A Unified Framework for Succinct Garbling from Homomorphic Secret Sharing

Yuval Ishai[1], Hanjun Li[2], and Huijia Lin[2]

[1] Technion, Haifa, Israel
yuvali@cs.technion.ac.il
[2] University of Washington, Seattle, WA, USA
{hanjul,rachel}@cs.washington.edu

Abstract. A major challenge in cryptography is the construction of *succinct garbling schemes* that have asymptotically smaller size than Yao's garbled circuit construction. We present a new framework for succinct garbling that replaces the heavy machinery of most previous constructions by lighter-weight *homomorphic secret sharing* techniques.

Concretely, we achieve *1-bit-per-gate* (amortized) garbling size for Boolean circuits under circular variants of standard assumptions in composite-order or prime-order groups, as well as a lattice-based instantiation. We further extend these ideas to *layered* circuits, improving the per-gate cost below 1 bit, and to *arithmetic* circuits, eliminating the typical $\Omega(\lambda)$-factor overhead for garbling mod-p computations. Our constructions also feature "leveled" variants that remove circular-security requirements at the cost of adding a depth-dependent term to the garbling size.

Our framework significantly extends a recent technique of Liu, Wang, Yang, and Yu (Eurocrypt 2025) for lattice-based succinct garbling, and opens new avenues toward practical succinct garbling. For moderately large circuits with a few million gates, our garbled circuits can be *two orders of magnitude smaller* than Yao-style garbling. While our garbling and evaluation algorithms are much slower, they are still *practically feasible*, unlike previous fully succinct garbling schemes that rely on expensive tools such as iO or a non-black-box combination of FHE and ABE. This trade-off can make our framework appealing when a garbled circuit is used as a functional ciphertext that is broadcast or stored in multiple locations (e.g., on a blockchain), in which case communication and storage may dominate computational cost.

Table of Contents

1 Introduction

Introduced by Yao in the 1980s, a garbling scheme [Yao82, BHR12] allows a "garbler" to transform a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a *garbled circuit* $\hat{C}$ along with a pair of short keys $\mathbf{k}_{i,0}, \mathbf{k}_{i,1}$ for each input bit x_i. Given the circuit C, the garbled circuit $\hat{C}$, and an *encoded input* consisting of the input labels $\mathbf{k}_x = \{\mathbf{k}_{i,x_i}\}_{i\in[n]}$ for an unknown input x, an efficient "evaluator" can compute $C(x)$ while learning nothing else about x. An important feature of garbled circuits is that the input keys are short, growing only with the security parameter λ and not with the size of the circuit C.

Garbling schemes serve as fundamental building blocks in cryptography and have found a variety of applications, including constant-round secure computation [Yao82, BMR90, FKN94, GS18, BL18, HIKR23], low-complexity cryptography [AIK05], proof systems [GGP10], single-key functional encryption [SS10], offline-online secure computation [CEMY09, AIKW13], and many more. See [App17] for a survey.

A central research direction is to minimize the size of garbled circuits. This is motivated by the fact that in typical applications, the garbler needs to communicate the garbled circuit to the evaluator. Minimizing size has a compounded impact when the same garbled circuit is distributed to multiple parties, translating into proportional bandwidth and storage savings across all recipients. For example, a garbled circuit implementing a complex algorithm can be generated and broadcasted to many receivers during an offline phase, or even stored on a blockchain, allowing fast online computation once the inputs are known.

Succinct Garbling. In this work, we focus on the task of obtaining *succinct* garbled circuits. Our default notion of succinctness requires that the bit-length of the garbled circuit be smaller than the description size of the original circuit. Concretely, we say that a garbling scheme (for Boolean circuits) is succinct, if for sufficiently large C we have $|\hat{C}| < |C| \log |C|$, where $|C|$ denotes the size of a Boolean circuit C (number of gates, with fan-in 2[3]) and $|\hat{C}|$ denotes the bit length of the string $\hat{C}$;[4] note that describing a general circuit C requires at least $|C| \log |C|$ bits. This is a natural threshold, since it implies that a succinct garbled circuit is less expensive to communicate than the original circuit. Once this minimal threshold of succinctness is crossed, we can aim for a full spectrum of better levels of succinctness. Concrete succinctness goals we consider in this work include garbling with 1-bit-per-gate, or even $O(1/\log\log\lambda)$-bits-per-gate, where λ is the security parameter. (These are all amortized costs, assuming $|C| \gg n, m, \lambda$.) The ultimate goal is achieving *full succinctness*, where the garbled circuit size is independent of the size of the original circuit.

Over the past four decades, tremendous progress has been made on reducing the garbled circuit size, even reaching the ultimate goal of full succinctness. However, significant gaps remain, especially when requiring the constructions to be efficient enough to be implemented.

– *Fast Non-Succinct Garbling Schemes.* Yao's original garbled circuit construction [Yao82] and its optimizations [BMR90, NPS99, KS08, PSSW09, KMR14, GLNP15, ZRE15, RR21] remain the most practical general-purpose garbling schemes, as they only rely on fast symmetric-key cryptography. The current state-of-the-art, due to Rosulek and Roy [RR21], garbles an

[3] By default, the number of gates counts all fan-in 2 gates, including XOR, AND, OR. But in fact, our results apply also to circuits with a much richer set of gates that include all possible fan-in-$O(\log\lambda)$ gates; see below.

[4] Note that C and $\hat{C}$ are syntactically different objects. C is a circuit, while $\hat{C}$ is a binary string. Adopting standard notation, $|C|$ denotes the number of *gates* in the original circuit C, while $|\hat{C}|$ denotes the bit-length of the garbled circuit.

AND gate using $1.5\lambda + 5$ bits, while XOR and NOT gates are free, using a random oracle. Note that this does not meet our succinctness criterion, since security against $\mathrm{poly}(|C|)$-time adversaries implies that $\lambda > \log |C|$. Indeed, high communication cost is typically the main practical bottleneck in applications that rely on Yao-style garbling.

– *Fully Succinct Garbling Schemes.* On the other extreme, fully succinct garbled circuits have been shown feasible under standard assumptions. Though theoretically optimal, the concrete garbling size is astronomically large, making these constructions practically infeasible. In addition, existing schemes rely either on indistinguishability obfuscation (iO) [KLW15, BCG+18] or on a non-black-box combination of fully homomorphic encryption (FHE) and attribute-based encryption (ABE) [GKP+13, BGG+14, HLL23]. These do not only incur a high computational overhead but also rely on a limited class of assumptions, such as circular-secure LWE (e.g., [Gen09, BV11, GSW13]) or combinations of multiple assumptions (e.g., LPN over large fields, local PRG, and DLin over bilinear groups).

As is often the case in cryptography, diversifying assumptions may also lead to efficiency benefits. This motivates the following question:

Can succinct garbling schemes be based on a broader set of assumptions, with improved efficiency?

Two recent works have made major progress on succinct garbling without the heavy machinery of iO, FHE, or ABE. The work of [LWYY24] presented garbled circuits with 1-bit-per-gate based on variants of RLWE or NTRU, while [ILL24] achieved *fully succinct* garbling for weak classes of programs, including truth-tables and DFAs, from a variety of group-based assumptions. The latter results build on a fully succinct *partial* garbling scheme for general circuits, applying a computation on a secret input on top of a computation on a public input. Here full succinctness requires the garbled circuit size to be independent of the complexity of the public part of the computation.

1.1 Our Results in a Nutshell

Following the recent momentum, we present a unified framework for constructing succinct garbled circuits with 1-bit-per-gate using techniques for *homomorphic secret sharing* (HSS) [BGI16, BGI+18, BKS19, OSY21, RS21, MORS24]. Our unified framework can be instantiated using pirme order groups or Paillier groups or lattices, relying on circular-security variants of the power-DDH assumption or a circular power-RLWE assumption. (See discussion of these assumptions below.) We further show how to avoid circular security altogether in a "leveled" variant of our unified framework, where the garbled circuits contain additional components of size $D \cdot \mathrm{poly}(\lambda)$ that depend the circuit depth D but not on the circuit size. Note that the leveled version already gives 1-bit-per-gate garbling for low-depth circuits, including NC^1 or depth-λ circuits, that arise in many applications. We summarize our results on succinct Boolean garbling in the following theorem, and compare it with prior schemes in Table 1.

Theorem (Succinct Boolean Garbling, Informal)**.** *Assuming either: (1) circular Power-DDH in Paillier groups (Definition 7), or (2) a variant of circular Power-DDH in prime-order groups (Definition 11), or (3) circular Power-RLWE (Definition 9), there is a garbling scheme for Boolean circuits C with garbled circuit size $|\widehat{C}| = |C| + \mathrm{poly}(\lambda)$.*

Assuming Power-DDH (Definition 6) in Paillier or prime-order groups, or Power-RLWE (Definition 8), there is a leveled variant with garbled circuit size $|\widehat{C}| = |C| + D \cdot \mathrm{poly}(\lambda)$ for circuits of depth D.

See Theorem 2 and Corollary 1 for the formal statements on instantiations from Paillier groups and lattices. The instantiation using prime order groups is slightly more complex and proceeds in two steps. In the first step, we obtain a 1-bit-per-gate Boolean garbling with inverse polynomial correctness and privacy errors assuming the circular Power-DDH assumption, as formally stated in Theorem 2 and Corollary 1. Then in the second step, we make the errors negligible, using correctness and privacy amplification, assuming a variant of the circular Power-DDH assumption (Definition 11). Importantly, it turns out that the amplification does not increase the (amortized) per-gate garbling size, keeping 1-bit-per-gate. See Section 5.5.

	Class	$\lvert\hat{C}\rvert + \lvert\hat{x}\rvert$	Tool	Assumption
Yao [Yao82]	general	$O(\lambda) \cdot \lvert C\rvert$	Symmetric	OWF
Fully Succinct e.g.,[BGG+14]	general	$\mathrm{poly}(\lambda) \cdot (n+m)$	iO FHE+ABE	 Lattice cir-LWE
Fully Succinct ILL24 [ILL24]	weak classes e.g. DFA	$\mathrm{poly}(\lambda) \cdot (n+m)$	HSS	Group cir-P-DDH
LWYY24 [LWYY24]	general	$\lvert C\rvert + \mathrm{poly}(\lambda) \cdot n$	SHE	Lattice cir-RLWE cir-NTRU
This Work	general	$\lvert C\rvert + \mathrm{poly}(\lambda) \cdot n$	HSS	Group cir-P-DDH Lattice cir-P-RLWE
	layered	$\frac{\lvert C\rvert}{\log\log\lambda} + \mathrm{poly}(\lambda) \cdot n$		
	general	$\lvert C\rvert + \mathrm{poly}(\lambda) \cdot (n+D)$		P-DDH P-RLWE
	layered	$\frac{\lvert C\rvert}{\log\log\lambda} + \mathrm{poly}(\lambda) \cdot (n+D)$		

Table 1. Comparison between Boolean garbling schemes, in terms of *the class of Boolean circuits handled*, *garbled circuit size*, *the cryptographic tool used*, and *assumptions*. For assumptions, we list both the mathematical structure and the concrete assumption. λ denotes the security parameter, $|C|$ the number of gates in C, n the input length, m the output length, and D the depth. OWF stands for one-way function and SHE for somewhat homomorphic encryption.

Extension 1: Beating 1-bit-per-gate. Our framework can be further extended in two interesting ways. First, for *layered* circuits, we can improve the garbling size to $O(1/\log\log\lambda)$-bits-per-gate. This gives the first garbling scheme for a natural and general class of circuits that goes below the bar of 1-bit-per-gate, without relying on iO or FHE plus ABE. Previously, this was only achieved for simple programs such as DFAs [ILL24]. In fact, this follows as a corollary of a more general construction of a garbling scheme for circuits built from "supergates", including all gates with $O(\log\lambda)$-fan-in, and the garbling size is 1 bit per "supergate."

Extension 2: Succinct arithmetic garbling. We extend our unified framework to garble *arithmetic* circuits, whose gates perform additions and multiplications modulo p or over the integers. Under the above assumptions, we garble arithmetic circuits modulo p with $O(\log p)$-bits-per-gate for general moduli p (for small modulus $p = \mathrm{poly}(\lambda)$, the constant behind the big-O is 1). Note that garbling schemes for mod-p computations automatically imply garbling schemes for bounded integer computations where the wire values are guaranteed to be smaller than p.

This represents significant progress on the front of succinct arithmetic garbling, without relying on iO or FHE plus ABE. As summarized in Table 2, the state-of-the-art arithmetic garbling schemes require $\tilde{\Omega}(\log p \cdot \lambda)$-bits-per-gate for $\mathbb{Z}_p$ computation [BLLL23, LL24, Hea24]. We eliminate the λ-multiplicative overhead.

For the simpler task of bounded integer garbling, prior works [BLLL23, MORS24] have shown how to trade the $\Omega(\lambda)$-multiplicative overhead for an additive $\mathrm{poly}(\lambda)$-overhead, achieving $(\log p + \mathrm{poly}(\lambda))$-bits-per-gate, based on DCR. We improve the state-of-the-art by diversifying the assumptions, adding prime-order groups and lattices, and removing the large additive $\mathrm{poly}(\lambda)$ overhead.

	Ring	$\lvert\hat{C}\rvert + \lvert\hat{x}\rvert$	Tool	Assumption	
[BLLL23]	$\mathbb{Z}$	$\lvert C\rvert \cdot (O(\ell) + \mathrm{poly}(\lambda))$	LHE	Paillier	DCR
[MORS24]	$\mathbb{Z}$	$\lvert C\rvert \cdot (\ell + \mathrm{poly}(\lambda))$	HSS	Paillier	cir-DCR
[BLLL23]	$\mathbb{Z}_p$	$\lvert C\rvert \cdot \lambda \cdot (O(\log p) + \mathrm{poly}(\lambda))$	LHE	Paillier	strong DCR
[LL24, Hea24]	$\mathbb{Z}_p$	$\lvert C\rvert \cdot \lambda \cdot O(\log p)$	Symmetric		CRH
This Work	$\mathbb{Z}_p$	$\lvert C\rvert \cdot O(\log p) + \mathrm{poly}(\lambda) \cdot n$ ★ $\lvert C\rvert \cdot O(\log p) + \mathrm{poly}(\lambda) \cdot (n + D)$ ★	HSS	Paillier Prime Group Lattice	cir-P-DDH cir-P-RLWE P-DDH P-RLWE

Table 2. Comparison between *arithmetic* garbling schemes, in terms of *ring supported*, *garbled circuit size*, *the cryptographic tool used*, and *assumptions*. For integer computations, the wire values must be smaller than an a priori upper bound 2^ℓ. For assumptions, we list both the mathematical object it relies on and the concrete assumption. λ denotes the security parameter, $|C|$ the number of gates in C, n the input length, and D the depth. CRH stands for Correlation Robust Hash, LHE for linearly homomorphic encryption. Strong DCR refers to DCR where the secret exponent of the hard subgroup is chosen to be a random λ-bit number, instead of $O(\log N)$-bit number. ★ indicates that when the prime is $O(\log(\lambda))$-bit long, the size of garbling is $|C| \cdot \log p + \mathrm{poly}(\lambda) \cdot n$ and $|C| \cdot \log p + \mathrm{poly}(\lambda) \cdot (n + D)$ respectively, eliminating the hidden constant factor multiplied with $\log p$.

Concrete Succinctness. Our 1-bit-per-gate Boolean garbling scheme improves the concrete garbling size even with just a moderately large number of gates. Recall that asymptotically the garbling size is $|\hat{C}| = |C| + \mathrm{poly}(\lambda)$. Here the $\mathrm{poly}(\lambda)$ additive term represents the size of some global public data pd. The concrete size of this global data determines when using our schemes yields smaller garbled circuits compared with Yao-style garbled circuits. The break-even point depends on the instantiation, as well as the choice of a PRG seed length. For our estimation below and in Section 7, we optimistically assume an "HSS-friendly" PRG with 128-bit seed and output length $\approx |C|$. Designing MPC/FHE/HSS-friendly PRGs is an active research direction; see, e.g., [ARS+15, GRR+16, BCG+17, CCKK21, ABG+24, FLLL24, CCH+24] and references therein. For such PRGs, there is typically a tradeoff between computational cost and seed length; the size of the global data pd in our constructions scales linearly with the seed length. On the other hand, the number of restricted multiplications (between an intermediate value and an input bit) needed for evaluating each output bit of the PRG, which is upper bounded by the branching program size, directly influences the computational cost. While research on HSS-friendly PRGs is still in its infancy, there is a large space of possible designs to explore. We hope that the goal of practical succinct garbling will further motivate research on the concrete efficiency of HSS-friendly PRGs.

Table 3 in Section 7 summarizes the concrete sizes of the global data. The Paillier instantiation has just 0.38MB global data, the simple instantiation using Prime-order groups *with inverse polynomial errors*[5] has 5.1MB global data which can be optimized down to just 0.13MB, and

[5] As mentioned above, these errors can be made negligible via correctness and security amplification. The amplification step increases the size of the global data and computational efficiency by a factor of $\omega(1)$ factor

our lattice instantiation has 71MB global data. Comparing with using optimized Yao's garbled circuits with estimated size[6] of $\lambda|C|$, our 1-bit-per-gate garbling is smaller when the original circuit satisfies $|C| > |\mathsf{pd}|/(\lambda - 1)$. Concretely, the break-even point is $|C| = 24K$ for Paillier instantiation, $8K$ for optimized prime order groups, and $4.5M$ for lattices. For moderately large circuits, e.g., of size 10^7 gates, our optimized prime order group instantiation is smaller than Yao-style garbling by a factor of 116 (from 160MB to 1.38MB), while our Paillier group instantiation is smaller by a factor of 98 with size 1.63MB. For reference, the plain circuit description size is at least 29MB.

Compared with the recent work of Liu et al. [LWYY24] that constructed 1-bit-per-gate garbled circuits based on circular RLWE, the public data in the latter has a much larger size of $10GB$. This results from the use of the Gentry-Sahai-Water fully homomorphic encryption scheme [GSW13], which has much larger ciphertexts than the HSS encodings used in this work.

Finally, we compare with fully succinct garbling schemes. The iO-based constructions are not currently implementable, while the FHE+ABE-based constructions [GKP+13, BGG+14, HLL23] have astronomically large input labels and/or computational costs, and hence are also impractical. The label size for each input bit of the RLWE instantiation of the succinct garbling scheme of [GKP+13, BGG+14] is $\Omega(n^2 \log q^4)$, where n and q are RLWE degree and modulus satisfying $n^{1-\epsilon} > \log q > D$ for some $\epsilon \in (0, 1)$ and D is the circuit depth. This means each input label has size $\omega(D^6)$, which is prohibitive even for small depth such as 100. The recent work [HLL23] removes the constraint of $\log q > D$, allowing for smaller modulus and degree. However, this requires performing "boostrapping" inside ABE, which is very computationally expensive.

In summary, for circuits of moderate size around 10^5 to 10^6 gates, the garbled circuits of our schemes are concretely smaller than all prior constructions.

Towards Practical Succinct Garbling. Concretely, our garbling schemes require evaluating a PRG using HSS, in addition to a few other HSS operations per gate. Assuming each output bit of the PRG can be evaluated using a restricted multiplication straightline (RMS) program of size S, or alternatively a branching program of size S, then garbling and evaluating a general Boolean circuit require $|C| \cdot (4S + O(1))$ homomorphic RMS operations. In particular, since the HSS restricted multiplication operation is much more expensive than the HSS addition operation, if the PRG requires $S_\times$ restricted multiplications, the per-gate cost of garbling is dominated by $4S_\times + O(1)$ homomorphic restricted multiplication. As discussed above, we optimistically conjecture an HSS-friendly PRG with a large stretch (as our garbling size scales linearly with the seed length), and where each output bit can be evaluated using a reasonably small number of restricted multiplication operations. Then in the lattice instantiation, the per-gate computation boils down to computing a small number of multiplication/addition of $\mathcal{R}_q$ elements and rounding.

Our Assumptions. The leveled version of our unified framework can be based on natural flavors of the Power Decisional Diffie Hellman (P-DDH) assumption (Definition 6), introduced in [GJM03, CNs07, AHI11] and further used in [GHKW17, KY18, AMN+18, BMZ19, ILL24], in Paillier or prime-order groups. P-DDH postulates that for appropriately sampled group el-

asymptotically, but does not increase the per-gate communication cost. For concrete efficiency, we consider the simpler instantiation without amplification.

[6] The state-of-the-art optimization over Yao's garbled circuit is by [RR21], which contains 1.5λ bits per AND gate, and garbling XOR is free. We use $\lambda|C|$ as a rough estimation of the garbled circuit size.

ement g and exponents s and a, b sampled randomly from a range $[\ell]$, the triple (g, g^s, g^{s^2}) is indistinguishable from (g, g^a, g^b).

To remove the $D \cdot \mathrm{poly}(\lambda)$ additive term in the size of "leveled" garbled circuits, we need the following circular-security variant of this assumptions. The Circular Power Decisional Diffie Hellman (CP-DDH) assumption (Definition 7) asserts that a circular encryption of bits of the secret key s using powers of the secret key is pseudorandom. More precisely, for appropriately sampled group elements g, f and random exponents s and $\{a_i, b_i, c_i\}_i$, the following computational indistinguishability holds:

$$\textbf{CP-DDH: } g, g^s, g^{s^2}, (g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{s[i]})_{i\in[\log s]} \quad \approx_c \quad g, g^s, g^d, (g^{a_i}, g^{b_i}, g^{c_i})_{i\in[\log s]}.$$ [7]

The P-DDH and CP-DDH assumptions can be postulated over Paillier or prime order groups. For the Paillier group, the assumption can be further simplified (still sufficient for succinct garbling) to $(g^r, g^{rs}, g^{rs^2}(1+N)^s)$ being pseudorandom, where g is a generator of the hard subgroup and the exponents r, s are randomly sampled. This optimization is introduced for concrete efficiency; see Section 7. For prime-order groups, Power-DDH and Circular Power-DDH hold in the standard generic group model (GGM) [Sho97], as shown in [ILL24]. In particular, our succinct garbling scheme can be instantiated in the (prime-order) GGM, under the mild assumption of a PRF in NC^1. Furthermore, under the CP-DDH assumption in prime order groups, we only obtain succinct garbling with inverse polynomial errors. As mentioned above, we can amplify correctness and privacy to make the errors negligible without hurting the amortized per-gate garbling size. This requires a variant of the CP-DDH assumption, which instead of hiding the bits of the secret s, hides bits of the secret shifted by a public constant s', $t = s + s'$.

$$\textbf{CP-DDH*: } g, g^s, g^{s^2}, s', (g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{t[i]})_{i\in[\log s]} \quad \approx_c \quad g, g^s, g^d, (g^{a_i}, g^{b_i}, g^{c_i})_{i\in[\log s]}\ , \quad \text{where } t = s + s'$$

Alternatively, our garbling schemes can be based on the Power-RLWE assumption (Definition 8) for the leveled version and circular Power-RLWE assumption (Definition 9) for the full-fledged version. Introduced in [ARS24], Power-RLWE postulates that RLWE samples with *small* secrets s and s^2, and the same public vector $\mathbf{a}$ in a polynomial ring $\mathcal{R}_q$, $(\mathbf{a}, s\mathbf{a}+\mathbf{e}_1, s^2\mathbf{a}+\mathbf{e}_2)$ is pseudorandom. The circular variant further uses the last sample to hide the secret s, assuming the pseudorandomness of $(\mathbf{a}, s\mathbf{a} + \mathbf{e}_1, s^2\mathbf{a} + \mathbf{e}_2 + s\Delta)$, where Δ is a constant.

1.2 Related Works

In this section we provide a detailed comparison between our results and prior or concurrent related works.

Comparison with [ILL24]. The work of [ILL24] constructed fully succinct garbling schemes for weak classes of programs, including truth tables, DFA, and decision trees, based on different group-based assumptions. This builds on a fully succinct *partial garbling schemes* (equivalently, conditional disclosure of secrets), where most of the input is public. cIn comparison, our garbling schemes achieve a weaker level of succinctness, but apply to all circuits while fully hiding the input. Our work provides a lattice-based instantiation of the succinct partial garbling scheme from [ILL24] and the underling homomorphic MAC primitive.

[7] Compared to [ILL24], our formulation here includes g^{s^2} in the indistinguishability, which we believe is more natural and easier to use. See also the remark under Definition 7.

Comparison with [LWYY24]. Another recent work [LWYY24] constructed 1-bit-per-gate garbled circuits using special somewhat homomorphic encryption schemes, namely the GSW scheme instantiated using circular variants of RLWE or NTRU. In comparison, our unified framework presents a more general design principle using HSS. It yields instantiations based on more diverse assumptions that include different group-based assumptions. Our garbled circuits have smaller concrete sizes as discussed in the introduction (also see Table 3), owing to the fact that HSS encoding is smaller than GSW ciphertexts. In addition, we show how to go below 1-bit-per-gate for layered circuits as well as an extension to arithmetic garbling.

Comparison with [MORS25]. The concurrent and independent work of [MORS25] achieves a similar set of results to this work based on similar techniques. We note the following differences. For Boolean garbling, both [MORS25] and this work achieve (amortized) 1 bit per gate for general circuits, and $O(1/\log\log\lambda)$ bits per gate for layered circuits under circular assumptions. Both works have leveled variants that avoid circular assumptions at the price of an additive $\mathrm{Depth}(C)\cdot\mathrm{poly}(\lambda)$ size overhead. The differences are in the underlying assumptions: the work of [MORS25] focuses on constructions in Paillier groups based on a circular DCR assumption (resp., standard DCR for the leveled variant), while our work presents a unified framework with instantiations in Paillier groups, prime-order groups, or lattices, based on the CP-DDH or CP-RLWE assumptions (resp. P-DDH or P-RLWE for the leveled variants).

The difference in assumptions stems from the that [MORS25] use a more sophisticated variant of the basic technique to base their construction (in the leveled case) on the standard DCR assumption. While DCR is more widely used than P-DDH in Paillier groups used in our work, these assumptions seem technically incomparable. We believe that adapting the technique from [MORS25] to our constructions will give leveled variants under Paillier groups, prime-order groups, or lattices based on the standard DDH or RLWE with small secret assumptions. However, this seems to come at the price of a higher concrete overhead. The current work initiates a study of the concrete efficiency of group-based and lattice-based garbling, including an effort to optimize the additive terms.

For arithmetic garbling, the work of [MORS25] constructs a scheme over *bounded integers* by 2^ℓ, with (amortized) $(\ell+\lambda)$ bits per gate for general circuits, and $O((\ell+\lambda)/\log\log\lambda)$ bits per gate for layered circuits. In contrast, our work constructs schemes for computation over $\mathbb{Z}_R$ computation for *any modulus* R of ℓ bits, with (amortized) $O(\ell)$ bits per gate for general circuits. We believe that we can also obtain additional savings in cost for layered arithmetic circuits. Besides the distinction between supporting bounded integers vs. $\mathbb{Z}_R$ computation, the above differences in assumptions also hold for the arithmetic garbling results.

Comparison with [CHHK25]. The concurrent and independent work of [CHHK25] constructed Boolean garbling schemes with amortized per-gate garbling size below λ. Their first scheme is proven in the Generic Group model (GGM), achieving $\lambda/\sqrt{\log\lambda}$-bit-per-gate garbling size. Their second scheme is proven in the plain model under the Power-DDH assumption together with the existence of a tweakable correlation robust hash, attaining a garbled circuit size of $\lambda\cdot|C|/\sqrt{\log\lambda}+\mathrm{poly}(\lambda)\cdot D$, where D is the depth of the circuit for *layered* circuits. In comparison, our Boolean garbling schemes achieve 1-bit-per-gate for general circuits, and $O(1/\log\log\lambda)$-bit-per-gate for layered circuits, again, removing the $\tilde{\Omega}(\lambda)$ multiplicative overhead. On the other hand, our schemes make a non-black-box use of a PRF (or high-stretch PRG), whereas their constructions can be cast unconditionally in the GGM.

Other Use of HSS in Garbling by [GN25]. The recent work of [GN25] constructed a garbling scheme that supports mixed circuits with both Boolean and bounded integer arithmetics using

HSS techniques. The core innovation is using HSS techniques to implement an efficient garbling gadget for bit-decomposition that is compatible with the state-of-art arithmetic garbling scheme of [MORS24]. Overall, their scheme has a garbling size of (amortized) $O(\lambda)$ bits per Boolean gate, $(\ell + \lambda_{\mathsf{DCR}})$ bits per arithmetic gate (over integers bounded by 2^ℓ), and $O(\ell \cdot \lambda_{\mathsf{DCR}}/\log(\lambda))$ bits per bit-decomposition gate. In comparison, we apply HSS techniques to construct significantly more succinct Boolean and arithmetic garbling with (amortized) 1 bit per Boolean gate, and $O(\ell)$ bits per arithmetic gate (over an ℓ-bit modulus).

Inspiration from Arithmetic Garbling. Our work builds upon a recent line of research [BLLL23, LL24, Hea24, MORS24] for improving the garbling size of *arithmetic* circuits. These circuits consist of addition and multiplication gates, evaluated over a ring, typically $\mathbb{Z}_p$ or $\mathbb{Z}$, and an input x consists of ring elements. Because there is a simple baseline solution that uses a Boolean garbling scheme to garble a Boolean circuit implementing the arithmetic circuit of interest, research naturally focuses on what can be done differently. The first work on arithmetic garbling by Applebaum et al. [AIK11] proposed an aritsshmetic generalization of input keys and labels – the keys of an input wire describes an affine functions $\mathbf{K}_i$ and the label for x_i is the output $\mathbf{K}_i(x_i)$. They then constructed an garbling scheme for bounded integer computation with such arithmetic input labels, based on LWE, which sends $\ell \cdot \mathrm{poly}(\lambda)$ bits per gate when the wire values are bounded by 2^ℓ. Building upon [AIK11] and a subsequent work by Ball et al. [BMR16], recent works [BLLL23, LL24, Hea24, MORS24] have renewed research on arithmetic garbling on several different fronts: 1) diversifying assumptions, 2) supporting more models of computing, such as, $\mathbb{Z}_p$ computation, and mixed circuits with both arithmetic and Boolean gates, and 3) optimizing succinctness.

We focus on the succinctness aspect. The baseline solution using Yao's garbled circuits requires $\Omega(\lambda\ell \log \ell)$-bits-per-gate. Interestingly, the work of Ball et. al. [BLLL23] showed that bounded integer computations can be garbled with $O(\ell + \mathrm{poly}(\lambda))$-bits-per-gates, trading the $O(\lambda \log \ell)$ multiplicative factor for an additive $\mathrm{poly}(\lambda)$ term, assuming the DCR assumption over Paillier/Damgård-Jurik groups. Their technique relies on simple additive homomorphism supported by DCR, rather than iO or FHE plus ABE underlying fully succinct garbling. The work of [MORS24] further improved size to exactly $\ell + \mathrm{poly}(\lambda)$-bits-per-gate, by applying HSS techniques, assuming the circular security of Damgård-Jurik encryption.

These works shed new light on how to avoid the $O(\lambda)$-multiplicative factor overhead associated with Yao's garbled circuits, using lightweight tools. But their techniques are limited in two ways. First, the additive $\mathrm{poly}(\lambda)$ is large, proportional to $\log N$ where N is the Paillier modulus, and dominates when wire values are relatively small $\ell = o(\log N)$. In particular, when used to garble Boolean computation $\ell = 1$, the size is $O(\log N)$-bits-per-gate, worse than Yao's garbled circuits. Second, their methods do not extend to garbling $\mathbb{Z}_p$-arithmetic circuits. Despite past efforts [AIK11, BMR16, BLLL23, LL24, Hea24], the most succinct garbled $\mathbb{Z}_p$-circuits have size $\Omega(\lambda \log p)$-bit-per-gate, carrying the $\Omega(\lambda)$-multiplicative factor overhead.

As discussed before, the current work overcomes the above two limitations. Our unified framework also gives a $\mathbb{Z}_p$-garbling scheme with $O(\log p)$-bits-per-gate for general p, based on various group and lattice assumptions. Our technique is inspired by techniques developed in the context of arithmetic garbling, particularly the HSS-based technique from [MORS24].

2 Technical Overview

Starting Point: Succinct Garbling of [ILL24]. Our starting point is the recent new approach to succinct garbling from [ILL24], which combines a new primitive called fully succinct *partial* garbling and fully homomorphic encryption (FHE) to obtain fully succinct standard garbling. This follows the FHE+ABE blueprint of [GKP+13, BGG+14], replacing succinct ABE by succinct partial garbling.

