^x to output $x_\sigma \bmod n$.

Correctness and security properties are defined as in Definition 24.

Any HSS following the RMS template as defined above satisfies the following lemma, which states that one can evaluate share of $z \cdot P(x^{(1)}, \ldots, x^{(\rho)})$ using only a memory value of z (instead of an input value) together with the input values of the rest of variables $(x^{(1)}, \ldots, x^{(\rho)})$. This property is key to our garbling scheme, as it allows non-interactive evaluations of pseudorandomly generated shares.

Lemma 27. *Let* $\mathsf{HSS} = (\mathsf{Setup}, \mathsf{Input}, \mathsf{MemGen}, \mathsf{Eval})$ *be scheme following the RMS template. There exists an extended evaluation algorithm* $\mathsf{ExtEval}$*:*

- $\mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P) \to y_\sigma$*: On input a party index* $\sigma \in \{0,1\}$*, an evaluation key* ek_σ*, a single memory value* M_σ*, a vector of* ρ *input values* $(\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)})$*, and an RMS program* P*, return a value* y_σ *such that the following holds.*

For any security parameter $\lambda \in \mathbb{N}$ *and any RMS program* P*, we have:*

$$\Pr\left[y_0 - y_1 = z \cdot P(x^{(1)}, \ldots, x^{(\rho)})\right] \geq 1 - \mathsf{negl}(\lambda) \ , \tag{3}$$

where the probability over $(\mathsf{pk}, (\mathsf{ek}_0, \mathsf{ek}_1)) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{I}_0^{(i)}, \mathsf{I}_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, x^{(i)})$, $\mathsf{M}_\sigma \leftarrow \mathsf{MemGen}(\sigma, \mathsf{ek}_\sigma, z)$*, and* $y_\sigma \leftarrow \mathsf{ExtEval}(\sigma, \mathsf{ek}_\sigma, \mathsf{M}_\sigma, (\mathsf{I}_\sigma^{(1)}, \ldots, \mathsf{I}_\sigma^{(\rho)}), P)$*, for* $\sigma \in \{0,1\}, i \in [\rho]$.

The proof of the above lemma is detailed in the supplementary material in Section B of [CMPR23]. In the HSS scheme from Paillier-ElGamal, the $\mathsf{Input}(\mathsf{pk}, x^{(i)})$ is a ciphertext of $x^{(i)}$.

B.2 HSS based on Paillier-ElGamal

In this work, we will be working heavily with the *Paillier-ElGamal Cryptosystem* ([CS02], [BCP03]), which is essentially the ElGamal cryptosystem over the group $(\mathbb{Z}/N^2\mathbb{Z})^\times$. We borrow the description of Paillier-ElGamal from Appendix B of [OSY21]. We will be working with a mild variant of Paillier-ElGamal (2) with the only semantic difference that the secret key is sampled from $[2^{2\lambda}]$ instead of $[N^2]$; the rest of the description is identical to that of [OSY21]. The security of this cryptosystem can be easily proven based on the DCR and SEI assumptions.

Definition 28 (S-bounded Short Exponent Indistinguishability Assumption). *Let* p *and* q *be primes of fixed length* $n(\lambda)$ *where* n *is some polynomial function of* λ *and let* $N = pq$ *be a Blum Integer. Consider a group* $\mathbb{G}$ *of unknown order* $N\phi(N)$ *with generator* g*. Let* $S < N^2$ *be an integer. The* S*-bounded Short Exponent Indistinguishability (SEI) assumption holds if for all PPT* $\mathcal{A}$*,*

$$\left|\Pr\left[z \xleftarrow{\$} [0, S-1] : \mathcal{A}(g^z) = 1\right] - \Pr\left[z \xleftarrow{\$} [N^2] : \mathcal{A}(g^z) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

In this work, we set $S = 2^{2\lambda}$.

Lemma 29 (Paillier-ElGamal Cryptosystem). *The Paillier-ElGamal cryptosystem is a CPA-secure encryption scheme iff the DCR assumption holds.*

Lemma 30 (Paillier-ElGamal Cryptosystem with Short Exponent). *The construction of 2 is a CPA-secure encryption scheme iff the DCR and the SEI assumptions hold.*

Proof. Consider an adversary $\mathcal{A}$ which can distingush between short-exponent Paillier-ElGamal ciphertexts with noticeable probability. We will show that, assuming the security of 'regular' Paillier-ElGamal, we can construct a distinguisher that can distinguish between elements of $N\phi(N)$ with small and large exponent. Our distinguisher works as follows. Given the challenge (g, g^x) (where x is sampled from either $[0, S-1]$ or $[N^2]$), the distinguisher D publishes $\mathsf{pk} = g^d$ and sends it to $\mathcal{A}$. In return, $\mathcal{A}$ responds with (m_0, m_1), and D chooses $r \xleftarrow{\$} [N^2]$ and responds with $(g^r, \mathsf{pk}^r(1+N)^{m_1})$ as the challenge ciphertext. If $\mathcal{A}$ responds with 0 then D outputs 1, otherwise D outputs 0. Since the ciphertext always corresponds to a correct 'left'-encryption of the adversary's challenge, $\mathcal{A}$ outputs 0 with noticeable probability if $x \xleftarrow{\$} [0, S-1]$ and 1 otherwise. Thus, D breaks the small exponent indistinguishability game, but this contradicts the SEI assumption. ■

Pailler-ElGamal Cryptosytem with Short Exponent

$\mathsf{PaillierEG.Gen}(1^\lambda)$

1 : Sample $(N, p, q) \leftarrow \mathsf{GenPQ}(1^\lambda)$.
2 : Sample a random $g' \leftarrow [N^2]$.
3 : Let $g = (g')^{2N} \mod N^2$.
4 : Sample $d \xleftarrow{\$} [2^{2\lambda}]$.
5 : **return** $(N, g, \mathsf{pk} = g^d \mod N^2, \mathsf{sk} = d)$

$\mathsf{PaillierEG.Enc}(\mathsf{pk}, x)$

1 : Sample $r \xleftarrow{\$} [N^2]$.
2 : **return** $\mathsf{ct} = (g^r, \mathsf{pk}^r(1+N)^x)$

$\mathsf{PaillierEG.Dec}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1))$

1 : Let $\mathsf{ct}' = \mathsf{ct}_1(\mathsf{ct}_0)^{-\mathsf{sk}} \mod N^2$.
2 : **return** $x = (\mathsf{ct}' - 1)/N$.

Fig. 2: The Paillier-ElGamal Cryptosystem with Short Exponent.

C Constant-Rate Arithmetic Garbling over Small Integers

C.1 Technical Overview: Constant-Rate Arithmetic Garbling over Small Integers

The starting point of our scheme is the recent work of [MORS24], which constructs constant-rate garbling for circuits with B-bounded integer arithmetic from the Decisional Composite Residuosity Assumption (DCR) using techniques derived from Homomorphic Secret Sharing (HSS). We begin by recalling the construction of [MORS24].

Homomorphic Secret Sharing Our techniques make heavy use of Homomorphic Secret Sharing (HSS). Informally, 2-party HSS allows parties to share an input into shares $x = (x_0, x_1)$ such that any party P_i can evaluate any arbitrary function f from a class $\mathcal{F}$ of admissible functions and obtain $y_i = \mathsf{Eval}(i, f, x_i)$. The correctness requirement states that $y_0 + y_1 = f(x)$, and security requires each share to computationally hide x. HSS is implied by a number of different standard assumptions in the literature ([BGI16], [OSY21], [CM21]), and a plethora of different constructions are known

for a variety of circuit classes. In this work, we will be interested in HSS schemes where the Share algorithm supports a public-key style setup, ie. each party publishes a public key, and consequently, shared public and evaluation keys can be derived non-interactively. We (very informally) term such cryptosystems 'HSS-compatible'.