In more detail, a partial garbling scheme generalizes standard garbling to consider computations with public and private parts, $C(\mathbf{x}, \mathbf{y}) = C_{\mathrm{Priv}}(\mathbf{y}, C_{\mathrm{Pub}}(\mathbf{x}))$. A partial garbling of C computes a garbling $\widehat{C}$ and a pair of short keys $\mathbf{k}_{x,i,0}, \mathbf{k}_{x,i,0}$ for every bit in $\mathbf{x}$, as well as $\mathbf{k}_{y,i,0}, \mathbf{k}_{y,i,1}$ for every bit in $\mathbf{y}$. The garbling $\widehat{C}$, the keys $\{\mathbf{k}_{x,i}\}$, $\{\mathbf{k}_{y,i}\}$ selected corresponding to inputs $\mathbf{x}, \mathbf{y}$, together with $\mathbf{x}$ in the clear reveals the computation result $\mathbf{z} = C(\mathbf{x}, \mathbf{y})$, and nothing else about the private input $\mathbf{y}$. The scheme of [ILL24] achieves a fully succinct garbling size $|\widehat{C}| \leq |C_{\mathrm{Priv}}| \cdot \mathrm{poly}(\lambda)$, independent of the complexity of C_{Pub}.

The observation from [ILL24] then is to apply partial garbling to the computation

$$C(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}) = \mathsf{Dec}(\mathsf{sk}, \mathsf{HEval}^f(\mathsf{ct}_{\mathbf{x}})),$$

i.e. with a public part $C_{\mathrm{Pub}}(\mathsf{ct}_{\mathbf{x}}) = \mathsf{HEval}^f(\mathsf{ct}_{\mathbf{x}}) = \mathsf{ct}^*_{\mathbf{z}}$ computing homomorphic evaluation of some function f over FHE ciphertexts $\mathsf{ct}_{\mathbf{x}}$, and a private part $C_{\mathrm{Priv}}(\mathsf{sk}, \mathsf{ct}^*_{\mathbf{z}})$ decrypting evaluated ciphertexts using the secret key sk as the private input. A partial garbling of C reveals the evaluation result $\mathbf{z} = f(\mathbf{x})$, and guarantees privacy of the secret key sk, which further guarantees privacy of $\mathbf{x}$ by FHE security. Therefore, a partial garbling of C can be viewed as a standard garbling of the function f. Furthermore, the size of $\widehat{C}$ only depends on the complexity of private computation, i.e. FHE decryption, $|\widehat{C}| \leq |\mathbf{z}| \cdot |\mathsf{Dec}| \cdot \mathrm{poly}(\lambda) = |\mathbf{z}| \cdot \mathrm{poly}(\lambda)$, and does not depend on the complexity of f. Hence the partial garbling of C is a fully succinct standard garbling of f.

While conceptually simple, this solution is far from practically useful due to the heavy computation complexity of FHE. A natural attempt is to use a less powerful, but much lighter-weight, homomorphic encryption (HE) scheme to obtain fully succinct garbling for many low-depth computations $\{f^i\}$, and composing them into a high-depth one: $f := f^T \circ f^{T-1} \circ \ldots \circ f_1$. However, some calculation shows a difficulty. Suppose each f^i is a Boolean circuit of depth D with width W. The succinct garbling of f^i costs $|\widehat{f^i}| = W \cdot \mathrm{poly}(\lambda)$, while our target size is $\leq WD \cdot \log(WD)$ to achieve succinctness. Without new ideas, we would need a powerful HE supporting $D = \mathrm{poly}(\lambda)$ depth computation to achieve succinctness.

Indeed, our new ideas require looking into the construction of [ILL24], and finding new ways to garble the private computation C_{Priv} more efficiently, at the cost of supporting only a restricted form of computation.

The Construction of [ILL24] in More Detail. The partial garbling construction of [ILL24] relies on a new primitive, algebraic homomorphic MAC (aHMAC), and the standard Yao's Boolean garbling to handle the public and private computations respectively. We give a simplified review here, assuming the the free-XOR [KS08] key format in Yao's garbling.

The aHMAC Scheme. An aHMAC scheme is run between an authenticator and an evaluator. They both hold an evaluation key evk. The authenticator additionally holds a PRF key k, and a global secret $s \in \mathbb{Z}$.

- The authenticator when given a bounded integer $x_i \in [B]$ as input, and an associated id, computes its tag as $\sigma_{x_i} := s \cdot x_i + k_x^{(i)}$ over $\mathbb{Z}$, where $k_x^{(i)} = \mathsf{PRF}(\mathsf{k}, \mathsf{id})$ is derived from the id.

– The evaluator when given inputs $\mathbf{x}$ and tags $\sigma_{\mathbf{x}} = \{\sigma_{x_i}\}$ can evaluate any arithmetic circuit C (with bounded intermediate values by B) using the evaluation key: $\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \sigma_{\mathbf{x}}, \mathbf{x})$.
– The authenticator when given only the ids, hence the derived keys $\mathbf{k}_x = \{k_x^{(i)}\}$, can evaluate the same circuit: $\mathbf{k}_z \leftarrow \mathsf{EvalKey}(\mathsf{evk}, C, \mathbf{k}_x)$.

The scheme guarantees the evaluated tags and keys are consistent: $\sigma_{\mathbf{z}} = s \cdot \mathbf{z} + \mathbf{k}_z$ over $\mathbb{Z}$, and also that the evaluation key evk and tags $\sigma_{\mathbf{x}}$ don't leak anything about the global secret s.

In this work, we view a pair of tag and key $\sigma_{x_i}, k_x^{(i)}$ as an additive share of sx_i over $\mathbb{Z}$, written as $\langle sx_i \rangle_0 = k_x^{(i)}$, and $\langle sx_i \rangle_1 = \sigma_{x_i}$. We view the algorithms $\mathsf{EvalKey}, \mathsf{EvalTag}$ as homomorphically evaluating additive shares of $s\mathbf{x}$ between a garbler P_G and an evaluator P_E, who both hold an evaluation key evk with respect to the global secret s. Note that the $\mathsf{EvalTag}$ algorithm by the evaluator also needs $\mathbf{x}$ in the clear.

$$\underline{P_G(\mathsf{evk})} \qquad\qquad \underline{P_E(\mathsf{evk}, \mathbf{x})}$$
$$\langle s\mathbf{z} \rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{evk}, C, \langle s\mathbf{x} \rangle_0), \quad \langle s\mathbf{z} \rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \langle s\mathbf{x} \rangle_1, \mathbf{x}).$$

The construction of [ILL24] guarantees that given any additive shares of $s\mathbf{x}$, as long as all intermediate values of $C(\mathbf{x})$ are bounded by B, the results of $\mathsf{EvalKey}, \mathsf{EvalTag}$ also form additive shares of $s\mathbf{z}$, where $\mathbf{z} = C(\mathbf{x})$.

Yao's Garbling. In Yao's garbling of a Boolean circuit C (assuming the free-XOR [KS08] key format), the garbler P_G samples a random key k_j for every wire j in C, and a global secret s. We view the keys $\{k_j\}$ and the global secret s as $O(\lambda)$-bit integers in this overview.

P_G provides a garbled table for each gate to the evaluator P_E, such that if P_E obtains a set of labels $\{l_i = s \cdot x_i + k_i\}$ according to an input $\mathbf{x} = \{x_i\}$, then she can use the garbled tables to recover a label $l_j = s \cdot v_j + k_j$ for every wire j corresponding to the correct wire value v_j. In order for P_E to recover the values z_o on the output wires o in C, a usual trick is to assume the least significant bit (LSB) of s is 1, so that $\mathrm{LSB}(l_o) = z_o \oplus \mathrm{LSB}(k_o)$. It suffices for P_G to send P_E $\mathrm{LSB}(k_o)$ for every output wire o.

We take an alternative view of Yao's garbling not as a static scheme, but as a protocol between a garbler P_G and an evaluator P_E.

– Initially P_G and P_E jointly hold additive shares of $s\mathbf{x}$ for some input $\mathbf{x} = \{x_i\}$: the garbler holds $\langle sx_i \rangle_0 = k_i$, and the evaluator holds $\langle sx_i \rangle_1 = l_i$.
– Then P_G sends garbled tables to P_E so that they jointly hold additive shares of sv_j for every wire value v_j in C.
– In the end, P_G and P_E jointly hold additive shares of $s\mathbf{z}$ for the output $\mathbf{z} = \{z_o\}$: the garbler holds $\langle sz_o \rangle_0 = k_o$, and the evaluator holds $\langle sz_o \rangle_1 = l_o$. P_G then sends $\{\mathrm{LSB}(k_o)\}$ to P_E to reveal $\mathbf{z}$.

The security of Yao's garbling guarantees that if the global secret s is not leaked by the initial additive shares $\langle s\mathbf{x} \rangle_1$ to P_E, then all communication from P_G to P_E can be simulated by P_E, given only the output $\mathbf{z}$. To summarize the protocol between P_G and P_E, we write

$$(P_G : \langle s\mathbf{z} \rangle_0), (P_E : \langle s\mathbf{z} \rangle_1, \mathbf{z}) \leftarrow \mathsf{Yao}^C\big((P_G : \langle s\mathbf{x} \rangle_0), (P_E : \langle s\mathbf{x} \rangle_1)\big).$$

Succinct Partial Garbling from aHMAC and Yao. We again describe the partial garbling scheme for evaluating $C(\mathbf{x}, \mathbf{y}) = C_{\mathrm{Priv}}(\mathbf{y}, C_{\mathrm{Pub}}(\mathbf{x}))$ as a protocol between the garbler P_G and the evaluator P_E, which we believe is more intuitive. (See Section 5 for viewing garbling as a 2PC protocol.) It represents a valid garbling scheme as long as P_G's communication is independent of the inputs $\mathbf{x}, \mathbf{y}$ except in an initialization phase.

1. In the initialization phase, P_G sets up the aHMAC scheme with a global secret s, and evaluation key evk. He then samples random additive shares $\langle s\mathbf{x}\rangle_0$, $\langle s\mathbf{x}\rangle_1$, for the public input $\mathbf{x}$, and $\langle s\mathbf{y}\rangle_0$, $\langle s\mathbf{y}\rangle_1$ for the private input $\mathbf{y}$. In the end, P_G sends evk, $\mathbf{x}$, $\langle s\mathbf{x}\rangle_1$ and $\langle s\mathbf{y}\rangle_1$ to P_E.
2. To evaluate the public computation C_{Pub}, [8] P_G and P_E locally run $\mathsf{EvalKey}$ and $\mathsf{EvalTag}$ respectively on their shares $\langle s\mathbf{x}\rangle_0$ and $\langle s\mathbf{x}\rangle_1$.

$$\begin{aligned} P_G: &\quad \langle s\mathbf{w}\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{evk}, C_{\mathrm{Pub}}, \langle s\mathbf{x}\rangle_0), \\ P_E: &\quad \langle s\mathbf{w}\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C_{\mathrm{Pub}}, \langle s\mathbf{x}\rangle_1, \mathbf{x}). \end{aligned} \tag{1}$$

3. To evaluate the private computation C_{Priv}, P_G and P_E jointly run Yao's garbling.

$$\begin{aligned} &(P_G: \langle s\mathbf{z}\rangle_0), (P_E: \langle s\mathbf{z}\rangle_1, \mathbf{z}) \\ &\quad \leftarrow \mathsf{Yao}^{C_{\mathrm{Priv}}}\big((P_G: \langle s\mathbf{y}\rangle_0, \langle s\mathbf{w}\rangle_0), (P_E: \langle s\mathbf{y}\rangle_1, \langle s\mathbf{w}\rangle_1)\big). \end{aligned} \tag{2}$$

   The evaluator P_E outputs $\mathbf{z}$ in the end.

In the above protocol, communication from P_G to P_E after the initialization phase corresponds to the garbling material $\widehat{C}$ in the garbling scheme. We note since the public computation C_{Pub} is evaluated by local procedures, with no communication, we indeed obtain a fully succinct partial garbling scheme.

Recall that in this work, we intend to perform homomorphic evaluation of some low-depth computation f^i over HE ciphertexts $\mathsf{ct}_{\mathbf{x}^i}$ using the public computation, and then HE decryption using the private computation. Furthermore, in order to compose multiple such evaluations, $\ldots f^{i+1} \circ f^i \circ \ldots$, we need to also implement HE re-encryption using the private computation. We illustrate the modified steps 2 and 3 below.

2' To evaluate the public computation $C_{\mathrm{Pub}} := \mathsf{HEval}^{f^i}$, P_G and P_E locally run $\mathsf{EvalKey}$ and $\mathsf{EvalTag}$ respectively on their shares $\langle s \cdot \mathsf{ct}_{\mathbf{x}^i}\rangle_0$, $\langle s \cdot \mathsf{ct}_{\mathbf{x}^i}\rangle_1$. [9]

$$\begin{aligned} P_G: &\quad \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}}\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{evk}, \mathsf{HEval}^{f^i}, \langle s \cdot \mathsf{ct}_{\mathbf{x}^i}\rangle_0), \\ P_E: &\quad \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}}\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{evk}, \mathsf{HEval}^{f^i}, \langle s \cdot \mathsf{ct}_{\mathbf{x}^i}\rangle_1, \mathsf{ct}_{\mathbf{x}^i}), \end{aligned}$$

   where ct^* denotes homomorphically evaluated ciphertexts.

3' To evaluate the private computation $C_{\mathrm{Priv}} := \mathsf{Enc} \circ \mathsf{Dec}$, P_G and P_E jointly run Yao's garbling.

$$\begin{aligned} &(P_G: \langle s \cdot \mathsf{ct}_{\mathbf{x}^{i+1}}\rangle_0), (P_E: \langle s \cdot \mathsf{ct}_{\mathbf{x}^{i+1}}\rangle_1, \mathsf{ct}_{\mathbf{x}^{i+1}}) \\ &\quad \leftarrow \mathsf{Yao}^{\mathsf{Enc}\circ\mathsf{Dec}}\big((P_G: \langle s \cdot \mathsf{sk}\rangle_0, \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}}\rangle_0), (P_E: \langle s \cdot \mathsf{sk}\rangle_1, \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}}\rangle_1)\big). \end{aligned}$$

   Note that the results are shares of fresh HE ciphertexts $\mathsf{ct}_{\mathbf{x}^{i+1}}$, so the parties can then repeat Step 2' and 3' for the next evaluation of f^{i+1}.

As explained, the communication by Yao's garbling to implement $\mathsf{Enc} \circ \mathsf{Dec}$ is too much for our purpose. Instead, our idea is to use homomorphic secret sharing (HSS) to replace Yao's garbling in the above protocol.

[8] A Boolean circuit C_{Pub} can be implemented by an arithmetic circuit over integers bounded by 2.

[9] Technically, we mean shares of $s \cdot \mathsf{Bits}(\mathsf{ct}_{\mathbf{x}^i})$ here, and shares of $s \cdot \mathsf{Bits}(\mathsf{sk})$ in step 3'. But we choose to abuse notations to avoid cluttering.

It may first seem a bit odd to consider HSS as a replacement for garbling. Indeed, in the setting of HSS, both parties *depend* on the input, while in the setting of garbling, P_G needs to be independent of the input. Our observation is that in the private computation implemented by Yao, $\mathsf{Enc} \circ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathbf{x}})$, the most complicated computations, e.g. evaluating a PRG, involve only the secret key sk, which is indeed independent of the actual input $\mathbf{x}$! One can therefore hope to rely on HSS for the complicated computations involving only sk, and in the end incorporate $\mathsf{ct}_{\mathbf{x}}$ in the remaining simpler steps.

Replacing Yao with HSS. An HSS scheme runs between two parties P_0, P_1. In common constructions, such as [ADOS22, BGI16, BKS19], they both hold encryptions of an input $\mathbf{y}$, denoted $I_{\mathbf{y}}$, and jointly an additive share of a global secret s over $\mathbb{Z}$ consistent with the encryptions. Each party P_b can locally evaluate any NC1 Boolean circuits C over the encrypted inputs via $\mathsf{HSS.Eval}_b$ such that the two outputs form additive shares of $s \cdot \mathbf{z}$ and $\mathbf{z}$, where $\mathbf{z} = C(\mathbf{y})$ is the evaluation result.

$$\underline{P_0(I_{\mathbf{y}}, \langle s \rangle_0)} \qquad \underline{P_1(I_{\mathbf{y}}, \langle s \rangle_1)}$$
$$\langle s\mathbf{z} \rangle_0, \langle \mathbf{z} \rangle_0 \leftarrow \mathsf{HSS.Eval}_0(I_{\mathbf{y}}, C, \langle s \rangle_0), \quad \langle s\mathbf{z} \rangle_1, \langle \mathbf{z} \rangle_1 \leftarrow \mathsf{HSS.Eval}_1(I_{\mathbf{y}}, C, \langle s \rangle_1).$$

In an additional step, the party P_0 may send its share $\langle \mathbf{z} \rangle_0 \pmod 2$ to P_1 to reveal the Boolean evaluation result $\mathbf{z}$.

It was observed in [CMPR23] that the above HSS schemes allow for an extended evaluation procedure, where if replacing the additive share of s with shares of $s \cdot w$ and w for some integer w, then the extended evaluation results form additive shares of $s \cdot w \cdot \mathbf{z}$ and $w \cdot \mathbf{z}$. In other words, the HSS evaluation results over encrypted inputs $\mathbf{y}$ can be additionally multiplied with an integer w, when the two parties hold additive shares of sw and w.

$$\underline{P_0(I_{\mathbf{y}}, \langle s \rangle_0)} \qquad \underline{P_1(I_{\mathbf{y}}, \langle s \rangle_1)}$$
$$\langle sw\mathbf{z} \rangle_0, \langle w\mathbf{z} \rangle_0 \qquad \langle sw\mathbf{z} \rangle_1, \langle w\mathbf{z} \rangle_1$$
$$\leftarrow \mathsf{ExtEval}(I_{\mathbf{y}}, C, \langle sw \rangle_0, \langle w \rangle_0), \qquad \leftarrow \mathsf{ExtEval}(I_{\mathbf{y}}, C, \langle sw \rangle_1, \langle w \rangle_1).$$

Taking the extended evaluation one step further, we can consider a matrix $\mathbf{W}$ as the additional input, and compute $\mathbf{z}' = \mathbf{W} \cdot C(\mathbf{y})$ (over $\mathbb{Z}$) as the final output. Including the additional step where P_0 sends its share of $\mathbf{z}' \pmod 2$ to P_1 to reveal $\mathbf{z}'$, we obtain an HSS evaluation "protocol" for NC1 Boolean circuits C, denoted HSS^C:

$$(P_G : \langle s\mathbf{z}' \rangle_0), (P_E : \langle s\mathbf{z}' \rangle_1, \mathbf{z}')$$
$$\leftarrow \mathsf{HSS}^C\big((P_G : I_{\mathbf{y}}, \langle s\mathbf{W} \rangle_0, \langle \mathbf{W} \rangle_0), (P_E : I_{\mathbf{y}}, \langle s\mathbf{W} \rangle_1, \langle \mathbf{W} \rangle_1)\big),$$

which we replace Yao's garbling with in step 3 (Equation 2) and 3' from the previous paragraph. [10] The communication cost from HSS is only $|\mathbf{z}'|$ bits, much smaller than that of Yao.

One detail to note is that in Yao's garbling we are free to use the global secret s from aHMAC also as the secret in Yao, but now with HSS, we need compatible instantiations with aHMAC, (see Section 4) so that they can share a common secret s. This usage of aHMAC and HSS requires us to prove security of the overall garbling scheme in a non-black-box way.

As anticipated, the computation implemented by this protocol is restricted: $\mathbf{z}' = \mathbf{W} \cdot C(\mathbf{y})$ over $\mathbb{Z}$, for an NC1 circuit C. In order to use $\mathsf{HSS}^{\mathsf{Enc} \circ \mathsf{Dec}}$ in place of $\mathsf{Yao}^{\mathsf{Enc} \circ \mathsf{Dec}}$ in Step 3', we need

[10] Readers may notice a mismatch, where from step 2 (Equation 1), the parties hold shares of $s\mathbf{w}$, but not of $\mathbf{w}$. This is not an issue, as P_E can compute $\mathbf{w}$ in the clear from the public input $\mathbf{x}$. The parties now hold a trivial share: $\langle \mathbf{w} \rangle_0 = 0$, $\langle \mathbf{w} \rangle_1 = \mathbf{w}$.

to find a suitable HE scheme where the $\mathsf{Enc} \circ \mathsf{Dec}$ circuit can be implemented in this restricted way:

$$\underbrace{\mathsf{ct}_{\mathbf{x}}}_{\mathbf{z}'} = \mathsf{Enc} \circ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*_{\mathbf{x}}) = \underbrace{\mathsf{Bits}(\mathsf{ct}^*_{\mathbf{x}})}_{\mathbf{W}} \cdot \underbrace{C(\mathsf{sk})}_{C(\mathbf{y})} \text{ over } \mathbb{Z}.$$

While such an HE scheme may seem hard to find, our observation is that the size of evaluated ciphertexts $|\mathsf{ct}^*_{\mathbf{x}}|$ don't matter in our scheme, as the communication cost from HSS is exactly the size of a fresh ciphertext $|\mathsf{ct}_{\mathbf{x}}|$. In fact, viewing one-time-pad as a trivial HE scheme suffices! We illustrate a simple case of homomorphically multiplying one-time-pad ciphertexts, and the $\mathsf{Enc} \circ \mathsf{Dec}$ computation.

$$\begin{aligned}
&\mathsf{ct}_x := x \oplus r_x, \quad \mathsf{ct}_y := y \oplus r_y, \quad \text{where } r_x = \mathsf{PRF}(\mathsf{sk}, 1), r_y = \mathsf{PRF}(\mathsf{sk}, 2).\\
&\mathsf{ct}^*_z = \mathsf{HMult}(\mathsf{ct}_x, \mathsf{ct}_y) = (\mathsf{ct}_x, \mathsf{ct}_y, \mathsf{ct}_x \cdot \mathsf{ct}_y).\\
&\mathsf{Enc} \circ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*_z) = (\mathsf{ct}_x \oplus r_x) \cdot (\mathsf{ct}_y \oplus p_2) \oplus r_z, \text{ where } r_z = \mathsf{PRF}(\mathsf{sk}, 3)\\
&\qquad\qquad = C_1(\mathsf{sk})\mathsf{ct}_x + C_2(\mathsf{sk})\mathsf{ct}_y + C_3(\mathsf{sk})\mathsf{ct}_x \cdot \mathsf{ct}_y + C_4(\mathsf{sk}) \text{ over } \mathbb{Z}.\\
&\qquad\qquad = \mathsf{Bits}(\mathsf{ct}^*_z) \cdot C(\mathsf{sk}) \text{ over } \mathbb{Z} \text{ where } C := (C_1, C_2, C_3, C_4).
\end{aligned}$$

The final equality, writing Boolean operations as a polynomial over $\mathbb{Z}$, uses the fact that $x \oplus y = x + y - 2xy$ over $\mathbb{Z}$ for $x, y \in \{0, 1\}$. In the following, we directly write $\overline{\mathbf{x}}$ to denote one-time-padded $\mathbf{x}$, instead of $\mathsf{ct}_{\mathbf{x}}$.

In summary, our final Boolean garbling scheme for a circuit C starts with two parties P_G, P_E holding additive shares $\langle s\overline{\mathbf{x}} \rangle$ and P_E holding $\overline{\mathbf{x}}$ in the clear, where $\overline{\mathbf{x}}$ represents one-time-padded inputs. For every gate in C, in a topological order, both parties applie aHMAC evaluations to "homomorphically" add or multiply two one-time-padded inputs, and then run HSS to decrypt and re-encrypt the resulting bit. The communication cost per gate is excatly 1-bit from the HSS protocol.

Generalization: Evaluating $O(\log \lambda)$-ary Gates. Observe that the technique of combining aHMAC and HSS from the previous paragraph can be viewed as a more general protocol for computation over some public masked input $\overline{\mathbf{x}}$ and a private secret key sk for deriving the masks. Using aHMAC we can evaluate any arithmetic circuit C_{Pub} (with bounded intermediate values) on $\overline{\mathbf{x}}$, and with HSS we can evaluate any NC1 Boolean circuit C_{Priv} on sk. The two results are then multiplied as an inner product over $\mathbb{Z}$. We summarize it as a protocol $\mathsf{aHMAC\text{-}HSS}^{C_{\text{Pub}}, C_{\text{Priv}}}$:

$$\begin{aligned}
&(P_G : \langle sz' \rangle_0), (P_E : \langle sz' \rangle_1, z')\\
&\qquad \leftarrow \mathsf{aHMAC\text{-}HSS}^{C_{\text{Pub}}, C_{\text{Priv}}}\big((P_G : I_{\mathsf{sk}}, \langle s\overline{\mathbf{x}} \rangle_0), (P_E : I_{\mathsf{sk}}, \langle s\overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}})\big),\\
&// \ z' = \langle C_{\text{Pub}}(\overline{\mathbf{x}}), C_{\text{Priv}}(\mathsf{sk}) \rangle.
\end{aligned}$$

The communication cost of this protocol is 1 bit. Note that the result z' is revealed to the evaluator P_E, hence should always be masked by a pseudo-random pad derived from sk.

Given this more general view, we can in fact use $\mathsf{aHMAC\text{-}HSS}^{C_{\text{Pub}}, C_{\text{Priv}}}$ to compute any function g over $O(\log \lambda)$ masked input bits, and re-mask the resulting value. In particular, we choose $C_{\text{Pub}}(\overline{\mathbf{x}})$ to compute a one-hot vector $(0, \ldots, 0, 1, 0, \ldots, 0)$, where all but the $\overline{\mathbf{x}}$-th component are 0. (See Fact 1.) And we choose $C_{\text{Priv}}(\mathsf{sk}) = (\ldots, C_{\text{Priv},\mathbf{v}}(\mathsf{sk}), \ldots)_{\mathbf{v}}$ to compute a vector listing evaluated values $g(\mathbf{x})$ (and then masked) for all possible values of $\overline{\mathbf{x}}$.

$$\begin{aligned}
\forall \mathbf{v} \in \{0,1\}^{|\overline{\mathbf{x}}|}, \quad & C_{\text{Pub},\mathbf{v}}(\overline{\mathbf{x}}) = 1 \text{ iff } \overline{\mathbf{x}} = \mathbf{v}\\
& C_{\text{Priv},\mathbf{v}}(\mathsf{sk}) = g(\mathbf{v} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{id})) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{id}').\\
// \ & z' = \langle C_{\text{Pub}}(\overline{\mathbf{x}}), C_{\text{Priv}}(\mathsf{sk}) \rangle = g(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{id}').
\end{aligned}$$

The $\mathsf{id}, \mathsf{id}'$ from the above means some distinct ids assigned to every wire of the overall circuit consisting of these $O(\log \lambda)$-ary gates.

In summary, our generalized technique can garble Boolean circuits consisting of arbitrary $O(\log \lambda)$-ary gates, costing 1 bit per such gate. As applications, we show how to obtain a scheme for *layered* circuits C^{Layer} with garbling size $|\widehat{C^{\mathsf{Layer}}}| \leq O(|C^{\mathsf{Layer}}|/\log\log\lambda)+\mathrm{poly}(\lambda)$ in Section 5, and a scheme for arithmetic circuits C over $\mathbb{Z}_R$ with garbling size $|\widehat{C}| \leq O(|C|\log R) + \mathrm{poly}(\lambda)$ in Section 6.

Other Extensions. Our techniques rely on two primitives:

- aHMAC which has been instantiated under the circular power-DDH (CP-DDH) assumptions in Paillier groups or prime-order groups in [ILL24];
- HSS which has been instantiated under the DDH assumption in Paillier groups [ADOS22], prime-order groups [BGI16], and the RLWE assumption [BKS19].

We introduce a new lattice assumption, CP-RLWE, analogous to the CP-DDH assumption in groups, and show three instantiations of our technique of combining aHMAC and HSS under either CP-DDH in Paillier groups, in prime-order groups, or CP-RLWE. As noted earlier, since we require using a common secret in both aHMAC and HSS, we have to prove the security of our garbling schemes in a non-black-box way.

The work of [ILL24] also constructed leveled variants of aHMAC that avoids the circular assumptions at the cost of a larger evaluation key evk with size linear in the supported evaluation depth. We also construct leveled garbling schemes using leveled aHMAC and (normal) HSS at the cost of increasing the garbling size by $\mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda)$ bits. They can be instantiated under P-DDH plus DDH in Paillier groups, P-DDH in prime-order groups, or P-RLWE. (See Section 5.2 for details.)

Finally, we note that existing aHMAC and HSS instantiations under prime-order groups suffer a $1/\mathrm{poly}(\lambda)$ correctness error. This causes a $1/\mathrm{poly}(\lambda)$ error for both correctness and *privacy* in our garbling scheme under prime-order groups. We show in Section 5.5 how to adapt existing HSS amplification techniques [BGI17] to our setting to remove the $1/\mathrm{poly}(\lambda)$ error at the price of increased computation cost and, in the non-leveled variant, assuming a variant of CP-DDH (Definition 11).

3 Preliminaries

Notations. We use bold letters $\mathbf{x}$ to denote a vector, and write $\mathbf{x}[i]$ to denote its i-th component. We write $\mathbf{x} \otimes \mathbf{y}$ to denote the tensor product between two vectors. For an integer value within some range $x \in [B]$, we write $\mathsf{Bits}(x)$ to denote its bit-representation as a Boolean vector of dimension $\lceil \log B \rceil$, and $\mathsf{BitComp}(\mathbf{x} \in \{0,1\}^{\lceil \log B \rceil})$ to denote the linear function that recovers x from its bit-representation.

We write $\langle x \rangle_0, \langle x \rangle_1$ to denote a pair of additive shares (over a ring $\mathcal{R}$) of the value x, i.e. the notation represents two arbitrary values $v_0, v_1 \in \mathcal{R}$ such that $v_1 = v_0 + x$ over $\mathcal{R}$. In this work we will consider additive shares over the integers $\mathcal{R} = \mathbb{Z}$ and over the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where n is a power-of-two.

When describing invocations of (sub-)protocols between two parties P_G, P_E, we write

$$(P_G : O_G), (P_E : O_E) \leftarrow \mathsf{Protocol}\,((P_G : I_G), (P_E : I_E))$$

to mean the parties respectively hold inputs I_G, I_E when entering the protocol, and obtain outputs O_G, O_E after the protocol.

We assume all gates and wires in a circuit are labeled by distinct ids in $\{0,1\}^\lambda$. We write $\mathsf{InWires}(C)$ to denote the ids of all input wires to C, and $\mathsf{InWires}(\mathsf{g}), \mathsf{OutWire}(\mathsf{g})$ to respectively denote the ids of input wires to, and output wire from a gate $\mathsf{g} \in C$.

When writing invocations of a function $f : \mathcal{X} \to \mathcal{Y}$, we use the short-hand $f(\mathbf{x} \in \mathcal{X}^\ell) \in \mathcal{Y}^\ell$ to mean parallel invocations of f on every component of the vector $\mathbf{x}$. For example, given a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$, we write

$$\overline{\mathbf{x}} = \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g}))$$

to mean computing masked inputs $\overline{\mathbf{x}}$ to some gate g using parallel invocations of a PRF (w.r.t. different wire ids) under a secret key sk.