Arithmetic Garbling from HSS At a high level, the garbling scheme of [MORS24] follows the template introduced in Section 2.2, where the scheme utilizes HSS to allow the parties to non-interactively compute shares of cross terms. In particular, the garbler begins by generating an HSS-compatible key pair $(\mathsf{sk}, \mathsf{pk})$. Then by setting $\Delta = \mathsf{sk}$, the garbler can provide $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk}^{-1})$ as one-time garbling cost, along with $\mathsf{Enc}_{\mathsf{pk}}(\langle x \rangle_{\mathsf{G}})$, $\mathsf{Enc}_{\mathsf{pk}}(\langle y \rangle_{\mathsf{G}})$ and $\mathsf{Enc}_{\mathsf{pk}}(\langle \mathsf{sk} \cdot x \rangle_{\mathsf{G}})$ for each gate as garbling material. Recall that the distributed discrete log algorithm (DDlog) allows parties to non-interactively compute, using as input an HSS-compatible encryption of A and a subtractive share of $\mathsf{sk} \cdot B$, sharings of $\mathsf{sk} \cdot AB$; full details can be found in Section 3.4 of [MORS24]. With this in place, the parties can now do the following:

1. Run $\mathsf{DDlog}(\mathsf{Enc}_{\mathsf{pk}}(\langle y \rangle_{\mathsf{G}}), \langle \mathsf{sk} \cdot x \rangle) \to \langle \mathsf{sk} \cdot x \cdot \langle y \rangle_{\mathsf{G}} \rangle$, $\mathsf{DDlog}(\mathsf{Enc}_{\mathsf{pk}}(\langle x \rangle_{\mathsf{G}}), \langle y \rangle) \to \langle \mathsf{sk} \cdot y \cdot \langle x \rangle_{\mathsf{G}} \rangle$, and $\mathsf{DDlog}(\mathsf{Enc}_{\mathsf{pk}}(\langle \mathsf{sk} \cdot x \rangle_{\mathsf{G}}), \langle y \rangle) \to \langle \mathsf{sk} \cdot y \cdot \langle \mathsf{sk} \cdot x \rangle_{\mathsf{G}} \rangle$.
2. Remove the sk factor in the sharings by running $\mathsf{DDlog}(\mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk}^{-1}), \langle \mathsf{sk} \cdot x \cdot \langle y \rangle_{\mathsf{G}} \rangle) \to \langle x \cdot \langle y \rangle_{\mathsf{G}} \rangle$, $\mathsf{DDlog}(\mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk}^{-1}), \langle \mathsf{sk} \cdot y \cdot \langle x \rangle_{\mathsf{G}} \rangle) \to \langle y \cdot \langle x \rangle_{\mathsf{G}} \rangle$, and $\mathsf{DDlog}(\mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk}^{-1}), \langle \mathsf{sk} \cdot y \cdot \langle \mathsf{sk} \cdot x \rangle_{\mathsf{G}} \rangle) \to \langle y \cdot \langle \mathsf{sk} \cdot x \rangle_{\mathsf{G}} \rangle$.

The garbler can then set his share of z to be $\langle x \rangle_{\mathsf{G}} \cdot \langle y \rangle_{\mathsf{G}} - \langle x \cdot \langle y \rangle_{\mathsf{G}} \rangle_{\mathsf{G}} - \langle y \cdot \langle x \rangle_{\mathsf{G}} \rangle_{\mathsf{G}}$ and vice-versa. Similarly, the parties can compute shares of $\mathsf{sk} \cdot z$, maintaining the invariant. The procedure is continued till the global output gates have been reached, at which point the garbler can send its (precomputed) shares of output wires to reconstruct.
The above scheme works assuming the KDM-security of any HSS-compatible encryption scheme. When instantiated using Damgard-Jurik ([RS21] shows that this scheme is HSS-compatible), the asymptotic rate of the scheme comes out to be approximately 1/3. [MORS24] also carries a number of optimizations to obtain a scheme with a rate arbitrarily close to 1, but the bare template remains the same. We stop and make a note of two observations here, motivated by [MORS24]'s use of the Damgard-Jurik cryptosystem.

- First, note that the Damgard-Jurik cryptosystem is parametrized by two numbers: a constant $\zeta > 2$ and an RSA modulus N, setting the plaintext space to be $\mathbb{Z}/N^{\zeta}\mathbb{Z}$ and the ciphertext space to be $\mathbb{Z}/N^{\zeta+1}\mathbb{Z}$. In particular, to preserve security, the scheme requires the share sizes to scale as $\mathsf{poly}(N)$.
- Secondly, the DDlog algorithm of [RS21] has a probability of failure with large plaintexts, restricting garbling to B-bounded integers where $B \approx N^{\zeta-1}/2^{\kappa}$ with a statistical security parameter κ.

Small-Integer Garbling with Paillier-ElGamal Our first contribution is a constant-rate, B-bounded arithmetic garbling scheme based on the circular security of the Paillier-ElGamal cryptosystem with a short exponent that supports small integers (of size $B \approx 2^{O(\lambda)}$). Our motivation is simple: an immediate way to overcome the problems inherent to the Damgard-Jurik garbling implementation is to instantiate HSS-based garbling from a different HSS-compatible encryption scheme. [OSY21] shows a promising construction of HSS from the Paillier-ElGamal Cryptosystem (Figure 2). However, simply instantiating the above HSS template using Paillier-ElGamal does not lead to any improvement in domain size.

- The plaintext space of Paillier-ElGamal is $\mathbb{Z}/N\mathbb{Z}$, which is still proportional to the size of the RSA modulus. Furthermore, this space also needs to be able to contain authenticated shares of wire values, i.e., $\langle \mathsf{sk} \cdot x \rangle$ for a wire x, where sk is chosen from $[N^2]$.
- The ciphertext space is $\mathbb{Z}/N^2\mathbb{Z}$, which is too large – simply instantiating the scheme above leads to a far worse rate of 1/6.

With this in mind, we outline a series of techniques that lead to an arithmetic garbling scheme for integers of size $2^{O(\lambda)}$ with rate (close to) 1/2.

Reducing the size of the plaintext space. We can alleviate the first issue by additionally relying on a well-known variant of the Paillier-ElGamal cryptosystem with *short exponent*. Informally

speaking, the S-bounded *Small Exponent Indistinguishability Assumption* (SEI, Definition 28) states that given a group $\mathbb{G}$ and generator g, a PPT adversary cannot distinguish between random g^a and g^b where $a \xleftarrow{\$} [0, S-1]$ and $b \xleftarrow{\$} \overline{\mathbb{G}}$. Applying this to the Paillier-ElGamal group of unknown order $N\phi(N)$, we can ensure that the encryption scheme remains secure even if sk is bounded above by a small constant, say $2^{2\lambda}$.

While the previous discussion allows us to reduce the size of sk, letting the authenticated share $\langle sk \cdot x \rangle$ fit into the plaintext space, the underlying problem with the size of the plaintext space being too large remains. However, we can, at this point, rely on an observation made in previous works (including [OSY21]) to reduce the bit-length of both shares to size $O(\lambda)$. The core idea is this: a single invocation of DDlog returns substractive shares of the output over $\mathbb{Z}/N\mathbb{Z}$. If we want to convert these to shares over the *integers*, however, then assuming that the output $\langle x \rangle$ of DDlog is randomly distributed[8], restricting x to be small immediately implies that the shares are *already* over $\mathbb{Z}$ with high probability. In particular, if x is bounded by $N/2^\kappa$ for some statistical parameter κ, the formed shares will be over $\mathbb{Z}$ with probability except 2^κ; if we restrict B to, say, $2^{2\lambda}$, this probability is negligible. Our key observation is that with this in place, both the garbler and the evaluator can perform a round of modular reduction over $2^{O(\lambda)}$ on their shares and still obtain valid (substractive) shares of x with negligible correctness error. Concretely, if the reduction is performed modulo $2^{3\lambda}$, then the probability of shares being malformed is negligible. Thus, the bitlength of the reduced shares is $\approx 3\lambda$. Setting this to be B, we obtain $B = 2^{3\lambda}$, as wished.