3.1 Definition of Garbling

Definition 1 (Garbling). *A garbling scheme consists of two efficient algorithms:*

- $\mathsf{Garb}(1^\lambda, C)$ *takes a circuit* $C : \mathcal{R}^{\ell_x} \to \mathcal{R}^{\ell_z}$*, over some ring* $\mathcal{R}$*, and outputs a garbling* $\widehat{C}$*, and input key functions* $\{K^{(i)}\}_{i\in[\ell_x]}$*, where each key function* $K^{(i)}$ *maps an input* $\mathbf{x}[i] \in \mathcal{R}$ *to a label* $L^{(i)} \in \{0,1\}^\ell$*, where the label length is bounded by a fixed polynomial in* λ *and the bit-length of* $\mathcal{R}$*, independent of the circuit size* $|C|$*:* $\ell \leq \mathrm{poly}(\lambda, |\mathcal{R}|)$
- $\mathsf{Eval}(C, \widehat{C}, \{L^{(i)}\}_{i\in[\ell_x]})$ *takes a circuit* C*, a garbling* $\widehat{C}$*, and input labels* $L^{(i)}$ *(corresponding to some input* $\mathbf{x} \in \mathcal{R}^{\ell_x}$*). It outputs the evaluation result* $\mathbf{z} \in \mathcal{R}^{\ell_z}$*.*

Correctness: *For every polynomials* $p(\lambda), p'(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, circuits* C *with size* $|C| \leq p(\lambda)$*, over rings* $\mathcal{R}$ *with bit-length* $|\mathcal{R}| \leq p'(\lambda)$*, and inputs* $\mathbf{x} \in \mathcal{R}^{\ell_x}$*, the following holds:*

$$\Pr\left[\begin{array}{l|r} \mathsf{Eval}(C, \widehat{C}, \{L^{(i)}\}) & (\widehat{C}, \{K^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C), \\ = C(\mathbf{x}) & L^{(i)} = K^{(i)}(\mathbf{x}[i]). \end{array}\right] \geq 1 - \mathrm{negl}(\lambda).$$

Security: *There exists an efficient simulator* Sim *such that for every polynomials* $p(\lambda), p'(\lambda)$*, sequence of circuits* $\{C_\lambda\}$ *where* $|C_\lambda| \leq p(\lambda)$*, over rings* $\mathcal{R}_\lambda$ *with bit-lengths* $|\mathcal{R}_\lambda| \leq p'(\lambda)$ *and sequence of inputs* $\{\mathbf{x}_\lambda \in \mathcal{R}_\lambda^{\ell_x}\}$*, the following holds (suppressing the subscript* λ *for brevity):*

$$\Big\{\mathsf{Sim}(1^\lambda, C, C(\mathbf{x}))\Big\}_\lambda \approx_c \left\{\widehat{C}, \{L^{(i)}\}, \left| \begin{array}{r} (\widehat{C}, \{K^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C), \\ L^{(i)} = K^{(i)}(\mathbf{x}[i]). \end{array}\right.\right\}_\lambda$$

Definition 2 (Succinct Garbling Schemes). *We say a garbling scheme is* succinct *if there exists a polynomial* $p(\lambda)$ *such that for every supported ring* $\mathcal{R}$ *and every* $\lambda \in \mathbb{N}$*, sufficiently large circuits* C *(over* $\mathcal{R}$*) with* $|C| > p(\lambda)$ *have garbling sizes* $|\widehat{C}| \leq |C| \cdot \log |C|$*.*

3.2 Hardness Assumptions in Paillier Groups

We consider two types of groups, Paillier groups (of composite orders) and prime-order groups in this work. We first provide a quick review of these groups and the standard DDH assumption in them. We next introduce two variants of the standard DDH assumption in those groups.

Definition 3 (Paillier Groups). *Paillier groups are defined by the following instance generation algorithm* Gen.

- $\mathsf{Gen}(1^\lambda, 1^\zeta)$ *uniformly samples two* λ*-bit primes* p, q *such that* $p = 2p' + 1$, $q = 2q' + 1$ *where* p', q' *are also primes. It outputs* $(N = pq, \zeta)$ *as the group description of* $G = \mathbb{Z}^*_{N^{\zeta+1}}$.

Lemma 1 (Facts about Paillier Groups [Pai99, DJ01]). *Let* $G = \mathbb{Z}^*_{N^{\zeta+1}}$ *be a Paillier group sampled by* $\mathsf{Gen}(1^\lambda, 1^\zeta)$ *for a polynomial* $\zeta(\lambda)$.

- G *has a subgroup* $F = \{(1+N)^x : x \in N^\zeta\}$ *where discrete log (i.e., finding* x*) can be efficiently solved.*
- G *has a subgroup* H *that's isomorphic to* $\mathbb{Z}^*_N$*, and* $G = F \times H$.
- *Consider a random element* $g \in G$ *such that the Jacobi symbol of* g *mod* N *is 1. Then* $\langle g \rangle$ *contains* F *except with negligible probability. We write* $g \leftarrow \mathsf{Samp}(N, \zeta)$ *to mean sampling such elements* g *with Jacobi symbol 1.*

Definition 4 (Prime-order Groups). *We consider prime-order groups defined by an instance generation algorithm* Gen *with the following syntax.*

- $\mathsf{Gen}(1^\lambda)$ *outputs* (G, p, g) *where* G *is a group description of prime order* $p > 2^\lambda$*, and* g *is a generator of* G.

The following DDH assumption in Paillier groups is adapted from the formulation by [ADOS22], where the authors formulate a separate "small exponent" assumption stating it's secure to sample the secret exponents in DDH from a smaller, but still sufficiently large, range than the order of g. We directly state the small-exponent variant of DDH in Paillier groups here, as it's required to obtain the HSS construction from [ADOS22] (Lemma 5).

Definition 5 (DDH Assumption). *We say DDH holds in Paillier groups if the following holds for every polynomial* $\zeta(\lambda)$*:*

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N]. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b, c \leftarrow [N^{\zeta+1}]. \end{array} \right\}_\lambda.$$

We say DDH holds in prime-order groups if the following holds:

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ a, b, c \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda.$$

Our first variant, power-DDH, was first introduced by [CNs07, AHI11] in prime-order groups, and formulated in Paillier groups (as an instance of the NIDLS framework) by [ARS24]. Roughly, the assumption states that a group element g raised to the powers of a random secret exponent $s, s^2, s^3, \ldots$ still "look random". In this work we only need the weaker version that consider the first and second powers s, s^2.

Definition 6 (Power-DDH Assumption [CNs07, AHI11, ARS24]). *We say the power-DDH assumption (P-DDH) holds in Paillier groups if the following holds for every polynomial $\zeta(\lambda)$:*

$$\left\{ \mathsf{pp}, g, g^s, g^{s^2} \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ s \leftarrow [N]. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N^{\zeta+1}]. \end{array} \right\}_\lambda .$$

We say P-DDH holds in prime-order groups if the following holds:

$$\left\{ \mathsf{pp}, g, g^s, g^{s^2} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda .$$

Remark 1. As remarked in [ILL24], in prime-order groups, power-DDH implies DDH: the reduction given a power-DDH tuple (g, g^s, g^{s^2}) samples $a, b \leftarrow \mathbb{Z}_p$ to re-randomize the tuple as $(g, g^{s\cdot a}, g^{s\cdot b}, g^{s^2 \cdot ab})$, which becomes a valid DDH tuple. If the reduction is given a random tuple (g, g^s, g^r), the re-randomized is also random. However, in Paillier groups there is no clear way to perform this re-randomization.

The next circular variant was first introduced by [ILL24] both over Paillier groups and prime-order groups. It further assumes that the DDH sample using s^2 as the secret exponent can securely hide (bits of) the secret s itself, after proper re-randomization.

Definition 7 (Circular-Power-DDH [ILL24]). *We say the circular-power-DDH assumption (CP-DDH) holds in Paillier groups if the following holds for every polynomial $\zeta(\lambda)$:*

$$\left\{ \begin{array}{c} \mathsf{pp}, g, g^s, g^{s^2}, g^{a_i}, g^{sa_i}, g^{s^2 a_i}(1+N)^{s[i]} \\ (\text{for } i \in \lceil \log N \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ s, \{a_i\} \leftarrow [N]. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \begin{array}{c} \mathsf{pp}, g, g^s, g^d, g^{a_i}, g^{b_i}, g^{c_i} \\ (\text{for } i \in \lceil \log N \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ s, d, \{a_i, b_i, c_i\} \leftarrow [N^{\zeta+1}]. \end{array} \right\}_\lambda .$$

We say CP-DDH holds in prime-order groups if the following holds:

$$\left\{ \begin{array}{c} \mathsf{pp}, g, g^s, g^{s^2}, g^{a_i}, g^{sa_i}, g^{s^2 a_i + s[i]} \\ (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, \{a_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \begin{array}{c} \mathsf{pp}, g, g^s, g^d, g^{a_i}, g^{b_i}, g^{c_i} \\ (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, d, \{a_i, b_i, c_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda .$$

Remark 2. We modify the formulation from [ILL24] to include the g^{s^2} term in the indistinguishability, so that it implies both DDH and P-DDH and looks more natural. The proof (Theorem 4 in [ILL24]) that CP-DDH in prime-order groups holds in the generic group model (GGM) still goes through for our variant.

3.3 Lattice Hardness Assumptions

In this work, we consider two variants to the standard RingLWE assumption over polynomial rings of the form $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, where $n(\lambda)$ is a power-of-2. Let $q(\lambda) > 2$ be a modulus, $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda) \subseteq \mathcal{R}$ be error and secret distributions. The standard RingLWE assumption (w.r.t. $\mathcal{R}, q, \mathcal{D}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{err}}$) states that for every polynomial $m(\lambda)$, the following computational indistinguishability holds:

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e} & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e} \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ (\text{over } \mathcal{R}_q) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b} \leftarrow \mathcal{R}_q^m \right\}_\lambda,$$

where $\mathcal{R}_q = \mathcal{R}/(q\mathcal{R})$. The first variant considers the case where two vectors of RingLWE samples are computed using the same public vector $\mathbf{a}$, correlated secrets s, s^2, and fresh errors $\mathbf{e}_1, \mathbf{e}_2$. This is a weaker version of the power RingLWE assumption first introduced in [ARS24], which considers multiple powers of s instead of just 2.

Definition 8 (Power RingLWE[ARS24]). *We say the power RingLWE (P-RLWE) assumption holds with respect to the ring $\mathcal{R}(\lambda)$, a modulus $q(\lambda)$, error and secret distributions $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ if the following holds for every polynomial $m(\lambda)$:*

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}_1,\ s^2 \cdot \mathbf{a} + \mathbf{e}_2 & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}_1, \mathbf{e}_2 \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ (\text{over } \mathcal{R}_\mathrm{q}) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

Remark 3. P-RLWE implies the standard RLWE, which just requires indistinguishability of the first 2 terms in the above.

Our next circular variant further assumes that the RingLWE sample using s^2 as the secret can securely hide the secret s itself.

Definition 9 (Circular Power RingLWE). *We say the circular power RingLWE (CP-RLWE) assumption holds with respect to the ring $\mathcal{R}(\lambda)$, two modulus $p(\lambda), q(\lambda)$ such that $q = p \cdot \Delta$, error and secret distributions $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ if the following holds for every polynomial $m(\lambda)$:*

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}_1,\ s^2 \cdot \mathbf{a} + \mathbf{e}_2 + s \cdot \Delta & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}_1, \mathbf{e}_2 \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ (\text{over } \mathcal{R}_\mathrm{q}) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

4 aHMAC and HSS as Evaluation Procedures

In this work, we make use of two tools from prior works, an algebraic homomorphic MAC scheme (aHMAC) [ILL24], and a homomorphic secret sharing scheme (HSS) with extended evaluations [CMPR23, ARS24]. At a highlevel, both schemes are run between a pair of parties, which we call the garbler and the evaluator, who jointly hold (not neccessarily additive) secret shares with respect to some input values $\mathbf{x}$.

- An aHMAC scheme allows the parties to locally evaluate arithmetic circuits (over bounded integers) on their input shares, if the evaluator additionally knows the inputs $\mathbf{x}$ in the clear.
- An HSS scheme allows the parties to locally evaluate a weaker program class (including NC1 Boolean circuits), but without requiring the evaluator to learn $\mathbf{x}$.

When garbling and evaluating a circuit, our techniques require interleaving aHMAC and HSS evaluations on secret shares of intermediate wire values. In particular, they require conversions between the two schemes' share formats.

For this reason, (except in the leveled variants of our grabling constructions,) we need to setup the two schemes using correlated secret randomness, and hence cannot directly invoke their standard security definitions. In the following lemmas we focus only on their correctness properties, and expose the underlying construction detail of their "setup" algorithm (for generating public data pd).

We stress that our garbling schemes will use the evaluation procedures of HSS and aHMAC as subroutines, and we will directly prove the security of our garbling schemes without relying on the security aHMAC and HSS in a black-box way.

4.1 aHMAC and HSS under Paillier Groups

The following lemmas summarize the aHMAC constructions under Paillier groups from [ILL24], including both the non-leveled and leveled variants. We refer readers to [ILL24] for more details. We note that [ILL24] presents the constructions in the language of NIDLS framework [ADOS22], which covers Paillier groups, class groups, and a variant of Joye-Libert encryption as known instantiations. In this work, we chose to focus on Paillier groups for clarity. Our results can be generalized to fit NIDLS framework, and enjoy other instantiations covered by it.

Lemma 2 (aHMAC Gate Evaluation under Paillier Groups). *Let $B < 2^{\mathrm{poly}(\lambda)}$ be a bound on input values, and $\zeta = \lceil \log B/(2\lambda) \rceil + 1$. There exist two pairs of efficient algorithms:*

- $\mathsf{MultKey}(\mathsf{pd}, w_0^x, w_0^y)$ *takes as inputs public data* pd *and two integer values $w_0^x, w_0^y \in \mathbb{Z}$. It outputs an integer $w_0^z \in \mathbb{Z}$.*
- $\mathsf{MultTag}(\mathsf{pd}, w_1^x, w_1^y, x, y)$ *takes as input public data* pd *and four integer values $w_1^x, w_1^y, x, y \in \mathbb{Z}$. It outputs an integer $w_1^z \in \mathbb{Z}$.*
- $\mathsf{AddKey}, \mathsf{AddTag}$ *have the same syntax as* $\mathsf{MultKey}, \mathsf{MultTag}$*, respectively.*

For every $\lambda \in \mathbb{N}$, $\mathsf{pp} = (N, \zeta)$ in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$, secret exponents, $s, s' \in [N]$, inputs $x, y \in [B]$ such that $xy < B$, and additive shares (over $\mathbb{Z}$) $\langle sx \rangle_0, \langle sx \rangle_1, \langle sy \rangle_0, \langle sy \rangle_1$, the following holds:

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s' \cdot xy & w_0^z = \mathsf{MultKey}(\mathsf{pd}, \langle sx \rangle_0, \langle sy \rangle_0) \\ (\text{over } \mathbb{Z}) & w_1^z = \mathsf{MultTag}(\mathsf{pd}, \langle sx \rangle_1, \langle sy \rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s \cdot (x+y) & w_0^z = \mathsf{AddKey}(\mathsf{pd}, \langle sx \rangle_0, \langle sy \rangle_0) \\ (\text{over } \mathbb{Z}) & w_1^z = \mathsf{AddTag}(\mathsf{pd}, \langle sx \rangle_1, \langle sy \rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed as follows:*

$$g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{r} \leftarrow [N]^{\lceil \log N \rceil},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda$$
$$\mathsf{pd} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, \{g^{\mathbf{r}[i]s^2} \cdot (1+N)^{\mathsf{Bits}(s')[i]}\}).$$

We write $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s')$ to denote public data pd *computed as above with freshly sampled g, $\mathbf{r}$, and* seed*.*

In the above, if we choose the secret exponents $s' = s$, then we can compose $\mathsf{MultKey}$, $\mathsf{MultTag}$, $\mathsf{MultKey}$, $\mathsf{MultTag}$ to obtain algorithms $\mathsf{EvalKey}, \mathsf{EvalTag}$ that respectively evaluates an arithmetic C over additive shares. Note that each invocation of those algorithms imposes a bound B on the underlying wire values. We therefore only consider bounded integer evaluations.

Definition 10 (Admissible Input w.r.t. B). *Let C be an arithmetic circuit (with ℓ_x inputs) over $\mathbb{Z}$. We say an input $\mathbf{x} \in \mathbb{Z}^{\ell_x}$ is admissible w.r.t. some positive integer B if all intermediate wire values of $C(\mathbf{x})$ are bounded by B.*

Lemma 3 (aHMAC Circuit Evaluation under Paillier Groups). *Under the same setting as Lemma 2, and assume the existence of a PRG, there exists a pair of efficient algorithms:*

- $\mathsf{EvalKey}(\mathsf{pd}, C, \mathbf{w}_0^x)$ *takes public data* pd*, an arithmetic circuit $C : \mathbb{Z}^{\ell_x} \to \mathbb{Z}^{\ell_z}$, and a vector $\mathbf{w}_0^x \in \mathbb{Z}^{\ell_x}$. It outputs a vector $\mathbf{w}_0^z \in \mathbb{Z}^{\ell_z}$.*
- $\mathsf{EvalTag}(\mathsf{pd}, C, \mathbf{w}_1^x, \mathbf{x})$ *takes public data* pd*, an arithmetic circuit C, two vectors $\mathbf{w}_0^x, \mathbf{x} \in \mathbb{Z}^{\ell_x}$. It outputs a vector $\mathbf{w}_1^z \in \mathbb{Z}^{\ell_z}$.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, $\mathsf{pp} = (N, \zeta)$ in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$, secret exponents, $s \in [N]$, arithmetic circuit C with $|C| \le p(\lambda)$, admissible inputs $\mathbf{x}$ w.r.t B, and additive shares (over $\mathbb{Z}$) $\langle s\mathbf{x}\rangle_0, \langle s\mathbf{x}\rangle_1$, the following holds:

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + s \cdot C(\mathbf{x}) & \mathbf{w}_0^z = \mathsf{EvalKey}(\mathsf{pd}, C, \langle s\mathbf{x}\rangle_0) \\ (\text{over } \mathbb{Z}) & \mathbf{w}_1^z = \mathsf{EvalTag}(\mathsf{pd}, C, \langle s\mathbf{x}\rangle_1, \mathbf{x}) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed as* $\mathsf{pd} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s)$.

Alternatively, by using a vector of different secret exponents $\mathbf{s} \in \mathbb{Z}^{d+1}$ we can compose $\mathsf{MultKey}$, $\mathsf{MultTag}$, $\mathsf{MultKey}$, $\mathsf{MultTag}$ to obtain *leveled* variants of algorithms $\mathsf{EvalKey}^d, \mathsf{EvalTag}^d$ that respectively evaluates an arithmetic C of depth bounded by d. [11]

Lemma 4 (aHMAC Leveled Circuit Evaluation under Paillier Groups). *Under the same setting as Lemma 2, assuming the existence of a PRG, for every polynomial depth bound $d(\lambda)$, there exists a pair of efficient deterministic algorithms:*

- $\mathsf{EvalKey}^d(\mathsf{pd}, C, \mathbf{w}_0^x)$ *takes public data* pd*, an arithmetic circuit $C : \mathbb{Z}^{\ell_x} \to \mathbb{Z}^{\ell_z}$ of depth at most d, and a vector $\mathbf{w}_0^x \in \mathbb{Z}^{\ell_x}$. It outputs a vector $\mathbf{w}_0^z \in \mathbb{Z}^{\ell_z}$.*
- $\mathsf{EvalTag}^d(\mathsf{pd}, C, \mathbf{w}_1^x, \mathbf{x})$ *takes public data* pd*, an arithmetic circuit C of depth at most d, two vectors $\mathbf{w}_0^x, \mathbf{x} \in \mathbb{Z}^{\ell_x}$. It outputs a vector $\mathbf{w}_1^z \in \mathbb{Z}^{\ell_z}$.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, $\mathsf{pp} = (N, \zeta)$ in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$, secret exponents, $\mathbf{s} \in [N]^{d+1}$, arithmetic circuit C with $|C| \le p(\lambda)$ and $\mathrm{Depth}(C) < d(\lambda)$, admissible inputs $\mathbf{x}$ w.r.t B, and additive shares (over $\mathbb{Z}$) $\langle \mathbf{s}[0] \cdot \mathbf{x}\rangle_0$, $\langle \mathbf{s}[0] \cdot \mathbf{x}\rangle_1$, the following holds:

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}[d] \cdot C(\mathbf{x}) & \mathbf{w}_0^z = \mathsf{EvalKey}^d(\mathsf{pd}, C, \langle \mathbf{s}[0] \cdot \mathbf{x}\rangle_0) \\ (\text{over } \mathbb{Z}) & \mathbf{w}_1^z = \mathsf{EvalTag}^d(\mathsf{pd}, C, \langle \mathbf{s}[0] \cdot \mathbf{x}\rangle_1, \mathbf{x}) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

[11] In the leveled variant, shares of two intermediate wires to a gate can have different secret "levels". We can artificially increase the lower wire by multiplying with a constant wire of value 1. In this work, we assume the inputs to a computation contains a constant wire 1 (with secret level 0). Multiplying 1 with itself then provides constant wires with any secret level as needed. This assumption does not affect the asymptotic size of our garbling schemes.

over the randomness of $\mathsf{pd} = \{\mathsf{pd}^{(j)}\}_{[d]}$*, computed as*

$$\mathsf{pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]).$$

The following lemma summarizes the HSS construction under Paillier groups from [ADOS22]. Again, the results in [ADOS22] are presented in the language of NIDLS framework, which covers Paillier groups as a particular instantiation. We refer readers to [ILL24] for more details.

Lemma 5 (HSS Extended Evaluation under DCR Groups [ADOS22]). *Under the same setting as Lemma 2, and assume the existence of a PRG, there exists a pair of efficient algorithms,* $\mathsf{ExtEval}_0$, $\mathsf{ExtEval}_1$*, where*

- $\mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \mathbf{w}_b^x, \mathbf{v}_b^x)$*: takes public data* $\mathsf{pd}_{\mathbf{y}}$ *(with respect to some vector* $\mathbf{y} \in \{0,1\}^{\ell_y}$*), an NC1 Boolean circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$*, and two vectors* $\mathbf{w}_b^x, \mathbf{v}_b^x \in \mathbb{Z}^{\ell_x}$*. It outputs a pair of integers* $w_b^z, v_b^z \in \mathbb{Z}$*.*

For every logarithmic function $d(\lambda) \le O(\log \lambda)$*, and every polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$*, Boolean circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$ *with* $|C| < p(\lambda)$ *and* $\mathrm{Depth}(C) < d(\lambda)$*, secret exponents* $s \in [N]$*, inputs* $\mathbf{x} \in [B]^{\ell_x}$ *and* $\mathbf{y} \in \{0,1\}^{\ell_y}$*, and additive shares (over* $\mathbb{Z}$*)* $\langle s\mathbf{x}\rangle_0, \langle s\mathbf{x}\rangle_1, \langle \mathbf{x}\rangle_0, \langle \mathbf{x}\rangle_1$*, the following holds:*

$$\Pr\left[\begin{array}{l} w_1^z = w_0^z + sz \\ v_1^z = v_0^z + z \\ (\text{over } \mathbb{Z}) \end{array} \;\middle|\; \begin{array}{r} w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \langle s\mathbf{x}\rangle_b, \langle \mathbf{x}\rangle_b) \\ z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y})) \text{ (over } \mathbb{Z}). \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_{\mathbf{y}}$*, which is computed as follows.*

$$\begin{aligned} g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{r}, \mathbf{r}' \leftarrow [N]^{\ell_y},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda \\ \mathsf{pd}_{\mathbf{y}}^{(i)} := \begin{pmatrix} g^{\mathbf{r}[i]}, & g^{\mathbf{r}[i]s} \cdot (1+N)^{\mathbf{y}[i]}, \\ g^{\mathbf{r}'[i]s}, & g^{\mathbf{r}'[i]} \cdot (1+N)^{\mathbf{y}[i]} \end{pmatrix} \quad \forall i \in [\ell_y], \\ \mathsf{pd}_{\mathbf{y}} := (\mathsf{pp}, \mathsf{seed}, \{\mathsf{pd}_{\mathbf{y}}^{(i)}\}). \end{aligned}$$

We write $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, \mathbf{y})$ *to denote public data* $\mathsf{pd}_{\mathbf{y}}$ *computed as above with freshly sampled* g*,* $\mathbf{r}$*,* $\mathbf{r}'$*, and* seed*.*

4.2 aHMAC and HSS under Prime-Order Groups

The following lemmas summarize the aHMAC constructions under prime-order groups from [ILL24], including both the non-leveled and leveled variants. The main difference between these constructions and those under Paillier groups is that these only achieve $\delta = 1/\mathrm{poly}(\lambda)$ correctness, and have computation costs scaling with $\sqrt{1/\delta}$. We refer readers to [ILL24], for more details.

Lemma 6 (aHMAC Gate Evaluation under Prime-Order Groups). *Let* $B < \mathrm{poly}(\lambda)$ *be a bound on input values,* $\delta = 1/\mathrm{poly}(\lambda)$ *be an error bound, and* $\mathsf{Pri.Gen}$ *be an instance generation algorithm for prime-order groups. There exists two pairs of efficient deterministic algorithms:* $\mathsf{MultKey}, \mathsf{MultTag}, \mathsf{AddKey}, \mathsf{AddTag}$ *with analogous syntax to Lemma 2.*

For every $\lambda \in \mathbb{N}$, $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$, *secret exponents,* $\mathbf{s}, \mathbf{s}' \in \{0,1\}^{\lceil \log p \rceil}$, *inputs* $x, y \in [B]$ *such that* $xy < B$, *and additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{s}x \rangle_0, \langle \mathbf{s}x \rangle_1, \langle \mathbf{s}y \rangle_0, \langle \mathbf{s}y \rangle_1$, *the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}' \cdot xy & w_0^z = \mathsf{MultKey}(\mathsf{pd}, \langle \mathbf{s}x \rangle_0, \langle \mathbf{s}y \rangle_0) \\ (\text{over } \mathbb{Z}) & w_1^z = \mathsf{MultTag}(\mathsf{pd}, \langle \mathbf{s}x \rangle_1, \langle \mathbf{s}y \rangle_1, x, y) \end{array}\right] > 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s} \cdot (x+y) & w_0^z = \mathsf{AddKey}(\mathsf{pd}, \langle \mathbf{s}x \rangle_0, \langle \mathbf{s}y \rangle_0) \\ (\text{over } \mathbb{Z}) & w_1^z = \mathsf{AddTag}(\mathsf{pd}, \langle \mathbf{s}x \rangle_1, \langle \mathbf{s}y \rangle_1, x, y) \end{array}\right] > 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of pd, *which is computed as follows:*

$$\mathbf{r} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil}, \mathsf{seed} \leftarrow \{0,1\}^\lambda, s := \mathsf{BitComp}(\mathbf{s}),$$
$$\mathsf{pd} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2 + \mathbf{s}'}).$$

We write $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{s}')$ *to denote public data* pd *computed as above with freshly sampled* $\mathbf{r}$, *and* seed.

Remark 4. The lemma also holds when the secret exponent $\mathbf{s}'$ has a different dimention $\ell \geq \lceil \log p \rceil$ than $\mathbf{s}$. In this case, the public data are computed with $\mathbf{r} \leftarrow \mathbb{Z}_p^\ell$ to match the dimension of $\mathbf{s}'$. We need to support this edge case when using aHMAC together with the HSS scheme based on BHHO encryption (Lemma 10), whose secret exponents has a dimension of $\lceil 3 \log p \rceil$.

Lemma 7 (aHMAC Circuit Evaluation under Prime-Order Groups). *Under the same setting as Lemma 6, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms:* $\mathsf{EvalKey}, \mathsf{EvalTag}$ *with analogous syntax to Lemma 3.*

For every polynomial $p(\lambda)$, *there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$, *secret exponents,* $\mathbf{s} \in \{0,1\}^{\lceil 3 \log p \rceil}$, *arithmetic circuit* C *with* $|C| \leq p(\lambda)$, *admissible inputs* $\mathbf{x}$ *w.r.t* B, *and additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{s} \otimes \mathbf{x} \rangle_0, \langle \mathbf{s} \otimes \mathbf{x} \rangle_1$, *the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{W}_1^z = \mathbf{W}_0^z + \mathbf{s} \otimes C(\mathbf{x}) & \mathbf{W}_0^z = \mathsf{EvalKey}(\mathsf{pd}, C, \langle \mathbf{s} \otimes \mathbf{x} \rangle_0) \\ (\text{over } \mathbb{Z}) & \mathbf{W}_1^z = \mathsf{EvalTag}(\mathsf{pd}, C, \langle \mathbf{s} \otimes \mathbf{x} \rangle_1, \mathbf{x}) \end{array}\right] > 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of pd, *which is computed as* $\mathsf{pd} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{s})$.

Lemma 8 (aHMAC Leveled Circuit Evaluation under Prime-Order Groups). *Under the same setting as Lemma 6, assuming the existence of a PRG, for every polynomial depth bound* $d(\lambda)$, *there exists a pair of efficient deterministic algorithms:* $\mathsf{EvalKey}^d, \mathsf{EvalTag}^d$ *with analogous syntax to Lemma 4.*

For every polynomial $p(\lambda)$, *there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$, *secret exponents* $\mathbf{S} \in \{0,1\}^{(d+1) \times \lceil 3 \log p \rceil}$, *arithmetic circuit* C *with* $|C| \leq p(\lambda)$ *and* $\mathrm{Depth}(C) < d(\lambda)$, *admissible inputs* $\mathbf{x}$ *w.r.t* B, *and additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_0, \langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_1$, *the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{W}_1^z = \mathbf{W}_0^z + \mathbf{S}[d] \otimes C(\mathbf{x}) & \mathbf{W}_0^z = \mathsf{EvalKey}^d(\mathsf{pd}, C, \langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_0) \\ (\text{over } \mathbb{Z}) & \mathbf{W}_1^z = \mathsf{EvalTag}^d(\mathsf{pd}, C, \langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_1, \mathbf{x}) \end{array}\right]$$
$$> 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd} = \{\mathsf{pd}^{(j)}\}_{[d]}$*, computed as*

$$\mathsf{pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{S}[j], \mathbf{S}[j+1]).$$

The following two lemmas summarize the HSS constructions under prime-order groups from [BGI16]. The first variant was proven secure assuming circular security of ElGamal encryption. We modify it slightly:

$$\begin{aligned} \text{ElGamal CT of } \mathbf{y} \text{ under s}: &\quad g^{\mathbf{r}}, g^{\mathbf{r}\cdot s+\mathbf{y}} \\ \text{modified}: &\quad g^{\mathbf{r}\cdot s}, g^{\mathbf{r}\cdot s^2+\mathbf{y}}. \end{aligned}$$

The modified ciphertext can be decrypted in the same way using s, but we can now use CP-DDH to replace circular security assumption of ElGamal when proving security of the joint usage of aHMAC and this variant is secure.

Lemma 9 (HSS Extended Evaluation Based on ElGamal [BGI16]). *Under the same setting as Lemma 6, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms,* $\mathsf{ExtEval}_0$, $\mathsf{ExtEval}_1$*, with analogous syntax to Lemma 5.*

For every logarithmic function $d(\lambda) \le O(\log \lambda)$*, polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$ *with* $|C| < p(\lambda)$*,* $\mathrm{Depth}(C) < d(\lambda)$*, secret exponents* $\mathbf{s} \in [\lceil \log p \rceil]$*, inputs* $\mathbf{x} \in [B]^{\ell_x}$*,* $\mathbf{y} \in \{0,1\}^{\ell_y}$*, and additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{x} \otimes \mathbf{s} \rangle_0, \langle \mathbf{x} \otimes \mathbf{s} \rangle_1, \langle \mathbf{x} \rangle_0, \langle \mathbf{x} \rangle_1$*, the following holds:*

$$\Pr\left[\begin{array}{l|r} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s} \cdot z & w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \langle \mathbf{x} \otimes \mathbf{s} \rangle_b, \langle \mathbf{x} \rangle_b) \\ v_1^z = v_0^z + z & z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y})) \ \text{(over } \mathbb{Z}\text{)}. \\ \text{(over } \mathbb{Z}\text{)} & \end{array}\right] > 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_{\mathbf{y}}$*, which is computed as follows.*

$$\begin{aligned} \mathbf{r} \leftarrow \mathbb{Z}_p^{\ell_y},\ \mathbf{R} \leftarrow \mathbb{Z}_p^{\ell_y \times \lceil \log p \rceil},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda,\ s := \mathsf{BitComp}(\mathbf{s}) \\ \mathsf{pd}_{\mathbf{y}} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}\cdot s},\ g^{\mathbf{r}\cdot s^2+\mathbf{y}},\ g^{\mathbf{R}\cdot s},\ g^{\mathbf{R}\cdot s^2+\mathbf{y}\otimes\mathbf{s}}). \end{aligned}$$

We write $\mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{y})$ *to denote public data* $\mathsf{pd}_{\mathbf{y}}$ *computed as above with freshly sampled* $\mathbf{r}$*,* $\mathbf{R}$*, and* seed*.*

The second variant was proven secure assuming only DDH, without assuming circular security, by using BHHO [BHHO08] encryption instead of ElGamal. We use this variant in our leveled garbling scheme, together with leveled aHMAC, so that we can use P-DDH instead of CP-DDH when proving security of garbling scheme.

Lemma 10 (HSS Extended Evaluation under BHHO [BGI16]). *Under the same setting as Lemma 6, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms,* $\mathsf{ExtEval}_0$, $\mathsf{ExtEval}_1$*, with analogous syntax to Lemma 5.*

For every logarithmic function $d(\lambda) \le O(\log \lambda)$*, polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, Boolean circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$ *with* $|C| < p(\lambda)$*,* $\mathrm{Depth}(C) < d(\lambda)$*, secret exponents*

$\mathbf{s} \in \{0,1\}^{\lceil 3\log p\rceil}$, *inputs* $\mathbf{x} \in [B]^{\ell_x}$, $\mathbf{y} \in \{0,1\}^{\ell_y}$, *and additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{x} \otimes \mathbf{s}\rangle_0, \langle \mathbf{x} \otimes \mathbf{s}\rangle_1$, $\langle \mathbf{x}\rangle_0, \langle \mathbf{x}\rangle_1$, *the following holds:*

$$\Pr\left[\begin{array}{l|r} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}\cdot z & w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_\mathbf{y}, C, \langle \mathbf{x}\otimes\mathbf{s}\rangle_b, \langle \mathbf{x}\rangle_b) \\ v_1^z = v_0^z + z & z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y})) \text{ (over } \mathbb{Z}). \\ \text{(over } \mathbb{Z}) & \end{array}\right] > 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_\mathbf{y}$*, which is computed as follows.*

$$\begin{aligned} \mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil 3\log p\rceil},\ \mathbf{r} \leftarrow \mathbb{Z}_p^{\ell_y},\ \mathbf{R} \leftarrow \mathbb{Z}_p^{\ell_y \times \lceil 3\log p\rceil},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda, \\ \mathsf{pd}_\mathbf{y} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{c}\otimes\mathbf{r}},\ g^{\mathbf{c}\otimes\mathbf{R}},\ g^{\mathsf{InnerPord}(\mathbf{c},\mathbf{s})\cdot\mathbf{r}+\mathbf{y}},\ g^{\mathsf{InnerPord}(\mathbf{c},\mathbf{s})\cdot\mathbf{R}+\mathbf{y}\otimes\mathbf{s}}). \end{aligned}$$

We write $\mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{y})$ *to denote public data* $\mathsf{pd}_\mathbf{y}$ *computed as above with freshly sampled* $\mathbf{c}$*,* $\mathbf{r}$*,* $\mathbf{R}$*, and* seed*.*

4.3 aHMAC and HSS under Lattices

The following lemmas summarize analgous aHMAC constructions to Lemma 2, 3, and 4 under lattices. We present details of these constructions (hence prove the lemmas) in Section 4.4.

Lemma 11 (aHMAC Gate Evaluation under Lattices). *Let* $B < 2^{\mathrm{poly}(\lambda)}$ *be a bound on input values,* $\mathcal{R}$ *be the polynomial ring* $\mathcal{R} = \mathbb{Z}[X]/(X^n+1)$ *where* $n(\lambda)$ *is a power-of-two,* $p \geq B\cdot\lambda^{\omega(1)}$*,* $q = p\cdot\Delta$ *be two moduli, where* $\Delta = B\cdot p\cdot\lambda^{\omega(1)}$ *is a scaling factor, and* $\mathcal{D}_{\mathsf{sk}}(\lambda), \mathcal{D}_{\mathsf{err}}(\lambda)$ *be error and secret distributions with coefficients bounded by* $\mathrm{poly}(\lambda)$*. There exists two pairs of efficient deterministic algorithms,* $\mathsf{MultKey}, \mathsf{MultTag}, \mathsf{AddKey}, \mathsf{AddTag}$*, with analogous syntax to Lemma 2.*

For every $\lambda \in \mathbb{N}$*, secret elements* $s, s' \in \mathcal{D}_{\mathsf{sk}}$*, inputs* $x, y \in [B]$ *such that* $xy < B$*, and additive shares (over* $\mathcal{R}$*)* $\langle sx\rangle_0, \langle sx\rangle_1, \langle sy\rangle_0, \langle sy\rangle_1$*, the following holds:*

$$\Pr\left[\begin{array}{l|r} w_1^z = w_0^z + s'\cdot xy & w_0^z = \mathsf{MultKey}(\mathsf{pd}, \langle sx\rangle_0, \langle sy\rangle_0) \\ \text{(over } \mathcal{R}) & w_1^z = \mathsf{MultTag}(\mathsf{pd}, \langle sx\rangle_1, \langle sy\rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

$$\Pr\left[\begin{array}{l|r} w_1^z = w_0^z + s\cdot(x+y) & w_0^z = \mathsf{AddKey}(\mathsf{pd}, \langle sx\rangle_0, \langle sy\rangle_0) \\ \text{(over } \mathcal{R}) & w_1^z = \mathsf{AddTag}(\mathsf{pd}, \langle sx\rangle_1, \langle sy\rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed (over* $\mathcal{R}_q$*) as follows:*

$$\begin{aligned} \mathsf{pp} &:= (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}}) \\ a \leftarrow \mathcal{R}_q,\ e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathsf{seed} &\leftarrow \{0,1\}^\lambda, \\ \mathsf{pd} &:= (\mathsf{pp}, \mathsf{seed}, a,\ s\cdot a + e_1,\ s^2\cdot a + e_2 - s'\cdot\Delta). \end{aligned}$$

We write $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, s, s')$ *to denote public data* pd *computed as above with freshly sampled* a*,* e_1, e_2*, and* seed*.*

Lemma 12 (aHMAC Circuit Evaluation under Lattices). *Under the same setting as Lemma 11, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms:* $\mathsf{EvalKey}, \mathsf{EvalTag}$ *with analogous syntax as Lemma 3.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, secret elements $s \in \mathcal{D}_{\mathsf{sk}}$, arithmetic circuit C with $|C| \le p(\lambda)$, admissible inputs $\mathbf{x}$ w.r.t B, and additive shares (over $\mathcal{R}$) $\langle s\mathbf{x}\rangle_0, \langle s\mathbf{x}\rangle_1$, the following holds:

$$\Pr\left[\begin{matrix}\mathbf{w}_1^z = \mathbf{w}_0^z + s\cdot C(\mathbf{x}) \\ (\text{over } \mathcal{R})\end{matrix} \;\middle|\; \begin{matrix}\mathbf{w}_0^z = \mathsf{EvalKey}(\mathsf{pd}, C, \langle s\mathbf{x}\rangle_0) \\ \mathbf{w}_1^z = \mathsf{EvalTag}(\mathsf{pd}, C, \langle s\mathbf{x}\rangle_1, \mathbf{x})\end{matrix}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd, which is computed as $\mathsf{pd} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, s, s)$.

Lemma 13 (aHMAC Leveled Circuit Evaluation under Lattices). *Under the same setting as Lemma 11, assuming the existence of a PRG, for every polynomial depth bound $d(\lambda)$, there exists a pair of efficient deterministic algorithms: $\mathsf{EvalKey}^d, \mathsf{EvalTag}^d$ with analogous syntax as Lemma 4.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, secret elements $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^{d+1}$, arithmetic circuit C with $|C| \le p(\lambda)$ and $\mathrm{Depth}(C) < d(\lambda)$, admissible inputs $\mathbf{x}$ w.r.t B, and additive shares (over $\mathcal{R}$) $\langle \mathbf{s}[0]\cdot\mathbf{x}\rangle_0$, $\langle \mathbf{s}[0]\cdot\mathbf{x}\rangle_1$, the following holds:

$$\Pr\left[\begin{matrix}\mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}[d]\cdot C(\mathbf{x}) \\ (\text{over } \mathcal{R})\end{matrix} \;\middle|\; \begin{matrix}\mathbf{w}_0^z = \mathsf{EvalKey}^d(\mathsf{pd}, C, \langle \mathbf{s}[0]\cdot\mathbf{x}\rangle_0) \\ \mathbf{w}_1^z = \mathsf{EvalTag}^d(\mathsf{pd}, C, \langle \mathbf{s}[0]\cdot\mathbf{x}\rangle_1, \mathbf{x})\end{matrix}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd} = \{\mathsf{pd}^{(j)}\}_{[d]}$, computed as

$$\mathsf{pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]).$$

The following lemma summarizes the HSS construction under lattices from [BKS19]. We refer readers to [BKS19] for more details.

Lemma 14 (HSS Extended Evaluation under Lattices [BKS19]). *Under the same setting as Lemma 11, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms, $\mathsf{ExtEval}_0$, $\mathsf{ExtEval}_1$, with analogous syntax to Lemma 5.*

For every logarithmic function $d(\lambda)$, every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, Boolean circuit $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$ with $|C| < p(\lambda)$, $\mathrm{Depth}(C) < d(\lambda)$, secret elements $s \in \mathcal{D}_{\mathsf{sk}}$, inputs $\mathbf{x} \in [B]^{\ell_x}$ and $\mathbf{y} \in \{0,1\}^{\ell_y}$, and additive shares (over $\mathcal{R}$) $\langle s\mathbf{x}\rangle_0, \langle s\mathbf{x}\rangle_1, \langle \mathbf{x}\rangle_0, \langle \mathbf{x}\rangle_1$, the following holds:

$$\Pr\left[\begin{matrix}w_1^z = w_0^z + sz \\ v_1^z = v_0^z + z \\ (\text{over } \mathcal{R})\end{matrix} \;\middle|\; \begin{matrix}w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \langle s\mathbf{x}\rangle_b, \langle \mathbf{x}\rangle_b) \\ z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y}))\ (\text{over } \mathbb{Z}).\end{matrix}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_{\mathbf{y}}$, which is computed (over $\mathcal{R}_q$) as follows.

$$\begin{aligned}
&\mathbf{a} \leftarrow \mathcal{R}_q^{\ell_y},\ r_1, r_2 \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_1', \mathbf{e}_2' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_y} \\
&\mathbf{b} := s\cdot\mathbf{a} + \mathbf{e}, \quad \mathbf{c}_1 := r_1\cdot\mathbf{a} + \mathbf{e}_1 + \mathbf{y}\cdot\Delta, \quad \mathbf{c}_1' := r_1\cdot\mathbf{b} + \mathbf{e}_1', \\
&\qquad\qquad \mathbf{c}_2 := r_2\cdot\mathbf{a} + \mathbf{e}_2, \qquad\qquad \mathbf{c}_2' := r_2\cdot\mathbf{b} + \mathbf{e}_2' + \mathbf{y}\cdot\Delta. \\
&\mathsf{pd}_{\mathbf{y}} := (\mathsf{pp}, \mathsf{seed}, \mathbf{c}_1, \mathbf{c}_1', \mathbf{c}_2, \mathbf{c}_2').
\end{aligned}$$

We write $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, s, \mathbf{y})$ to denote public data $\mathsf{pd}_{\mathbf{y}}$ computed as above with freshly sampled $\mathbf{a}$, r_1, r_2, the errors, and seed.

4.4 aHMAC Constructions under Lattices

We show a construction of $\mathsf{MultKey}$, $\mathsf{MultTag}$, AddKey and AddTag, which proves Lemma 2.

Construction 1 (aHMAC Gate Evaluation under Lattices). The construction is with respect to the following public parameters $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$:

- a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where n is a power-of-two;
- two modulus $p > B \cdot \lambda^{\omega(1)}$, and $q = p \cdot \Delta$, where $\Delta > B^2\lambda^{\omega(1)}$, for some input bound B.
- error and secret distributions $\mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}} \subseteq \mathcal{R}$ with coefficients bounded by $\mathrm{poly}(\lambda)$.

As described in Lemma 2, the public data with respect to two secrets $s, s' \in \mathcal{D}_{\mathsf{sk}}$ are sampled as follows

$$a \leftarrow \mathcal{R}_q,\ e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda,$$
$$\mathsf{pd} := (\mathsf{pp}, \mathsf{seed}, a, \underbrace{s \cdot a + e_1}_{b}, \underbrace{s^2 \cdot a + e_2 - s' \cdot \Delta}_{c}).$$

$w_0^z \leftarrow \mathsf{MultKey}(\mathsf{pd}, w_0^x \in \mathcal{R}, w_0^y \in \mathcal{R})$: Read seed from pd, and expand from it pseudo-random "shifting factors" $r^x, r^y, r^z \in \mathcal{R}_p$.

1. Shift the coefficients of w_0^x, w_0^y by the random factors $r^x, r^y \in \mathcal{R}_p$, and then reduce them mod p.
$$w_0^x \leftarrow (w_0^x + r^x \bmod p), \quad w_0^y \leftarrow (w_0^y + r^y \bmod p).$$
2. Read a from pd, and compute the output w_0^z as follows.
$$w_0^z \leftarrow (\lfloor a w_0^x w_0^y / \Delta \rfloor + r^z) \bmod p.$$

$w_1^z \leftarrow \mathsf{MultTag}(\mathsf{pd}, w_1^x \in \mathcal{R}, w_1^y \in \mathcal{R}, x \in \mathbb{Z}_B, y \in \mathbb{Z}_B)$: Read seed from pd, and expand from it pseudo-random "shifting factors" $r^x, r^y, r^z \in \mathcal{R}_p$.

1. Shift the coefficients of w_1^x, w_1^y by the random factors $r^x, r^y \in \mathcal{R}_p$, and then reduce them mod p. As a result, we have $\|w_1^x\|_\infty, \|w_1^y\|_\infty < p$.
$$w_1^x \leftarrow (w_1^x + r^x \bmod p), \quad w_1^y \leftarrow (w_1^y + r^y \bmod p).$$
Note that if the input satsify $w_1^x = sx + w_0^x$ over $\mathcal{R}$, where $x \in [B]$, then it also holds, except with negligible probability, that $(w_1^x + r^x \bmod p) = sx + (w_0^x + r^x \bmod p)$ over $\mathcal{R}$, as long as $\|sx\|_\infty \ll p$. (See Lemma 2 in [BKS19].) The same holds for w_1^y.
2. Read $a, b, c \in \mathcal{R}_q$ from pd, and compute the following over $\mathcal{R}_q$:
$$d = -a \cdot w_1^x \cdot w_1^y + b \cdot (x \cdot w_1^y + y \cdot w_1^x) - c \cdot x \cdot y.$$
Assuming $w_1^x = sx + w_0^x$, and $w_1^y = sy + w_0^y$, where $x, y \in [B]$ and $xy < B$, then the above computation equals
$$\begin{aligned} d =& -a \cdot (s^2xy + sxw_0^y + syw_0^x + w_0^x w_0^y) \\ &+ (sa + e_1) \cdot (2sxy + xw_0^y + yw_0^x) - (s^2a + e_2 - s'\Delta) \cdot xy \\ =& s'xy\Delta - aw_0^x w_0^y + \underbrace{e_1(xw_1^y + yw_1^x) + e_2xy}_{\|error\|_\infty \le B \cdot p \cdot \mathrm{poly}(\lambda) \ll \Delta} \end{aligned}$$

3. Round the coefficients of d by Δ, and shift resulting coefficients again by the random factor $r^z \in \mathcal{R}_p$.

$$w_1^z \leftarrow (\lfloor d/\Delta \rfloor + r^z \bmod p).$$

We have shown that the error term from d is much smaller than Δ. Hence the rounding step removes it, except with negligible probability. (See Lemma 1 in [BKS19].)

$$w_1^z = \lfloor d/\Delta \rfloor + r^z = s'xy + \underbrace{\lfloor aw_0^x w_0^y/\Delta \rfloor + r^z}_{w_0^z} \bmod p.$$

Shifting by the random factor r^z ensures that $w_1^z = s'xy + w_0^z$ over $\mathcal{R}$ holds except with negligible probability, as long as $\|s'xy\|_\infty \ll p$.

$w_0^z \leftarrow \mathsf{AddKey}(\mathsf{pd}, w_0^x, w_0^y)$: output $w_0^z = w_0^x + w_0^y$ over $\mathcal{R}$.

$w_1^z \leftarrow \mathsf{AddTag}(\mathsf{pd}, w_0^x, w_0^y, x, y)$: output $w_1^z = w_1^x + w_1^y$ over $\mathcal{R}$.

Note that assuming $w_1^x = sx + w_0^x$, and $w_1^y = sy + w_0^y$, then we have

$$w_1^z = s(x+y) + \underbrace{w_0^x + w_0^y}_{w_0^z}.$$

We have directly analyzed the correctness in the construction, and have proven Lemma 2. By composing the algorithms $\mathsf{MultKey}$, $\mathsf{MultTag}$, AddKey and AddTag in an analogous way to the instantiations under Paillier groups (see Section 4.1), we derive Lemma 11 and 12 as corollaries.

Additionally, we note that our new lattice construction implies an aHMAC scheme (and a leveled variant) as originally defined in [ILL24]. We refer readers to [ILL24] for the definition of an aHMAC scheme.

Theorem 1 (aHMAC Under Lattices). *Assuming CP-RLWE (Definition 9) with respect to the public parameters $\mathsf{pp}^{\mathsf{Lat}}$ specified in Section 5.3, there exists an aHMAC scheme achieving* negl*-correctness, with an evk of size $\ell_z \cdot \mathrm{poly}(\lambda)$ bits.*

Alternatively, assuming P-RLWE (Definition 8, with respect to $\mathsf{pp}^{\mathsf{Lat}}$), there exists a leveled aHMAC scheme achieving negl*-correctness, with an evk of size $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ bits.*

5 Succinct Boolean Garbling Schemes

For a more intuitive presentation, we first show a 2PC protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 1, 2, under Paillier groups) for evaluating Boolean circuits between a garbler P_G and an evaluator P_E:

- In an Init phase, the garbler P_G sends public data and input shares w.r.t. a Boolean vector $\mathbf{x} \in \{0,1\}^{\ell_x}$ to the evaluator P_E;
- In an Eval phase, the two parties jointly evaluate gates of a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ in topological order;
- In a Final phase, the garbler P_G sends some decryption data to reveal the final output $\mathbf{z} \in \{0,1\}^{\ell_z}$ to the evaluator P_E.

We note that all messages in this protocol are from the garbler P_G to the evaluator P_E. We further divide them into two parts, each satisfying a special property.[12]

[12] Without the special properties, a trivial protocol is letting the garbler P_G directly send $\mathbf{z} := C(\mathbf{x})$ to the evaluator P_E.

1. *Input shares w.r.t. to the vector* $\mathbf{x}$. We ensure they are decomposable, i.e., each bit in this communication depends only on a single bit of $\mathbf{x}$.
2. *Garbling materials.* These include the public data during Init, the decryption data during Final, and all communication during Eval. We ensure they are independent of the input $\mathbf{x}$.

Therefore, we can directly "compile" the above 2PC protocol into a garbling scheme as follows.

- The Garb algorithm outputs (1) key functions $\{K_x^{(i)}\}$ such that labels $\{L_x^{(i)} := K_x^{(i)}(\mathbf{x}[i])\}$ exactly equal to the input shares w.r.t. $\mathbf{x}$, and (2) a garbling $\widehat{C}$ that contains the garbling materials.
- The Eval algorithm performs all steps of P_E in the 2PC protocol to recover the final output.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC evaluation procedures $\mathsf{EvalKey}, \mathsf{EvalTag}$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}$ under Paillier groups; (See Lemma 3;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}$ under paillier groups; (See Lemma 5;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1. [a]

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.
Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init :
  1. P_G sends public data pd to the evaluator P_E.

$$\begin{aligned} &\zeta := 2,\ \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ &s \leftarrow [N],\ \mathsf{sk} \leftarrow \{0,1\}^\lambda \\ &\mathsf{pd} := \big(\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s), \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, \mathsf{sk})\big). \end{aligned} \tag{3}$$

  2. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}}\rangle_1$ to P_E. [b]

$$\begin{aligned} &\overline{\mathbf{x}} = \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)), \\ &\langle s\overline{\mathbf{x}}\rangle_1 := s\overline{\mathbf{x}} + \langle s\overline{\mathbf{x}}\rangle_0 \text{ (over } \mathbb{Z}), \text{ where } \langle s\overline{\mathbf{x}}\rangle_0 \leftarrow [N \cdot \lambda^{\omega(1)}]^{\ell_x}. \end{aligned}$$

[a] With a bound on $|C|$, the PRF can be replaced with a PRG where each bit can be evaluated in NC1.
[b] $(\overline{\mathbf{x}}, \langle s\overline{\mathbf{x}}\rangle_1)$ jointly is what we call input shares in the overview text.

Fig. 1. The Init phase of our 2PC protocol for Boolean circuits under Paillier groups.

The core of our construction is a sub-protocol $\mathsf{BoolGateEval}$ (Figure 3, and Section 5.1) for evaluating an arbitrary Boolean gate with up to $O(\log \lambda)$ input wires, and a single output, costing 1 bit per gate.

The sub-protocol itself stays unchanged when we instantiate the underlying primitives aHMAC and HSS under Paillier groups, prime-order groups, or lattices. The only changes under different instantiations lie in the Init phases of the main protocol, during which P_G computes public data pd differently. We describe the Init phase under Paillier groups in Figure 1, under lattices in Section 5.3, and under prime-order groups in Section 5.4. In summary, we obtain the following theorem.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ Continued

– Eval : P_G, P_E evaluate gates $\mathsf{g} \in C$ in the topological order while maintaining the following invariant:
1. P_G, P_E jointly hold additive shares $\langle s\overline{\mathbf{x}}_{\mathsf{g}}\rangle$, where $\overline{\mathbf{x}}_{\mathsf{g}}$ are masked input wire values to the gate g

$$\overline{\mathbf{x}}_{\mathsf{g}} = \mathbf{x}_{\mathsf{g}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g})). \tag{4}$$

2. P_E holds the masked wire values $\overline{\mathbf{x}}_{\mathsf{g}}$.

To evaluate the gate g, P_G, P_E jointly call the sub-protocol $\mathsf{BoolGateEval}$.

$$(P_G : \langle s\overline{z}_{\mathsf{g}}\rangle_0), (P_E : \langle s\overline{z}_{\mathsf{g}}\rangle_1, \overline{z}_{\mathsf{g}}) \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}}\left((P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}}_{\mathsf{g}}\rangle_0), (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}}_{\mathsf{g}}\rangle_1, \overline{\mathbf{x}}_{\mathsf{g}})\right)$$

– Final : P_G sends masks $\mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$ mod 2 on all output wires to P_E, who then recovers the output $\mathbf{z}$ by removing the masks mod2. [a]

[a] The final message from P_G to P_E can be avoided via an optimization: let $\mathsf{BoolGateEval}$ compute z, instead of masked $\overline{z}$, for values on the output wires.

Fig. 2. The $\mathsf{Eval}, \mathsf{Final}$ phases of our 2PC protocol for Boolean circuits.

Theorem 2 (Garbling $O(\log\lambda)$-ary Gates). *Let $\mathcal{C}^{\mathsf{Arb}} = \{\mathcal{C}^{\mathsf{Arb}}_\lambda\}$ be the class of circuits (of unbounded size) consisting of arbitrary gates with $O(\log\lambda)$ input wires and 1 output wires.*

Assuming CP-DDH in Paillier groups or CP-RLWE with respect to the public parameters $\mathsf{pp}^{\mathsf{Lat}}$ specified in Section 5.3, there exists a garbling scheme for $\mathcal{C}^{\mathsf{Arb}}$ over $\mathbb{Z}_2$, where the garbling size $\widehat{C}$ for a circuit $C \in \mathcal{C}^{\mathsf{Arb}}_\lambda$ is $|\widehat{C}| \le |C| + \mathrm{poly}(\lambda)$.

Assuming CP-DDH in prime-order groups, there exists a garbling scheme for $\mathcal{C}^{\mathsf{Arb}}$ over $\mathbb{Z}_2$ achieving the same garbling size as above, but with 1/poly correctness and privacy errors. *The errors can be made negligible assuming a variant of CP-DDH (Definition 11).*

The proof of Theorem 2 follows from Proposition 1, 3, and 5, which are proven in Section 5.1, 5.3, and 5.4 respectively. We show amplification techniques in Section 5.5 for removing correctness and privacy errors from prime-order group instantiations. Applying Theorem 2 to garbling standard Boolean circuits with binary gates gives a scheme costing 1 bit per gate.

Corollary 1 (Boolean Garbling). *Assuming any of the assumptions in Theorem 2, there exists a garbling scheme for all Boolean circuits C (with binary gates) with garbling size $|\widehat{C}| \le |C| + \mathrm{poly}(\lambda)$.*

The scheme assuming CP-DDH in prime-order groups has 1/poly correctness and privacy errors*, which can be made negligible assuming a variant of CP-DDH (Definition 11).*

In the special case of layered circuits C^{Layer}, we can re-write C^{Layer} into another circuit C' in terms of general gates for $\log\log\lambda$-depth computations, with the guarantee that $|C'| < O(|C^{\mathsf{Layer}}|/\log\log\lambda)$. (See Lemma 4.12 in [BGI16].) Since each general gate depends on at most $\log\lambda$ input values, we can apply Theorem 2 to garble C' which yields a scheme costing $O(1/\log\log\lambda)$ bits per gate.

Corollary 2 (Boolean Garbling for Layered Circuits). *Assuming any of the assumptions in Theorem 2, there exists a garbling scheme for all layered Boolean circuits C^{Layer} (with binary gates) with garbling size $|\widehat{C}^{\mathsf{Layer}}| \le O(|C^{\mathsf{Layer}}|/\log\log\lambda) + \mathrm{poly}(\lambda)$.*

The scheme assuming CP-DDH in prime-order groups has 1/poly correctness and privacy errors*, which can be made negligible assuming a variant of CP-DDH (Definition 11).*

In Section 5.2, we describe a leveled variant of the 2PC protocol $\mathsf{LBoolCircEval}$ (Figure 4 , 5, under Paillier groups), based on a leveled variant of the core sub-protocol $\mathsf{LBoolGateEval}$ (Figure 6). We describe the Init phases of this variant under lattices and prime-order groups in Section 5.3 and 5.4 respectively. In summary, we obtain the following theorem.

Theorem 3 (Leveled Garbling of $O(\log\lambda)$-ary Gates). *Let $\mathcal{C}^{\mathsf{Arb}} = \{\mathcal{C}^{\mathsf{Arb}}_\lambda\}$ be the class of circuits (of unbounded size) consisting of arbitrary gates with $O(\log\lambda)$ input wires and 1 output wires.*

Assuming P-DDH and DDH in Paillier groups, or P-RLWE with respect to the public parameters $\mathsf{pp}^{\mathsf{Lat}}$ specified in Section 5.3, there exists a garbling scheme for $\mathcal{C}^{\mathsf{Arb}}$ over $\mathbb{Z}_2$, where the garbling size $\widehat{C}$ for a circuit $C \in \mathcal{C}^{\mathsf{Arb}}_\lambda$ is $|\widehat{C}| \le |C| + \mathrm{Depth}(C)\cdot\mathrm{poly}(\lambda)$.

The proof of Theorem 3 follows from Proposition 2, 4, and 6, which are proven in Section 5.1, 5.3, and 5.4 respectively. Security amplification for the leveled variant under prime-order groups is analogous to the non-leveled variant as described in Section 5.5. We obtain two corollaries analogous to the non-leveled case.

Corollary 3 (Leveled Boolean Garbling). *Assuming any of the assumptions in Theorem 3, there exists a garbling scheme for all Boolean circuits C (with binary gates) with garbling size $|\widehat{C}| \le |C| + \mathrm{Depth}(C)\cdot\mathrm{poly}(\lambda)$.*

Corollary 4 (Leveled Boolean Garbling for Layered Circuits). *Assuming any of the assumptions in Theorem 2, there exists a garbling scheme for all layered Boolean circuits C^{Layer} (with binary gates) with garbling size $|\widehat{C}^{\mathsf{Layer}}| \le O(|C^{\mathsf{Layer}}|/\log\log\lambda) + \mathrm{Depth}(C)\cdot\mathrm{poly}(\lambda)$.*

5.1 Sub-protocol for Garbling $O(\log\lambda)$-ary Boolean Gates

In the sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$, both parties P_G, P_E hold public data $\mathsf{pd} = (\mathsf{aHMAC.pd}, \mathsf{HSS.pd}_{\mathsf{sk}})$ prepared in the Init phase of the main protocol (Figure 1), and jointly hold additive shares $\langle s\overline{\mathbf{x}}\rangle$, where s is a global secret exponent sampled during Init, and $\overline{\mathbf{x}}$ represent masked input values to the gate $\mathsf{g} \in C$ (with masks derived from sk). Additionally, P_E holds $\overline{\mathbf{x}}$ in the clear.

$$\text{Inputs: } (P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), \quad (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}).$$

Their goal is to jointly obtain shares of $\langle s\cdot\overline{z}\rangle$, where $\overline{z}$ is the masked output of g. Additionally, P_E should hold $\overline{z}$ in the clear.

$$\text{Outputs: } (P_G : \langle s\overline{z}\rangle_0), \quad (P_E : \langle s\overline{z}\rangle_1, \overline{z}).$$

Their first steps are local aHMAC and HSS evaluations by both parties over the input shares $\langle s\overline{\mathbf{x}}\rangle$. For now, assume there are arithmetic circuits $C_{\mathbf{v}}$ (Fact 1) and Boolean circuits $C_{\mathsf{g},\mathbf{v}}$ (Equation 6) which satsify

$$\overline{z} = \sum_{\mathbf{v}\in\{0,1\}^{\ell_x}} C_{\mathbf{v}}(\overline{\mathbf{x}})\cdot C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) \text{ over } \mathbb{Z}. \tag{5}$$

The parties apply aHMAC to locally evaluate $C_{\mathbf{v}}$ over $\langle s\overline{\mathbf{x}}\rangle$ to obtain shares $\langle s\cdot C(\overline{\mathbf{x}})\rangle$. They also locally hold additive shares of $\langle C(\overline{\mathbf{x}})\rangle$ as P_E can compute $C(\overline{\mathbf{x}})$ on its own.

$$P_G : \langle s\cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_0),$$
$$P_E : \langle s\cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}).$$

The parties next apply HSS to locally evaluate $C_{\mathsf{g},\mathbf{v}}$ over the public data $\mathsf{HSS.pd}_{\mathsf{sk}}$ and additionally "multiply" the results with $C(\overline{\mathbf{x}})$ to obtain shares $\langle s\overline{z}\rangle$ and $\langle \overline{z}\rangle$.

$$P_G : (\langle s\overline{z}\rangle_0, \langle \overline{z}\rangle_0) \leftarrow \mathsf{ExtEval}_0(\mathsf{HSS.pd}_{\mathsf{sk}}, (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots), \{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\}_{\mathbf{v}})$$
$$P_E : (\langle s\overline{z}\rangle_1, \langle \overline{z}\rangle_1) \leftarrow \mathsf{ExtEval}_1(\mathsf{HSS.pd}_{\mathsf{sk}}, (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots), \{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1\}_{\mathbf{v}}).$$

In summary, the parties now hold shares $\langle s\overline{z}\rangle$, $\langle \overline{z}\rangle$ through local computations. In the last step, P_G sends its share $\langle \overline{z}\rangle$ mod 2 to P_E, who then recovers $\overline{z}$.

It remains to specify the arithmetic circuits $C_{\mathbf{v}}$ and Boolean circuits $C_{\mathsf{g},\mathbf{v}}$ satisfying Equation 5. For this, we let $C_{\mathbf{v}}$ implement the indicator polynomial $p_{\mathbf{v}}$ specified as follows.

Fact 1 (Indicator Polynomial). For every positive integer $\ell_x \in \mathbb{N}$, every vector $\mathbf{v} \in \{0,1\}^{\ell_x}$, there exists a polynomial $p_{\mathbf{v}}$ (over $\mathbb{Z}$) such that

$$p_{\mathbf{v}}(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{x} = \mathbf{v} \\ 0 & \text{for } \mathbf{x} \in \{0,1\}^{\ell_x}, \mathbf{x} \neq \mathbf{v}. \end{cases}$$

Furthermore, $p_{\mathbf{v}}$ can be implemented by an arithmetic circuit $C_{\mathbf{v}} : \mathbb{Z}^{\ell_x} \to \mathbb{Z}$ of size $|C_{\mathbf{v}}| \leq O(\ell_x)$, $\mathrm{Depth}(C_{\mathbf{v}}) \leq O(\log \ell_x)$ and such that all Boolean inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$ are admissible w.r.t. the bound $B = 2$.