Combining the two approaches, our scheme achieves a quantitative reduction of share size (and hence, B). In practice, this allows us to set B as low as 256 bits, an improvement from the previously known best value of ≈ 4000 bits.

Choosing garbler shares pseudorandomly. The second problem is harder to solve. It is clear that in order to achieve a competitive rate, the garbling material cannot consist of ciphertexts. We make two key observations here: first, HSS allows parties to evaluate more than just a single DDlog instance; it allows a non-interactive share conversion for *any* NC^1 function of an encrypted input, and second the garbler's shares contain no information about the secret (by definition, since they must be formed before the evaluator receives the garbled circuit) and hence can be defined to be pseudorandom.

We construct our scheme as follows. Let id_x be the label referring to the identity of the wire carrying the value x; this value is completely independent of x. The garbler can sample a key k for a low-depth PRF[9] and set for any wire id_x (which we can colloquially refer to as x) its shares $(\langle x \rangle_\mathsf{G}, \langle \mathsf{sk} \cdot x \rangle_\mathsf{G}) := \mathsf{PRF}_k(\mathsf{id}_x)$. For convenience, we can view $\mathsf{PRF}_k(w) = (\mathsf{PRF}^1_k(w), \mathsf{PRF}^2_k(w))$. As before, parties maintain the invariant $(\langle x \rangle, \langle sk \cdot x \rangle)$ for every wire x. However, note now that addition gates are no longer free since the garbler's share of the output of an addition gate $z = x \pm y$ is defined to be $\mathsf{PRF}_k(\mathsf{id}_z) \neq \mathsf{PRF}_k(\mathsf{id}_x) + \mathsf{PRF}_k(\mathsf{id}_y)$ except with negligible probability. Thus, the garbler sends a fresh *shift*

$$(\langle x \rangle_\mathsf{G}, \langle \mathsf{sk} \cdot x \rangle_\mathsf{G}) \pm (\langle y \rangle_\mathsf{G}, \langle \mathsf{sk} \cdot y \rangle_\mathsf{G}) - \mathsf{PRF}_k(\mathsf{id}_z)$$

for each z, which the evaluator adds to its share to maintain the invariant.

Now assume the parties encounter a multiplication gate with input wires x and y and output wire z. The garbler can then provide the evaluator with global ciphertexts $\mathsf{Enc}_\mathsf{pk}(k)$ and $\mathsf{Enc}_\mathsf{pk}(\mathsf{sk} \cdot k)$. Assume that the chosen PRF has depth $O(\log(\lambda))$. The key ingredient of our construction is a procedure introduced in [CMPR23] that lets two parties, given ciphertexts $\mathsf{Enc}_\mathsf{pk}(A), \mathsf{Enc}_\mathsf{pk}(\mathsf{sk} \cdot A)$ for a PRF key A and a substractive input sharing $(\langle B \rangle, \langle \mathsf{sk} \cdot B \rangle)$ over $\mathbb{Z}$, use a special form of HSS (called *HSS with simulatable memory shares* in [CMPR23]) to compute substractive shares of $(\langle B \cdot \mathsf{PRF}_A(c) \rangle, \langle \mathsf{sk} \cdot B \cdot \mathsf{PRF}_A(c) \rangle)$ over $\mathbb{Z}$ for any c of their choice. We refer the reader to Appendix B.1 for an outline of the method of [CMPR23].

In particular, the parties can use the ciphertexts of k to compute $\langle y \cdot \mathsf{PRF}^1_k(\mathsf{id}_x) \rangle$, $\langle \mathsf{sk} \cdot y \cdot \mathsf{PRF}^2_k(\mathsf{id}_x) \rangle$ and $\langle x \cdot \mathsf{PRF}^1_k(\mathsf{id}_y) \rangle$, which are the required cross-terms in Equation (1) and Equation (2), and hence can set their shares appropriately. However, note that with this in place, there is also another invariant: the garbler's share of $(z, \mathsf{sk} \cdot z) = (x \cdot y, \mathsf{sk} \cdot x \cdot y)$ is necessarily defined to be $\mathsf{PRF}_k(\mathsf{id}_z)$. Let $(\langle z \rangle'_\mathsf{G}, \langle \mathsf{sk} \cdot z \rangle'_\mathsf{G})$

[8] In general, this is non-trivial; however the parties can ensure it to be so by adding, say, a PRG output with range $[N]$ to their obtained shares.

[9] The construction of such a PRF is known from several standard assumptions, including DDH and DCR.

be the *computed* share of the garbler immediately after the HSS evaluation, defined as

$$\langle z\rangle'_{\mathsf{G}} := \mathsf{PRF}^1_k(\mathsf{id}_x)\cdot\mathsf{PRF}^1_k(\mathsf{id}_y) - \langle x\cdot\mathsf{PRF}^1_k(\mathsf{id}_y)\rangle_{\mathsf{G}} - \langle y\cdot\mathsf{PRF}^1_k(\mathsf{id}_x)\rangle_{\mathsf{G}} \tag{4}$$

$$\langle \mathsf{sk}\cdot z\rangle'_{\mathsf{G}} := \mathsf{PRF}^2_k(\mathsf{id}_x)\cdot\mathsf{PRF}^2_k(\mathsf{id}_y) - \langle \mathsf{sk}\cdot x\cdot\mathsf{PRF}^1_k(\mathsf{id}_y)\rangle_{\mathsf{G}} - \langle y\cdot\mathsf{PRF}^2_k(\mathsf{id}_x)\rangle_{\mathsf{G}} \tag{5}$$

Contrast this with the invariant, which sets the *actual* share $(\langle z\rangle_{\mathsf{G}}, \langle \mathsf{sk}\cdot z\rangle_{\mathsf{G}}) := \mathsf{PRF}_k(\mathsf{id}_z)$. To maintain this, the garbler sends two (plaintext) *shifts* $\langle z\rangle_{\mathsf{G}} - \langle z\rangle'_{\mathsf{G}}$ and $\langle \mathsf{sk}\cdot z\rangle_{\mathsf{G}} - \langle \mathsf{sk}\cdot z\rangle'_{\mathsf{G}}$ as garbling material which the evaluator can add to its own shares to adjust them appropriately. The resulting invariant is shares of $(z, \mathsf{sk}\cdot z)$ where the garbler's share is defined to be $\mathsf{PRF}_k(\mathsf{id}_z)$, and the evaluation can continue.

The per-gate garbling material is the size of two plaintexts, setting the scheme's rate close to 1/2. When instantiated with Paillier-ElGamal and the small exponent assumption, the scheme also supports small integers of size $2^{O(\lambda)}$.

We briefly mention the security of the above sketch requires a variant of circular security for the underlying Paillier-ElGamal cryptosystem, owing to the fact that the evaluator can see an encryption of the PRF key, which is used to pseudorandomly mask (the garbler's computed share of) $\mathsf{sk}\cdot z$. A more involved technical discussion can be found in Appendix C.2.

C.2 Small-Integer Garbling from Circular Security of Paillier-ElGamal with Constant Rate

HSS Share Reduction with Paillier-ElGamal The starting point of our construction for small-integer garbling with constant rate is a *share reduction* mechanism that allows parties to non-interactively perform homomorphic secret sharing from DCR that supports smaller share sizes. Recall from the discussion in Appendix C.1 that our techniques heavily rely on homomorphic secret sharing instantiated with the Paillier-ElGamal Cryptosystem with Small Exponent (henceforth referred to as simply Paillier-ElGamal). Natively, this form of HSS follows the HSS template for RMS programs described in Definition 26. In this setting, both parties maintain *Memory Shares* M^x of any intermediary value x required in the computation; these memory shares are of the form $\mathsf{M}^x := (\langle x\rangle, \langle \mathsf{sk}\cdot x\rangle)$ where the shares are interpreted as *subtractive shares* over the integers modulo N, where N is the RSA modulus associated with the Paillier-ElGamal instantiation. A full description of how HSS is constructed from Paillier-ElGamal can be found in Appendix B of [OSY21].

The key to our construction involves a technique that allows parties to reduce the size of intermediate memory shares from a subtractive sharing over N to a subtractive sharing over the integers modulo $2^{O(\lambda)}$. In this section, we introduce our (slightly modified) DDlog algorithm over reduced shares for Paillier-ElGamal.

Let $(N, g, k_{\mathsf{DDlogRS}})$ be a CRS where $(N, p, q) \xleftarrow{\$} \mathsf{GenPQ}(1^\lambda)$, $g = (g')^{2N} \mod N^2$ where $g' \xleftarrow{\$} [N^2]$, and k_{DDlogRS} is a randomly sampled (public) PRF key for a pseudorandom function $\mathsf{PRF}_{\mathsf{DDlogRS}} : \{0,1\}^\lambda \times \{0,1\}^{\mathsf{id}(\lambda)} \to [N]$. Let $x, y \in \{0,1\}^{k\lambda}$ be integers such that arithmetic is $2^{(k-1)\lambda}$-*bounded*, ie. xy does not exceed $2^{(k-1)\lambda}$ where $k > 1$. Assume that $(\mathsf{pk}, \mathsf{sk})$ is an honestly generated Paillier-ElGamal key pair. The $\mathsf{DDlogRS}$ algorithm then takes as input a Paillier-ElGamal ciphertext pair $c = \mathsf{Enc}_{\mathsf{pk}}(x), c' = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk}\cdot x)$ and subtractive memory shares $(\langle y\rangle, \langle \mathsf{sk}\cdot y\rangle)$ over the integers and outputs correct subtractive shares $(\langle xy\rangle, \langle \mathsf{sk}\cdot xy\rangle)$ over $[2^{k\lambda}]\times[2^{(k+2)\lambda}]$ with overwhelming probability. Here, we use $\sigma \in \{0,1\}$ to denote the party ID.

$\mathsf{DDlogRS}(N, k, \sigma, c = (c_0, c_1), c' = (c'_0, c'_1), (\langle y\rangle_\sigma, \langle \mathsf{sk}\cdot y\rangle_\sigma))$

1 : Define $(g_\sigma, g'_\sigma) := (c_1^{\langle y\rangle_\sigma}\cdot c_0^{-\langle \mathsf{sk}\cdot y\rangle_\sigma}, c_1'^{\langle y\rangle_\sigma}\cdot c_0'^{-\langle \mathsf{sk}\cdot y\rangle_\sigma})$
2 : Compute $a_\sigma, a'_\sigma < N$ such that $g_\sigma = a_\sigma + a'_\sigma N$
3 : Compute $b_\sigma, b'_\sigma < N$ such that $g'_\sigma = b_\sigma + b'_\sigma N$
4 : Set $z_\sigma := (a'_\sigma a_\sigma^{-1} \mod N) + \mathsf{PRF}_{\mathsf{DDlogRS}}(\mathsf{id}) \mod 2^{k\lambda}$
5 : Set $z'_\sigma := (b'_\sigma b_\sigma^{-1} \mod N) + \mathsf{PRF}_{\mathsf{DDlogRS}}(\mathsf{id}') \mod 2^{(k+2)\lambda}$
6 : **return** (z_σ, z'_σ)

Lemma 31. *The outputs* (z_σ, z'_σ) *of* $\mathsf{DDlogRS}$ *are correctly obtained subtractive shares of* $(xy, \mathsf{sk}\cdot xy)$.

Proof. We note that the only difference between our algorithm $\mathsf{DDlogRS}$ and the multiplication procedure outlined in Section B.1 of [OSY21] is that we perform a second reduction modulo $2^{k\lambda}$ and $2^{(k+2)\lambda}$ in Steps 4 and 5. Hence, the shares $A_\sigma := (a'_\sigma a_\sigma^{-1} \mod N) + \mathsf{PRF}_{\mathsf{DDlogRS}}(\mathsf{id})$ and $B_\sigma := (b'_\sigma b_\sigma^{-1} \mod N) + \mathsf{PRF}_{\mathsf{DDlogRS}}(\mathsf{id}')$ are correctly computed substractive shares over $[N]$. Furthermore, by the security of $\mathsf{PRF}_{\mathsf{DDlogRS}}$, the shares are also indistinguishable from uniform in $\mathbb{Z}_N$. We thus only need to show that z_σ and z'_σ are substractive shares over $[2^{k\lambda}]$ and $[2^{(k+2)\lambda}]$ respectively. To do this, note that $z_1 - z_0 \geq xy$ so long as $z_1 \geq xy$. However, this is only possible for $xy \leq 2^{(k-1)\lambda}$ possible values of z_1, and since z_1 is uniform, it happens with probability $2^{-\lambda} = \mathsf{negl}(\lambda)$. Thus, the shares are well-formed with overwhelming probability in λ. ■

By replacing every instance of input multiplication in the HSS scheme from Paillier-ElGamal in Section B.1 of [OSY21] with $\mathsf{DDlogRS}$, we immediately obtain an HSS scheme with small shares from Paillier-ElGamal with small exponent. Since this scheme follows the HSS template for RMS programs in Definition 26, it can be equipped with the $\mathsf{ExtEval}$ algorithm that allows for computation of small shares of $(y \cdot P(x))$ for any input share x if $(y \cdot P(x))$ is $2^{k\lambda}$-bounded.

Remark 32. For our purposes, we will require a slightly modified version of the algorithm that outputs complete memory shares $\mathsf{M}^{y \cdot P(x)}$ of $(y \cdot P(x))$. Henceforth, we assume the output of $\mathsf{ExtEval}$ as defined in Lemma 27 to output a memory value M^y_σ, where $\mathsf{M}^y_0 - \mathsf{M}^y_1 = (z \cdot P(x^{(1)}, \cdots, x^{(\rho)}), \mathsf{sk} \cdot z \cdot P(x^{(1)}, \cdots, x^{(\rho)}))$ with all but negligible probability. When equipped with $\mathsf{DDlogRS}$, this scheme outputs reduced memory shares over the integers. Our algorithm $\mathsf{ExtEval}$ as used in Appendix C.2 uses this modified $\mathsf{ExtEval}$ procedure as an intermediary.