We define $C_{\mathsf{g},\mathbf{v}}(\mathsf{sk})$ to compute a masked output $\overline{z}$ pretending $\overline{\mathbf{x}} = \mathbf{v}$.

$$C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \oplus \mathsf{g}\big(\mathbf{v} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g}))\big). \tag{6}$$

Effectively, Equation 5 computes all possible evaluation results of $\overline{z}$ via $C_{\mathsf{g},\mathbf{v}}(\mathsf{sk})$, and selects the correct one via $C_{\mathbf{v}}(\overline{\mathbf{x}})$, which only equals 1 when $\mathbf{v} = \overline{\mathbf{x}}$. Assuming PRF is in NC1 and $\ell_x = O(\log \lambda)$, $C_{\mathsf{g},\mathbf{v}}$ can indeed be evaluated using HSS.

We summarize the sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ in Figure 3. Note that in each invocation, the only communication is *one bit* $b := \langle \overline{z}\rangle_0$ sent from the garbler P_G to the evaluator P_E. We summarize its correctness and security in the following lemmas.

Lemma 15 (Correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Paillier Groups). *Let $\ell(\lambda) \leq O(\log \lambda)$ be a bound on input length. There exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (N, \zeta)$ in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^2)$, secret exponent $s \in [N]$, additive shares (over $\mathbb{Z}$) $\langle s\overline{\mathbf{x}}\rangle_0, \langle s\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds:*

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s\overline{z}, & \mathsf{pd} \text{ sampled per Equation 3}, \\ \quad z = \mathsf{g}(\mathbf{x}) & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \\ & \quad \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)),\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \mathrm{negl}(\lambda).$$

Proof. The correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ follows from that of $\mathsf{EvalKey}$, $\mathsf{EvalTag}$ and that of $\mathsf{ExtEval}_b$. □

Sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean gate $\mathsf{g} \in C$.
Inputs: P_G, P_E both hold public data $\mathsf{pd} = (\mathsf{aHMAC.pd}, \mathsf{HSS.pd}_{\mathsf{sk}})$ (as defined in Equation 3), and jointly hold additive shares $\langle s\overline{\mathbf{x}}\rangle$, where $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$ is a masked input vector. P_E additionally holds the vector $\overline{\mathbf{x}}$.
Outputs: P_G, P_E jointly output additive shares $\langle s\overline{z}\rangle$, where $\overline{z} \in \{0,1\}$ is the masked output. P_E additionally holds the bit $\overline{z}$.

– P_G, P_E obtain additive shares $\langle s\overline{z}\rangle$ and $\langle \overline{z}\rangle$ through local computations, where

$$\overline{z} := z \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)),\ \ z := \mathsf{g}(\mathbf{x}),\ \ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)). \tag{7}$$

Let $C_{\mathbf{v}}$ and $C_{\mathsf{g},\mathbf{v}}$ be arithmetic and Boolean circuits specified in Fact 1 and Equation 6, respectively. Further define $C_{\mathsf{g}} := (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots)$.

1. P_G, P_E locally runs $\mathsf{EvalKey}, \mathsf{EvalTag}$, respectively, to obtain additive shares $\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ and $\langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ for all $\mathbf{v} \in \{0,1\}^{\ell_x}$.

$$\begin{aligned} P_G :& \langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_0), && \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow 0 \\ P_E :& \langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}), && \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow C_{\mathbf{v}}(\overline{\mathbf{x}}). \end{aligned}$$

2. P_G, P_E locally runs $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$, respectively, to obtain additive shares $\langle s\overline{z}\rangle$ and $\langle \overline{z}\rangle$.

$$\begin{aligned} P_G :& (\langle s\overline{z}\rangle_0, \langle \overline{z}\rangle_0) \leftarrow \mathsf{ExtEval}_0(\mathsf{HSS.pd}_{\mathsf{sk}}, C_{\mathsf{g}}, \{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\}_{\mathbf{v}}) \\ P_E :& (\langle s\overline{z}\rangle_1, \langle \overline{z}\rangle_1) \leftarrow \mathsf{ExtEval}_1(\mathsf{HSS.pd}_{\mathsf{sk}}, C_{\mathsf{g}}, \{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1\}_{\mathbf{v}}). \end{aligned}$$

– P_G sends a bit $b := \langle \overline{z}\rangle_0 \bmod 2$ to P_E, who can then locally recover $\overline{z}$.

$$\overline{z} := \langle \overline{z}\rangle_1 - b \bmod 2.$$

Fig. 3. Our 2PC subprotocol for $O(\log \lambda)$-ary Boolean gates.

Lemma 16 (Security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Paillier Groups). *Under the same setting as Lemma 15, there exists an efficient simulator* Sim *that, given the masked output* $\overline{z}$, statistically *simulates* P_G*'s message in the sub-protocol* $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, *every Boolean circuit* C *with a gate* g *of* $\ell_x \le \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^2)$, *secret exponent* $s \in [N]$, *additive shares (over* $\mathbb{Z}$*)* $\langle s\overline{\mathbf{x}}\rangle_0, \langle s\overline{\mathbf{x}}\rangle_1$, *and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$, *the following holds.*

$$\mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), \mathsf{Sim}(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \le \mathrm{negl}(\lambda), \quad \left| \begin{array}{l} \mathsf{pd} \text{ sampled per Equation 3}, \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array} \right.$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

Proof. The simulator computes $\langle \overline{z}\rangle_1$ following exactly P_E's steps, and then simulates $\langle \overline{z}\rangle_0 \bmod 2$ (which is the message from P_G) as $\overline{z} - \langle \overline{z}\rangle_1 \bmod 2$. □

Using the correctness and security of the core sub-protocol, $\mathsf{BoolGateEval}$ under Paillier groups, we can now prove those of our garbling scheme under Paillier groups (compiled from the 2PC protocol $\mathsf{BoolCircEval}$).

Proposition 1 (Garbling of $O(\log \lambda)$-ary Gates under Paillier Groups). *Assuming CP-DDH in Paillier groups, the garbling scheme compiled from the protocol* $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ *(Figure 1, 2) is correct and secure.*

Proof of Proposition 1. The correctness of the protocol follows from that of $\mathsf{BoolGateEval}$ (Lemma 15). Hence the correctness of the compiled garbling scheme follows. We focus now on proving security.

First, we recap the compiled garbling scheme. Given a circuit C, the garbler proceeds as follows.

- Sample Paillier public parameters $\mathsf{pp} = (N, \zeta)$, a secret exponent $s \leftarrow [N]$, a PRF key $\mathsf{sk} \leftarrow \{0,1\}^\lambda$, and compute public data pd per Equation 3:

$$\begin{aligned} &\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2)\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathsf{seed} \leftarrow \{0,1\}^\lambda, \\ &\ s \leftarrow [N],\ \mathbf{r} \leftarrow [N]^{\lceil \log N \rceil}, \quad \mathbf{r}', \mathbf{r}'' \leftarrow [N]^\lambda, \\ &\mathsf{pd} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2} \odot (1+N)^{\mathsf{Bits}(s)}, \\ &\qquad g^{\mathbf{r}'}, g^{\mathbf{r}'s} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})}, g^{\mathbf{r}''s}, g^{\mathbf{r}''} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})}). \end{aligned} \tag{8}$$

  where $\odot$ denotes component-wise multiplication between two vectors.
- For every input wire i, sample a pad $k^{(i)} \leftarrow [N \cdot \lambda^{\omega(1)}]$, and define the key function $K^{(i)}$ as follows:

$$K^{(i)}(b) := (\bar{b}, s \cdot \bar{b} + k^{(i)}), \qquad \text{where } \bar{b} := b \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)[i]).$$

- For every gate $\mathsf{g} \in C$, in the topological order, where a pad $k^{(i)}$ is defined for all $i \in \mathsf{InWires}(g)$, follow the subprotocol $\mathsf{BoolGateEval}$ as P_G with $(\mathsf{pd}, \{k^{(i)}\}_{\mathsf{InWires}(g)})$ as inputs, and compute its message b_{g}. The output of the subprotocol defines a pad $k^{(j)}$ for the output wire $j = \mathsf{OutWire}(\mathsf{g})$.
- Compute the masks on output wires $\mathbf{o} = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$.
- Output the garbling $\widehat{C} = (\mathsf{pd}, \{b_{\mathsf{g}}\}_C, \mathbf{o})$ consisting of the public data, the bit b_{g} for every $\mathsf{g} \in C$, and the masks on the output wires. Output the key functions $\{K^{(i)}\}$ as defined above.

Next, we describe the simulator Sim required by Definition 1. It takes the circuit C and the evaluation results $\mathbf{z}$ as input, and simulates the garbling $\widehat{C}$ and input labels $\{L^{(i)}\}$ as follows.

- Sample Paillier public parameter $\mathsf{pp} = (N, \zeta)$ honestly, and sample random elements as public data $\widetilde{\mathsf{pd}}$:

$$\begin{aligned} &\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2)\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathsf{seed} \leftarrow \{0,1\}^\lambda, \\ &\ s \leftarrow [N],\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow [N^2]^{\lceil \log N \rceil},\ \mathbf{a}', \mathbf{b}', \mathbf{a}'', \mathbf{b}'' \leftarrow [N^2]^\lambda, \\ &\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, g^{\mathbf{a}'}, g^{\mathbf{b}'}, g^{\mathbf{a}''}, g^{\mathbf{b}''}). \end{aligned} \tag{9}$$

- For every input wire i, sample a label $\tilde{l}^{(i)} \leftarrow [N \cdot \lambda^\omega]$, and a masked bit $\tilde{x}^{(i)} \leftarrow \{0,1\}$. The simulated input labels are $\widetilde{L}^{(i)} = (\tilde{x}^{(i)}, \tilde{l}^{(i)})$. Further sample a masked bit $\tilde{x}^{(j)} \leftarrow \{0,1\}$ for every wires j in C, including the output wires.
- For every gate $g \in C$, in the topological order, where a label $\tilde{l}^{(i)}$ and a masked bit $\tilde{x}^{(i)}$ are defined for all $i \in \mathsf{InWires}(\mathsf{g})$, follow the the subprotocol $\mathsf{BoolGateEval}$ as P_E except the last step (See Figure 3) with $(\widetilde{\mathsf{pd}}, \{\tilde{l}^{(i)}, \tilde{x}^{(i)}\})$ as inputs. The computation results (corresponding to $\langle s\overline{z} \rangle_1$ in Figure 3) define a label $\tilde{l}^{(j)}$ for the output wire $j = \mathsf{OutWire}(\mathsf{g})$. Then run the simulator guaranteed by the security of the subprotocol (Lemma 16):

$$\tilde{b}_{\mathsf{g}} \leftarrow \mathsf{Sim}'(\widetilde{\mathsf{pd}}, \{\tilde{l}^{(i)}, \tilde{x}^{(i)}\}_{\mathsf{InWires}(\mathsf{g})}, \tilde{z}^{(j)}),$$

  where $\tilde{z}^{(i)}$ is the masked bit assigned to wire $j = \mathsf{OutWire}(\mathsf{g})$.

– Let $\widetilde{\mathbf{z}}$ be the masked bits assigned to output wires $\mathsf{OutWires}(C)$, simulate the masks on output wires as $\tilde{\mathbf{o}} = \widetilde{\mathbf{z}} \oplus \mathbf{z}$.
– Output the simulated garbling $\widetilde{C} = (\widetilde{\mathsf{pd}}, \{\tilde{b}_{\mathsf{g}}\}, \tilde{\mathbf{o}})$, and the simulated input labels $\{\widetilde{L}^{(i)}\}$.

We now show a series of hybrid experiments, where Hyb_0 describe the honest distribution of $\widehat{C}$ and $\{L^{(i)} = K^{(i)}(\mathbf{x}[i])\}$ for some input $\mathbf{x}$, and Hyb_5 describe the simulated distribution $\widetilde{C}$ and $\{\widetilde{L}^{(i)}\}$.

Hyb_0 The real distribution of $\widehat{C}$ and $\{L^{(i)} = K^{(i)}(\mathbf{x}[i])\}$ computed according to the garbling scheme. (See the recap earlier.)

Hyb_1 In this hybrid, instead of computing the bits $\{b_{\mathsf{g}}\}$ as the garbler's message following the subprotocol $\mathsf{BoolGateEval}$, simulate them using the simulator Sim' guaranteed by the security of the subprotocol:
– First compute the correct wire value $x^{(j)}$ on each wire j in C, and then the masked bit $\overline{x}^{(j)} = x^{(j)} \oplus \mathsf{PRF}(\mathsf{sk}, j)$.
– Let $l^{(i)} = k^{(i)} + s\overline{x}^{(i)}$ be the labels on input wires. For every gate $\mathsf{g} \in C$, in topological order, follow the subprotocol as P_E (except the last step) with $(\mathsf{pd}, \{l^{(i)}, \overline{x}^{(i)}\}_{\mathsf{InWires}(\mathsf{g})})$ as inputs to compute a label $l^{(j)}$ for the output wire $j = \mathsf{OutWire}(\mathsf{g})$. Then run the simulator

$$\tilde{b}_{\mathsf{g}} \leftarrow \mathsf{Sim}'(\mathsf{pd}, \{\tilde{l}^{(i)}, \tilde{x}^{(i)}\}, \overline{z}^{(j)}),$$

where $\overline{z}^{(j)}$ is the masked bit on wire j.

By the correctness and security of $\mathsf{BoolGateEval}$ (Lemma 15 and 16), the simulated bits b_{g} are statistically close to the correctly computed ones in Hyb_0. Hence we have $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$. Note that in Hyb_1, the pads $k^{(i)}$ sampled for the input wires are not used for computing the bits b_{g} anymore.

Hyb_2 In this hybrid, instead of computing the input labels as $l^{(i)} = \overline{x}^{(i)}s + k^{(i)}$, directly sample $\tilde{l}^{(i)} \leftarrow [N \cdot \lambda^{\omega(1)}]$.

The two ways of sample $l^{(i)}$ and $\tilde{l}^{(i)}$ are statistically close, hence we have $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_1$. Note that in Hyb_2, the secret exponent s within pd is not used for computing the input labels or anywhere else.

Hyb_3 In this hybrid, instead of computing the masks on the output wires directly using the PRF, $\mathbf{o} = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$, compute it as $\tilde{\mathbf{o}} = \overline{\mathbf{z}} \oplus \mathbf{z}$, where $\overline{\mathbf{z}}$ are the masked wire values on the output wires. As the two ways of computing $\mathbf{o}$ and $\tilde{\mathbf{o}}$ are equivalent, we have $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 In this hybrid, instead of computing the public data pd as in Equation 8, simulate it with random elements as in Equation 9.

We claim the two ways of sampling pd are computationally indistinguishable (Claim 1). Hence we have $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$. Note that in Hyb_4, the PRF key sk are only used for computing masked wire values $\overline{x}^{(i)} = x^{(i)} \oplus \mathsf{PRF}(\mathsf{sk}, i)$, and in particular not the public data anymore.

Hyb_5 In this hybrid, instead of computing the masked wire values $\overline{x}^{(i)}$ using a PRF, directly sample them at random $\tilde{x}^{(i)} \leftarrow \{0,1\}$.

By the security of PRF, we have $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_4$. Note that Hyb_5 computes exactly the simulated distribution of $\widetilde{C}$ and $\{\widetilde{L}^{(i)}\}$.

By a hybrid argument, we conclude $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_5$. It remains to prove the following claim.

Claim 1. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 8 and* $\widetilde{\mathsf{pd}}$ *by Equation 9 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 8 to Equation 9.

Hyb'_0 This is the distribution of Equation 8.

Hyb'_1 In this hybrid, instead of computing the aHMAC public data as

$$g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2} \odot (1+N)^{\mathsf{Bits}(s)}$$

where $\mathbf{r}, s$ are random exponents from $[N]$, simulate them as random elements

$$g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}},$$

where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are random exponents from $[N^2]$. By CP-DDH in Paillier groups (Definition 7), we have $\mathsf{Hyb}'_1 \approx_c \mathsf{Hyb}'_0$.

Hyb'_2 In this hybrid, instead of computing the HSS public data as

$$g^{\mathbf{r}'}, g^{\mathbf{r}'s} \odot (1+N)^{\mathsf{Bits(sk)}}, g^{\mathbf{r}''s}, g^{\mathbf{r}''} \odot (1+N)^{\mathsf{Bits(sk)}},$$

where $\mathbf{r}', \mathbf{r}'', s$ are random exponents from $[N]$, simulate them as

$$g^{\mathbf{a}'}, g^{\mathbf{b}'} \odot (1+N)^{\mathsf{Bits(sk)}}, g^{\mathbf{a}''}, g^{\mathbf{b}''} \odot (1+N)^{\mathsf{Bits(sk)}},$$

where $\mathbf{a}', \mathbf{b}', \mathbf{a}'', \mathbf{b}''$ are random exponents from $[N^2]$. By DDH (Definition 5, which is implied by CP-DDH) in Paillier groups, we have $\mathsf{Hyb}'_2 \approx_c \mathsf{Hyb}'_1$.

Hyb'_3 In this hybrid, instead of multiplying the term $(1+N)^{\mathsf{Bits}(sk)}$ to random elements $g^{\mathbf{b}'}$ and $g^{\mathbf{b}''}$ as above, directly compute HSS public data at random

$$g^{\mathbf{a}'}, g^{\mathbf{b}'}, g^{\mathbf{a}''}, g^{\mathbf{b}''}.$$

By Lemma 1, the element g sampled by $\mathsf{Pai.Samp(pp)}$ has the guarantee that $\langle g \rangle$ contains the subgroup generated by $(1+N)$. Therefore, $g^{\mathbf{b}'}$ with a random exponent $\mathbf{b}'$ from $[N^2]$ perfectly hides the multiplicative factor $(1+N)^{\mathsf{Bits(sk)}}$. We have $\mathsf{Hyb}'_3 \equiv \mathsf{Hyb}'_2$.
Note that Hyb'_3 computes exactly the distribution of Equation 9.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_3$, which proves the claim. □

5.2 A Leveled Variant under Paillier Groups

Compared to the non-leveled protocol, the main changes in the leveled variant are (1) in the core sub-protocol LBoolGateEval both parties now run *leveled* aHMAC local evaluations, and (2) the leveled aHMAC and (normal) HSS instances no longer rely on common secret exponents.

The two differences together allow us to avoid circular security arguments in this variant. On the other hand, they require much larger public data, of size linear in the circuit depth. We now explain the differences in more detail.

– During Init, the garbler prepares appropriate public data (Equation 10) for $\mathrm{Depth}(C)$ instances of leveled aHMAC and HSS to support the following two types of local evaluations. Assume P_G, P_E jointly holds additive shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$, and P_E additionally holds $\overline{\mathbf{x}}$.

$$\text{Inputs: } (P_G : \mathsf{pd}, \langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle_0), \quad (P_E : \mathsf{pd}, \langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}).$$

1. In the first type, they locally run leveled aHMAC evaluations on the input shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$ over arithmetic circuits $C_{\mathbf{v}}$ (of depth d_{Ind}; Fact 1) to obtain additive shares $\langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle$, where $\mathbf{k}[t]$ is an independent secret exponent in the t-th HSS instance.

$$\text{Via aHMAC Eval: } (P_G : \langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle_0), \quad (P_E : \langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle_1).$$

They then locally run HSS to evaluate $C^{(i)}_{\mathsf{g},\mathbf{v}}$ over public data $\mathsf{HSS.pd}_{\mathsf{sk},s^{(t+1)}}$ where $s^{(t+1)}$ is an independent secret exponent in the $(t+1)$-th leveled aHMAC instance, and $C^{(i)}_{\mathsf{g},\mathbf{v}}(\mathsf{sk}, s^{(t+1)})$ is defined to compute $C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) \cdot \mathsf{Bits}(s^{(t+1)})[i]$ (see Equation 6). The result can be additionally "multiplied" by $C_{\mathbf{v}}(\overline{\mathbf{x}})$ via HSS extended evaluation. In the end, they jointly hold additive shares of $\langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle$ and $\langle \overline{z} \rangle$.

$$\begin{aligned}\text{Via HSS Eval: } &(P_G : \langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle_0, \langle \overline{z} \rangle_0),\\ &(P_E : \langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle_1, \langle \overline{z} \rangle_1).\end{aligned}$$

Finally, they locally linearly combine the shares $\langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle$ into shares $\langle s^{(t+1)} \cdot \overline{z} \rangle$.

$$\text{Via Linear Comb.: } (P_G : \langle s^{(t+1)} \cdot \overline{z} \rangle_0), \quad (P_E : \langle s^{(t+1)} \cdot \overline{z} \rangle_1).$$

2. In the second type, they locally run leveled aHMAC evaluations on the input shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$ over the identity arithmetic circuit C_{id} (of appropriate depth) to obtain additive shares $\langle s^{(t')} \cdot \overline{\mathbf{x}} \rangle$, where $t' > t$, and $s^{(t')}$ is an independent secret exponent in the t'-th leveled aHMAC instance.

$$\text{Via aHMAC Eval: } (P_G : \langle s^{(t')} \cdot \overline{\mathbf{x}} \rangle_0), \quad (P_E : \langle s^{(t')} \cdot \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}).$$

– During Eval, for every gate $\mathsf{g} \in C$ of depth t, assume P_G, P_E jointly holds additive shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$, and P_E additionally holds $\overline{\mathbf{x}}$.
First, they apply type-1 local evaluations to obtain additive shares of $\langle s^{(t+1)} \overline{z} \rangle, \langle \overline{z} \rangle$. Next, P_G sends his share $\langle \overline{z} \rangle$ mod 2 to P_E who then recovers $\overline{z}$ mod 2. Finally, for every gate $\mathsf{g}' \in C$ of depth $t' > t$ that uses $\overline{z}$ as an input, they apply type-2 local evaluations to obtain additive shares of $\langle s^{(t')} \overline{z} \rangle$.

Implementing the leveled variant requires careful book-keeping of the public data. We give full details of the leveled variant of our 2PC protocol under Paillier groups in Figure 4 , 5, and the leveled variant of the core sub-protocol in Figure 6.

Note that the total communication from P_G to P_E consists of *one bit* per invocation of the sub-protocol $\mathsf{LBoolGateEval}$, plus public data of size $\mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda)$, assuming all gates in C has fan-in $O(\log(\lambda))$. We summarize the correctness and security of the sub-protocol $\mathsf{LBoolGateEval}$ in the following lemmas.

Lemma 17 (Correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Paillier Groups). *Let $\ell(\lambda) \leq O(\log \lambda)$ be a bound on input length, and $d_{\mathrm{Ind}} = O(\log \log \lambda)$ be the depth of the indicator arithmetic circuit over ℓ inputs (Fact 1).*

There exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C (of depth d_C) with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (N, \zeta)$ in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^2)$, secret exponent $\mathbf{s} \in [N]^{d_C \cdot d_{\mathrm{Ind}}+1}$, additive shares (over $\mathbb{Z}$)*

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC *leveled* evaluation procedures $\mathsf{EvalKey}^{d_{\mathrm{Ind}}}, \mathsf{EvalTag}^{d_{\mathrm{Ind}}}$ for bounded depth computations by $d_{\mathrm{Ind}} = O(\log\log\lambda)$ [a] over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}$ under Paillier groups; (See Lemma 4;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}$ under paillier groups; (See Lemma 5;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.
Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init : Let $d_C = \mathrm{Depth}(C)$, and $d = d_C \cdot d_{\mathrm{Ind}}$.
  1. P_G sends public data pd to the evaluator P_E.

$$
\begin{aligned}
&\zeta := 2,\ \mathsf{pp} = (N,\zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta),\\
&\mathbf{s} \leftarrow [N]^{d+1},\ \mathbf{k} \leftarrow [N]^{d_C},\ \mathsf{sk} \leftarrow \{0,1\}^\lambda\\
&\quad // \text{ For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}],\ s^{(t)}_{\mathsf{end}} = \mathbf{s}[(t+1)\cdot d_{\mathrm{Ind}} - 1].\\
&\forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]),\\
&\forall t \in [d_C], \quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s^{(t)}_{\mathsf{end}}, \mathbf{k}[t]),\\
&\forall t \in [d_C], \quad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{k}[t], \mathsf{sk} \| \mathsf{Bits}(s^{(t+1)})),\\
&\mathsf{pd} := \left(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\right).
\end{aligned}
\tag{10}
$$

  2. Let $s = \mathbf{s}[0]$. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}}\rangle_1$ to P_E as in $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 1).

[a] d_{Ind} is the depth of the indicator arithmetic circuit over $O(\log\lambda)$ inputs (Fact 1).

Fig. 4. The Init phase of leveled 2PC for Boolean circuits under Paillier groups.

$\langle s^{(t)}\overline{\mathbf{x}}\rangle_0, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds: (where we use the shorthand $s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}]$)

$$
\Pr\left[\begin{array}{l} w_1^z = w_0^z + s^{(t+1)}\overline{z}, \\ z = \mathsf{g}(\mathbf{x}) \end{array} \middle| \begin{array}{l} \mathsf{pd} \text{ sampled per Equation 10}, \\ (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}} \\ \quad \left((P_G : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\right) \\ z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]
$$

$$
\geq 1 - \mathrm{negl}(\lambda).
$$

Proof. The correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ follows from that of the leveled variants of $\mathsf{EvalKey}$, $\mathsf{EvalTag}$ (Lemma 4) and that of $\mathsf{ExtEval}_b$ (Lemma 5). □

Lemma 18 (Security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Pailler Groups). *Under the same setting as Lemma 17, there exists an efficient simulator* Sim *that, given the masked output $\overline{z}$,* statistically *simulates P_G's message in the sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.*

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (N,\zeta)$*

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ Continued

– Eval : P_G, P_E evaluate gates $\mathsf{g} \in C$ (at depth t) in the topological order while maintaining the following invariant. (We write $s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}]$ for short.)
  1. P_G, P_E jointly hold additive shares $\langle s^{(t)} \overline{\mathbf{x}}_{\mathsf{g}} \rangle$, where $\overline{\mathbf{x}}_{\mathsf{g}}$ are masked input wire values to the gate g as in $\mathsf{BoolCircEval}^{C}$ (Equation 4).
  2. P_E holds the masked wire values $\overline{\mathbf{x}}_{\mathsf{g}}$.

  To evaluate the gate g, P_G, P_E call the sub-protocol $\mathsf{LBoolGateEval}$.

$$(P_G : \langle s^{(t+1)} \overline{z}_{\mathsf{g}} \rangle_0), (P_E : \langle s^{(t+1)} \overline{z}_{\mathsf{g}} \rangle_1, \overline{z}_{\mathsf{g}}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}} \left((P_G : \mathsf{pd}, \langle s^{(t)} \overline{\mathbf{x}}_{\mathsf{g}} \rangle_0), (P_E : \mathsf{pd}, \langle s^{(t)} \overline{\mathbf{x}}_{\mathsf{g}} \rangle_1, \overline{\mathbf{x}}_{\mathsf{g}}) \right).$$

  Then, for every gate g' (at depth $t' > t+1$) taking z as an input, P_G, P_E obtain shares $\langle s^{(t')} \overline{z} \rangle$ through local computations.

$$\mathsf{diff} := (t' - t - 1) \cdot d_{\mathrm{Ind}},\ \mathsf{pd}_{\mathsf{diff}} := \{\mathsf{aHMAC.pd}^{((t+1) \cdot d_{\mathrm{Ind}} + j)}\}_{j \in [\mathsf{diff}+1]}$$
$$P_G : \langle s^{(t')} \overline{z} \rangle_0 \leftarrow \mathsf{EvalKey}_0^{\mathsf{diff}}(\mathsf{pd}_{\mathsf{diff}}, C_{\mathsf{id}}, \langle s^{(t+1)} \cdot \overline{z} \rangle_0),$$
$$P_E : \langle s^{(t')} \overline{z} \rangle_1 \leftarrow \mathsf{EvalTag}_1^{\mathsf{diff}}(\mathsf{pd}_{\mathsf{diff}}, C_{\mathsf{id}}, \langle s^{(t+1)} \cdot \overline{z} \rangle_1, \overline{z}),$$

  where C_{id} (with depth $= \mathsf{diff}$) computes the identity function.

– Final : The same as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 2).

Fig. 5. The $\mathsf{Eval}, \mathsf{Final}$ phases of the leveled 2PC protocol for Boolean circuits under Paillier groups.

in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^2)$*, secret exponents* $\mathbf{s} \in [N]^{d_C \cdot d_{\mathrm{Ind}}+1}$*, additive shares (over* $\mathbb{Z}$*)* $\langle s^{(t)} \overline{\mathbf{x}} \rangle_0, \langle s^{(t)} \overline{\mathbf{x}} \rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds.*

$$\mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)} \overline{\mathbf{x}} \rangle_0), \mathsf{Sim}(\mathsf{pd}, \langle s^{(t)} \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \le \mathrm{negl}(\lambda), \quad \left| \begin{array}{l} \mathsf{pd} \text{ sampled per Equation 10}, \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array} \right.$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)} \overline{\mathbf{x}} \rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.

Proof. Analogous to the proof of Lemma 16. □

Using the correctness and security of $\mathsf{LBoolGateEval}$ under Paillier groups, we can now prove those of our leveled garbling scheme under Paillier groups (compiled from the 2PC protocol $\mathsf{LBoolCircEval}$).

Proposition 2 (Leveled Garbling of $O(\log \lambda)$-ary Gates under Paillier Groups). *Assuming P-DDH and DDH in Paillier groups, the garbling scheme compiled from the protocol* $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ *(Figure 4, 5) is correct and secure.*

Proof of Proposition 2. The correctness of the protocol follows from that of $\mathsf{LBoolGateEval}$ (Lemma 17). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as

Sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean gate $\mathsf{g} \in C$.
Inputs: P_G, P_E both hold public data $\mathsf{pd} = \{\mathsf{aHMAC.pd}^{(j)}\}, \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}, \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}\}$ (as defined in Equation 10), and jointly hold additive shares $\langle s^{(t)}\overline{\mathbf{x}}\rangle$, where $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$ is a masked input vector. P_E additionally holds the vector $\overline{\mathbf{x}}$.
Outputs: P_G, P_E jointly output additive shares $\langle s^{(t+1)}\overline{z}\rangle$, where $\overline{z} \in \{0,1\}$ is the masked output. P_E additionally outputs the bit $\overline{z}$.