Rate-1/2 Garbling with Pseudorandom Garbler Shares We provide a formal specification along with proofs of correctness and security of our rate-1/2 garbling scheme below; a high-level description was covered in Appendix C.1. Our scheme requires a circular-secure variant of Paillier-ElGamal, which (as far as we know) is not equivalent to any known circular-security assumptions.
Recall from Appendix C.1 that our scheme requires the garbler's share to be defined *pseudorandomly* as $\mathsf{PRF}(k_{\mathsf{PRF}}, \mathsf{id}_w)$ for some each wire w. In order to be able to compute shares of the cross terms defined in Equation (1) and Equation (2) the parties employ the $\mathsf{ExtEval}$ procedure to compute shares of the form $(x \cdot \mathsf{PRF}(k_{\mathsf{PRF}}, \mathsf{id}_y))$ for each multiplication gate with input wires x and y; these correspond to the required cross terms since the garbler's share of y is defined as $\mathsf{PRF}(k_{\mathsf{PRF}}, \mathsf{id}_y)$. However, the garbler's *calculated* cross terms, which define its *calculated* share of xy are different from $\mathsf{PRF}(k_{\mathsf{PRF}}, \mathsf{id}_{xy})$. In order to maintain this invariant, the garbler must send a shift $(\langle xy \rangle_G, \langle \mathsf{sk} \cdot xy \rangle_{\mathsf{G}}) - \mathsf{PRF}(k_{\mathsf{PRF}}, \mathsf{id}_{xy})$ which the evaluator adds to its own share, setting it to be $(xy, \mathsf{sk} \cdot xy) - \mathsf{PRF}(k_{\mathsf{PRF}}, \mathsf{id}_{xy})$.
In order to calculate this value using HSS, the evaluator receives an input share $\mathsf{I}^{k_{\mathsf{PRF}}}$ corresponding to the input key. Hence, the evaluator's view includes two separate ciphertexts: a public-key encryption of a symmetric key k_{PRF}, and a symmetric key encryption of the secret key sk. This is similar to the setting of *hybrid encryption* in the real world, in which parties first use a PKI to establish a symmetric key and then use that symmetric key for efficient encryption and decryption. While the security of hybrid encryption can be proved secure in the standard model, the view of the evaluator in our construction consists of ciphertexts that are *key-dependent*, which is not covered by standard security definitions of hybrid encryption. We thus introduce a natural variant of circular security for the hybrid setting that formally captures the relevant security notion. We term this assumption as *circular security of the hybrid* and provide a full specification below.
In the following definition, F and F' are function classes corresponding to the functions of the secret key that are being encrypted by the public-key and symmetric-key encryption scheme respectively.

Definition 33 (F, F'-Circular Security of $(\mathcal{E}, \mathcal{E}')$-Hybrid). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with plaintext space $\mathcal{M}$, ciphertext space $\mathcal{C}$ and key spaces $\mathcal{SK}$ and $\mathcal{PK}$, and let $\mathcal{E}' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be a symmetric-key encryption scheme with key space $\mathcal{K}$, plaintext space $\mathcal{M}'$, and ciphertext space $\mathcal{C}'$. Furthermore, let $F \subseteq \mathcal{K} \times \mathcal{SK} \to \mathcal{M}'$ and $F' \subseteq \mathcal{K} \times \mathcal{SK} \to \mathcal{M}$ be function classes. We say that the $(\mathcal{E}, \mathcal{E}')$-hybrid is F, F'-circular secure if for every polynomial time algorithm $\mathcal{A}$,*

$$\left|\Pr\left[\mathcal{A}^{\mathcal{O}_{F,k,\mathsf{sk},0},\mathcal{O}_{F',k,\mathsf{sk},0}}(1^\lambda)=1 : \begin{matrix} k \xleftarrow{\$} \mathcal{K} \\ (\mathsf{sk},\mathsf{pk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)\end{matrix}\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_{F,k,\mathsf{sk},1},\mathcal{O}_{F',k,\mathsf{sk},1}}(1^\lambda)=1 : \begin{matrix} k \xleftarrow{\$} \mathcal{K} \\ (\mathsf{sk},\mathsf{pk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)\end{matrix}\right]\right| \leq \mathsf{negl}(\lambda)$$

where $\mathcal{O}_{F,k,\mathsf{sk},0}$, $\mathcal{O}_{F,k,\mathsf{sk},1}$, $\mathcal{O}_{F',k,\mathsf{sk},0}$ *and* $\mathcal{O}_{F',k,\mathsf{sk},1}$ *are as defined below.*

$\mathcal{O}_{F,k,\mathsf{sk},0}(f)$

1 : **if** $f \notin F$, **return** $\perp$
2 : **else** $c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, f(k,\mathsf{sk}))$
3 : **return** c

$\mathcal{O}_{F,k,\mathsf{sk},1}(f)$

1 : **if** $f \notin F$, **return** $\perp$
2 : **else** $c \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, 0^{|f(k,\mathsf{sk})|})$
3 : **return** c

$\mathcal{O}_{F',k,\mathsf{sk},0}(f)$

1 : **if** $f \notin F'$, **return** $\perp$
2 : **else** $c \xleftarrow{\$} \mathsf{Enc}'(k, f(k,\mathsf{sk}))$
3 : **return** c

$\mathcal{O}_{F',k,\mathsf{sk},1}(f)$

1 : **if** $f \notin F'$, **return** $\perp$
2 : **else** $c \xleftarrow{\$} \mathsf{Enc}'(k, 0^{|f(k,\mathsf{sk})|})$
3 : **return** c

Remark 34. Our security proof will require a relatively strong variant of the above assumption. In particular, let $\mathcal{E}$ be the Paillier-ElGamal Encryption Scheme with Short Exponent, and let $\mathcal{E}'$ be a one-time pseudorandom mask for a fixed number of messages defined as $\{\mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_i) \oplus m_i\}_{i \in \mathsf{poly}(\lambda)}$ where id_i is public, $m \in \{0,1\}^{(2k+2)\lambda}$ and k_{PRF} is a uniform secret key; with our parameters, the above scheme is equivalent to a computational one-time pad encryption of a message of length $(n+|C|)(2k+2)$. Furthermore, we require F to support only the *identity* and *product* functions $\mathsf{I}_{\mathcal{K}} : (k,\mathsf{sk}) \mapsto k$ and $\times : (k,\mathsf{sk}) \mapsto k \cdot \mathsf{sk}$. The function class we require F' to support is stronger and is defined recursively in Figure 3. Note here that our function class F' is *circuit-dependent*, ie. the exact function depends on the topology of the circuit. In our reduction, we will consider an adversary that is allowed to make exactly *one* call to $\mathcal{O}_{F',k,\mathsf{sk},b}(F_{\mathsf{prf-mask},C}(\cdot,\cdot,\mathsf{id}_i))$ for each gate i (by the security of our one-time pseudorandom mask encryption scheme, which breaks if multiple messages are allowed to be encrypted under the same mask). We note that the function is computable in time $\mathsf{poly}(\lambda)$.

Remark 35. Implicit in our construction below is the fact that each gate of the circuit can have arbitrary fan-out. Note that our required garbling material per gate is a single plaintext shift_z, which the evaluator uses to output a memory share M_E^z of the output wire of the gate. By the security of HSS, this share computationally hides the value of z, and can thus be reused as an input share to the next $\mathsf{ExtEval}$. The evaluator does not get any additional information about z by reusing the input share.

Remark 36. While the scheme as described in Protocol 2 does not allow the evaluation of additive/subtractive gates for free, there is a simple modification that enables this change. Note that polynomially-many linear combinations of PRF evaluations can be evaluated in parallel and reconstructed in logarithmic depth, and are hence representable in NC^1. Thus, the garbler and the evaluator can simply perform addition/subtraction on their shares when encountering an additive/subtractive gate, and the garbler can set the obtained linear combination of PRF values as its share of an input wire of a multiplicative gate. The evaluator (who knows the topology of the circuit, and thus can determine the precise combination) simply performs an $\mathsf{ExtEval}$ with the adjusted program description accordingly.

With this in mind, we can now give a full description of our scheme.

The Function $F_{\mathsf{prf-mask},C}$

Parameters: Let C be a circuit and k, sk be the symmetric and secret key used in Definition 33 respectively. The inputs to $\mathsf{ExtEval}$ are exactly as defined in Protocol 2.