– P_G, P_E obtain additive shares $\langle s^{(t)}\overline{z}\rangle$ and $\langle \overline{z}\rangle$ through local computations, where $\overline{z}$ is defined as in $\mathsf{BoolGateEval}$ (Equation 7). Let $C_{\mathbf{v}}$ and $C_{\mathsf{g},\mathbf{v}}$ be arithmetic and Boolean circuits specified in Fact 1 and Equation 6, respectively. Further define $C_{\mathsf{g}} := (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots)$, and $C_{\mathsf{g}}^{(i)} : \{0,1\}^{\ell_x} \times \{0,1\}^{\lceil \log \ell_x \rceil} \to \{0,1\}^{2^{\ell_x} \cdot \lceil \log \ell \rceil}$.

$$C_{\mathsf{g}}^{(i)}(\mathsf{sk}, s^{(t+1)}) = (\ldots, C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) \cdot \mathsf{Bits}(s^{(t+1)})[i], \ldots)_{\mathbf{v}\in\{0,1\}^{\ell_x}}.$$

1. P_G, P_E locally runs $\mathsf{EvalKey}^{d_{\mathrm{Ind}}}, \mathsf{EvalTag}^{d_{\mathrm{Ind}}}$, respectively, to obtain additive shares $\langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ and $\langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ for all $\mathbf{v} \in \{0,1\}^{\ell_x}$.

$$\begin{aligned}
&\mathsf{pd}_{\mathrm{Ind}} := \{\mathsf{aHMAC.pd}^{(t\cdot d_{\mathrm{Ind}}+j)}\}_{j\in[d_{\mathrm{Ind}}]} \cup \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}\}\\
&P_G : \langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow \mathsf{EvalKey}^{d_{\mathrm{Ind}}}(\mathsf{pd}_{\mathrm{Ind}}, C_{\mathbf{v}}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), \qquad \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow 0\\
&P_E : \langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow \mathsf{EvalTag}^{d_{\mathrm{Ind}}}(\mathsf{pd}_{\mathrm{Ind}}, C_{\mathbf{v}}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}), \quad \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow C_{\mathbf{v}}(\overline{\mathbf{x}}).
\end{aligned}$$

2. P_G, P_E locally runs $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$, respectively, to obtain additive shares $\langle s^{(t+1)}\overline{z}\rangle$ and $\langle \overline{z}\rangle$. ($P_E$'s computation is analogous P_G's.)

$$\begin{aligned}
P_G : &(_, \langle \overline{z}\rangle_0) \leftarrow \mathsf{ExtEval}_0(\mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}, C_{\mathsf{g}}, \{\langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\}_{\mathbf{v}}),\\
&(_, \langle \overline{z} \cdot \mathsf{Bits}(s^{(t+1)})[i]\rangle_0)\\
&\qquad \leftarrow \mathsf{ExtEval}_0(\mathsf{HSSpd}_{\mathsf{sk},\mathbf{s}}^{(t)}, C_{\mathsf{g}}^{(i)}, \{\langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\}_{\mathbf{v}}),\\
&\langle s^{(t+1)}\overline{z}\rangle_0 \leftarrow \mathsf{BitComp}\left(\langle \overline{z} \cdot \mathsf{Bits}(s^{(t+1)})[i]\rangle_0\right) \text{ over } \mathbb{Z}.
\end{aligned}$$

– P_G sends a bit $b := \langle \overline{z}\rangle_0 \bmod 2$ to P_E, who can then locally recover $\overline{z}$.

$$\overline{z} := \langle \overline{z}\rangle_1 - b \bmod 2.$$

Fig. 6. Leveled 2PC protocol for Boolean gates.

follows according to Equation 10, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^{\lambda}$:

$$\begin{aligned}
&\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai.Gen}(1^{\lambda}, 1^2),\ \mathbf{s} \leftarrow [N]^{d+1},\ \mathbf{k} \leftarrow [N]^{d_C},\\
&\quad // \text{ For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}],\ s_{\mathsf{end}}^{(t)} = \mathbf{s}[(t+1) \cdot d_{\mathrm{Ind}} - 1].\\
&\quad \forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]),\\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s_{\mathsf{end}}^{(t)}, \mathbf{k}[t]),\\
&\quad \forall t \in [d_C], \quad \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{k}[t], \mathsf{sk} \| \mathsf{Bits}(s^{(t+1)})),\\
&\mathsf{pd} := \left(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}, \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}\}_{t\in[d_C]}\right).
\end{aligned} \tag{11}$$

In the simulation, the are computed as follows:

$$\begin{aligned}
&\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2),\\
&\quad \forall j \in [d], \qquad \mathsf{aHMAC.}\widetilde{\mathsf{pd}}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}),\\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC.}\widetilde{\mathsf{pd}'}^{(t)}, \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}),\\
&\quad \forall t \in [d_C], \qquad \mathsf{HSS.}\widetilde{\mathsf{pd}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil + \lambda}),\\
&\widetilde{\mathsf{pd}} := \big(\{\mathsf{aHMAC.}\widetilde{\mathsf{pd}}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.}\widetilde{\mathsf{pd}'}^{(t)}, \mathsf{HSS.}\widetilde{\mathsf{pd}}^{(t)}\}_{t\in[d_C]}\big),
\end{aligned} \tag{12}$$

where $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)$ is as follows

$$\begin{aligned}
&g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow [N^2]^\ell,\ \mathsf{seed} \leftarrow \{0,1\}^\lambda\\
&\mathsf{aHMAC}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}),
\end{aligned} \tag{13}$$

and $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)$ as as follows

$$\begin{aligned}
&g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \leftarrow [N^2]^\ell,\ \mathsf{seed} \leftarrow \{0,1\}^\lambda\\
&\mathsf{HSS}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, g^{\mathbf{d}}).
\end{aligned} \tag{14}$$

We show an analogous claim (to Claim 1) which completes the proof.

Claim 2. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 11 and* $\widetilde{\mathsf{pd}}$ *by Equation 12 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 11 to Equation 12.

Hyb'_0 This is the distribution of Equation 11.

$\mathsf{Hyb}'_{0,1}$ Instead of computing the first instance of aHMAC public data $\mathsf{aHMAC.pd}^{(0)}$ as $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[0], \mathbf{s}[1])$, simulate it as $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil})$. We claim (Claim 3) the two ways of generating $\mathsf{aHMAC.pd}^{(0)}$ are computationally indistinguishable. Hence we have $\mathsf{Hyb}'_{0,1} \approx_c \mathsf{Hyb}'_0$.

$\mathsf{Hyb}'_{0,j}$ for $1 < j < d_{\mathrm{Ind}} - 1$, instead of computing the j-th instance of aHMAC public data from

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]),$$

simulate is as

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}).$$

By Claim 3 again, we have $\mathsf{Hyb}'_{0,j} \approx_c \mathsf{Hyb}'_{0,j-1}$.

$\mathsf{Hyb}'_{0,j}$ for $j = d_{\mathrm{Ind}} - 1$, instead of computing the aHMAC public data $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}^{(j)}, \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}'^{(0)}$ from (where $s^{(0)}_{\mathsf{end}} := \mathbf{s}[d_{\mathrm{Ind}} - 1]$)

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[s^{(0)}_{\mathsf{end}}, \mathbf{s}[d_{\mathrm{Ind}}]), \quad \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[s^{(0)}_{\mathsf{end}}], \mathbf{k}[0]),$$

simulate them as

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}), \quad \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}).$$

By Claim 3 again, we have $\mathsf{Hyb}'_{0,j} \approx_c \mathsf{Hyb}'_{0,j-1}$.

$\mathsf{Hyb}'_{0,j}$ for $j = d_{\mathrm{Ind}}$, instead of computing the HSS public data $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}^{(0)}$ from

$$\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{k}[0], \mathsf{sk} \| \mathsf{Bits}(s^{(1)})),$$

simulate it as

$$\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil + \lambda}).$$

We claim (Claim 4) the two ways of generating $\mathsf{HSS}.\mathsf{pd}^{(0)}$ are computationally indistinguishable. Hence we have $\mathsf{Hyb}'_{0,j} \approx_c \mathsf{Hyb}'_{0,j-1}$.

$\mathsf{Hyb}'_{t,j}$ for $1 < t < d_C$, $1 \le j \le d_{\mathrm{Ind}}$ is analogous to the case of $\mathsf{Hyb}'_{0,j}$, except replacing the 0 in its description with t.

We have $\mathsf{Hyb}'_{t,1} \approx_c \mathsf{Hyb}'_{t-1,d_{\mathrm{Ind}}}$, and $\mathsf{Hyb}'_{t,j} \approx_c \mathsf{Hyb}'_{t,j-1}$. Note that $\mathsf{Hyb}'_{d_C-1,d_{\mathrm{Ind}}}$ computes exactly the distribution of Equation 12.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_{d_C-1,d_{\mathrm{Ind}}}$, which proves the claim. It remains to prove the following sub-claims.

Claim 3. *For all $s' \in \mathbb{Z}$, with bit-length $\ell \le \mathrm{poly}(\lambda)$ the following computational indistinguishability holds*

$$\begin{aligned}&\left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s') \,\middle|\, s \leftarrow [N]\right\}_\lambda \\ \approx_c &\left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda\end{aligned}$$

where the public parameter pp is sampled as $\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai}.\mathsf{Gen}(1^\lambda, 1^2)$ in both sides.

Proof. This follows directly from P-DDH and DDH (Definition 7 and 6) in Paillier groups. □

Claim 4. *For all $s' \in \mathbb{Z}$, with bit-length $\ell \le \mathrm{poly}(\lambda)$, and $\mathsf{sk} \in \{0,1\}^\lambda$ the following computational indistinguishability holds*

$$\begin{aligned}&\left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, \mathsf{sk} \| \mathsf{Bits}(s')) \,\middle|\, s \leftarrow [N]\right\}_\lambda \\ \approx_c &\left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda\end{aligned}$$

where the public parameter pp is sampled as $\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai}.\mathsf{Gen}(1^\lambda, 1^2)$ in both sides.

Proof. This follows directly from DDH (Definition 5) in Paillier groups. □

5.3 Instantiations under Lattices

In this section, we instantiate the non-leveled and leveled variants of our 2PC protocols under lattices, $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$, $\mathsf{LBoolCircEval}^{C,\mathsf{Lat}}$. As explained in the begining of Section 5, the protocols stay mostly unchanged, except for the Init phases, during which P_G computes public data pd differently. We show them in Figure 7 and 8 respectively.

Parameter Settings. Our instantiations under lattices uses the following public parameter settings: A polynomial ring $\mathcal{R}(\lambda) = \mathbb{Z}[X]/(X^{n(\lambda)} + 1)$, two moduli $p(\lambda), q(\lambda)$, and error and secret distributions $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ where

- $n \le \mathrm{poly}(\lambda)$ is a power-of-two,
- $p \ge \lambda^{\omega(1)}$, $q = p \cdot \Delta$, and $\Delta \ge p \cdot \lambda^{\omega(1)}$;
- $\mathcal{D}_{\mathsf{err}}(\lambda)$, $\mathcal{D}_{\mathsf{sk}}(\lambda)$ have coefficients bounded by $\mathrm{poly}(\lambda)$.

We write $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$.

The Non-leveled Variant. The non-leveled 2PC protocol is shown in Figure 7. It uses the same core sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ (Figure 3) which stays unchanged. We summarize the correctness and security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under lattices in the following lemmas. Their proofs are completely analogous to those of Lemma 15 and 16, hence are omitted.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ specified in Section 5.3;
- aHMAC evaluation procedures $\mathsf{EvalKey}, \mathsf{EvalTag}$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 12;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 14;)
- a $\mathsf{PRF} : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.
Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init :
  1. P_G sends public data pd to the evaluator P_E.

$$s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathsf{sk} \leftarrow \{0,1\}^{\lambda}$$
$$\mathsf{pd} := \big(\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, s), \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, \mathsf{sk})\big). \tag{15}$$

  2. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}}\rangle_1$ to P_E.

$$\overline{\mathbf{x}} = \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)),$$
$$\langle s\overline{\mathbf{x}}\rangle_1 := s\overline{\mathbf{x}} + \langle s\overline{\mathbf{x}}\rangle_0 \text{ (over } \mathcal{R}\text{)}, \text{ where } \langle s\overline{\mathbf{x}}\rangle_0 \leftarrow \mathcal{R}^{\ell_x}_{\lambda^{\omega(1)}}.$$

- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 2).

Fig. 7. Our 2PC protocol for Boolean circuits under lattices.

Lemma 19 (Correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Lattices). *Let $\ell(\lambda) \le O(\log \lambda)$ be a bound on input length, and $\mathsf{pp}^{\mathsf{Lat}}$ be the public parameters specified in Section 5.3. There exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \le \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, secret exponent $s \in \mathcal{D}_{\mathsf{sk}}$, additive shares (over $\mathcal{R}$) $\langle s\overline{\mathbf{x}}\rangle_0, \langle s\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^{\lambda}$, the following holds:*

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s\overline{z}, & \mathsf{pd} \text{ sampled per Equation 15}, \\ z = \mathsf{g}(\mathbf{x}) & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \\ & \quad \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \quad \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\ge 1 - \mathrm{negl}(\lambda).$$

Lemma 20 (Security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Lattices). *Under the same setting as Lemma 19, there exists an efficient simulator Sim that, given the masked output $\overline{z}$,* statistically *simulates P_G's message in the sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.*

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$*, secret exponent* $s \in \mathcal{D}_{\mathsf{sk}}$*, additive shares (over* $\mathcal{R}$*)* $\langle s\overline{\mathbf{x}}\rangle_0, \langle s\overline{\mathbf{x}}\rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds:*

$$\mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), \mathsf{Sim}(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \mathrm{negl}(\lambda), \;\middle|\; \begin{array}{l} \mathsf{pd} \text{ sampled per Equation 15}, \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{BoolGateEval}^{C,\mathsf{g}}$*.*

Using the correctness and security of the core sub-protocol, BoolGateEval under lattices, we can now prove those of our garbling scheme under lattices (compiled from the 2PC protocol BoolCircEval).

Proposition 3 (Garbling of $O(\log\lambda)$-ary Gates under Lattices). *Assuming CP-RLWE with respect to the public parameters* $\mathsf{pp}^{\mathsf{Lat}}$ *specified in Section 5.3, the garbling scheme compiled from the protocol* $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$ *(Figure 7) is correct and secure.*

Proof of Proposition 3. The correctness of the protocol follows from that of BoolGateEval (Lemma 19). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as follows according to Equation 15, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, and the public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ described in Section 5.3.

$$\begin{aligned} \mathsf{seed} \leftarrow \{0,1\}^\lambda,\ s, r_1, r_2 \leftarrow \mathcal{D}_{\mathsf{sk}},\ a \leftarrow \mathcal{R}_q, \mathbf{a}' \leftarrow \mathcal{R}_q^\lambda, \\ e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathbf{e}', \mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}''_1, \mathbf{e}''_2 \leftarrow \mathcal{D}_{\mathsf{err}}^\lambda,\ \mathbf{b} := s\mathbf{a}' + \mathbf{e}' \\ \mathsf{pd} = (\mathsf{pp}^{\mathsf{Lat}}, \mathsf{seed}, a, sa + e_1, s^2a + e_2 - s\Delta \\ r_1\mathbf{a}' + \mathbf{e}'_1 + \mathsf{Bits}(\mathsf{sk})\Delta,\ r_1\mathbf{b} + \mathbf{e}''_1, \\ r_2\mathbf{a}' + \mathbf{e}'_2,\ r_2\mathbf{b} + \mathbf{e}''_2 + \mathsf{Bits}(\mathsf{sk})\Delta). \end{aligned} \tag{16}$$

In the simulation, they are computed as random elements:

$$\begin{aligned} \mathsf{seed} \leftarrow \{0,1\}^\lambda,\ a, b, c \leftarrow \mathcal{R}_q,\ \mathbf{b}', \mathbf{c}', \mathbf{b}'', \mathbf{c}'' \leftarrow \mathcal{R}_q^\lambda, \\ \widetilde{\mathsf{pd}} = (\mathsf{pp}^{\mathsf{Lat}}, \mathsf{seed}, a,\ b,\ c,\ \mathbf{b}'\,\mathbf{c}', \mathbf{b}'',\ \mathbf{c}''). \end{aligned} \tag{17}$$

We show the analogous claim (to Claim 1) which completes the arguments for this proof.

Claim 5. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 16 and* $\widetilde{\mathsf{pd}}$ *by Equation 17 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 16 to Equation 17.

Hyb'_0 This is the distribution of Equation 16.

Hyb'_1 In this hybrid, instead of computing the aHMAC public data, together with the intermediate value $\mathbf{b}$ as

$$a, sa + e_1, s^2a + e_2 - s\Delta, \mathbf{b} := s\mathbf{a}' + \mathbf{e}'$$

where $a, \mathbf{a}'$ are random elements in $\mathcal{R}_q$, s is a secret sampled from $\mathcal{D}_{\mathsf{sk}}$, and $e_1, e_2, \mathbf{e}'$ are errors from $\mathcal{D}_{\mathsf{err}}$, simulate them as random elements $a, b, c, \mathbf{b}$ from $\mathcal{R}_q$. By CP-RLWE (Definition 9), we have $\mathsf{Hyb}'_1 \approx_c \mathsf{Hyb}'_0$.

Hyb'_2 In this hybrid, instead of computing the HSS public data as

$$r_1\mathbf{a}' + \mathbf{e}'_1 + \mathsf{Bits}(\mathsf{sk})\Delta,\ r_1\mathbf{b} + \mathbf{e}''_1,\ r_2\mathbf{a}' + \mathbf{e}'_2,\ r_2\mathbf{b} + \mathbf{e}''_2 + \mathsf{Bits}(\mathsf{sk})\Delta,$$

where $\mathbf{a}, \mathbf{a}', \mathbf{b}$ are random elements from $\mathcal{R}_q$, r_1, r_2 secrets sampled from $\mathcal{D}_{\mathsf{sk}}$, and $\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}''_1, \mathbf{e}''_2$ are errors from $\mathcal{D}_{\mathsf{err}}$, simulate them as

$$\mathbf{b}' + \mathsf{Bits}(\mathsf{sk})\Delta,\ \mathbf{c}',\ \mathbf{b}'',\ \mathbf{c}'' + \mathsf{Bits}(\mathsf{sk})\Delta,$$

where $\mathbf{b}', \mathbf{c}', \mathbf{b}'', \mathbf{c}''$ are random elements from $\mathcal{R}_q$. By RLWE (which is implied by CP-RLWE) we have $\mathsf{Hyb}'_2 \approx_c \mathsf{Hyb}'_1$.

Hyb'_3 In this hybrid, instead of adding the term $\mathsf{Bits}(sk)\Delta$ to random elements $\mathbf{b}'$ and $\mathbf{c}''$ as above, directly compute HSS public data as random elements $\mathbf{b}', \mathbf{c}', \mathbf{b}'', \mathbf{c}''$ from $\mathcal{R}_q$.
Since $\mathbf{b}', \mathbf{c}''$ are random, they perfectly hide the additive factor $\mathsf{Bits}(sk)\Delta$. We have $\mathsf{Hyb}'_3 \equiv \mathsf{Hyb}'_2$. Note that Hyb'_3 computes exactly the distribution of Equation 17.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_3$, which proves the claim. □

The Leveled Variant. The leveled 2PC protocol is shown in Figure 8. It uses the same core sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ (Figure 6) which stays unchanged. We summarize the correctness and security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under lattices in the following lemmas. Their proofs are completely analogous to those of Lemma 17 and 18, hence are omitted.

Lemma 21 (Correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Lattices). *Let $\ell(\lambda) \le O(\log \lambda)$ be a bound on input length, $\mathsf{pp}^{\mathsf{Lat}}$ be the public parameters specified in Section 5.3, and $d_{\mathrm{Ind}} = O(\log\log\lambda)$ be the depth of the indicator arithmetic circuit over ℓ inputs (Fact 1).*

There exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C (of depth d_C) with a gate g of $\ell_x \le \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, secret exponents $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^{d_C \cdot d_{\mathrm{Ind}}+1}$, additive shares (over $\mathcal{R}$) $\langle s^{(t)}\overline{\mathbf{x}}\rangle_0, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds: (where we use the shorthand $s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}]$)

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s^{(t+1)}\overline{z}, & \mathsf{pd} \text{ sampled per Equation 18}, \\ z = \mathsf{g}(\mathbf{x}) & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}} \\ & \quad \big((P_G : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \\ & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\ge 1 - \mathrm{negl}(\lambda).$$

Lemma 22 (Security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Lattices). *Under the same setting as Lemma 21, there exists an efficient simulator Sim that, given the masked output $\overline{z}$,* statistically *simulates P_G's message in the sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.*

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Lat}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ specified in Section 5.3;
- aHMAC *leveled* evaluation procedures $\mathsf{EvalKey}^{d_{\mathrm{Ind}}}, \mathsf{EvalTag}^{d_{\mathrm{Ind}}}$ for bounded depth computations by $d_{\mathrm{Ind}} = O(\log\log\lambda)$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 13;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 14;)
- a $\mathsf{PRF} : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.
Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init : Let $d_C = \mathrm{Depth}(C)$, and $d = d_C \cdot d_{\mathrm{Ind}}$.
  1. P_G sends public data pd to the evaluator P_E.

$$
\begin{aligned}
&\mathbf{s} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d+1},\ \mathbf{k} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d_C},\ \mathsf{sk} \leftarrow \{0,1\}^{\lambda} \\
&\text{// For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}],\ s^{(t)}_{\mathsf{end}} = \mathbf{s}[(t+1)\cdot d_{\mathrm{Ind}} - 1]. \\
&\forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{s}[j], \mathbf{s}[j+1]), \\
&\forall t \in [d_C], \quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s^{(t)}_{\mathsf{end}}, \mathbf{k}[t]), \\
&\forall t \in [d_C], \quad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{k}[t], \mathsf{sk} \| \mathsf{Bits}(s^{(t+1)})), \\
&\mathsf{pd} := \big(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\big).
\end{aligned}
\tag{18}
$$

  2. Let $s = \mathbf{s}[0]$. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}}\rangle_1$ to P_E as in $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$ (Figure 7).
- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ (Figure 5).

Fig. 8. Our leveled 2PC protocol for Boolean circuits under lattices.

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, every Boolean circuit* C *with a gate* g *of* $\ell_x \le \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$*, secret exponents* $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^{d_C \cdot d_{\mathrm{Ind}}+1}$*, additive shares (over* $\mathbb{Z}$*)* $\langle s^{(t)}\overline{\mathbf{x}}\rangle_0, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^{\lambda}$*, the following holds:*

$$
\mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0),\ \mathsf{Sim}(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \le \mathrm{negl}(\lambda), \ \Big|\ \begin{array}{l} \mathsf{pd} \text{ sampled per Equation 18}, \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}
$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$*.*

Using the correctness and security of $\mathsf{LBoolGateEval}$ under lattices, we can now prove those of our leveled garbling scheme under lattices (compiled from the 2PC protocol $\mathsf{LBoolCircEval}$).

Proposition 4 (Leveled Garbling of $O(\log\lambda)$-ary Gates under Lattices). *Assuming P-RLWE with respect to the public parameters* $\mathsf{pp}^{\mathsf{Lat}}$ *specified in Section 5.3, the garbling scheme compiled from the protocol* $\mathsf{LBoolCircEval}^{C,\mathsf{Lat}}$ *(Figure 8) is correct and secure.*

Proof of Proposition 4. The correctness of the protocol follows from that of $\mathsf{LBoolGateEval}$ (Lemma 21). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as follows according to Equation 18, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$ and the public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ described in Section 5.3:

$$\begin{aligned}
&\mathbf{s} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d+1},\ \mathbf{k} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d_C},\\
&\text{// For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}],\ s_{\mathsf{end}}^{(t)} = \mathbf{s}[(t+1) \cdot d_{\mathrm{Ind}} - 1].\\
&\forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{s}[j], \mathbf{s}[j+1]),\\
&\forall t \in [d_C], \quad \mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s_{\mathsf{end}}^{(t)}, \mathbf{k}[t]),\\
&\forall t \in [d_C], \quad \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{k}[t], \mathsf{sk} \| \mathsf{Bits}(s^{(t+1)})),\\
&\mathsf{pd} := \left(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}, \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}\}_{t\in[d_C]}\right).
\end{aligned} \tag{19}$$

In the simulation, the are computed as follows:

$$\begin{aligned}
&\forall j \in [d], \quad \widetilde{\mathsf{aHMAC.pd}}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}),\\
&\forall t \in [d_C], \quad \widetilde{\mathsf{aHMAC.pd'}}^{(t)}, \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}),\\
&\forall t \in [d_C], \quad \widetilde{\mathsf{HSS.pd}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}),\\
&\widetilde{\mathsf{pd}} := \left(\{\widetilde{\mathsf{aHMAC.pd}}^{(j)}\}_{j\in[d]}, \{\widetilde{\mathsf{aHMAC.pd'}}^{(t)}, \widetilde{\mathsf{HSS.pd}}^{(t)}\}_{t\in[d_C]}\right),
\end{aligned} \tag{20}$$

where $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp})$ is as follows

$$\begin{aligned}
&a, b, c \leftarrow \mathcal{R}_q,\ \mathsf{seed} \leftarrow \{0,1\}^\lambda\\
&\mathsf{aHMAC}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, a, b, c),
\end{aligned} \tag{21}$$

and $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp})$ as as follows

$$\begin{aligned}
&\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \leftarrow \mathcal{R}_q^{n \log q + \lambda},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda\\
&\mathsf{HSS}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}).
\end{aligned} \tag{22}$$

We show an analogous claim (to Claim 2) which completes the proof.

Claim 6. *For all $\mathsf{sk} \in \{0,1\}^\lambda$, the distribution of pd defined by Equation 19 and $\widetilde{\mathsf{pd}}$ by Equation 20 are computationally indistinguishable.*

Proof. The proof is again analogous to that of Claim 2, based on the following two sub-claims.

Claim 7. *For all $s' \in \mathcal{D}_{\mathsf{sk}}$, the following computational indistinguishability holds*

$$\begin{aligned}
&\left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, s') \,\middle|\, s \leftarrow \mathcal{D}_{\mathsf{sk}}\right\}_\lambda\\
&\approx_c \left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}})\right\}_\lambda
\end{aligned}$$

Proof. This follows directly from P-RLWE (Definition 8). □

Claim 8. *For all* $s' \in [N]$*, and* $\mathsf{sk} \in \{0,1\}^\lambda$ *the following computational indistinguishability holds*

$$\left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, \mathsf{sk} \| \mathsf{Bits}(s')) \,\middle|\, s \leftarrow \mathcal{D}_{\mathsf{sk}} \right\}_\lambda$$
$$\approx_c \left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}})\right\}_\lambda.$$

Proof. This follows from RLWE, which is implied by P-RLWE. □

5.4 Instantiations under Prime-Order Groups

In this section, we instantiate the non-leveled and leveled variants of our 2PC protocols under prime-order groups, $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$, $\mathsf{LBoolCircEval}^{C,\mathsf{Pri}}$. As explained in the begining of Section 5, the protocols stay mostly unchanged, except for the Init phases, during which P_G computes public data pd differently. We show them in Figure 9 and 10 respectively.

The Non-leveled Variant. The non-leveled 2PC protocol is shown in Figure 9. It uses the same core sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ (Figure 3) which stays unchanged. We summarize the correctness and security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under prime-order groups in the following lemmas. Their proofs are completely analogous to those of Lemma 15 and 16, hence are omitted.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC evaluation procedures, with error bound $\delta = 1/(\mathrm{poly}(\lambda) \cdot |C|)$, $\mathsf{EvalKey}, \mathsf{EvalTag}$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}$ under prime-order groups; (See Lemma 7;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}$ under ElGamal; (See Lemma 9;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.
Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init :
  1. P_G sends public data pd to the evaluator P_E.

$$\begin{aligned} \mathsf{pp} = (G, p, g) &\leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s \leftarrow \mathbb{Z}_p,\ \mathbf{s} &:= \mathsf{Bits}(s),\ \mathsf{sk} \leftarrow \{0,1\}^\lambda \\ \mathsf{pd} &:= \left(\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{s}), \mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathsf{sk})\right). \end{aligned} \tag{23}$$

  2. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1$ to P_E.

$$\begin{aligned} \overline{\mathbf{x}} &= \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)), \\ \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1 &:= \mathbf{s} \otimes \overline{\mathbf{x}} + \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0 \text{ (over } \mathbb{Z}), \\ &\text{where } \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0 \leftarrow [\lambda^{\omega(1)}]^{\lceil 3 \log p \rceil \times \ell_x}. \end{aligned}$$

- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 2), except for syntactical changes from using dot products $\cdot$ when multiplying with a scalar s to using tensor products $\otimes$ when multiplying with a vector $\mathbf{s}$.

Fig. 9. Our 2PC protocol for Boolean circuits under prime-order groups.

Lemma 23 (Correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups). *Let $\ell(\lambda) \le O(\log\lambda)$ be a bound on input length, and $\delta = 1/(\mathrm{poly}(\lambda)\cdot|C|)$ be the error bound specified in Figure 9. There exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \le \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (G,p,g)$ in the support of $\mathsf{Pri.Gen}(1^\lambda)$, secret exponents $\mathbf{s} \in \{0,1\}^{\lceil \log p\rceil}$, additive shares (over $\mathbb{Z}$) $\langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_0, \langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds:*

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + \mathbf{s}\overline{z}, & \mathsf{pd} \text{ sampled per Equation 23},\\ z = \mathsf{g}(\mathbf{x}) & (P_G : w_0^z), (P_E : w_1^z, \overline{z})\\ & \quad \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\big)\\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \quad \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\ge 1 - \delta(\lambda) - \mathrm{negl}(\lambda).$$

Lemma 24 (Security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups). *Under the same setting as Lemma 23, there exists an efficient simulator Sim that, given the masked output $\overline{z}$,* statistically *simulates P_G's message in the sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.*

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \le \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (G,p,g)$ in the support of $\mathsf{Pri.Gen}(1^\lambda)$, secret exponents $\mathbf{s} \in \{0,1\}^{\lceil \log p\rceil}$, additive shares (over $\mathbb{Z}$) $\langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_0, \langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds:

$$\mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_0), \mathsf{Sim}(\mathsf{pd}, \langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \le \mathrm{negl}(\lambda) + \delta(\lambda), \left|\begin{array}{l} \mathsf{pd} \text{ sampled per Equation 23},\\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)),\\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}\right.$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s}\otimes\overline{\mathbf{x}}\rangle_0)$ denotes P_G's message to P_E in $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

Using the correctness and security of the core sub-protocol, $\mathsf{BoolGateEval}$ under prime-order groups, we can now prove those of our garbling scheme under prime-order groups (compiled from the 2PC protocol $\mathsf{BoolCircEval}$).

Proposition 5 (Garbling $O(\log\lambda)$-ary Gates under Prime-Order Groups). *Assuming CP-DDH in prime-order groups, the garbling scheme compiled from the protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$ (Figure 9) achieves $1/\mathrm{poly}$ correctness and privacy error.*

Proof of Proposition 5. The correctness of the protocol, with an error $\delta\cdot|C| = 1/\mathrm{poly}(\lambda)$ follows from that of $\mathsf{BoolGateEval}$ (Lemma 23), and a union bound on the error probability over all gates in C. Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 1, except for two differences.

- In Hyb_1, which changes from following the subprotocol $\mathsf{BoolGateEval}$ as P_G with $(\mathsf{pd}, \{k^{(i)}\})$ as inputs, into following the subprotocol as P_E with $(\mathsf{pd}, \{l^{(i)}, \overline{x}^{(i)}\})$ as inputs, there is an error probability for every gate in C. Therefore, the statistical distance between Hyb_1 and Hyb_0 is bounded by $\mathrm{negl}(\lambda) + \delta(\lambda)\cdot|C| \le 1/\mathrm{poly}(\lambda)$.
- The public data pd are computed and simulated differently as explained below. We need to argue the honestly computed pd and the simulated are computationally indistinguishable, which completes the argument for this proof.

In the honest protocol, the public data pd are computed as follows according to Equation 23, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$.

$$\begin{aligned} \mathsf{seed} \leftarrow \{0,1\}^\lambda,\ \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda),\\ s \leftarrow \mathbb{Z}_p, \mathbf{r} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil},\ \mathbf{r}' \leftarrow \mathbb{Z}_p^{\lambda},\ \mathbf{R} \leftarrow \mathbb{Z}_p^{\lambda \times \lceil \log p \rceil},\\ \mathsf{pd} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2 + \mathsf{Bits}(s)},\\ g^{\mathbf{r}'s}, g^{\mathbf{r}'s^2 + \mathsf{Bits}(\mathsf{sk})}, g^{\mathbf{R}s}, g^{\mathbf{R}s^2 + \mathsf{Bits}(\mathsf{sk}) \otimes \mathsf{Bits}(s)}). \end{aligned} \tag{24}$$

In the simulation, they are computed as random elements:

$$\begin{aligned} \mathsf{seed} \leftarrow \{0,1\}^\lambda,\ \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda),\\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil},\ \mathbf{a}', \mathbf{b}' \leftarrow \mathbb{Z}_p^{\lambda},\ \mathbf{A}, \mathbf{B} \leftarrow \mathbb{Z}_p^{\lambda \times \lceil \log p \rceil},\\ \mathsf{pd} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, g^{\mathbf{a}'}, g^{\mathbf{b}'}, g^{\mathbf{A}}, g^{\mathbf{B}}). \end{aligned} \tag{25}$$

We show the analogous claim (to Claim 1) which completes the arguments for this proof.

Claim 9. *For all $\mathsf{sk} \in \{0,1\}^\lambda$, the distribution of pd defined by Equation 24 and $\widetilde{\mathsf{pd}}$ by Equation 25 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 24 to Equation 25.

Hyb'_0 This is the distribution of Equation 16.

Hyb'_1 In this hybrid, instead of computing the last two terms of HSS public data as

$$\mathbf{H}_1 = g^{\mathbf{R}s}, \mathbf{H}_2 = g^{\mathbf{R}s^2 + \mathsf{Bits}(\mathsf{sk}) \otimes \mathsf{Bits}(s)},$$

where $\mathbf{R}$ are random exponents, simulate them based on the aHMAC public data $g^{\mathbf{r}s}, g^{\mathbf{r}s^2 + \mathsf{Bits}(s)}$ as follows.

$$\widetilde{\mathbf{H}}_1 = g^{\mathsf{Bits}(\mathsf{sk}) \otimes \mathbf{r}s + \mathbf{R}s}, \quad \widetilde{\mathbf{H}}_2 = g^{\mathsf{Bits}(\mathsf{sk}) \otimes (\mathbf{r}s^2 + \mathsf{Bits}(s)) + \mathbf{R}s^2}.$$

By the randomness of $\mathbf{R}$, we have $\mathsf{Hyb}'_1 \equiv \mathsf{Hyb}'_0$.

Hyb'_2 In this hybrid, instead of computing the aHMAC public data as

$$g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2 + \mathsf{Bits}(s)},$$

where $\mathbf{r}, s$ are random exponents, simulate them as random elements

$$g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}},$$

for random exponents $\mathbf{a}, \mathbf{b}, \mathbf{c}$. By CP-DDH in prime-order groups, (Definition 7), we have $\mathsf{Hyb}'_2 \approx_c \mathsf{Hyb}'_1$.

Hyb'_3 In this hybrid, instead of computing the HSS public data as

$$g^{\mathbf{r}'s}, g^{\mathbf{r}'s^2 + \mathsf{Bits}(\mathsf{sk})}, \widetilde{\mathbf{H}}_1 = g^{\mathsf{Bits}(\mathsf{sk}) \otimes \mathbf{b} + \mathbf{R}s}, \quad \widetilde{\mathbf{H}}_2 = g^{\mathsf{Bits}(\mathsf{sk}) \otimes \mathbf{c} + \mathbf{R}s^2}.$$

where $\mathbf{r}', \mathbf{R}$, $\mathbf{b}, \mathbf{c}$ are random exponents, simulate them as

$$g^{\mathbf{a}'}, g^{\mathbf{b}' + \mathsf{Bits}(\mathsf{sk})} \widetilde{\mathbf{H}}_1, = g^{\mathsf{Bits}(\mathsf{sk}) \otimes \mathbf{b} + \mathbf{A}}, \quad \widetilde{\mathbf{H}}_2 = g^{\mathsf{Bits}(\mathsf{sk}) \otimes \mathbf{c} + \mathbf{B}}.$$

where $\mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}', \mathbf{A}, \mathbf{B}$ are exponents. By P-DDH (Definition 6, which is implied by CP-DDH) in prime-order groups, we have $\mathsf{Hyb}'_3 \approx_c \mathsf{Hyb}'_2$.

Hyb'_4 In this hybrid, remove the additive terms involving $\mathsf{Bits}(\mathsf{sk})$ from the exponents.

Due to the randomness of $\mathbf{b}', \mathbf{A}, \mathbf{B}$, We have $\mathsf{Hyb}'_4 \equiv \mathsf{Hyb}'_3$. Note that Hyb'_4 computes exactly the distribution of Equation 25.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_4$, which proves the claim. □

The Leveled Variant. The leveled 2PC protocol is shown in Figure 10. It uses the same core sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ (Figure 6) which stays unchanged. We summarize the correctness and security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under prime-order groups in the following lemmas. Their proofs are completely analogous to those of Lemma 17 and 18, hence are omitted.

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Pri}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC *leveled* evaluation procedures, with error bound $\delta = 1/(\mathrm{poly}(\lambda) \cdot |C|)$, $\mathsf{EvalKey}^{d_{\mathrm{Ind}}}, \mathsf{EvalTag}^{d_{\mathrm{Ind}}}$ for bounded depth computations by $d_{\mathrm{Ind}} = O(\log\log\lambda)$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}$ under lattices; (See Lemma 8;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}$ under BHHO; (See Lemma 10;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.
Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init : Let $d_C = \mathrm{Depth}(C)$, and $d = d_C \cdot d_{\mathrm{Ind}}$.
  1. P_G sends public data pd to the evaluator P_E.

$$
\begin{aligned}
&\mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda),\ \mathbf{K} \leftarrow \{0,1\}^{d_C \times \lceil 3\log p\rceil},\ \mathsf{sk} \leftarrow \{0,1\}^\lambda \\
&\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil\log p\rceil} \text{ where } s_j \leftarrow \mathbb{Z}_p,\ \mathbf{S}[j] := \mathsf{Bits}(s_j), \\
&\text{// For short, write } \mathbf{s}^{(t)} = \mathbf{S}[t\cdot d_{\mathrm{Ind}}],\ \mathbf{s}^{(t)}_{\mathsf{end}} = \mathbf{S}[(t+1)\cdot d_{\mathrm{Ind}} - 1]. \\
&\forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{S}[j], \mathbf{S}[j+1]), \\
&\forall t \in [d_C], \quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}^{(t)}_{\mathsf{end}}, \mathbf{K}[t]), \\
&\forall t \in [d_C], \quad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}, \mathbf{K}[t], \mathsf{sk}\|\mathsf{Bits}(\mathbf{s}^{(t+1)})), \\
&\mathsf{pd} := \big(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\big).
\end{aligned}
\tag{26}
$$

  2. Let $\mathbf{s} = \mathbf{S}[0]$. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1$ to P_E as in $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$ (Figure 9).
- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ (Figure 5) except for syntactical changes from using dot products $\cdot$ when multiplying with a scalar s to using tensor products $\otimes$ when multiplying with a vector $\mathbf{s}$.

Fig. 10. Our leveled 2PC protocol for Boolean circuits under prime-order groups.

Lemma 25 (Correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups). *Let $\ell(\lambda) \le O(\log\lambda)$ be a bound on input length, $\delta = 1/(\mathrm{poly}(\lambda)\cdot|C|)$ be the error bound specified in Figure 10, and $d_{\mathrm{Ind}} = O(\log\log\lambda)$ be the depth of the indicator arithmetic circuit over ℓ inputs (Fact 1).*

There exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C (of depth d_C) with a gate g of $\ell_x \le \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (G,p,g)$ in the support of $\mathsf{Pri.Gen}(1^\lambda)$, secret exponents $\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil\log p\rceil}$, additive shares (over $\mathbb{Z}$)

$\langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1$, *and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$, *the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}^{(t+1)}\overline{z}, & \mathsf{pd} \text{ sampled per Equation 26}, \\ z = \mathsf{g}(\mathbf{x}) & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}} \\ & \quad \big((P_G : \mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0), (P_E : \mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \\ & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \delta(\lambda) - \mathrm{negl}(\lambda).$$

Lemma 26 (Security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups). *Under the same setting as Lemma 25, there exists an efficient simulator* Sim *that, given the masked output* $\overline{z}$, statistically *simulates* P_G*'s message in the sub-protocol* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, *every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$, *secret exponents* $\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil \log p \rceil}$, *additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1$, *and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$, *the following holds:*

$$\mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0), \mathsf{Sim}(\mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \mathrm{negl}(\lambda) + \delta(\lambda), \left| \begin{array}{l} \mathsf{pd} \text{ sampled per Equation 26}, \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}\right.$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.

Using the correctness and security of LBoolGateEval under prime-order groups, we can now prove those of our leveled garbling scheme under prime-order groups (compiled from the 2PC protocol LBoolCircEval).

Proposition 6 (Leveled Garbling of $O(\log \lambda)$-ary Gates under Prime-Order Groups). *Assuming P-DDH in prime-order groups, the garbling scheme compiled from the protocol* $\mathsf{LBoolCircEval}^{C,\mathsf{Pri}}$ *(Figure 10) achieves* $1/\mathrm{poly}$ *correctness and privacy error.*

Proof of Proposition 6. The correctness of the protocol follows from that of LBoolGateEval (Lemma 25). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as follows according to Equation 26, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$:

$$\begin{aligned} &\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ &\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil \log p \rceil} \text{ where } s_j \leftarrow \mathbb{Z}_p,\ \mathbf{S}[j] := \mathsf{Bits}(s_j), \\ &\mathbf{K} \leftarrow \{0,1\}^{d_C \times \lceil 3 \log p \rceil},\ \mathsf{sk} \leftarrow \{0,1\}^\lambda \\ &\quad // \text{ For short, write } \mathbf{s}^{(t)} = \mathbf{S}[t \cdot d_{\mathrm{Ind}}],\ \mathbf{s}_{\mathsf{end}}^{(t)} = \mathbf{S}[(t+1) \cdot d_{\mathrm{Ind}} - 1]. \\ &\quad \forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Pri}}, \mathbf{S}[j], \mathbf{S}[j+1]), \\ &\quad \forall t \in [d_C], \quad \mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Pri}}, \mathbf{s}_{\mathsf{end}}^{(t)}, \mathbf{K}[t]), \\ &\quad \forall t \in [d_C], \quad \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{K}[t], \mathsf{sk} \| \mathsf{Bits}(\mathbf{s}^{(t+1)})), \\ &\mathsf{pd} := \big(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}, \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}\}_{t\in[d_C]}\big). \end{aligned} \tag{27}$$

In the simulation, the are computed as follows:

$$\begin{aligned}
&\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda),\\
&\quad \forall j \in [d], \quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log p \rceil}),\\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil 3 \log p \rceil}),\\
&\quad \forall t \in [d_C], \quad \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}),\\
&\widetilde{\mathsf{pd}} := \big(\{\mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)}\}_{t\in[d_C]}\big),
\end{aligned} \tag{28}$$

where $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)$ is as follows

$$\begin{aligned}
&\mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathbb{Z}_p^\ell, \mathsf{seed} \leftarrow \{0,1\}^\lambda\\
&\mathsf{aHMAC}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}),
\end{aligned} \tag{29}$$

and $\mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{Sim}(\mathsf{pp})$ as as follows

$$\begin{aligned}
&\mathbf{a}, \mathbf{b}, \leftarrow \mathbb{Z}_q^{\lceil \log q \rceil + \lambda}, \mathbf{C}, \mathbf{D} \leftarrow \mathbb{Z}_q^{(\lceil \log q \rceil + \lambda) \times \lceil 3 \log p \rceil}, \mathsf{seed} \leftarrow \{0,1\}^\lambda\\
&\mathsf{HSS}^{\mathsf{BHHO}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{C}}, g^{\mathbf{D}}).
\end{aligned} \tag{30}$$

We show an analogous claim (to Claim 2) which completes the proof.

Claim 10. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 27 and* $\widetilde{\mathsf{pd}}$ *by Equation 28 are computationally indistinguishable.*

Proof. The proof is again analogous to that of Claim 2, based on the following two sub-claims.

Claim 11. *For all* $\mathbf{s}' \in \{0,1\}^\ell$*, with* $\ell \le \mathrm{poly}(\lambda)$ *the following computational indistinguishability holds*

$$\begin{aligned}
&\left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{s}') \,\middle|\, s \leftarrow \mathbb{Z}_p, \mathbf{s} := \mathsf{Bits}(s)\right\}_\lambda\\
&\approx_c \left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda
\end{aligned}$$

where the public parameter pp *is sampled as* $\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda)$ *in both sides.*

Proof. This follows from P-DDH (Definition 6) and DDH (Definition 5, which is implied by P-DDH) in prime-order groups. □

Claim 12. *For all* $\mathbf{s}' \in \{0,1\}^\ell$*, with* $\ell \le \mathrm{poly}(\lambda)$ *and* $\mathsf{sk} \in \{0,1\}^\lambda$ *the following computational indistinguishability holds*

$$\begin{aligned}
&\left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}, s, \mathsf{sk} \| \mathsf{Bits}(s')) \,\middle|\, s \leftarrow \mathcal{D}_{\mathsf{sk}}\right\}_\lambda\\
&\approx_c \left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda.
\end{aligned}$$

where the public parameter pp *is sampled as* $\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda)$ *in both sides.*

Proof. This follows from the security of the BHHO [BHHO08] encryption scheme, which is based on DDH (Definition 5) in prime-order groups. □

5.5 Security Amplification for Prime-Order Group Instantiations.

In this section, we show how to adapt the amplification techniques from [BGI17] to remove the 1/poly privacy and correctness errors of our garbling scheme under prime-order groups. For simplicity, we focus on the non-leveled variant in this section. The leveled variant can be amplified in the analogous way.

The Reason For the Errors. The the 1/poly privacy and correctness errors of our garbling scheme both come from the 1/poly correctness errors in the aHMAC and HSS constructions under prime-order groups. While it's clear correctness of our garbling scheme depends on those of aHMAC and HSS, it's less obvious how privacy depends on those. We briefly review our proof strategy (see the proof of Proposition 5) to illustrate the cause of this error.

The relevant step in our proof consists the following hybrid experiments for computing the garbled circuits $\widehat{C}$, and labels $\{L^{(i)}\}$.

Hyb_0 : This is the real world distribution.
- First sample a global secret $\mathbf{s}$ and a PRF key sk. Compute public data pd w.r.t. $\mathsf{seed}, \mathbf{s}, \mathsf{sk}$ following Equation 23.
- Next sample a random pad $\mathbf{k}^{(i)}$ for every input wire i in C, and compute the labels $L^{(i)}$ as $L^{(i)} = \mathbf{s} \cdot (\mathbf{x}[i] \oplus \mathsf{PRF}(\mathsf{sk}, i)) + \mathbf{k}^{(i)}$, where $\mathbf{x}$ is the input.
- For every gate g in C, in topological order, run aHMAC and HSS (as P_G described in Figure 3) evaluations over $\{\mathbf{k}^{(i)}\}$, for input wires i to g. The results are a pad $\mathbf{k}^{(j)}$ and an integer $r^{(j)}$. Set $b^{(j)} = r^{(j)} \bmod 2$.
- In the end, compute $\mathbf{o} = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$ and set $\widehat{C} = (\mathsf{pd}, \{b^{(j)}\}, \mathbf{o})$.

Hyb_1 : In this hybrid, compute the bits $\{b^{(j)}\}$ differently.
- For every output wire j of some gate $\mathsf{g} \in C$, compute the correct wire value $x^{(j)}$ according to the input $\mathbf{x}$. Then set $\overline{x}^{(j)} = x^{(j)} \oplus \mathsf{PRF}(\mathsf{sk}, j)$.
- For every gate g in C, in topological order, run aHMAC and HSS (as P_E described in Figure 3) evaluations over $\{L^{(i)}, x^{(i)}\}$, for the input wires i to g. The results are a label $L^{(j)}$ and an integer $u^{(j)}$. Set the bit $b^{(j)} = \overline{x}^{(j)} + u^{(j)} \bmod 2$.

If there are no error in the aHMAC and HSS evaluations, then we have $L^{(j)} = \mathbf{s} \cdot \overline{x}^{(j)} + \mathbf{k}^{(j)}$, and $u^{(j)} = \overline{x}^{(j)} + r^{(j)}$ for every output wire j of some gate $\mathsf{g} \in C$. Hence conditioned on no error occurs, $\mathsf{Hyb}_0, \mathsf{Hyb}_1$ compute the same distribution.

In our construction (Figure 9) we set the error chance of each aHMAC and HSS evaluation to be $\leq 1/(\mathrm{poly}(\lambda)|C|)$. Hence by a union bound, no error occurs except with 1/poly chance, creating a 1/poly statistical distance between Hyb_0 and Hyb_1.

Removing the Correctness Error. The correctness errors from both aHMAC and HSS evaluations stem from the following distributed discrete logarithm (DDLog) technique, which underlies their constructions (Lemma 7 and 9). In particular, the following DDLog evaluation is invoked for every intermediate multiplication within aHMAC and HSS.

Lemma 27 (Distributed Discrete Log with Error [BGI16, DKK18]). *For any cyclic group G with order p and a generator g, there exists an algorithm $\mathsf{DDLog}_{G,g}$:*

- $\mathsf{DDLog}_{G,g}(\delta \in (0,1], B \in [p], \phi : G \to \{0,1\}^{\lceil \log(2B/\delta) \rceil}, a \in G)$ *takes an error bound δ, a message bound B, a function ϕ mapping group elements to bit strings, and an element a. It outputs a value $\alpha \in \mathbb{Z}_p$.*

The algorithm requires $O(\sqrt{B/\delta})$ group operations, and has the guarantee that for all $0 < \delta \leq 1$, $B < p$, $a \in G$, and $m \leq B$:

$$\Pr\left[\begin{array}{l}\mathsf{DDLog}_{G,g}(\delta, B, \phi, a \cdot g^m) \\ =\mathsf{DDLog}_{G,g}(\delta, B, \phi, a) + m \bmod p\end{array}\middle|\ \phi \leftarrow \$\right] \geq 1 - \delta,$$

where $\phi \leftarrow \$$ means sampling at random from all possible mappings.

The DDLog algorithm is setup with a sufficiently small error bound δ such that the overall error probability (through a union bound) of all aHMAC and HSS multiplications is bounded by 1/poly. A (pseudo-)random mapping function ϕ used for DDLog is specified by the public PRG seed included in the public data pd of aHMAC and HSS.

To remove the correctness error, we follow the observation from [BGI17] that when two parties locally run DDLog on two elements $a, a \cdot g^m$, one of the party, which we call the left party, can actually detect potential errors as long as there is a bound B on the value m:

- The left party aborts with probability $\leq \delta$ over the randomness of ϕ;
- When the left party doesn't abort, both parties output the correct results except with negligible probability.

Armed with this detection technique, we can remove the correctness error from our grabling scheme: when the garbler – who acts as the left party in DDLog – aborts, it restarts with fresh randomness.

By setting the error probability δ in each DDLog invocation to be sufficiently small, $\leq 1/(\text{poly}(\lambda) \cdot |C|)$, the garbler only restarts with $1/\text{poly}(\lambda)$ probability. In expectation, it takes a constant number of restarts before the garbler produces a garbled circuit that's guaranteed to be correct.

Removing the Privacy Error. While restarting removes the 1/poly correctness error, there is still a 1/poly privacy error in the resulting scheme.

Looking again at the two hybrids $\mathsf{Hyb}_0, \mathsf{Hyb}_1$ in our proof strategy, it may seem with restarting we have ensured no error occurs during all aHMAC and HSS evaluations, and have removed the 1/poly statistical difference between Hyb_0 and Hyb_1. However, the subtle issue is that in Hyb_0, the experiment runs DDLog as the left party to detect errors and restarts, while in Hyb_1 the experiment runs DDLog as the right party, who does not have the same restarting pattern.

To simulate the restarting pattern of Hyb_0, our first step is to use another observation from [BGI17]: the right party running DDLog can actually predict potential abort from the left party with possibly false positives:

- The right party can additionally output a bit pred, which equals 1 with probability $\leq 2\delta$ over the randomness of ϕ;
- When $\mathsf{pred} = 0$, the left party does not abort.

Armed with this prediction technique, in Hyb_1, the experiment can proceed as the right party running DDLog as long as $\mathsf{pred} = 0$.

However, when the experiment sees $\mathsf{pred} = 1$ during some DDLog invocation, it then needs to re-run this particular DDLog as the left party to check if there needs to be a restart (as $\mathsf{pred} = 1$ may be a false positive). As we explain next, re-running the DDLog as the left party relies on some leakages on the global secret s and the PRF key sk. We then show how to deal with those leakages by adapting techniques from [BGI17].

Leakages in aHMAC and HSS. In order to explain the leakage, we now expose a bit more detail on how the left and right parties, which correspond to P_G and P_E respectively in Figure 3, run DDLog within aHMAC and HSS evaluations.

- aHMAC evaluations are over input shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle$, where $\mathbf{s}$ is the global secret. The results are output shares $\langle \mathbf{s} \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle$, where $C_{\mathbf{v}}$ is an arithmetic circuit with intermediate values bounded by $B = 2$. The right party additionally holds $\overline{\mathbf{x}}$ in the clear.
  In each invocation of DDLog, the left and right parties respectively hold inputs of the form $a, a \cdot g^{\mathbf{s}[i] \cdot v}$:
  $$\begin{aligned} \text{Left}: &\quad \mathsf{DDLog}(\delta, \phi, B, a), \\ \text{Right}: &\quad \mathsf{DDLog}(\delta, \phi, B, a \cdot g^{\mathbf{s}[i] \cdot v}), \end{aligned}$$
  where v is an intermediate wire value of $C_{\mathbf{v}}(\overline{\mathbf{x}})$.
  When the right party predicts a potential abort and needs to re-run DDLog as the left party, it needs to know both $\mathbf{s}[i]$ and v. As all intermediate wire values v are known to the right party in the clear, the only leakage required is a certain bit $\mathbf{s}[i]$ from the global secret.
- HSS evaluations are over input shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle$, and $\langle \overline{\mathbf{x}} \rangle$. The results are output shares $\{\langle \mathbf{s} \cdot \mathsf{InnerPord}(\overline{\mathbf{x}}, C_{\mathsf{g}}(\mathsf{sk})) \rangle\}$, where C_{g} is a Boolean circuit (implementable by an arithmetic circuit with $B = 2$), and sk is the global PRF key. The right party additionally holds $\overline{\mathbf{x}}$ in the clear.
  In each invocation of DDLog, the left and right parties respectively hold inputs of the form $a, a \cdot g^{\mathbf{s}[i] \cdot \overline{\mathbf{x}}[j] \cdot v}$:
  $$\begin{aligned} \text{Left}: &\quad \mathsf{DDLog}(\delta, \phi, B = 2, a), \\ \text{Right}: &\quad \mathsf{DDLog}(\delta, \phi, B = 2, a \cdot g^{\mathbf{s}[i] \cdot \overline{\mathbf{x}}[j] \cdot v}), \end{aligned}$$
  where v is an intermediate wire value of $C_{\mathsf{g}}(\mathsf{sk})$.
  When the right party predicts a potential abort and needs to re-run DDLog as the left party, it needs to know $\mathbf{s}[i]$, $\overline{\mathbf{x}}[j]$, and v. As $\overline{\mathbf{x}}$ is known to the right party in the clear, the leakage required is a certain bit $\mathbf{s}[i]$ from the global secret, and an intermediate wire value v from $C_{\mathsf{g}}(\mathsf{sk})$.

In summary, the Hyb_1 experiment proceeds as the righ party running DDLog in aHMAC and HSS evaluations, as long as $\mathsf{pred} = 0$. In the case $\mathsf{pred} = 1$, it relies on the following leakage to re-run the DDLog as the left party.

- If the DDLog is within an aHMAC evaluation, the leakage is a bit $\mathbf{s}[i]$ from the global secret $\mathbf{s}$.
- If the DDLog is within an HSS evaluation, the leakage is a bit $\mathbf{s}[i]$ and an intermediate wire value v in $C_{\mathsf{g}}(\mathsf{sk})$.

If the re-run as the left party indeed aborts, then Hyb_1 restarts with fresh randomness, and all the leakages have no effect. However if the re-run does not abort, (i.e. $\mathsf{pred} = 1$ is a false positive), then Hyb_1 continues as the right party. This restarting pattern exactly simulates that of Hyb_0, so we have $\mathsf{Hyb}_0 \equiv \mathsf{Hyb}_1$.

Note that as $\mathsf{pred} = 1$ happens independently in each DDLog invocation with $\leq 2\delta \leq 1/(\mathrm{poly}(\lambda) \cdot |C|)$ probability, in an eventual accepting Hyb_1 experiment with no aborts, there are at most some $\omega(1) \leq \lambda$ instances of leakages except with negligible probability.

In conclusion, the overall leakage in Hyb_1 are (1) $\leq \lambda$ bits from the global secret $\mathbf{s}$ and (2) $\leq \lambda$ intermediate values in the circuit $C_{\mathsf{g}}(\mathsf{sk})$.

Removing the Leakages. We explain the solutions to each leakage type in more detail. They are adapted from the techniques introduced in [BGI17] for dealing with similar types leakages.

1. The global secret $\mathbf{s} \in \{0,1\}^{\lceil \log p \rceil}$ in our garbling scheme is sampled as follows, per Equation 23:

$$s \leftarrow \mathbb{Z}_p, \quad \mathbf{s} := \mathsf{Bits}(s).$$

Our security proof (of Proposition 5) relies on the s being a random exponent and the CP-DDH assumption (Definition 7) to argue that the public data pd leaks nothing about the PRF key sk. With $\leq \lambda$ bits of leakage from $\mathbf{s}$, the secret exponent s is no longer random. Our solution is to create $\lambda+1$ additive shares of s, and define $\mathbf{s}$ to be the bits of all $\lambda+1$ shares. Any $\leq \lambda$ bits leaked from $\mathbf{s}$ are now statistically independent of the secret exponent s, which remains random. In more detail, we modify Equation 23 and correspondingly $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}$, $\mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}$ as follows:

Modified Equation 23 :

$$\forall i \in [\lambda+1],\ s_i \leftarrow \mathbb{Z}_p, \quad s := \sum_i s_i \bmod p, \quad \mathbf{s} := (\ldots \| \mathsf{Bits}(s_i) \| \ldots).$$

Modified $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}, \mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}$:

$$\text{parse } \mathbf{s} = (\ldots \|\mathbf{s}_i\| \ldots), \quad s := \sum_i \mathsf{BitComp}(\mathbf{s}_i) \bmod p.$$

Now our proof argument of pd computed using the modified Equation 23 goes through, under a slight variant of the CP-DDH assumption that incorporates the extra secret sharing steps. (In the leveled variants, the P-DDH assumption unmodified suffices.)

Definition 11 (CP-DDH* Over Prime-Order Groups). *We say CP-DDH* holds in prime-order groups if the following holds:*

$$\left\{ \begin{array}{l} \mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^{s^2}, \\ g^{\mathbf{a}}, g^{s\cdot\mathbf{a}}, g^{s^2\cdot\mathbf{a}+(\ldots\|\mathsf{Bits}(s_i)\|\ldots)} \end{array} \middle| \begin{array}{l} \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s_0, s_1, \ldots, s_\lambda \leftarrow \mathbb{Z}_p,\ s := \sum_i s_i \bmod p \\ \mathbf{a} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil \cdot (\lambda+1)} \end{array} \right\}_\lambda$$

$$\approx_c \left\{ \begin{array}{l} \mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^d, \\ g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}} \end{array} \middle| \begin{array}{l} \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, s_1, \ldots, s_\lambda, d \leftarrow \mathbb{Z}_p, \\ \mathbf{a},\mathbf{b},\mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil}. \end{array} \right\}_\lambda .$$

Remark 5. This formulation can be further simplified to

$$\begin{aligned} &\left\{ \begin{array}{l} \mathsf{pp}, s', g, g^s, g^{s^2}, \\ g^{\mathbf{a}}, g^{s\cdot\mathbf{a}}, g^{s^2\cdot\mathbf{a}+\mathsf{Bits}(s+s' \bmod p)} \end{array} \middle| \begin{array}{l} \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, s' \leftarrow \mathbb{Z}_p,\ \mathbf{a} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil} \end{array} \right\}_\lambda \\ \approx_c &\left\{ \begin{array}{l} \mathsf{pp}, s', g, g^s, g^d, \\ g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}} \end{array} \middle| \begin{array}{l} \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, s', d \leftarrow \mathbb{Z}_p,\ \mathbf{a},\mathbf{b},\mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil}. \end{array} \right\}_\lambda . \end{aligned} \tag{31}$$

We sketch this through the following hybrid arguments:

Hyb'_0 This is the left-hand-side distribution from the CP-DDH* assumption, in a slightly more convenient form:

$$\mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^{s^2}, \{g^{\mathbf{a}_i}, g^{s\cdot\mathbf{a}_i}, g^{s^2\cdot\mathbf{a}_i+\mathsf{Bits}(s_i)}\}_{i=0,1,\ldots,\lambda} \cdot \left| \begin{array}{l} \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s_0, s_1, \ldots, s_\lambda \leftarrow \mathbb{Z}_p,\ s := \sum_i s_i \bmod p \\ \\ \mathbf{a}_i \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil} \end{array} \right.$$

Hyb'_1 Equivalently sample $s \leftarrow \mathbb{Z}_p$ and set $s' := \sum_{i>0} s_i \bmod p$, and $s_0 := s - s' \bmod p$. The distribution is:

$$\mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^{s^2}, \\ g^{\mathbf{a}_0}, g^{s\cdot\mathbf{a}_0}, g^{s^2\cdot\mathbf{a}_0+\mathsf{Bits}(s-s' \bmod p)}, \{g^{\mathbf{a}_i}, g^{s\cdot\mathbf{a}_i}, g^{s^2\cdot\mathbf{a}_i+\mathsf{Bits}(s_i)}\}_{i=1,\ldots,\lambda}.$$

We have $\mathsf{Hyb}'_0 \equiv \mathsf{Hyb}'_1$.

Hyb'_2 Replace the terms $g^{s^2}, g^{s\cdot\mathbf{a}_0}, g^{s^2\cdot\mathbf{a}_0+\mathsf{Bits}(s-s' \bmod p)}$, with $g^d, g^{\mathbf{b}_0}, g^{\mathbf{c}_0}$ for random exponents $d, \mathbf{b}_0, \mathbf{c}_0$:

$$\mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^d, \\ g^{\mathbf{a}_0}, g^{\mathbf{b}_0}, g^{\mathbf{c}_0}, \{g^{\mathbf{a}_i}, g^{s\cdot\mathbf{a}_i}, g^{d\cdot\mathbf{a}_i+\mathsf{Bits}(s_i)}\}_{i=1,\ldots,\lambda}.$$

By the simplied assumption in Equation 31, we have $\mathsf{Hyb}'_2 \approx_c \mathsf{Hyb}'_1$.

Hyb'_3 Replace the terms $g^{s\cdot\mathbf{a}_i}$ and $g^{d\cdot\mathbf{a}_i}$ with $g^{\mathbf{b}_i}, g^{\mathbf{c}_i}$ for random exponents $\mathbf{b}_i, \mathbf{c}_i$:

$$\mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^d, \\ g^{\mathbf{a}_0}, g^{\mathbf{b}_0}, g^{\mathbf{c}_0}, \{g^{\mathbf{a}_i}, g^{\mathbf{b}_i}, g^{\mathbf{c}_i+\mathsf{Bits}(s_i)}\}_{i=1,\ldots,\lambda}.$$

By DDH (which is implied by Equation 31), we have $\mathsf{Hyb}'_3 \approx_c \mathsf{Hyb}'_2$.

Hyb'_4 Remove the additive terms $\mathsf{Bits}(s_i)$ from the exponents. By the randomness of exponents $\mathbf{c}_i$, we have $\mathsf{Hyb}'_4 \equiv \mathsf{Hyb}'_3$. Note that the resulting is exactly the right-hand-side distribution from the CP-DDH* assumption.

2. To deal with the second leakage type, we need to ensure the leaked intermediate values from $C_\mathsf{g}(\mathsf{sk})$, for all gates $\mathsf{g} \in C$, are independent of the PRF key sk.
The solution is to replace C_g with a leakage resilient circuit $\overline{C}_\mathsf{g}$ such that any set of $\leq \lambda$ intermediate values can be computationally simulated from the evaluation result only. We cite the result from [BGI17] that there exists a compiler from any NC1 Boolean circuit C_g to a leakage resilient one $\overline{C}_\mathsf{g}$ also in NC1, together with a compiler for the inputs sk to $\overline{\mathsf{sk}}$ such that $C_\mathsf{g}(\mathsf{sk}) = \overline{C}_\mathsf{g}(\overline{\mathsf{sk}})$.

Lemma 28 (Leakage Resilient Circuits for NC1 [BGI17]). *Assuming there is a PRF in NC1. There exists a pair of compilers* $\mathsf{LR}^{\mathsf{Circ}}$ $\mathsf{LR}^{\mathsf{Input}}$ *satisfy the following.*

Correctness: *For every logarithmic depth bound $d \leq O(\log\lambda)$, there exists another logarithmic bound $d' \leq O(\log\lambda)$, and a polynomial $p \leq \mathrm{poly}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}$ of depth $\leq d(\lambda)$, and inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$:*
- $\overline{C} \leftarrow \mathsf{LR}^{\mathsf{Circ}}(C)$ *has depth* $\leq d'(\lambda)$;
- $\overline{\mathbf{x}} \leftarrow \mathsf{LR}^{\mathsf{Input}}(\mathbf{x})$ *has bit-length* $\leq \ell_x \cdot p(\lambda)$;
- $C(\mathbf{x}) = \overline{C}(\overline{\mathbf{x}})$.