1. For input wires, $F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_z) = 0^{(2k+2)\lambda}$.
2. For an additive/subtractive gate $z = x \pm y$, $F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_z) = -F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_x) \mp F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_y)$.
3. For a multiplicative gate $z = x \cdot y$,

$$F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_z) = (F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_z)_1, F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_z)_2),$$

where

$$\begin{aligned} F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_z)_1 &= F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_x)_1 \cdot F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_y)_1 \\ &\quad - \mathsf{ExtEval}(1, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_G\}, \mathsf{M}_G^x, \mathsf{I}^k, \mathsf{PRF}_\lambda^1(\cdot, \mathsf{id}_y))_1 \\ &\quad - \mathsf{ExtEval}(1, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_G\}, \mathsf{M}_G^y, \mathsf{I}^k, \mathsf{PRF}_\lambda^1(\cdot, \mathsf{id}_x))_1 \end{aligned}$$

$$\begin{aligned} F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_z)_2 &= F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_x)_2 \cdot F_{\mathsf{prf-mask},C}(k, \mathsf{sk}, \mathsf{id}_y)_1 \\ &\quad - \mathsf{ExtEval}(1, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_G\}, \mathsf{M}_G^x, \mathsf{I}^k, \mathsf{PRF}_\lambda^1(\cdot, \mathsf{id}_y))_2 \\ &\quad - \mathsf{ExtEval}(1, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_G\}, \mathsf{M}_G^y, \mathsf{I}^k, \mathsf{PRF}_\lambda^2(\cdot, \mathsf{id}_x))_1 \end{aligned}$$

Fig. 3: Function supported by the circular security assumption.

Protocol Small Integer Garbling from Paillier-ElGamal with Small Exponent $\mathsf{AGC}_{\mathsf{PaillierEG}}$

Parameters. Let λ be a security parameter and $k > 1$ be a global parameter. We consider B-bounded arithmetic circuits where $B \leq 2^{(k-1)\lambda}$. We consider a circuit C of size at most $\mathsf{poly}(\lambda)$ with n inputs, m outputs and $s_\times$ multiplication gates; we represent the ith output wire by out_i. Let $\mathsf{PaillierEG} = (\mathsf{PaillierEG.Gen}, \mathsf{PaillierEG.Enc}, \mathsf{PaillierEG.Dec})$ be the Paillier-ElGamal cryptosystem with short exponent defined in Figure 2. Further define $\mathsf{PRF}_\lambda : \{0,1\}^\lambda \times (n+|C|) \to \{0,1\}^{2k+2}$ to be a family of PRFs of depth at most $O(\log \lambda)$. For convenience, we define $\mathsf{PRF}_\lambda^1 : \{0,1\}^\lambda \times (n+|C|) \to \{0,1\}^k$ and $\mathsf{PRF}_\lambda^2 : \{0,1\}^\lambda \times (n+|C|) \to \{0,1\}^{k+2}$ as partial outputs of the PRF, ie. $\mathsf{PRF}_\lambda = (\mathsf{PRF}_\lambda^1, \mathsf{PRF}_\lambda^2)$. Furthermore, note that $\mathsf{PRF}_\lambda(\cdot, \mathsf{id}_x)$ for fixed x is expressible as an RMS program, since $\mathsf{PRF} \in \mathsf{NC}^1$.

$\mathsf{AGC}_{\mathsf{PaillierEG}}.\mathsf{Garble}(1^\lambda, C)$**:**

1. The garbler samples $k_{\mathsf{PRF}}, k_{\mathsf{DDlogRS}} \xleftarrow{\$} \{0,1\}^\lambda$ and $(N, g, \mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{PaillierEG.Gen}(1^\lambda)$.
2. The garbler sets $\mathsf{I}^{k_{\mathsf{PRF}}} \leftarrow (\mathsf{PaillierEG.Enc}(\mathsf{pk}, k_{\mathsf{PRF}}), \mathsf{PaillierEG.Enc}(\mathsf{pk}, \mathsf{sk} \cdot k_{\mathsf{PRF}}))$ and adds it to $\widehat{C}$.
3. The garbler chooses a subtractive sharing of 1 $(\langle 1 \rangle_i, \langle \mathsf{sk} \rangle_i)_{i \in \{E,G\}}$ and adds one share $\mathsf{M}_E := (\langle 1 \rangle_E, \langle \mathsf{sk} \rangle_E)$ to $\widehat{C}$. Here M_E is the evaluator's memory share of 1 that it uses to perform an HSS evaluation. The garbler initiates its own HSS with the share $\mathsf{M}_G := (\langle 1 \rangle_G, \langle \mathsf{sk} \rangle_G)$.
4. For each input wire i and admissible value x, the garbler sets its share as $\mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_i)$. The label associated with x is $L_x^i \leftarrow (x, \mathsf{sk} \cdot x) + \mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_i)$.
5. For each internal wire w, the garbler maintains the sharing $(\langle w \rangle_{\mathsf{G}}, \langle \mathsf{sk} \cdot w \rangle_{\mathsf{G}})$. For each gate in parallel, the garbler can compute:

   Addition/Subtraction Gate $z = x \pm y$**.** The garbler sets its label of wire z to be $\mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)$ and adds $\mathsf{shift}_z := -(\langle x \rangle_{\mathsf{G}}, \langle (\mathsf{sk} \cdot x) \rangle_{\mathsf{G}}) \mp (\langle y \rangle_{\mathsf{G}}, \langle (\mathsf{sk} \cdot y) \rangle_{\mathsf{G}}) - \mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)$ to $\widehat{C}$.

Multiplication Gate $z = x \cdot y$**.** The garbler sets its label of wire z to be $\mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)$. It then computes

$$\begin{aligned}
&(\langle x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}},\mathsf{id}_y)\rangle_{\mathsf{G}}, \langle \mathsf{sk} \cdot x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{G}}) \\
&\qquad \leftarrow \mathsf{ExtEval}(1, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_G\}, \mathsf{M}^x_G, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{PRF}^1_\lambda(\cdot, \mathsf{id}_y)) \\
&(\langle y \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}},\mathsf{id}_x)\rangle_{\mathsf{G}}, \langle \mathsf{sk} \cdot y \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{G}}) \\
&\qquad \leftarrow \mathsf{ExtEval}(1, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_G\}, \mathsf{M}^y_G, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{PRF}^1_\lambda(\cdot, \mathsf{id}_x)) \\
&(\langle y \cdot \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}},\mathsf{id}_x)\rangle_{\mathsf{G}}, \langle \mathsf{sk} \cdot y \cdot \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{G}}) \\
&\qquad \leftarrow \mathsf{ExtEval}(1, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_G\}, \mathsf{M}^y_G, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{PRF}^2_\lambda(\cdot, \mathsf{id}_x))
\end{aligned}$$

and sets

$$\mathsf{shift}_z := (\mathsf{shift}^1_z, \mathsf{shift}^2_z),$$

where

$$\begin{aligned}
\mathsf{shift}^1_z = \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x) \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y) - \langle x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{G}} \\
- \langle y \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{G}} - \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z) \\
\mathsf{shift}^2_z = \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x) \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y) - \langle \mathsf{sk} \cdot x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{G}} \\
- \langle y \cdot \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{G}} - \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)
\end{aligned}$$

and adds shift_z to $\widehat{C}$.

6. The garbler outputs $((L^i_x)_{i\in[n]}, \widehat{C} = (N, g, k_{\mathsf{DDlogRS}}, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{M}_E, (\mathsf{shift}_i)_{i\in C}, \{\langle \mathsf{out}_i\rangle_{\mathsf{G}}\}_{i\in m}))$ where x is over the set of admissible inputs.

$\mathsf{AGC}_{\mathsf{PaillierEG}}.\mathsf{Eval}((L_i)_{i\in[n]}, \widehat{C})$:

1. The evaluator sets its sharing of input wire w_i as $(\langle w_i\rangle_{\mathsf{E}}, \langle \mathsf{sk} \cdot w_i\rangle_{\mathsf{E}}) := L_i$. It interprets $\widehat{C}$ as $(N, g, k_{\mathsf{DDlogRS}}, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{M}_E, (\mathsf{shift}_i)_{i\in C}, \{\langle \mathsf{out}_i\rangle_{\mathsf{G}}\}_{i\in m})$.
2. For each internal wire w, the evaluator maintains the sharing $(\langle w\rangle_{\mathsf{E}}, \langle \mathsf{sk}\cdot w\rangle_{\mathsf{E}})$. It then proceeds gate-by-gate in topological order:

   Addition/Subtraction Gate $z = x \pm y$**.** The evaluator sets its share of wire z as $(\langle x\rangle_{\mathsf{E}}, \langle(\mathsf{sk}\cdot x)\rangle_{\mathsf{E}}) \pm (\langle y\rangle_{\mathsf{E}}, \langle(\mathsf{sk}\cdot y)\rangle_{\mathsf{E}}) + \mathsf{shift}_z$.