Leakage Resilience: *There exists a simulator* Sim *such that for every logarithmic depth bound* $d \le O(\log \lambda)$*, Boolean circuits* $\{C_\lambda\}$ *of depth* $\le d(\lambda)$*, inputs* $\{\mathbf{x}_\lambda\}$*, and sets of leakage wires* $\{S_\lambda\}$ *of size* $\le \lambda$*:*

$$\left\{ \begin{array}{l|l} \textit{Wire values} & \overline{C} \leftarrow \mathsf{LR}^{\mathsf{Circ}}(C), \\ \textit{in } S \textit{ of } \overline{C}(\overline{\mathbf{x}}) & \overline{\mathbf{x}} \leftarrow \mathsf{LR}^{\mathsf{Input}}(\mathbf{x}), \end{array} \right\}_\lambda \approx_c \{\mathsf{Sim}(C(\mathbf{x}))\}_\lambda$$

Now we just need to modify Equation 23 to compile the sampled PRF key sk into leakage resilient inputs $\overline{\mathsf{sk}}$, and correspondingly modify Figure 3 to use leakage relient circuits in HSS evaluations:

$$\begin{array}{ll} \text{Modified Equation 23 :} & \mathsf{sk} \leftarrow \{0,1\}^\lambda, \quad \overline{\mathsf{sk}} \leftarrow \mathsf{LR}^{\mathsf{Input}}(\mathsf{sk}) \\ \text{Modified Figure 3 Step 2 :} & \text{run } \mathsf{ExtEval}_b \text{ with } \overline{C}_{\mathsf{g}} \leftarrow \mathsf{LR}^{\mathsf{Circ}}(C). \end{array}$$

After applying the two solutions, we can now conclude that in Hyb_1 the leakages (required to simulating restarting patterns of Hyb_0) do not affect the remaining proof arguments.

The overhead of our solutions for removing leakages are (1) larger public data pd caused by a larger global secret vector $\mathbf{s}$ and a larger leakage resilient version of the PRF key $\overline{\mathsf{sk}}$ and (2) heavier computation in HSS evaluations caused by the leakage resilient circuits $\overline{C}_{\mathsf{g}}$.

6 Efficient Arithmetic Garbling Schemes

Our observation is that the Boolean garbling schemes from Theorem 2 (and their leveled variants), supporting evaluations of arbitrary $O(\log \lambda)$-ary gates, can implement arithmetics over small modulus $R(\lambda) \le \mathrm{poly}(\lambda)$ very efficeintly. This is because multiplications and additions between two $\mathbb{Z}_R$ values can be implemented by $(2 \log R)$-ary Boolean gates, costing $\log R$ bits per multiplication or addition.

A small issue prevents directly using Theorem 2 to obtain arithmetic garbling schemes for small modulus. An arithmetic garbling scheme requires arithmetic labels for its inputs $\mathbf{x} \in \mathbb{Z}_R^{\ell_x}$, while the schemes from Theorem 2 require Boolean labels for the bit representations $\mathsf{Bits}(\mathbf{x})$.

Therefore, to obtain arithmetic garbling over polynomial modulus $R(\lambda) < \mathrm{poly}(\lambda)$, we need a special garbling scheme in which the evaluation algorithm Eval takes in arithmetic labels for some input $\mathbf{x} \in \mathbb{Z}_R^{\ell_x}$, and outputs Boolean labels for their bit-representations $\mathsf{Bits}(\mathbf{x})$, as required by the scheme from Theorem 2. Fortunately, such schemes exist based on Chinese Remainder Theorem and the minimal assumption of one-way functions. (See Section 8 in [AIK11], and also [LL24, Hea24] for more efficient constructions based on stronger assumptions.)

Lemma 29 (Bit-Decomposition Garbling Scheme [AIK11]). *Assuming one-way functions exist, there exists a garbling scheme for the class of functions* $C : \mathbb{Z}_R \to \{0,1\}^{\lceil \log R \rceil \times \ell}$ [13] *specified by any modulus* R*, and any set of Boolean key functions* $K^{(i)} : \{0,1\} \to \{0,1\}^\ell$ *for* $i \in [\lceil \log R \rceil]$*:*

$$C(x) = \{K^{(i)}(\mathsf{Bits}(x)[i])\}.$$

The garbling size is $\mathrm{poly}(\log R, \ell, \lambda)$*.*

[13] Technically we are generalizing Definition 1 to allow functions with different input and output rings: $\mathbb{Z}_R$ and $\mathbb{Z}_2$.

Composing a bit-decomposition garbling scheme with Theorem 2 results in an arithmetic garbling scheme satisfying Definition 1, and creates only an additive cost of $|\mathbf{x}| \cdot \mathrm{poly}(\lambda)$ to the garbling size. We therefore obtain the following corollary.

Corollary 5 (Arithmetic Garbling for Small Modulus). *Let $R(\lambda) \leq \mathrm{poly}(\lambda)$ be a modulus. Assuming any of the assumptions in Theorem 2, there exists a garbling scheme for all arithmetic circuits C (with binary gates, ℓ_x inputs) over $\mathbb{Z}_R$ with garbling size*

$$|\widehat{C}| \leq |C| \cdot \log R + \ell_x \cdot \mathrm{poly}(\lambda).$$

The scheme assuming CP-DDH in prime-order groups has $1/\mathrm{poly}$ correctness and privacy errors, *which can be made negligible assuming a variant of CP-DDH (Definition 11).*

Alternatively, assuming any of the assumptions in Theorem 3, there exists a garbling scheme for all arithmetic circuits C (with binary gates, ℓ_x inputs) over $\mathbb{Z}_R$ with garbling size

$$|\widehat{C}| \leq |C| \cdot \log R + (\ell_x + \mathrm{Depth}(C)) \cdot \mathrm{poly}(\lambda).$$

Using Chinese Remainder Theorem, we can further compose multiple schemes, supporting co-prime polynomial moduli $\{R_i\}$, into one supporting a large modulus $R^* = \prod_i R_i$. We additionally show how to emulate an arbitrary modulus R using a sufficiently large one $R^* = O(R^2)$ in Section 6.1 (protocol $\mathsf{ArithCircEval}^C$, Figure 2). We show its instantiation under Paillier groups to illustrate our techniques. Other instantiations under prime-order groups and lattices differ only by how the public data are generated during the Init phase, analogous to the Boolean case. In summary, we obtain the following result.

Theorem 4 (Arithmetic Garbling for Large Modulus). *Assuming any of the assumptions in Theorem 2, there exists a garbling scheme for all arithmetic circuits C (with binary gates, ℓ_x inputs) over an arbitrary modulus $\mathbb{Z}_R$ with garbling size*

$$|\widehat{C}| \leq O(|C| \cdot \log R) + \ell_x \cdot \mathrm{poly}(\lambda, \log R).$$

The scheme assuming CP-DDH in prime-order groups has $1/\mathrm{poly}$ correctness and privacy errors, *which can be made negligible assuming a variant of CP-DDH (Definition 11).*

We can also obtain leveled variants analogous to the Boolean case to avoid circular assumptions at the cost of an additive $\mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda, \log R)$ term in the size of the garbling.

Theorem 5 (Leveled Arithmetic Garbling for Large Modulus). *Assuming any of the assumptions in Theorem 3, there exists a garbling scheme for all arithmetic circuits C (with binary gates, ℓ_x inputs) over an arbitrary modulus $\mathbb{Z}_R$ with garbling size*

$$|\widehat{C}| \leq O(|C| \cdot \log R) + (\ell_x + \mathrm{Depth}(C)) \cdot \mathrm{poly}(\lambda, \log R).$$

Note that we cannot use the schemes from Theorem 2 (and their leveled variants) to support general arithmetic gates with $\ell_x = \omega(1)$ inputs over polynomial modulus $R(\lambda) < \mathrm{poly}(\lambda)$, as their computation cost would become super-polynomial $R^{\ell_x} = \lambda^{\omega(1)}$. Therefore, we *do not* directly obtain analogous (to Theorem 2 and 4) arithmetic garbling schemes for layered circuits.

Protocol $\mathsf{ArithCircEval}^{C,\mathsf{Pai}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate an crithmetic circuit over $\mathbb{Z}_R$, $C : \mathbb{Z}_R^{\ell_x} \to \mathbb{Z}_R^{\ell_z}$. It uses the same ingradients as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 1), except with a different PRF:

- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to [R^2 \cdot \lambda^{\omega(1)}]$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \mathbb{Z}_R^{\ell_x}$, while P_E holds notinog.
Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \mathbb{Z}_R^{\ell_z}$.

- Init : P_G sends public data pd to P_E computed in the same way as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 1). Then P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle_1$ to P_E.

$$\overline{\mathbf{x}}[i] = \mathbf{x}[i] + r^{(i)} \bmod R,\ \langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle := s\mathsf{Bits}(\overline{\mathbf{x}}) + \langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle_0 \text{ over } \mathbb{Z},$$
$$\text{where } r^{(i)} \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)[i]),\ \langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle_0 \leftarrow [N\lambda^{\omega(1)}]^{\ell_x \cdot \lceil \log R \rceil}.$$

- Eval : P_G, P_E evaluate gates $\mathsf{g} \in C$ in the topological order while maintaining the following invariant:
  1. P_G, P_E jointly hold additive shares $\langle s\mathsf{Bits}(\overline{\mathbf{x}}_\mathsf{g})\rangle$, where $\overline{\mathbf{x}}_\mathsf{g}$ are masked input wire values to the gate g
  $$\overline{\mathbf{x}}_\mathsf{g}[i] = \mathbf{x}_\mathsf{g}[i] + r^{(i)} \bmod R \text{ where } r^{(i)} \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g})[i]). \tag{32}$$
  2. P_E holds the masked wire values $\overline{\mathbf{x}}_\mathsf{g}$.

  To evaluate the gate g, P_G, P_E jointly call the sub-protocol $\mathsf{ArithGateEval}$.

$$(P_G : \langle s\mathsf{Bits}(\overline{z}_\mathsf{g})\rangle_0), (P_E : \langle s\mathsf{Bits}(\overline{z}_\mathsf{g}\rangle_1, \overline{z}_\mathsf{g}))$$
$$\leftarrow \mathsf{ArithGateEval}^{C,\mathsf{g}}\left((P_G : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{\mathbf{x}}_\mathsf{g})\rangle_0), (P_E : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{\mathbf{x}}_\mathsf{g})\rangle_1, \overline{\mathbf{x}}_\mathsf{g})\right)$$

- Final : P_G sends masks $\mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(\mathsf{g}))$ mod R on all output gates $\mathsf{g} \in C$ to P_E, who can then recovers the output $\mathbf{z}$ by removing the masks mod R.

Fig. 11. 2PC protocol for Arithmetic circuits with large modulus.

6.1 Handling Large R using Chinese Remainder Theorem

The overall protocol $\mathsf{ArithCircEval}^{C,\mathsf{Pai}}$ (under Paillier groups) is shown in Figure 2. It's mostly the same as the Boolean protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ except how wire values are masked.

- In the Boolean protocol, each wire value (on wire i) is masked by a single bit derived by $\mathsf{PRF}(\mathsf{sk}, i)$.
- In the arithmetic protocol, each wire value (on wire i) is masked by an integer mod R derived by $\mathsf{PRF}(\mathsf{sk}, i)$.

The wire values are always represented as bits. In particular, the input shares during the Init phases are defined as additive shares of bit representations of the masked inputs. We can still compile such a protocol to an arithmetic garbling scheme with arithmetic input labels, relying on existing techniques [AIK11, LL24, Hea24] as explained in Section 6.

As in the Boolean case, we rely on a core sub-protocol $\mathsf{ArithGateEval}^{C,\mathsf{g}}$ (Figure 12) to evaluate arithmetic gates ($+$, $\times$). It proceeds in the following steps.

- First find enough number of $O(\log \lambda)$-bit primes $\{p_i\}$ such that their product $Q = \prod_i p_i$ is sufficiently large, $Q > R^2 \cdot \lambda^{\omega(1)}$. Let ℓ be the number of primes needed. Also define CRT representations with respect to Q as

$$\mathsf{CRT}(x) = \{x_i\} \text{ where } x_i := x \bmod p_i, \quad \mathsf{CRT}^{-1}(\{x_i\}) = x, \tag{33}$$

Sub-protocol $\mathsf{ArithGateEval}^{C,\mathsf{g}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate an arithmetic gate ($+$ or $\times$) $\mathsf{g} \in C$.

Inputs: P_G, P_E both hold public data $\mathsf{pd} = (\mathsf{aHMAC.pd}, \mathsf{HSS.pd}_{\mathsf{sk}})$ (as defined in Equation 3), and jointly hold additive shares $\langle s\mathsf{Bits}(\overline{x})\rangle$, $\langle s\mathsf{Bits}(\overline{y})\rangle$, where $\overline{x}, \overline{y} \in \mathbb{Z}_R$ are masked inputs. P_E additionally holds the values $\overline{x}$, $\overline{y}$.

Outputs: P_G, P_E jointly output additive shares $\langle s\mathsf{Bits}(\overline{z})\rangle$, where $\overline{z} \in \mathbb{Z}_R$ is the masked output. P_E additionally holds the value $\overline{z}$.

- Let $\mathsf{CRT}, \mathsf{CRT}^{-1}$ be functions defined in Equation 33, and C^{CRT}, C^{InvCRT} be Boolean circuits implementing them (Equation 34).
- P_G, P_E obtain additive shares $\langle s\mathsf{Bits}(\mathsf{CRT}(\overline{x}))\rangle$ through local computations:

$$\begin{aligned} P_G &: \{\langle s\mathsf{Bits}(\overline{x}_i)\rangle_0\} \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C^{\mathsf{CRT}}, \langle s\mathsf{Bits}(\overline{x})\rangle_0), \\ P_E &: \{\langle s\mathsf{Bits}(\overline{x}_i)\rangle_1\} \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C^{\mathsf{CRT}}, \langle s\mathsf{Bits}(\overline{x})\rangle_1, \overline{x}), \end{aligned}$$

  where $\overline{x}_i := \overline{x} \bmod p_i$. Similarly obtain shares of $\langle s\mathsf{Bits}(\mathsf{CRT}(\overline{y}))\rangle$.
- $\forall i \in [\ell]$, P_G, P_E apply $\mathsf{BoolGateEval}'$ over the shares $\langle s\mathsf{Bits}(\overline{x}_i)\rangle$, $\langle s\mathsf{Bits}(\overline{y}_i)\rangle$.

$$\begin{aligned} &(P_G : \langle s\mathsf{Bits}(\overline{z}_i)\rangle_0), (P_E : \langle s\mathsf{Bits}(\overline{z}_i)\rangle_0, \overline{z}_i) \\ &\quad \leftarrow \mathsf{BoolGateEval}'^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{x}_i, \overline{y}_i)\rangle_0, \\ &\qquad\qquad (P_E : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{x}_i, \overline{y}_i)\rangle_1, \overline{x}_i, \overline{y}_i), \big) \end{aligned}$$

  where $\mathsf{BoolGateEval}'$ is a slight variant of $\mathsf{BoolGateEval}$ (Figure 3) as explained in Section 6.1.
- P_G, P_E obtain additive shares $\langle s\mathsf{Bits}(\overline{z}')\rangle$ where $\overline{z}' := \mathsf{CRT}^{-1}(\{\overline{z}_i\})$ through local computations.

$$\begin{aligned} P_G &: \langle s\mathsf{Bits}(\overline{z}')\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C^{\mathsf{InvCRT}}, \{\langle s\mathsf{Bits}(\overline{z}_i)\rangle_0\}), \\ P_E &: \langle s\mathsf{Bits}(\overline{z}')\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C^{\mathsf{InvCRT}}, \{\langle s\mathsf{Bits}(\overline{z}_i)\rangle_1\}, \{\overline{z}_i\}), \end{aligned}$$

  Then obtain additive shares $\langle s\mathsf{Bits}(\overline{z})\rangle$ where $\overline{z} := \overline{z}' \bmod R$ through local computations.

$$\begin{aligned} P_G &: \langle s\mathsf{Bits}(\overline{z})\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, \bmod R, \langle s\mathsf{Bits}(\overline{z}')\rangle_0), \\ P_E &: \langle s\mathsf{Bits}(\overline{z})\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, \bmod R, \langle s\mathsf{Bits}(\overline{z}')\rangle_1, \overline{z}'). \end{aligned}$$

Fig. 12. 2PC protocol for Arithmetic gates.

and Boolean circuits computing those conversions.

$$\begin{aligned} C^{\mathsf{CRT}}(\mathsf{Bits}(x)) &:= \{\mathsf{Bits}(x \bmod p_i)\}_i = \mathsf{Bits}(\mathsf{CRT}(x)), \\ C^{\mathsf{InvCRT}}(\mathsf{Bits}(\mathsf{CRT}(x))) &:= \mathsf{Bits}(x). \end{aligned} \tag{34}$$

- P_G, P_E apply the aHMAC evaluations locally on the shares $\langle s\mathsf{Bits}(\overline{x})\rangle$, $\langle s\mathsf{Bits}(\overline{y})\rangle$ to obtain shares of their CRT representations:

$$\begin{array}{ll} \text{Input} & \langle s\mathsf{Bits}(\overline{x})\rangle, \langle s\mathsf{Bits}(\overline{y})\rangle \\ \text{via aHMAC} & \{\langle s\mathsf{Bits}(\overline{x}_i)\rangle\}, \{\langle s\mathsf{Bits}(\overline{y}_i)\rangle\}. \end{array}$$

- After decomposing the large values $\overline{x}, \overline{y}$ into small CRT representations, P_G, P_E jointly call $\mathsf{BoolGateEval}'^{C,\mathsf{g}}$ to evaluate the gate function g on each CRT components $\overline{x}_i$, $\overline{y}_i$:

$$\text{via } \mathsf{BoolGateEval}' \quad \{\langle s\mathsf{Bits}(\overline{z}_i)\rangle\},$$

  where $\mathsf{BoolGateEval}'^{C,\mathsf{g}}$ is the same as $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ (Figure 3) except using a different Boolean circuit $C'_{g,\mathbf{v}}(\mathsf{sk})$ defined as follows.

1. Parse $\mathbf{v}$ as two values $\overline{x}_i, \overline{y}_i \in \mathbb{Z}_{p_i}$.
2. Compute x'_i, y'_i as

$$x'_i = \overline{x}_i - (r^x \bmod R), \quad y'_i = \overline{y}_i - (r^y \bmod R) \text{ where}$$
$$r^x \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)[0]), \quad r^y \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)[1]).$$

3. Outputs $\mathsf{Bits}(\overline{z}_i)$ where $\overline{z}_i$ is computed as

$$\overline{z}_i = \mathsf{g}(x'_i, y'_i) + r^{(z)} \bmod p_i, \text{ where } r^z \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)).$$

We note two facts of the values $\overline{z}_i$ computed by $\mathsf{BoolGateEval}'^{C,\mathsf{g}}$.

$$\begin{aligned} &\mathsf{CRT}^{-1}(\{\overline{z}_i\}) \equiv g(x,y) + \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \bmod R, \\ &|\mathsf{CRT}^{-1}(\{\overline{z}_i\})| \leq R^2 \cdot \lambda^{\omega(1)}. \end{aligned} \tag{35}$$

where x, y are the actual wire values to the gate g.

– Finally, convert the shares of small CRT components $\langle s\overline{z}_i \rangle$ back into a share of an integer, and then compute the mod R circuit on it.

$$\begin{aligned} &\text{via aHMAC} \quad \{\langle s\mathsf{Bits}(\overline{z}') \rangle\}, \text{ where } \overline{z}' = \mathsf{CRT}^{-1}(\{\overline{z}_i\}) \\ &\text{via aHMAC} \quad \{\langle s\mathsf{Bits}(\overline{z}) \rangle\}, \text{ where } \overline{z} = \overline{z}' \bmod R. \end{aligned}$$

The security of the subprotocol $\mathsf{ArithGateEval}^{C,\mathsf{g}}$ and of the overall protocol $\mathsf{ArithCircEval}^{C,\mathsf{Pai}}$ can be proved analogously to the Boolean case. Hence we omit them here.

7 Concrete Efficiency Analysis

In this section, we analyze the concrete garbling sizes of our non-leveled schemes, which corresponds to the communication sizes in the protocols $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 1, 2), $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$ (Figure 9), and $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$ (Figure 7). They consist of two parts: (1) 1-bit per gate in the circuit (during the Eval phase), and (2) public data pd (during the Init phase). We analyze the size of the public data pd in different instantiations below, and summarize them in Table 3.

	Concrete Size	Asymptotic
Ours (Paillier)	0.38 MB	$8\lambda\lceil \log N \rceil$
Ours (Prime-Order)	5.1 MB	$(4/\beta)\lambda^2\lceil \log p \rceil$
size opt. ver.	0.13 MB	
Ours (Lattice)	71 MB	$4\lambda n\lceil \log q \rceil$
[LWYY24] (Lattice)	10 GB	$4\lambda n\lceil \log q \rceil^2$

Table 3. Concrete sizes for the public data pd in our non-leveled schemes, and an optimistic estimation for [LWYY24]. Our scheme under prime-order groups has a $1/\mathrm{poly}$ correctness and privacy error. Here N is the Paillier modulus, p is the prime-order group size, β denotes digit-decomposition by 2^β (explained below), and n, q are the degree and the modulus of the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$.

In the following, we use $\lambda = 128$ as the computational security parameter, and $\kappa = 40$ as the statistical security parameter. We also recall the following parameters:

– Under Paillier groups, N and ζ specify the group $\mathbb{Z}^*_{N^{\zeta+1}}$.
– Under prime-order groups, p denotes the group size.
– Under lattices, n and q specify the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$, where the degree n is a power-of-two.

Paillier Groups Instantiation. The public data for aHMAC evaluations contains a λ-bit seed, and $3\lceil \log N \rceil$ elements in $\mathbb{Z}^*_{N^{\zeta+1}}$. (See Lemma 2.) The public data for HSS evaluations can share the seed from aHMAC, and additionally contains 4λ elements in $\mathbb{Z}_{N^{\zeta+1}}$. In total:

$$|\mathsf{pd}^{\mathsf{Pai}}| = \lambda + (3\lceil \log N \rceil + 4\lambda) \cdot (\zeta+1) \cdot \lceil \log N \rceil.$$

An optimization to reduce this size is to compress the public data of aHMAC. As long as the order of the sub-group generated by $(1+N)$ is sufficiently larger than the secret exponent s, i.e., $N^\zeta \gg |s|$, we can compress the public data from consisting $3\lceil \log N \rceil$ elements to 3 elements:

$$g^{r^*},\ g^{r^* s},\ g^{r^* s^2} \cdot (1+N)^s, \text{ where } r^* = \sum_i \mathbf{r}[i], \text{ and } \mathbf{r} \leftarrow [N]^{\lceil \log N \rceil}.$$

Note that the compressed version of pd can be derived from the original, hence is still secure. Furthermore, if aggressively assuming the secret exponent in CP-DDH only needs to have 2λ [14] instead of $\lceil \log N \rceil$ bits, it suffices to set $\zeta = 1$ to guarantee $N^\zeta = N \gg 2\lambda$. In total

$$|\mathsf{pd}^{\mathsf{Pai}}| = \lambda + (3+4\lambda) \cdot 2 \cdot \lceil \log N \rceil.$$
$$\text{// w/ compressed } \mathsf{aHMAC}\ \mathsf{pd} \text{ and small exponents.}$$

Concretely, we set the Paillier modulus N to have 3072 bits (which is believed to provides 128 bits of security), which gives $|\mathsf{pd}^{\mathsf{Pai}}| = 0.38\text{MB}$.

Prime-Order Groups Instantiation. The public data for aHMAC evaluations contains a λ-bit seed, and $3\lceil \log p \rceil$ elements in $\mathbb{Z}_p$. (See Lemma 6.) The public data for HSS evaluations can share the seed from aHMAC, and additionally contains $2\lambda + 2\lambda\lceil \log p \rceil$ elements in $\mathbb{Z}_p$. (See Lemma 9.) In total:

$$|\mathsf{pd}^{\mathsf{Pri}}| = \lambda + (3\lceil \log p \rceil + 2\lambda + 2\lambda\lceil \log p \rceil) \cdot \lceil \log p \rceil.$$

We can reduce this size by similarly assuming the secret exponent only needs 2λ instead of $\lceil \log p \rceil$ bits. Furthermore, we can considering digit-decomposition instead of bit-decomposition of the secret s. When using a base 2^β, the public data for aHMAC now only needs to contain $6\lambda/\beta$ elements, and the public data for HSS only needs to contain $2\lambda + (4/\beta)\lambda^2$ elements. [15] A final optimization is to use a random oracle (RO) to obtain the first elements in all ElGamal ciphertexts for free, as suggested in [BGI16], which reduces the public data for HSS by a factor of 2. In total:

$$|\mathsf{pd}^{\mathsf{Pri}}| = \lambda + \left((6/\beta+1)\lambda + (2/\beta)\lambda^2\right) \cdot \lceil \log p \rceil.$$
$$\text{// w/ small exponents, digit decomposition, and RO.}$$

Concretely, we consider two settings. First, optimizing for computation time, we follow the optimized implementation from [BGI17] to use "conversion friendly" primes for p. As noted

[14] We estimate a 2λ-bit exponent to have λ-bit security following the estimation for small-exponent ElGamal in [BGI17].

[15] As a consequence of using digit decomposition, the computation cost of our scheme will increase (by a at least a factor of 2^β).

there, compared to a general prime, such conversion friendly primes needs to have a 50% larger bit-length to provide a similar level of security. Therefore, we estimate $\lceil \log p \rceil = 5000$ to provide 128 bits of security. We also follow [BGI17] to set $\beta = 4$, which gives $|\mathsf{pd}^{\mathsf{Pri}}| = 5.07$MB.

Second, optimizing for garbling size, we choose elliptic curves of 256-bit as the prime-order group, and more aggressively set $\beta = 8$, which gives $|\mathsf{pd}^{\mathsf{Pri}}| = 0.13$MB.

Lattice Instantiation. The public data for aHMAC evaluations contains a λ-bit seed, and 3 elements in $\mathcal{R}_q$. (See Lemma 11.) The public data for HSS evaluations can share the seed from aHMAC, and additionally contains 4λ elements in $\mathcal{R}_q$. (See Lemma 14.) In total:

$$|\mathsf{pd}^{\mathsf{Lat}}| = \lambda + (3 + 4\lambda) \cdot n \cdot \lceil \log q \rceil.$$

Concretely, we follow [BKS19] to use uniform ternary secrets with coefficients from $\{0, -1, 1\}$, and rounded Gaussian error distributions with parameter $\sigma = 8/\sqrt{2\pi}$. We choose a modulus with $\lceil \log q \rceil = 142$ bits, and the polynomial ring with degree $n = 2^{13} = 8192$. These settings are estimated [16] to achieve 128 bits of security, and a correctness error 2^{-40}. We get $|\mathsf{pd}^{\mathsf{Lat}}| = 71.42$MB.

Comparing with the Scheme of [LWYY24] Based on FHE. The scheme of [LWYY24] is based on the GSW [GSW13] fully homomorphic encryption (FHE) scheme (and assuming its KDM-security). Under the RLWE version of GSW, the garbling material contains 1 bit per gate, a λ-bit seed, public parameters pp, and λ FHE ciphertexts. We refer to the seed, pp and the ciphertexts as the public data $\mathsf{pd}^{\mathsf{GSW}}$ in this scheme. In more detail, pp consists of $2 + 8\lceil \log q \rceil$ elements in $\mathcal{R}_q$, and each ciphertext consists of $4\lceil \log q \rceil$ elements in $\mathcal{R}_q$. In total:

$$|\mathsf{pd}^{\mathsf{GSW}}| = \lambda + (2 + 8\lceil \log q \rceil + 4\lceil \log q \rceil \lambda) \cdot n \cdot \lceil \log q \rceil.$$

Compared to our lattice instantiation, the public data in [LWYY24] is asymptotically greater by a factor of $\log q$, assuming the polynomial ring degree n and the modulus q being equal.

In the GSW FHE scheme, the modulus q not only needs to satisfy a similar set of constraints to our lattice instantiation, but also needs to support homomorphic evaluations of a low-depth PRG. Therefore, we expect concretely the scheme of [LWYY24] needs a much larger q than ours, and consequently also a larger n, to achieve 128 bits of estimated security. Optimistically, under the same settings of $\lceil \log q \rceil = 142$ and $n = 2^{13} = 8192$, we get $|\mathsf{pd}^{\mathsf{GSW}}| = 10.00$GB.

Acknowledgments. Y. Ishai was supported by ISF grants 2774/20 and 3527/24, BSF grant 2022370, and ISF-NSFC grant 3127/23. H. Lin and H. Li were supported by NSF grant CNS-2026774, and a Simons Collaboration on the Theory of Algorithmic Fairness.

[16] Using the LWE security estimator: https://github.com/malb/lattice-estimator

Bibliography

[ABG+24] Amit Agarwal, Elette Boyle, Niv Gilboa, Yuval Ishai, Mahimna Kelkar, and Yiping Ma. Compressing unit-vector correlations via sparse pseudorandom generators. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VIII*, volume 14927 of *LNCS*, pages 346–383. Springer, Cham, August 2024.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 421–452. Springer, Cham, August 2022.

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In Bernard Chazelle, editor, *ICS 2011*, pages 45–60. Tsinghua University Press, January 2011.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 260–274. IEEE Computer Society, 2005.

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

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 166–184. Springer, Berlin, Heidelberg, August 2013.

[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, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 543–574. Springer, Cham, August 2018.

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

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Berlin, Heidelberg, April 2015.

[ARS24] Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 301–330. Springer, Cham, May 2024.

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

[BCG+18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishabil-

ity obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018.

[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 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, 2014.

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

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

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Berlin, Heidelberg, August 2008.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

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

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Cham, April / May 2018.

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

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

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

[BMZ19] James Bartusek, Fermi Ma, and Mark Zhandry. The distinction between fixed and random generators in group-based assumptions. In Alexandra Boldyreva and

Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 801–830. Springer, Cham, August 2019.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[CCH+24] Mingyu Cho, Woohyuk Chung, Jincheol Ha, Jooyoung Lee, Eun-Gyeol Oh, and Mincheol Son. FRAST: tfhe-friendly cipher based on random s-boxes. *IACR Trans. Symmetric Cryptol.*, 2024(3):1–43, 2024.

[CCKK21] Jung Hee Cheon, Wonhee Cho, Jeong Han Kim, and Jiseung Kim. Adventures in crypto dark matter: Attacks and fixes for weak pseudorandom functions. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 739–760. Springer, Cham, May 2021.

[CEMY09] Seung Geol Choi, Ariel Elbaz, Tal Malkin, and Moti Yung. Secure multi-party computation minimizing online rounds. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 268–286. Springer, Berlin, Heidelberg, December 2009.

[CHHK25] Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. $o(1/\lambda)$-rate boolean garbling scheme from generic groups. Cryptology ePrint Archive, Paper 2025/268, 2025.

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

[CNs07] Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 573–590. Springer, Berlin, Heidelberg, May 2007.

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

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

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563. ACM, 1994.

[FLLL24] Ximing Fu, Mo Li, Shihan Lyu, and Chuanyi Liu. Bit-fixing correlation attacks on goldreich's pseudorandom generators. *IACR Cryptol. ePrint Arch.*, page 1594, 2024.

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

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

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 537–566. Springer, Cham, November 2017.

[GJM03] Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In Matt Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 120–135. Springer, Berlin, Heidelberg, March 2003.

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

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

[GN25] Jian Guo and Wenjie Nan. Efficient mixed garbling from homomorphic secret sharing and GGM-tree. Cryptology ePrint Archive, Paper 2025/207, 2025.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 430–443. ACM Press, October 2016.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499. Springer, 2018.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 3–31. Springer, Cham, May 2024.

[HIKR23] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Additive randomized encodings and their applications. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 203–235. Springer, Cham, August 2023.

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

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

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 419–428. ACM Press, June 2015.

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

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

[KY18] Ilan Komargodski and Eylon Yogev. Another step towards realizing random oracles: Non-malleable point obfuscation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 259–279. Springer, Cham, April / May 2018.

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

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. Garbled circuits with 1 bit per gate. Cryptology ePrint Archive, Paper 2024/1988, 2024.

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part IV*, volume 15367 of *Lecture Notes in Computer Science*, pages 71–97. Springer, 2024.

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

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

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

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Berlin, Heidelberg, May 1999.

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

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Cham.

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021,*

Part III, volume 12827 of *LNCS*, pages 687–717, Virtual Event, August 2021. Springer, Cham.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Berlin, Heidelberg, May 1997.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 463–472. ACM Press, October 2010.

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

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