   Multiplication Gate $z = x \cdot y$**.** The evaluator computes

$$\begin{aligned}
&(\langle x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}},\mathsf{id}_y)\rangle_{\mathsf{E}}, \langle \mathsf{sk} \cdot x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{E}}) \\
&\qquad \leftarrow \mathsf{ExtEval}(0, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_E\}, \mathsf{M}^x_E, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{PRF}^1_\lambda(\cdot, \mathsf{id}_y)) \\
&(\langle y \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}},\mathsf{id}_x)\rangle_{\mathsf{E}}, \langle \mathsf{sk} \cdot y \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{E}}) \\
&\qquad \leftarrow \mathsf{ExtEval}(0, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_E\}, \mathsf{M}^y_E, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{PRF}^1_\lambda(\cdot, \mathsf{id}_x)) \\
&(\langle y \cdot \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}},\mathsf{id}_x)\rangle_{\mathsf{E}}, \langle \mathsf{sk} \cdot y \cdot \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{E}}) \\
&\qquad \leftarrow \mathsf{ExtEval}(0, \{k_{\mathsf{DDlogRS}}, \mathsf{M}_E\}, \mathsf{M}^y_E, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{PRF}^2_\lambda(\cdot, \mathsf{id}_x))
\end{aligned}$$

   and sets its shares to be

$$\begin{aligned}
\langle z\rangle_{\mathsf{E}} := \langle x\rangle_{\mathsf{E}} \cdot \langle y\rangle_{\mathsf{E}} + \langle x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{E}} \\
+ \langle y \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{E}} + \mathsf{shift}^1_z \\
\langle \mathsf{sk} \cdot z\rangle_{\mathsf{E}} := \langle \mathsf{sk} \cdot x\rangle_{\mathsf{E}} \cdot \langle y\rangle_{\mathsf{E}} + \langle \mathsf{sk} \cdot x \cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{E}} \\
+ \langle y \cdot \mathsf{PRF}^2_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{E}} + \mathsf{shift}^2_z
\end{aligned}$$

3. The evaluator outputs $\{\langle \mathsf{out}_i\rangle_{\mathsf{E}} - \langle \mathsf{out}_i\rangle_{\mathsf{G}}\}_{i\in m}$.

Protocol 2: Small Integer Garbling Scheme from Paillier-ElGamal.

Theorem 37 (Small Integer Garbling from the Circular-Security of Paillier-ElGamal with Small Exponent). *Let λ be a security parameter and $\ell(\lambda)$ be the bitlength of a corresponding RSA modulus. We consider a circuit C consisting of n input wires and m output wires. Assume the $(\{\times, \mathsf{I}_{\mathcal{K}}\}, \{F_{\mathsf{prf}-\mathsf{mask},C}\}_{C\in\mathcal{C}_\lambda})$-circular security of the (Paillier-ElGamal with Small Exponent, Pseudorandom Mask)-Hybrid (Definition 33, Remark 34) and the existence of a pseudorandom function in NC^1. Then for every integer $k > 1$, the construction $\mathsf{AGC}_{\mathsf{PaillierEG}}$ (Protocol 2) is an arithmetic garbling scheme for B-bounded integer computation where B is any integer less than $2^{(k-1)\lambda}$. Moreover, the garbled circuit has bit-size*

$$6\ell(\lambda) + (n + |C| + 1)(2k + 2)\lambda + (km + 1)\lambda$$

where the size of each input label is $k\lambda$ bits.

Proof. We show correctness and Privacy.

Correctness. We will show that for any wire w carrying the secret x, the following invariant holds: $\langle w\rangle_{\mathsf{E}} - \langle w\rangle_{\mathsf{G}} = (x, \mathsf{sk}\cdot x)$.

- **Input Labels:** For the input labels, the invariant holds by Step 4 of the scheme.
- **Addition/Subtraction Gates:** Consider the output wires of addition/subtraction gates and assume WLOG that the gate is an addition gate. The garbler sets its share to be $\mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)$. Assuming that the invariant holds for input gates, the evaluator calculates its share as $(\langle x\rangle_{\mathsf{E}}, \langle(\mathsf{sk}\cdot x)\rangle_{\mathsf{E}}) + (\langle y\rangle_{\mathsf{E}}, \langle(\mathsf{sk}\cdot y)\rangle_{\mathsf{E}}) + \mathsf{shift}_z = (\langle x\rangle_{\mathsf{E}}, \langle(\mathsf{sk}\cdot x)\rangle_{\mathsf{E}}) + (\langle y\rangle_{\mathsf{E}}, \langle(\mathsf{sk}\cdot y)\rangle_{\mathsf{E}}) - (\langle x\rangle_{\mathsf{G}}, \langle(\mathsf{sk}\cdot x)\rangle_{\mathsf{G}}) - (\langle y\rangle_{\mathsf{G}}, \langle(\mathsf{sk}\cdot y)\rangle_{\mathsf{G}}) - \mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z) = (x + y, \mathsf{sk}\cdot(x + y)) - \mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)$, which maintains the invariant.
- **Multiplication Gates:** Consider the output wire of a multiplication gate. We use here the correctness of the $\mathsf{ExtEval}$ algorithm. First, note that we require the PRF to be in NC^1 since $\mathsf{ExtEval}$ requires the program P to be defined as a polynomial-size RMS program; this class encapsulates NC^1 circuits. As stated in Lemma 27, the outputs of $\mathsf{ExtEval}$ in Step 5 of the protocol form subtractive shares over $[N]$. Along with the correctness of Lemma 31, these are shares over $2^{k\lambda}$ and $2^{(k+2)\lambda}$ with all but negligible probability. Hence, note that

$$\begin{aligned}\Pr[\langle z\rangle_{\mathsf{E}} - \langle z\rangle_{\mathsf{G}} \neq z] \leq\ & \Pr[\langle x\cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{E}} - \langle x\cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)\rangle_{\mathsf{G}} \neq x\cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_y)] \\ &+ \Pr[\langle y\cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{E}} - \langle y\cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)\rangle_{\mathsf{G}} \neq y\cdot \mathsf{PRF}^1_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_x)] \\ \leq\ & 2\mathsf{negl}(\lambda) \\ =\ & \mathsf{negl}(\lambda)\end{aligned}$$

  Thus, the output wire satisfies the above invariant with all but negligible probability. Note that the depth of the circuit is polynomially bounded, and hence the label values of each output wire go through at most polynomially many iterations of $\mathsf{ExtEval}$. It follows that that each such wire then maintains the above invariant with all but negigible probability.
- **Output Wires:** From the previous discussion, the above invariant is also maintained for each output wire of the circuit. Hence, the output wires reconstruct, and the scheme is correct.

Privacy. We describe our simulator in Algorithm 4. We will argue indistinguishability via a series of hybrids.

Algorithm Small Integer Garbling from Paillier-ElGamal with Small Exponent: $\mathsf{Sim}(1^\lambda, C, z)$

Parameters. Let λ be a security parameter and $k > 1$ be a global parameter. We consider a circuit C of size at most $\mathsf{poly}(\lambda)$ with n inputs, m outputs and $s_\times$ multiplication gates; we represent the ith output wire by out_i. Let $\mathsf{PaillierEG} = (\mathsf{PaillierEG.Gen}, \mathsf{PaillierEG.Enc}, \mathsf{PaillierEG.Dec})$ be the Paillier-ElGamal cryptosystem with short exponent defined in Figure 2.

$\mathsf{Sim}(1^\lambda, C, z)$**:**

1. The simulator samples $k_{\mathsf{DDlogRS}} \xleftarrow{\$} \{0,1\}^\lambda$ and $(N, g, \mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{PaillierEG.Gen}(1^\lambda)$.

2. The simulator sets $\mathsf{I}^{k_{\mathsf{PRF}}} \leftarrow (\mathsf{PaillierEG.Enc}(\mathsf{pk}, 0), \mathsf{PaillierEG.Enc}(\mathsf{pk}, 0))$ and adds it to $\widehat{C}$.
3. The simulator chooses a subtractive sharing of 1 and adds one share $\mathsf{M}_E := (\langle 1 \rangle_E, \langle \mathsf{sk} \rangle_E)$ to $\widehat{C}$.
4. The simulator chooses $L_x^i \xleftarrow{\$} \{0,1\}^{(2k+2)\lambda}$.
5. For output wire i of any gate, the simulator chooses $\mathsf{shift}_i \xleftarrow{\$} \{0,1\}^{(2k+2)\lambda}$.
6. The simulator runs $\mathsf{AGC}_{\mathsf{PaillierEG}}.\mathsf{Eval}$ to find the outputs $\{\langle \mathsf{out}_i \rangle_{\mathsf{E}}\}_{i \in N}$ and chooses $\langle \mathsf{out}_i \rangle_{\mathsf{G}} := \langle \mathsf{out}_i \rangle_{\mathsf{E}} - z_i$. Then it outputs $((L_x^i)_{i \in [n]}, \widehat{C} = (N, g, k_{\mathsf{DDlogRS}}, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{M}_E, (\mathsf{shift}_i)_{i \in C}, \{\langle \mathsf{out}_i \rangle_{\mathsf{G}}\}_{i \in m}))$.

Algorithm 4: Simulator for Small Integer Garbling Scheme from Paillier-ElGamal.

– Hybrid 0: The real-world protocol.
– Hybrid 1: The garbler runs Step 6 of the simulator instead of the garbling scheme. The only difference between this hybrid and the previous one is how the simulator programs its output shares. The garbler's output shares, however, are statistically hidden by the correctness of the protocol; thus the hybrids are statistically indistinguishible.
– Hybrid 2: In this hybrid, the garbler replaces $\mathsf{I}^{k_{\mathsf{PRF}}}$ with encryptions of 0 instead of k and $\mathsf{sk} \cdot x$, and replaces the shifts shift_z with PRF outputs $\mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)$. The indisinguishability of this hybrid from the previous one follows via a direct application of the circular security of the (Paillier-ElGamal, Pseudorandom Mask)-Hybrid parametrized with Remark 34.
  Suppose that there is a PPT distinguisher $\mathsf{Dist}(1^\lambda)$ that can distinguish between Hybrid 1 and Hybrid 2 with greater than negligible probability. We will show an adversary $\mathcal{A}$ that, given access to this distinguisher and the oracles $\mathcal{O}_{F,k,\mathsf{sk},b}$ and $\mathcal{O}_{F',k,\mathsf{sk},b}$, can break the circular security of the (Paillier-ElGamal, Pseudorandom Mask)-Hybrid. The adversary $\mathcal{A}(1^\lambda, N, g)$ works as follows. It runs $\mathsf{AGC}_{\mathsf{PaillierEG}}.\mathsf{Garble}(1^\lambda, C)$ for some B-bounded circuit C, with the following changes:
  - In step 1, it does not sample k_{PRF} or $\mathsf{PaillierEG.Gen}(1^\lambda)$.
  - In step 2, it makes the oracle queries $\mathcal{O}_{F,k,\mathsf{sk},b}(\mathsf{I}_{\mathcal{K}})$ and $\mathcal{O}_{F,k,\mathsf{sk},b}(\times)$ where $\mathsf{I}_{\mathcal{K}}$ and $\times$ are the functions as defined in Remark 34.
  - In step 4, the adversary sets its share by querying $\mathcal{O}_{F',k,\mathsf{sk},b}(F_{\mathsf{prf-mask},C}(\cdot,\cdot,\mathsf{id}_{\mathsf{in}_i}))$ for each input wire in_i.
  - In step 5, the adversary sets $\mathsf{shift}_z \leftarrow \mathcal{O}_{F',k,\mathsf{sk},b}(F_{\mathsf{prf-mask},C}(\cdot,\cdot,\mathsf{id}_z))$.

  Note that since the pseudorandom mask is a *one-time* encryption scheme, the adversary is only allowed to make a single call to the oracle per gate. Nevertheless, the adversary sends the output $((L_x^i)_{i \in [n]}, \widehat{C} = (N, g, k_{\mathsf{DDlogRS}}, \mathsf{I}^{k_{\mathsf{PRF}}}, \mathsf{M}_E, (\mathsf{shift}_i)_{i \in C}, \{\langle \mathsf{out}_i \rangle_{\mathsf{G}}\}_{i \in m}))$ to Dist. If $b = 0$, then the above procedure corresponds to Hybrid 1, while if $b = 1$ then the above procedure corresponds to Hybrid 2. The adversary outputs whatever Dist outputs, however since Dist is able to distinguish between the two hybrids with more than negligible probability, the adversary can differentiate between $b = 0$ and $b = 1$ with more than negligible probability as well, breaking the $(\{\times, \mathsf{I}_{\mathcal{K}}\}, \{F_{\mathsf{prf-mask},C}\}_{C \in \mathcal{C}_\lambda})$-circular security of the (Paillier-ElGamal, Pseudorandom Mask)-Hybrid.
– Hybrid 3: In this hybrid, the garbler replaces both the labels L_i and the shifts shift_z with truly random values. Note that there is no information about k_{PRF} remaining in the view. The indistinguishability of this hybrid follows from the security of the PRF. Our reduction works briefly as follows: assume that there is a distinguisher $\mathsf{Dist}(1^\lambda)$ which can distinguish between Hybrids 2 and 3 with noticeable probability. Then consider a PPT adversary $\mathcal{A}^{\mathcal{O}(\cdot)}$, where $\mathcal{O}(\cdot)$ either $\mathsf{PRF}(k_{\mathsf{PRF}}, \cdot)$ or $R(\cdot)$ where R is a truly random function. $\mathcal{A}$ can run the algorithm of Hybrid 2, making calls to either $\mathsf{PRF}_\lambda(k_{\mathsf{PRF}}, \mathsf{id}_z)$ or $R(\mathsf{id}_z)$ as given. It sends the output of the garbled circuit to Dist, then outputs as the distinguisher does accordingly. Since only a polynomial number of calls to the oracle have been made, the adversary then can output whether $\mathcal{O}(\cdot)$ is the PRF or a truly random function with noticeable probability, breaking the underlying security of the PRF.
– Hybrid 4: The simulator's algorithm of Algorithm 4. The hybrids are identical.

With this we complete the proof of privacy.

Size. Finally, we compute the size of the garbling output. Note that N is of size $\ell(\lambda)$, g is an element of N and is hence also of size $\ell(\lambda)$. k_{DDlogRS} is of length λ and M_1, being a share, is of length $(2k+2)\lambda$,

which is also the length of each one of the shifts (there are $n + |C|$) of them. Furthermore, $\mathsf{I}^{k_{\mathsf{PRF}}}$ is two ciphertexts and is hence of size $4\ell(\lambda)$, and the output wires are of length $k\lambda$ each. Together, this adds up to

$$6\ell(\lambda) + (n + |C| + 1)(2k + 2)\lambda + (km + 1)\lambda.$$

It is easy to see that the rate of the scheme goes rapidly to $1/2$ as $k \to \infty$.

■